UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

VINICIUS MOREIRA

ESTUDO E APLICAÇÃO DE *TUNING* EM SISTEMA GERENCIADOR DE BANCO DE DADOS ORIENTADO A GRAFOS

PONTA GROSSA

2023

VINICIUS MOREIRA

ESTUDO E APLICAÇÃO DE *TUNING* EM SISTEMA GERENCIADOR DE BANCO DE DADOS ORIENTADO A GRAFOS

Study and application of tuning in a graph-oriented database manager system

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação da Universidade Tecnológica Federal do Paraná (UTFPR).
Orientador: Tarcizio Alexandre Bini.

PONTA GROSSA

2023

VINICIUS MOREIRA

ESTUDO E APLICAÇÃO DE *TUNING* EM SISTEMA GERENCIADOR DE BANCO DE DADOS ORIENTADO A GRAFOS

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 23/maio/2023

_____

Tarcizio Alexandre Bini
Doutorado
Universidade Tecnológica Federal do Paraná

_____

Simone de Almeida
Doutorado
Universidade Tecnológica Federal do Paraná

_____

João Paulo Aires
Doutorado
Universidade Tecnológica Federal do Paraná

PONTA GROSSA

2023

RESUMO

O volume de dados armazenados em bancos cresce exponencialmente a cada dia. Para o gerenciamento de grandes quantidades de dados, o modelo relacional apresenta limitações quanto à escalabilidade, o que levou ao surgimento de novas soluções, como é o caso dos Sistemas Gerenciadores de Banco de Dados NoSQL. Quando há necessidade de gerenciar dados altamente conectados, como por exemplo, redes sociais, o modelo de armazenamento mais adequado a ser utilizado é o orientado a grafos. Porém, mesmo ele executando satisfatoriamente em sua área de aplicação, há possibilidade de melhorias no quesito desempenho, trazendo assim, vantagens competitivas para as empresas. Uma forma de melhorar o desempenho dos Sistemas Gerenciadores de Banco de Dados é ajustar e modificar o sistema gerenciador, sistema operacional, esquema da base, ou equipamento disponível. O objetivo deste trabalho é aplicar ajustes em um Sistema Gerenciador de Banco de Dados orientado a grafos. Para isso, foi realizado o levantamento de possíveis técnicas a serem empregadas juntamente com a criação de um ambiente de testes. Os experimentos demonstram redução de mais de 80 minutos no tempo de execução de 57,14% das consultas da carga de trabalho, ao alterar as configurações de uso de memória RAM. Quando se trata de índices, a adição de novos tipos reduziu em torno de 72 minutos o tempo de execução de 66,87% das consultas da carga de trabalho utilizada.

Palavras-chave: Sistemas Gerenciadores de Banco de Dados; NoSQL; Grafos; Otimização de desempenho; Memória RAM; Índices.

ABSTRACT

The volume of data stored in database grows exponentially every day. For the management of large amounts of data, the relational model has limitations related to scalability, which led to the emergence of new solutions, as is the case of NoSQL Database Management Systems. When there is a need to manage highly connected data, such as social networks, the most suitable storage model to use is the graph-oriented one. However, even though it performs satisfactorily in his application area, there is the possibility of improvements in terms of performance, thus bringing competitive advantages to companies. One way to improve the performance of Database Management Systems is adjust and modify the management system, the operating system, base schema, or available hardware. The objective of this work is to apply tuning in a graph-oriented Database Management System. For this, a survey of possible techniques to be applied was carried out together with the creation of a test environment. The experiments demonstrate a reduction of more than 80 minutes in the execution time of 57.14% of the workload queries, when changing the RAM memory usage settings. When it comes to indexes, the addition of new types reduced the execution time of 66.87% of queries in the workload used by around 72 minutes.

Keywords: Database Management System; NoSQL; Graph; Tuning; RAM memory; Index.

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE ABREVIATURAS E SIGLAS

ACID	Atomicidade, Consistência, Isolamento, Durabilidade
BASE	*Basically Available, Soft State, Eventual Consistency*
BI	*Business Inteligence*
BSON	*Binary* JSON
CAP	*Consistency*, *Availability*, *Partition Tolerance*
CMS	*ContentManagementSystem*
COMOM	*Commission Communautaire Commune*
DBA	*Database Administrator*
GB	*Giga Byte*
GiB	*Giga Binary Byte*
JSON	*JavaScript Object Notation*
JVM	*Java Virtual Machine*
LDBC	*Linked Data Benchmark Council*
MB	*Mega Byte*
NOSQL	*Not Only SQL*
OLAP	*Online Analytical Processing*
OLTP	*Online Transaction Processing*
RAM	*Random Access Memory*
RPM	Rotações Por Minuto
SF	*Scale Factor*
SGBD	Sistemas Gerenciadores de Banco de Dados
SGBDR	Sistemas Gerenciadores de Banco de Dados Relacionais
SNB	*Social Network Benchmark*
SO	Sistema Operacional
SQL	*Structured Query Language*
TCO	*Total Cost of Ownership*
TPC	*Transaction Processing Performance Council*)
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Desde que surgiram, em meados da década de 70, os Sistemas Gerenciadores de Banco de Dados Relacionais (SGDBR) são aplicados em larga escala (DB-ENGINES RANKING, 2023) para o armazenamento e gestão de massas de dados. Possuem como características marcantes o emprego do modelo relacional como forma de armazenamento (SILBERSCHATZ, 2020) e uma linguagem padrão de consultas e manipulação de dados, o SQL (*Structured Query Language*). Também caracterizam-se pela forte consistência e integridade dos dados no gerenciamento de transações, pautado nas propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) (ZDEPSKI; BINI; MATOS, 2018).

Porém, segundo Sadalage e Fowler (2019), os SGBDRs possuem limitações como a incompatibilidade de impedância: simplicidade da tipagem relacional (inteiro, real, caracter), não podendo armazenar nada diferente de tuplas. Caso seja necessário armazenar algo que não tem a mesma estrutura das tuplas relacionais, como objetos da programação orientada a objeto, é preciso fazer a tradução desse tipo de dado, que pode ser mapeado por *frameworks*, como o Hibernate (HIBERNATE, 2022) e o iBATIS (IBATIS, 2022).

Outra limitação está relacionada à escalabilidade (BRITO, 2010). Devido ao crescente aumento do volume de dados e número de usuários, a queda de desempenho é evidente nesse tipo de base de dados. Distribuir o armazenamento em vários nós (escalabilidade horizontal) ou realizar um upgrade no servidor (escalabilidade vertical) apresentam-se como alternativas para obter ganhos de desempenho. Para Sadalage e Fowler (2019), o escalonamento pode não ser plausível ou mesmo tornar-se inviável: no caso do escalonamento vertical, o custo financeiro pode se tornar elevado. Quanto ao escalonamento horizontal, os SGBDRs priorizam a consistência das bases, pelo controle de concorrência no acesso e atualização dos dados, não operando bem de forma distribuída no quesito desempenho.

Como alternativa às soluções relacionais, surgiram uma ampla classe de SGBDs que não utilizam o modelo relacional, tão pouco esquemas, como forma de armazenamento, chamada de NoSQL (*Not Only SQL*). Motivados pela escalabilidade, possibilitam o armazenamento em varias máquinas (*clusters*), pois

foram projetados para executarem de forma distribuída, tornando-se assim uma boa opção para melhoria de desempenho com menor custo. Segundo Sadalage e Fowler (2019), a categorização dos SGBDs NoSQL pode ser realizada de acordo com quatro principais modelos de armazenamento: chave-valor, família de colunas, orientado a documentos e orientado a grafos.

Bancos de dados chave-valor utilizam uma coleção de chaves únicas e um conjunto de valores associados a elas. Na orientação a documentos, o armazenamento de dados é estruturado (listas, registros aninhados) e salvo como coleções em documentos, geralmente nos formatos JSON (*JavaScript Object Notation*), XML (*Extensible Markup Language*) e BSON (*Binary* JSON). Logo, SGBDs orientados a famílias de colunas são capazes de armazenar e carregar colunas inteiras de dados por meio de uma chave.

Os bancos de dados orientados a grafo utilizam a teoria dos grafos como forma de armazenamento. Sendo a representação de um conjunto de objetos, onde pares de objetos são conectados por ligações, um grafo é um par de conjuntos (V, E), onde V é o conjunto de vértices e E é o conjunto de arestas (do inglês, *edges*) (PI; KOROGLU, 2017). Para a classificação, os vértices são conhecidos como nós e armazenam os dados, já as arestas definem o relacionamento entre os nós e também podem conter dados associados.

Independentemente de qual modelo de armazenamento for empregado, obter redução no tempo de resposta de comandos submetidos à base de dados é essencial às aplicações modernas. Para este intuito, administradores de bancos de dados (*Database Administrator - DBAs*) podem fazer uso de técnicas de *tuning*, que segundo Tramontina (2008, p. 1), "é um processo de refinamento que envolve modificações em vários aspectos". Tais modificações podem ser efetivadas refinando-se o esquema do banco de dados e consultas submetidas, efetuando-se configurações nos parâmetros do SGBD e/ou Sistema Operacional (SO), entre outros. Portanto, de forma resumida, este trabalho buscou com a aplicação de técnicas de *tuning* em SGBDs NoSQL orientados a Grafos, obter ganhos de desempenho, fator este, muito desejável por empresas que buscam vantagens competitivas.

1.1 Objetivos

Aplicar técnicas de *tuning* em um SGBD não relacional orientado a Grafos, na tentativa de reduzir o tempo de execução dos comandos a ele submetidos.

Os objetivos específicos deste trabalho são:

- Aplicar o modelo de armazenamento de dados NoSQL orientado a Grafos;
- Analisar e escolher o SGBD orientado a Grafos para ser utilizado nos experimentos;
- Levantar possíveis técnicas de *tuning* para serem aplicadas ao SGBD escolhido;
- Definir cenários para realização de testes de desempenho;
- Aplicar técnicas de *tuning* sobre o ambiente de armazenamento;
- Avaliar os resultados obtidos verificando se houve ou não melhoria de desempenho.

1.2 Justificativa

O modelo orientado a Grafos permite o armazenamento de dados, ao criar interligações entre os nós de forma simples com desempenho superior em operações de escrita, quando comparado ao modelo relacional, pois para escrever um novo nó, é necessário gerar um agregado de dados e relacioná-lo a um nó existente na base de dados, dispensando qualquer verificação em relação a esquemas ou tipagem de dados. SGBDs NoSQL orientados a Grafos são amplamente aplicados para a detecção de fraudes, buscas orientadas a grafos, mídias sociais e sistemas de recomendações de produtos (PI; KOROGLU, 2017). Desta forma, pesquisar e aplicar técnicas de *tuning* para SGBDs NoSQL orientado a Grafos é interessante, tanto para a comunidade científica quanto empresarial.

Pesquisas relacionadas à *tuning* em SGBDRs, como Dominico (2013) e Bini (2014), apresentam redução no tempo de execução em consultas submetidas aos SGBDRs. Porém, o levantamento bibliográfico exigido por este trabalho demonstrou um número inferior de trabalhos relacionados ao *tuning* de bases orientadas a Grafos em comparação às bases relacionais.

Há registros sobre mudanças nas configurações de SGBDs orientados a Grafos, que podem ser aplicadas para melhorar seu desempenho sob cargas de trabalho do tipo OLAP (*Online Analitical Processing*). Um exemplo no Neo4j (NEO4J, 2022) é o uso de nós escravos replicados (vértices em outros participantes do *cluster*) (SALADAGE; FOWLER, 2019), utilizados para aumentar a disponibilidade, sincronizando a gravação de conteúdo com o nó mestre, podendo ser empregados apenas para a leitura de dados, diminuindo assim o tempo de execução de consultas. Outro registro que trata a respeito da otimização de desempenho de bases orientadas a Grafos é o capítulo 11 do guia do usuário do SGBD Neo4J (NEO4J LABS, 2022). Neste capítulo é abordado o tema Inferência/Raciocínio (*Inferencing/Reasoning*): descobrir um conjunto de dados que não estão explicitamente armazenados utilizando hierarquia de categorias. O guia descreve como utilizar hierarquias para consultar detalhadamente a base de dados, trazendo o máximo de dados relevantes, e também para atualizá-la de uma forma otimizada, melhorando o desempenho nesta tarefa.

A partir de buscas em manuais e demais referências, a contribuição deste trabalho tanto para a sociedade quanto para a comunidade acadêmica é a aplicação e análise de técnicas de *tuning* sugeridas na literatura em SGBDs orientados a grafos. Os resultados obtidos e as técnicas de *tuning* empregadas foram documentados e servem de auxílio à DBAs e pesquisadores da área de banco de dados que objetivam melhorias de desempenho em suas bases de dados. Os experimentos realizados poderão ser replicados seguindo os passos do Apêndice A.

1.3 Organização do trabalho

O presente trabalho encontra-se dividido da seguinte forma. O Capítulo 2 define os modelos de armazenamento comumente encontrados na literatura. No Capítulo 3, é apresentada a escolha devidamente justificada do SGBD empregado no trabalho, assim como sua história e aplicação no mercado, descrevendo suas principais funcionalidades. O Capítulo 4 define e detalha as técnicas de *tuning* empregadas no SGBD, justifica suas escolhas e importância. No Capítulo 5 é apresentado o ambiente experimental, assim como são detalhados os resultados obtidos pela aplicação das técnicas de *tuning* no SGBD orientado a grafos. O

Capítulo 6 discute as conclusões obtidas nesta monografia, assim como apresenta possibilidades de trabalhos futuros.

2 MODELOS DE ARMAZENAMENTO DE DADOS

Criados com o objetivo de descrever como os dados são organizados, os modelos de armazenamento determinam o comportamento de um SGBD, e categorizam-se pela forma que estruturam os dados e por quais operações fornecem suporte (MICROSOFT, 2022). Inserções, remoções, atualizações e consultas são realizadas com base no modelo de armazenamento empregado.

O modelo relacional por sua vez, foi criado no ano 1970 com base na teoria dos conjuntos e álgebra linear (CODD, 1970). O modelo emprega relações (tabelas), que são compostas por tuplas (linhas) e atributos (colunas), caracterizadas pela simplicidade dos dados (inteiro, real, caracter) (SADALAGE; FOWLER, 2019). Uma de suas principais características é a restrição de integridade (chaves primárias, chave estrangeiras) (BRITO, 2010). As chaves primárias asseguram a identificação única das tuplas, enquanto as chaves estrangeiras criam uma dependência do valor à um atributo contido em outra relação. É possível observar na Figura 1 a disposição de uma relação no modelo relacional onde o “ID” é a chave primaria que identifica a tupla.

Figura 1 – Exemplo de uma tabela no modelo relacional

ID	nome	email	cargo
0000000011	Silvana Moreira	sm@hotmail.com	CargoExemplo1
0000000016	Vinicius Moreira	viynni1@hotmail.com	CargoExemplo2
0000000006	Vinicius Moreira	vinim@hotmail.com	CargoExemplo1
0000000017	Gustavo Henrique Salina	gustavo@leno.com	CargoExemplo3
0000000023	teste	teste@teste	CargoExemplo2

Fonte: Autoria própria (2022)

Os SGBDRs utilizam o modelo relacional e a maioria emprega a SQL como linguagem padrão. Geralmente o esquema é definido antecipadamente e todas as transações, ou seja, toda e qualquer atividade que o próprio SGBD executa após o usuário ter uma interação com o banco (BARBOSA, 2018), devem utilizá-lo (MICROSOFT, 2022). De acordo com Khasawneh (2020) alguns dos exemplos de SGBDRs são: Oracle (ORACLE, 2022), PostgreSQL (POSTGRESQL, 2022) e Microsoft SQL Server (MICROSOFT, 2022).

A integridade e a consistência dos dados são garantidas em qualquer transação. Isto se deve as propriedades ACID, que são essenciais para o desenvolvimento de sistemas bancários, financeiros e de segurança (MUS, 2019).

- Atomicidade: garante que a transação seja feita de forma completa. Se algum erro ocorrer durante a execução da transação, o SGBD se responsabiliza por desfazer todas as alterações retornando para um estado consistente;
- Consistência: a partir das regras de integridade que o DBA impôs ao SGBD, após o término de uma transação, o banco de dados deve passar de um estado consistente para outro estado consistente. Caso isto não ocorra, é dever do DBA rever as regras de integridade;
- Isolamento: evita o acesso concorrente aos dados por diferentes transações. Pois caso ocorra, não haverão garantias de que os dados alterados estarão consistentes;
- Durabilidade: somente transações podem fazer alterações nos banco de dados, portanto, as modificações feitas por uma transação devem ser mantidas, até outra transação alterar o banco de dados novamente.

Para Mus (2019) e Khasawneh (2020), uma das principais limitações que os SGBDRs possuem é o escalabilidade. Esta permite obter melhorias no desempenho dos SGBDs, e existem duas formas para realizá-la: verticalmente e horizontalmente. No escalonamento vertical, a capacidade de armazenamento e o poder de computação do servidor de banco de dados são expandidos. No escalonamento horizontal, o processamento e o armazenamento são divididos em várias máquinas conectadas em *cluster.* Algumas formas desta divisão são a fragmentação e a replicação (SADALAGE; FOWLER, 2019). A fragmentação consiste em dividir uma grande massa de dados em componentes menores, que poderão ser armazenados em diferentes servidores. Já a replicação tem como principal objetivo a disponibilidade, onde várias cópias idênticas dos dados estarão armazenadas em diferentes servidores do *cluster*, ajudando em tarefas como a leitura e o *backup* dos dados.

À medida que a quantidade de dados aumenta, o desempenho dos SGBDRs piora (SAHATQIJA, 2018), devido aos gargalos que são causados pelas suas características e propriedades. A solução aos SGBDRs é empregar escalonamento vertical, já que não foram projetados para serem executados de forma distribuída, em razão das limitações de disponibilidade causadas pelas propriedades ACID

(ZDEPSKI, 2019). Porém, o custo de um *upgrade* se torna cada vez mais elevado, tornando o escalonamento inviável de ser realizado, já que o custo de implementação aumenta com o crescimento da quantidade de dados.

Devido às limitações de escalabilidade e flexibilidade dos SGBDRs, os bancos de dados NoSQL (*Not only SQL*) surgiram, não para substituir, mas como uma alternativa para superar as limitações, que não permitiam um desempenho satisfatório aos SGBDRs (SAHATQIJA, 2018). O termo NoSQL surgiu pela primeira vez no ano de 1998 como nome de um SGBD desenvolvido por Carlos Strozzi (STROZZI,1998). Porém esta definição está relacionada a aplicação criada e não tem influência em como o termo foi empregado posteriormente. Somente no ano de 2009, como resultado de uma reunião em São Francisco, Estados Unidos, organizada por Johan Oskarsson, com o objetivo de levantar bancos de dados não relacionais, distribuídos e com código aberto, o termo NoSQL, sugerido por Eric Evans (*NoSQL Meetup*), passou a ser empregado (SADALAGE; FOWLER, 2019).

SGBDs NoSQL não utilizam o modelo relacional, e também não fazem uso da SQL como linguagem padrão para a manipulação dos dados. As principais características segundo Lóscio et al. (2011) são a escalabilidade horizontal (execução em *cluster*), a ausência de esquema ou esquema flexível, permissão de replicação, já que foram projetados para serem executados de forma distribuída, e a consistência eventual. Estas características que os tornam adequados para armazenar grandes volumes de dados não estruturados ou semi-estruturados (SOUZA, 2014).

A consistência eventual se deve ao fato dos SGBDs NoSQL abdicarem das propriedades ACID. Com a necessidade de obter disponibilidade e tolerância a partições, manter a propriedade da consistência torna-se uma tarefa impossível, afirmação que tem como princípio o teorema CAP (Consistência, Disponibilidade (*Availability*), e tolerância a Partições) (SADALAGE; FOWLER, 2019), onde dadas as três propriedades, é possível obter apenas duas. Em substituição às propriedades ACID, um novo modelo de transações é utilizado, conhecido como BASE (*Basically Available, Soft State, Eventual Consistency*), caracterizado pela não exigência da consistência, permitindo que o banco de dados esteja eventualmente consistente em um instante definido. De acordo com Chandra (2015), as propriedades BASE são definidas como:

- *Basically Available:* disponibilidade alta, garantida pela distribuição dos dados em *cluster* e pela replicação dos dados*;*
- *Soft State:* os dados podem sofrer alterações mesmo não acontecendo nenhuma transação*;*
- *Eventual Consistency:* Os dados poderão estar inconsistentes em algum instante de tempo, devido ao fato deles estarem distribuídos, sendo necessário propagá-los pelo *cluster.*

A ausência de um esquema pré-definido nos bancos de dados NoSQL torna possível adotar abordagens diferentes para o armazenamento de dados com estruturas complexas (SADALAGE; FOWLER, 2019). Um exemplo de abordagem é conhecido como orientação a agregados, termo utilizado para descrever uma unidade formada por diversos registros. Nesta, podem ser armazenadas listas e outras estruturas de dados complexas que estejam aninhadas dentro do agregado. A agregação visa a abstração dos dados, delimitando o que pertence ao modelo e entidades externas, e são tratados como uma unidade para fins de alteração dos dados (ZDEPSKI, 2019). Agregados facilitam a execução dos bancos de dados em *cluster*, já que constituem uma unidade natural para replicação e fragmentação.

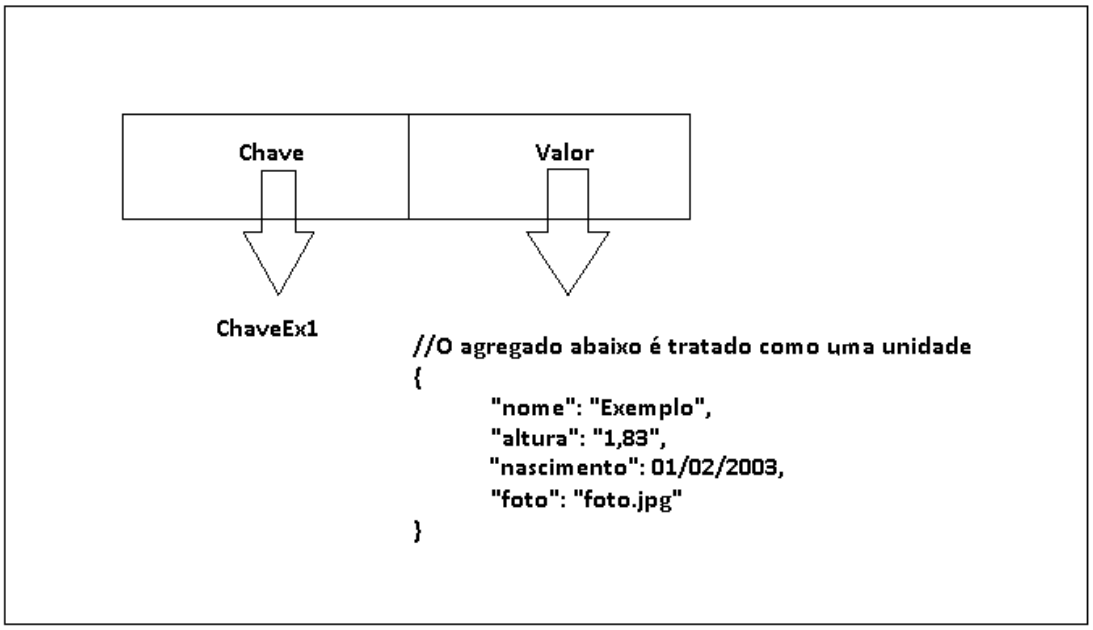
Segundo Sadalage e Fowler (2019), a classificação dos modelos NoSQL é dada a partir da forma com que o armazenamento de dados é realizado, sendo elas: chave-valor, documentos, família de colunas, que utilizam orientação a agregados, e grafos, que se beneficiam das conexões que são criadas entre os dados para manipulá-los. Este Capítulo está dividido em Seções, sendo elas 2.1, 2.2, 2.3 e 2.4, que descrevem os modelos chave-valor, orientado a documentos, família de colunas e grafos respectivamente. A Seção 2.5 apresenta às considerações finais deste Capítulo.

2.1 Modelo chave-valor

O modelo chave-valor é fortemente pautado em agregados, pois armazena os dados em uma tabela composta de dois campos: uma chave e seu campo, que será um agregado de valores associado a ela, semelhante a uma tabela *hash* (SADALAGE; FOWLER, 2019). A chave é um atributo de identificação, assim como a chave primaria dos SGBDRs, e é a partir dela que as transações dos dados são

realizadas. A Figura 2 é a representação de como os dados são armazenados neste modelo.

Figura 2 – Exemplo de armazenamento chave-valor



Fonte: Autoria própria (2022)

As possíveis operações no modelo chave-valor são: adicionar ou excluir uma chave, adicionar um agregado ou recuperar os valores associados à chave. Caso seja necessário manipular os dados contidos no agregado, a aplicação deve ser responsável pelo tratamento dos dados, já que o modelo prevê o agregado como uma unidade, e não prove nenhuma forma de manipulá-lo.

O ponto forte do modelo chave-valor é a alta escalabilidade, devido a sua simplicidade e a não existência de esquema para o armazenamento dos dados. Por conta destas características, alguns de seus principais exemplos de aplicação são guardar as sessões *web, cache* de navegação e armazenamento de dados do perfil dos usuários. Sadalage e Fowler (2019) citam como exemplo um carrinho de compras, onde uma chave irá salvar todos os dados do cliente e os produtos escolhidos por ele. Riak (RIAK, 2022) e Redis (REDIS, 2022) são exemplos de bancos de dados chave-valor.

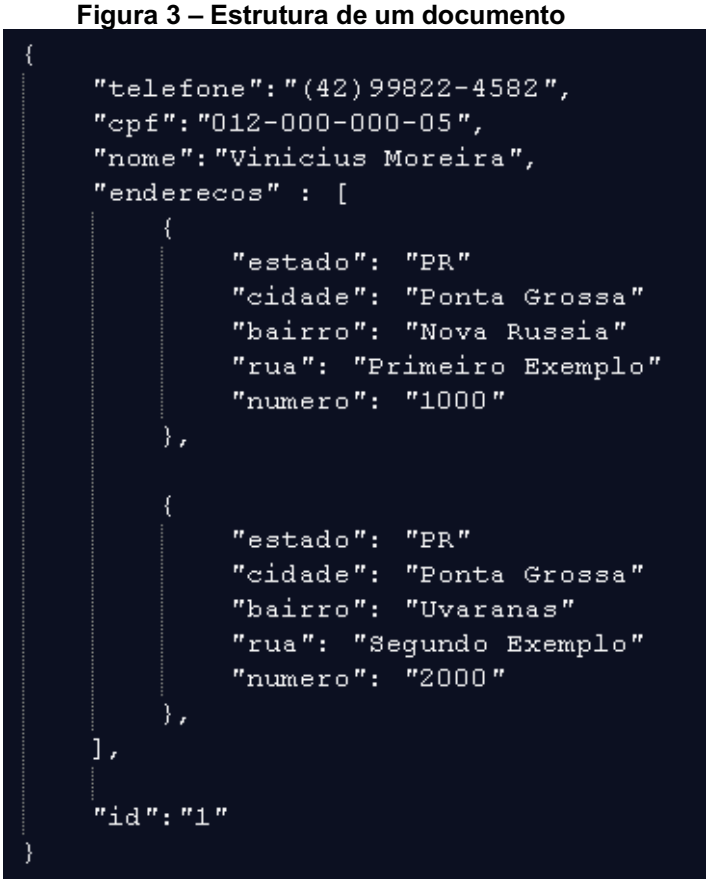
2.2 Modelo orientado a documentos

Bancos de dados orientados a documentos utilizam o conceito de documentos nas transações, onde sua estrutura é parecida com uma árvore

hierárquica (SADALAGE; FOWLER, 2019). Pode haver semelhança entre os documentos armazenados, porém não necessariamente precisam ter a mesma estrutura.

Da mesma forma que o modelo chave-valor, no modelo orientado a documentos o armazenamento é feito em agregados associados a uma chave, geralmente nos formatos JSON, XML ou BSON (WANZELLER, 2013). A diferença está em como o agregado é organizado, já que no modelo orientado a documentos é permitida a criação de identificação para partes do agregado. Com isso, as consultas poderão trazer uma parte específica do agregado, enquanto no modelo chave-valor o resultado de uma consulta por uma chave é o agregado inteiro. Na Figura 3 há um exemplo da estrutura de como um documento é armazenado.

Figura 3 – Estrutura de um documento



Fonte: Autoria própria (2022)

Com base na forma que os dados são armazenados, uma das principais características do modelo orientado a documentos é a flexibilidade. Já que também não possuem esquema pré-definido e, diferente do modelo relacional que precisa traduzir agregados de valores para armazenar os dados em tuplas, é possível armazenar e manipular agregados, sendo uma das principais vantagens desse

modelo. MongoDB (MONGODB, 2022) é o SGBD mais popular nativo para orientação em documentos, seguido pelo Couchbase (COUCHBASE, 2022).

2.3 Modelo família de colunas

De forma semelhante aos relacionais, os SGBDs orientados a família de colunas utilizam tabelas com linhas e colunas para fazer o armazenamento (ZDEPSKI, 2019). Porém, enquanto no modelo relacional os dados associados a uma chave estão armazenados em uma tupla (linha), a chave, no modelo família de colunas, carrega colunas inteiras com agregados de dados contidos em várias linhas. O Quadro 2 representa um armazenamento realizado no modelo família de colunas, onde o conteúdo nos campos em cinza representa as chaves, o conteúdo em verde representa o nome da família de coluna, o conteúdo em amarelo é o nome da coluna e o conteúdo em branco é o valor da coluna.

Quadro 1 – Exemplo de armazenamento no modelo família de colunas

<table>
<tr><td rowspan="3">Chave1</td><td colspan="3">Dados Pessoais</td><td colspan="3">Endereço</td></tr>
<tr><td>Nome</td><td>CPF</td><td>Telefone</td><td>Estado</td><td>Cidade</td><td>Bairro</td></tr>
<tr><td>Exemplo1</td><td>000-000-000-00</td><td>(99)99999-9999</td><td>Paraná</td><td>Ponta Grossa</td><td>Nova Rússia</td></tr>
<tr><td rowspan="3">Chave2</td><td colspan="2">Profissão</td><td colspan="4">Empresa</td></tr>
<tr><td>Nome</td><td>Descrição</td><td>Nome</td><td colspan="3">CNPJ</td></tr>
<tr><td>Exemplo2</td><td>Exemplo de profissão</td><td>Empresa do Exemplo 2</td><td colspan="3">11. 111. 111/0001-11</td></tr>
</table>

Fonte: Autoria própria (2022)

O conceito do modelo família de colunas é agregar uma grande quantidade de linhas com a mesma tipagem. Os bancos de dados neste modelo têm como forma de armazenamento uma tripla ordenada, formada por linha, coluna e *timestamp* (identificador de todas as versões de um dado), que contém o número em milissegundos do instante em que os dados foram armazenados.

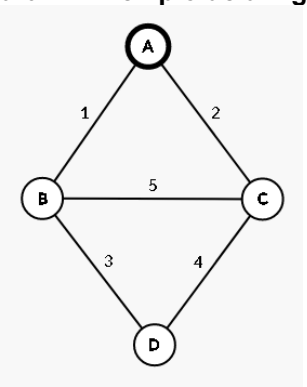
A aplicabilidade dos SGBDs orientados a família de colunas é fazer processamento de grandes massas de dados. Sadalage e Fowler (2019) utilizam como exemplos de seu emprego registros de eventos (*logs*), sistemas de

gerenciamento de conteúdo (CMS) e contadores de páginas visitadas. O fato de uma coluna com varias linhas de mesma tipagem estar comprimida, torna o modelo ideal para o armazenamento e distribuição em *cluster*, onde é possível fazer fragmentação, distribuindo as chaves com as colunas de dados em vários servidores diferentes. Os SGBDs Cassandra (CASSANDRA, 2022) e HBase (HBASE, 2022) são muito utilizados para a modelagem orientada a família de colunas.

2.4 Modelo orientado a grafos

Bancos de dados orientados a grafos utilizam a teoria dos grafos para fazer o armazenamento e manipulação dos dados. A definição formal de um grafo é uma tripla ordenada (*V*, *E*, função de incidência), onde dado um grafo *G*, os vértices *V* são elementos inter-relacionados, pertencentes a um conjunto não vazio *V*(*G*). As arestas *E* (do inglês *edges)*, são elementos de um conjunto discreto *E*(*G*), disjunto do conjunto dos vértices, que descrevem as relações entre os vértices do grafo. E uma função de incidência $\psi(G)$, que mapeia um par de vértices para cada aresta do grafo (BONDY, 1976). A Figura 4 é um exemplo de um grafo, onde o conjunto de vértices é representado por $V(G) = \{A, B, C, D\}$, o conjunto de arestas é $E(G) = \{1, 2, 3, 4, 5\}$, e as funções de incidência são $\psi G(1) = \{A, B\}$, $\psi G(2) = \{A, C\}$, $\psi G(3) = \{B, D\}$, $\psi G(4) = \{C, D\}$ e $\psi G(5) = \{B, C\}$.

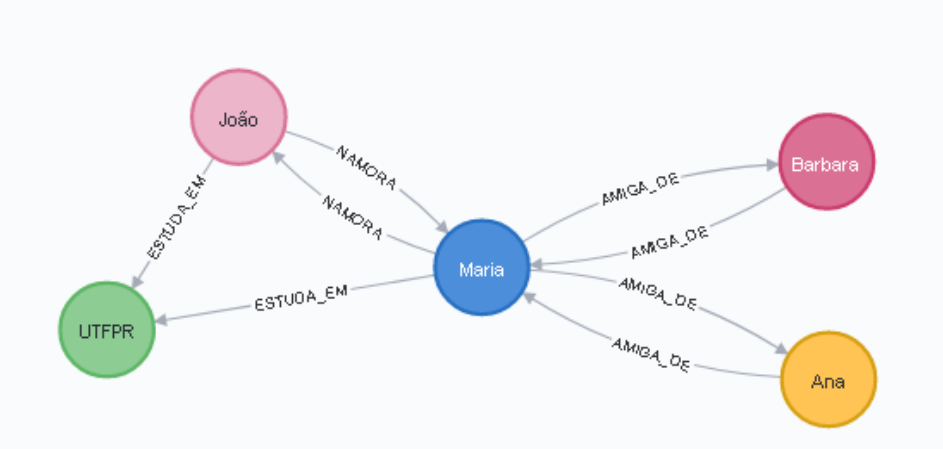
Figura 4 - Exemplo de um grafo



Fonte: Autoria própria (2022)

No modelo de armazenamento orientado a grafo, os SGBDs criam nós (vértices), que podem armazenar um dado ou um conjunto de dados. Já os relacionamentos (arestas), criam ligações entre nós, e também podem conter dados associados. A Figura 5 representa como a estrutura de um banco de dados orientado a grafos é constituída.

Figura 5 - Exemplo de uma estrutura em grafos



Fonte: Autoria própria (2022)

Os SGBDs orientados a grafos são aplicados quando o relacionamento dos dados é importante (CHAO, 2018), pois o fato dos dados estarem conectados proporciona melhor desempenho na recuperação dos mesmos junto com suas associações. Os principais exemplos de sua aplicação são: detecção de fraudes, buscas orientadas a grafos, mídias sociais e sistemas de recomendações de produtos (PI; KOROGLU, 2017), onde as buscas geralmente são realizadas por associações, tornando o modelo de armazenamento orientado a grafos mais adequado.

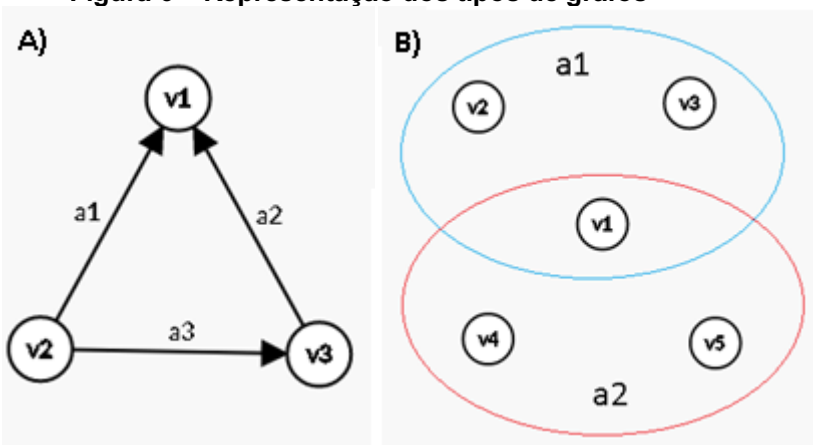
Segundo Chao (2018) existem duas formas de fazer o armazenamento nos Bancos de Dados orientados a Grafos: as não nativas e as nativas. As não nativas são feitas através de SGBDs modelados para as outras classificações NoSQL, como por exemplo SGBDs orientados a Documentos, Chave-Valor e Família de Colunas, que são dependentes de índices para realizar buscas. Já as nativas utilizam grafos como padrão de entrada e não dependem totalmente de indexação (KHAN; SHAHZAD, 2018), executando consultas através dos relacionamentos entre nós. Consultas neste tipo de base de dados tendem a apresentar ganhos de desempenho e escalabilidade.

Os SGBDs orientados a grafos são classificados com base nas características dos vértices, arestas e do próprio grafo (ANGLES, 2012). Com relação aos vértices, a classificação pode ser: vértices rotulados ou com atributos. Quanto à classificação das arestas pode ser: direcionada ou não, com rótulos ou atributos. Vértices e arestas rotulados contêm apenas um rótulo que os identificam.

Com atributos, possuem estruturas como um par chave-valor ou uma tupla relacional, que contêm vários dados associados.

A classificação dos SGBDs com relação ao tipo de grafo é dada a partir de seu formato, sendo elas: grafos simples, hipergrafos (*hypergraph*), grafos com atributos e grafos aninhados. Os grafos simples são formados a partir da definição formal de grafos, um par de vértices conectados por uma aresta conforme a Figura 6 (A). A definição de um hipergrafo é a extensão do conceito de grafo, que acontece quando uma aresta, conhecida como hiperaresta (*hyperedge*) conecta um conjunto de vértices conforme a Figura 6 (B). Os grafos aninhados permitem que contenham grafos inteiros como valores de um vértice, conhecidos como hipernós (*hypernodes*), um exemplo disso seria caso o vértice "v1" da Figura 6 (A) tivesse como valor um grafo inteiro. E os grafos com atributos contêm vértices e arestas com atributos para descrever as suas propriedades, como se os vértices e arestas da Figura 6 (A) tivessem como valor uma estrutura em documentos como na Figura 3.

Figura 6 – Representação dos tipos de grafos



Grafo simples e hipergrafo. Fonte: Autoria própria (2022)

Alguns exemplos de SGBDs orientado a grafos estão disponíveis Quadro 2, o qual apresenta como são implementados os grafos, de acordo com o seu tipo, vértice e aresta. Conforme o Quadro 2 e o autor Angles (2012), a maioria dos SGBDs utilizam grafos simples, por ser a representação que requer menos recurso, ou com atributo, por facilitar o acesso à informação dos vértices ERVEN (2015). Apenas dois SGBDs empregam hipergrafos, que geralmente é utilizado para criar subdivisões dos vértices, como por exemplo, a representação de diversos sabores de uma mesma marca de refrigerante. Por fim, nenhum deles utiliza grafo aninhado.

Quadro 2 - SGBDs orientados a grafos e suas classificações

	Grafos				Vértices		Arestas		
SGBD	**Simples**	**Hipergrafo**	**Aninhados**	**Com Atributos**	**Rotulados**	**Com Atributos**	**Direcionadas**	**Rotuladas**	**Com Atributos**
AllegroGraph	●				●		●	●	
DEX				●	●	●	●	●	●
Filament	●				●		●	●	
G-Store	●				●		●	●	
HyperGraphDB		●			●		●	●	
InfiniteGraph				●	●	●	●	●	●
Neo4j				●	●	●	●	●	●
Sones		●		●	●	●	●	●	●
vertexDB	●				●		●	●	

Fonte: Adaptado de Angles (2012)

Para iniciar um banco de dados orientado a grafos é necessário criar pelo menos dois nós e ligá-los a partir dos relacionamentos. A Figura 7 exemplifica a criação de dois nós e do relacionamento entre eles no SGBD Neo4j (NEO4J, 2022).

Figura 7 – Criação de nós e relacionamento no SGBD Neo4j

```
1 Create (p:Pessoa {cpf: '016.809.390.57',nome: 'Vinicius'})
2 Create (u:Universidade {nome: 'UTFPR'})
3 Create (p)-[e:ESTUDA_EM]→(u)
4 RETURN *
```

Fonte: Autoria própria (2022)

Os nós dos bancos de dados orientados a grafos não podem estar em servidores diferentes, já que eles operaram conectados (SADALAGE; FOWLER, 2019). Para distribuí-los em *cluster*, usa-se replicação, geralmente do tipo mestre-

escravo (*master-slave*), onde os nós escravos sincronizam os dados gravados com o nó mestre. Pelo fato de usarem replicação, a disponibilidade destes bancos é alta. Dentro de uma máquina do *cluster,* as transações são totalmente compatíveis com as propriedades ACID, pois elas são executadas de forma atômica (ZDEPSKI, 2019).

Para realizar consultas em bancos orientados a grafos, são utilizados linguagens de consulta específicas para percorrer grafos, como por exemplo, a Gremlin (GREMLIN, 2022). Os nós e suas propriedades podem ser indexados, e consultados a partir do índice, e de forma semelhante, os relacionamentos também estão suscetíveis a indexação. Para iniciar uma consulta, recomenda-se buscar nós indexados, pois o desempenho para buscá-los é melhor em comparação a nós não indexados, e algum deles podem ser utilizados como nó inicial para realizar uma travessia. A criação de um índice para um nó e como consultá-lo está representado na Figura 8.

Figura 8 – Indexação de um nó e como consultá-lo

Fonte: Autoria própria (2022)

2.5 Considerações

Neste Capitulo foram abordados os modelos de armazenamento de dados, definindo o modelo relacional e os modelos NoSQL chave-valor, documentos, família de colunas e grafos. Levando em consideração a aplicabilidade de cada modelo, é possível utilizar mais de um modelo de armazenamento em um sistema onde são

desenvolvidas diversas aplicações. A escolha do modelo deverá favorecer o melhor desempenho para cada aplicação, portanto se faz necessário uma análise das vantagens e desvantagens em cada caso.

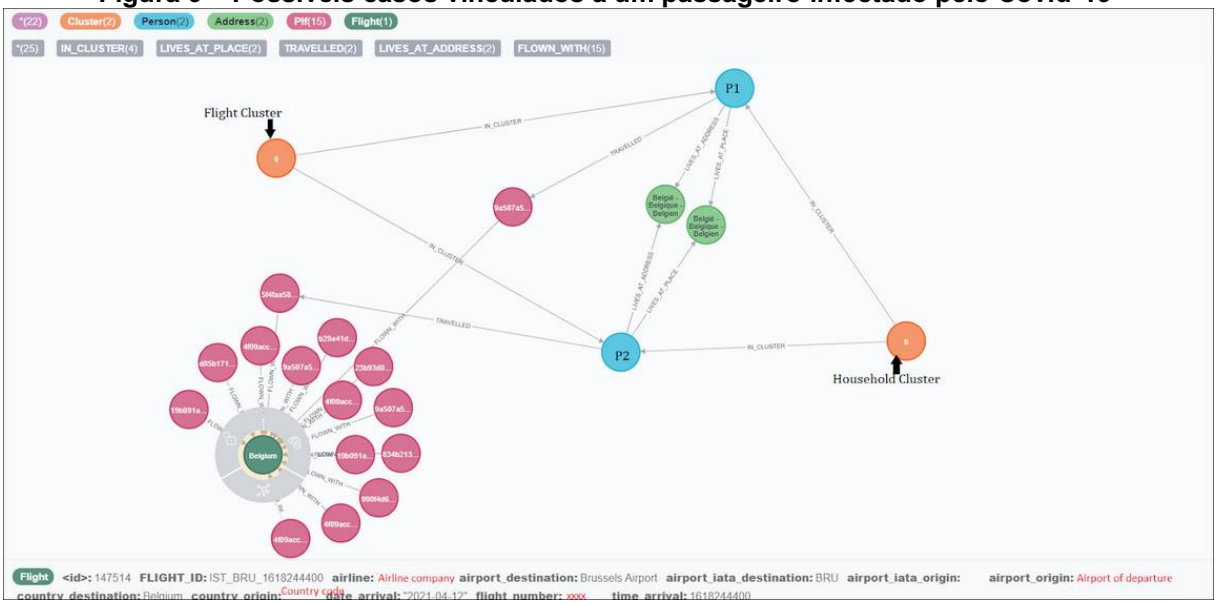
Utilizar vários modelos de armazenamento em uma aplicação é um tema abordado por Sadalage e Fowler (2019) com o nome de persistência poliglota. O tema também é abordado por Zdepski, Bini e Matos (2018), que propõem um método de projeto de banco de dados aplicado a sistemas que utilizam persistência poliglota, fazendo uso do modelo relacional ou orientado a agregados.

O Capítulo 3 dará ênfase no modelo orientado a grafos. Justificando a escolha de um SGBD orientado a grafos detalhando suas principais características e funcionalidades.

3 SISTEMAS DE ARMAZENAMENTO ORIENTADOS A GRAFOS

A utilização de grafos como forma de armazenamento principalmente quando o relacionamento entre os dados importa é uma característica que facilita o descobrimento de padrões e tendências. Um exemplo de aplicabilidade é o estudo de caso “Covid-19 *Contact Tracing with* Neo4j” (NEO4J CASE STUDIES, 2022), onde o desafio era encontrar qual local da cidade de Bruxelas, capital da Bélgica, estava ocorrendo surto de Covid-19, criando uma base com dados conectados, a partir de casos confirmados da doença. Desta maneira, a COMOM (*Commission Communautaire Commune*), responsável por detectar surtos da doença, poderia identificar em qual área da cidade estariam os grupos de infecção, aumentando o tempo de reposta para a triagem, diminuindo o contágio em até dez vezes. A identificação da concentração dos casos a partir de grafos criados facilitava o trabalho da COMOM, já que com os nós conectados era possível antecipar qual parte da população tinha tendência a se infectar. Um exemplo de uma base de dados criada por um SGBD orientado a grafos está ilustrado na Figura 9, onde os possíveis contágios eram encontrados a partir de consultas proporcionadas pela navegação entre os relacionamentos, sendo os nós em rosa pessoas que estavam em um mesmo vôo com uma pessoa infecta pelo Covid-19.

Figura 9 – Possíveis casos vinculados a um passageiro infectado pelo Covid-19



Fonte: Neo4j Case Studies (2022)

Outro exemplo marcante do uso de um SGBD orientado a Grafo está presente no estudo de caso da eBay (NEO4J CASE STUDIES, 2022) relacionado a criação de um aplicativo utilizando Inteligência Artificial junto ao *Google Assistant*, com o objetivo de identificar com precisão qual produto o usuário está pesquisando. A estratégia adotada foi construir um sistema de recomendação, que entendia e aprendia o contexto da linguagem utilizada pelos usuários, trazendo um resultado específico para cada pesquisa. O sucesso da criação da ferramenta se deve a utilização do conhecimento sobre grafos e de um banco de dados orientado a grafos, já que seriam empregadas uma quantidade muito grande de dados conectados, e a resposta deveria ser a mais rápida possível.

O sucesso dos estudos de casos da COMOM e eBay se deve a utilização de SGBDs orientados a grafos. Conforme a Figura 10, extraída do site DB-Engines (DB-ENGINES RANKING, 2023), há diversos SGBDs orientados a grafo disponíveis no mercado. Para classificá-los, os seguintes critérios são utilizados: número de menções do SGBD em *websites* (Google, Bing), frequência com que são pesquisados, vagas de emprego relacionadas ao SGBD, número de perfis pessoais que o mencionam em redes profissionais, e relevância nas redes sociais.

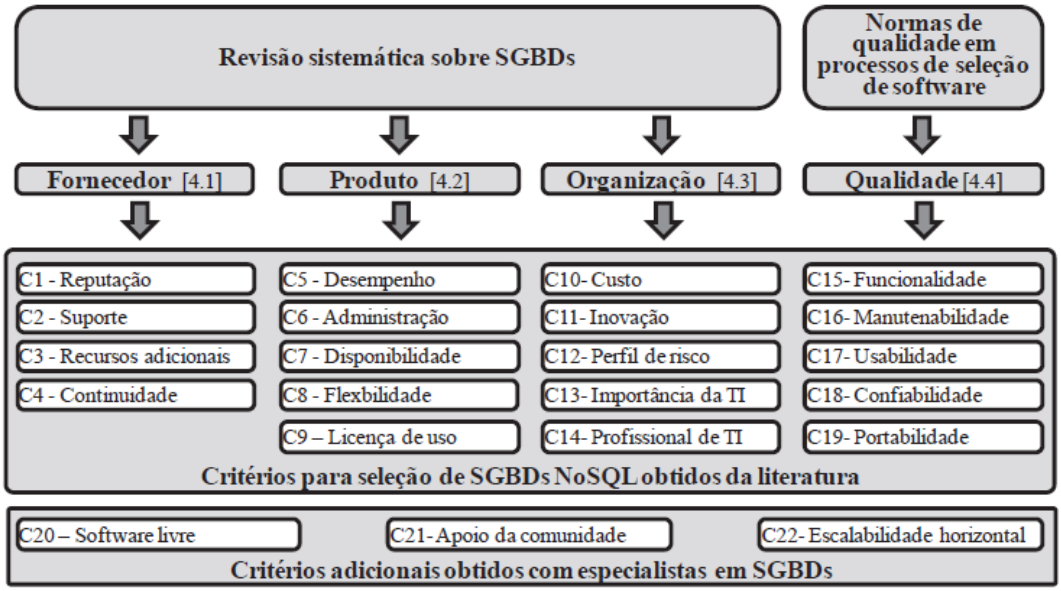
Figura 10 - Ranking dos SGBDs orientados a grafos

Rank			DBMS	Database Model	Score		
Apr 2023	Mar 2023	Apr 2022			Apr 2023	Mar 2023	Apr 2022
1.	1.	1.	Neo4j	Graph	51.60	-1.91	-7.92
2.	2.	2.	Microsoft Azure Cosmos DB	Multi-model	35.08	-1.03	-5.26
3.	3.	3.	Virtuoso	Multi-model	6.24	-0.16	+0.57
4.	4.	4.	ArangoDB	Multi-model	4.80	-0.24	-0.85
5.	5.	5.	OrientDB	Multi-model	4.06	-0.23	-1.01
6.	6.	↑ 7.	Amazon Neptune	Multi-model	2.69	+0.10	-0.08
7.	7.	↑ 8.	JanusGraph	Graph	2.44	-0.11	-0.02
8.	8.	↓ 6.	GraphDB	Multi-model	2.33	+0.01	-0.47
9.	↑ 10.	↑ 15.	NebulaGraph	Graph	2.17	+0.34	+1.04
10.	↑ 11.	↑ 20.	Memgraph	Graph	2.12	+0.34	+1.76

Fonte: DB-Engines Ranking (2023)

Na Figura 11, o autor Souza (2014) define 22 critérios para a seleção de um SGBD, onde 19 deles foram obtidos fazendo um levantamento na literatura. Estão reunidos em três grupos (Fornecedores, Produto, e Organização), junto com as normas de qualidade. Outros 3 critérios foram obtidos junto a especialistas em SGBDs. A maioria dos critérios são empregados no meio empresarial, porém alguns são aplicáveis à pesquisa. Ao comparar os critérios utilizados pelo DB-Engines com os de Souza (2014), neste último, a classificação não fica restrita somente à reputação, mas também detalhes técnicos, como desempenho, escalabilidade horizonal, disponibilidade, entre outros.

Figura 11 - Critérios para a seleção de SGBD



Fonte: Souza (2014)

Tanto os critérios utilizados pelo ranking do DB-Engines, quanto os critérios definidos por Souza (2014) foram considerados na escolha do SGBD para a condução dos experimentos neste trabalho. A partir destes critérios, principalmente a popularidade, desempenho, reputação e o fato de haver opções sem custo para adquirir o software, foram fatores decisivos para a escolha do SGBD Neo4j (NEO4J, 2022). Além de possuir a melhor pontuação, sendo o mais popular, o Neo4j é classificado como nativo para a orientação a grafos, o que significa que tanto seu desempenho quanto a escalabilidade serão superiores por utilizar grafos como padrão de entrada.

3.1 Neo4j

O desenvolvimento do SGBD Neo4j teve inicio no ano 2000 após seus desenvolvedores enfrentarem problemas de desempenho utilizando SGBDRs em algumas de suas aplicações. Em 2002 aconteceu o lançamento da sua primeira versão. Logo, em 2007 foi reconhecido como o primeiro SGBD orientado a grafos. Atualmente é o mais popular segundo o DB-Engines, não somente entre os nativos, mas entre todos os SGBDs que podem utilizar grafos como forma de armazenamento.

O Neo4j é um SGBD escrito na linguagem Java e uma de suas principais características é ser nativo à orientação a grafos. Isto implica que suas consultas

não necessariamente utilizam índices, podendo navegar entre nós através dos relacionamentos. Segundo Chao (2018) este conceito é chamado de adjacência livre de indexação, do inglês "*index-free adjacency*". Por utilizarem uma estrutura de armazenamento direcionada a grafos, os dados são eficientemente armazenados, ou seja, os dados referentes a nós e relacionamentos são armazenados junto uns dos outros. Esta forma de armazenamento garante a não necessidade de índices, aumentando a velocidade de processamento nas consultas.

Outra característica do Neo4j é a utilização de grafos com atributos, onde seus vértices e arestas podem conter agregados de dados como valores. Na criação tanto de um vértice, quanto de uma aresta do grafo, é possível armazenar somente um dado ou um agregado de dados, associando-os a um rótulo, assim como está ilustrado na Figura 12, onde há a representação de dois nós criados junto a um relacionamento. É possível realizar consultas a partir dos dados armazenados no agregado ou pelo rótulo, sendo esta a maneira mais eficiente, já que o intuito dos rótulos existirem é para facilitar a localização dos nós.

Figura 12 – Nós e relacionamento com propriedades


```
{
  "identity": 187,
  "labels": [
    "Exemplo1"
  ],
  "properties": {
"nome": "Exemplo1",
"propriedade2": "p2",
"propriedade1": "p1",
"propriedade3": "p3"
  }
}
```

```
{
  "identity": 261,
  "start": 187,
  "end": 188,
  "type": "RELACAO1",
  "properties": {
"r1propriedade": "r1p"
  }
}
```

```
{
  "identity": 188,
  "labels": [
    "Exemplo2"
  ],
  "properties": {
"nome": "Exemplo2",
"propriedade2": "p2",
"propriedade1": "p1",
"propriedade3": "p3"
  }
}
```

Fonte: Autoria própria (2022)

Além de utilizar a linguagem Gremlin (GREMLIN, 2022) para percorrer grafos, a linguagem Cypher também é empregada para transações pelo Neo4j.

Segundo os desenvolvedores do SGBD, "*Cypher é o SQL para grafos*", sendo a linguagem empregada para a criação, leitura, atualização e remoção de dados. Para a criação de nós é utilizado o comando "*create*". Já para a leitura dos dados, o comando "*match*" procura o nó com os dados passados por parâmetro. Na atualização dos dados o comando "*match*" procura o nó, e o comando "*set*" altera o campo da variável com os novos valores passados por parâmetro. A remoção dos dados é realizada utilizando o comando "*delete*", que, da mesma forma que na atualização, é executada no nó encontrado com o comando "*match*". Para exibir o resultado do comando utilizado, basta digitar o comando "*return*" junto com a variável que indica o nó ou o relacionamento a ser apresentado. O Quadro 3 exemplifica cada um dos comandos utilizados nas transações dos dados em Cypher, comparando-os com os comandos na linguagem SQL.

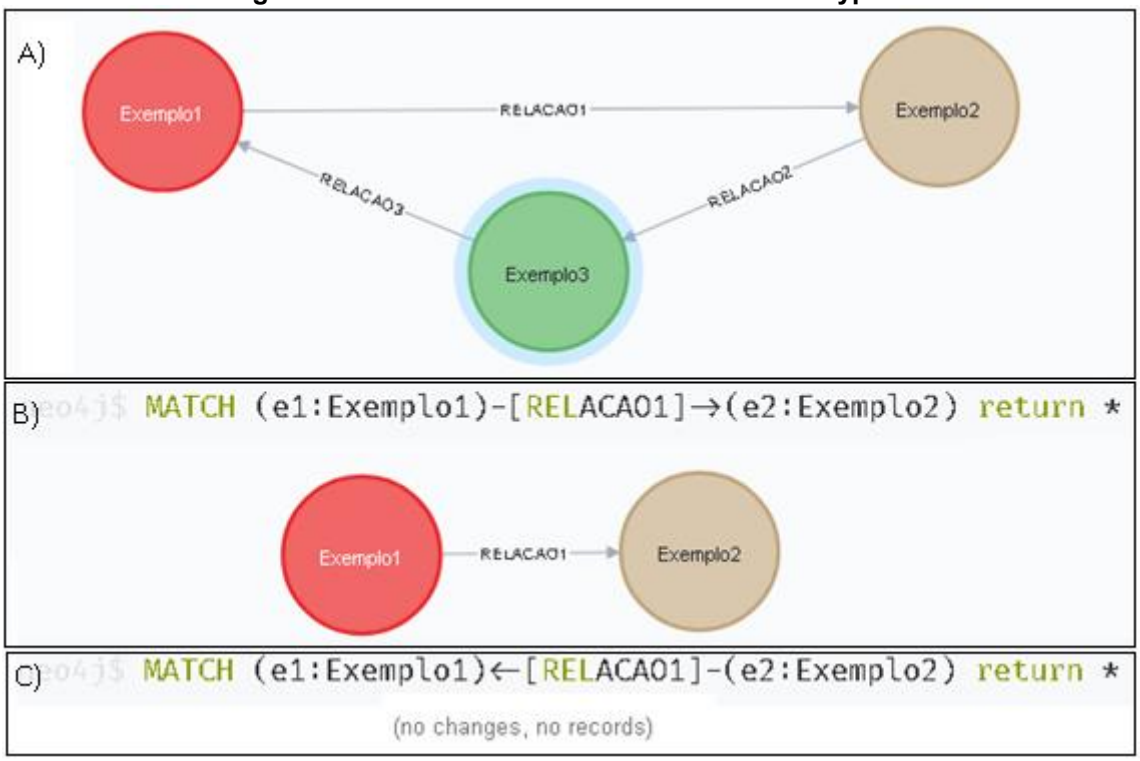
Quadro 3 – Comparativo comandos em Cypher e SQL

Cypher	SQL
CREATE(p:Pessoa{nome:'Vinicius', CPF:'016.908.930.65'})	INSERT INTO Pessoa (nome, CPF) VALUES ('Vinicius', '016.908.930.65')
MATCH (p:Pessoa{nome:'Vinicius'})	SELECT * FROM Pessoa WHERE nome='Vinicius'
MATCH (p:Pessoa {nome: 'Vinicius'}) SET p.CPF = '015.907.931.55'	UPDATE Pessoa SET CPF='015.907.931.55' WHERE nome='Vinicius'
MATCH (p:Pessoa {nome: 'Vinicius'}) DELETE p	DELETE FROM Pessoa WHERE nome='Vinicius'

Fonte: Autoria própria (2022)

O relacionamento entre os dados na linguagem Cypher é determinado no momento da criação da aresta que ligará dois nós. A ligação só pode ser direcionada, e na criação são utilizados os símbolos "<-" ou "->", e caso não contenha o direcionamento, uma mensagem de erro é retornada. Esta determinação é importante principalmente na consulta dos relacionamentos, já que a direção determina o inicio e o fim da aresta. A Figura 13 exemplifica os resultados obtidos em uma consulta apenas mudando o parâmetro de direcionamento do relacionamento. Sendo a Figura 13 A um grafo completo. A Figura 13 B é o resultado do comando "*match*", procurando por um relacionamento entre os nós "Exemplo1" e "Exemplo2" com o direcionamento à direita. E a Figura 13 C é o mesmo comando da Figura 13 B, mudando apenas o direcionamento do relacionamento para a esquerda.

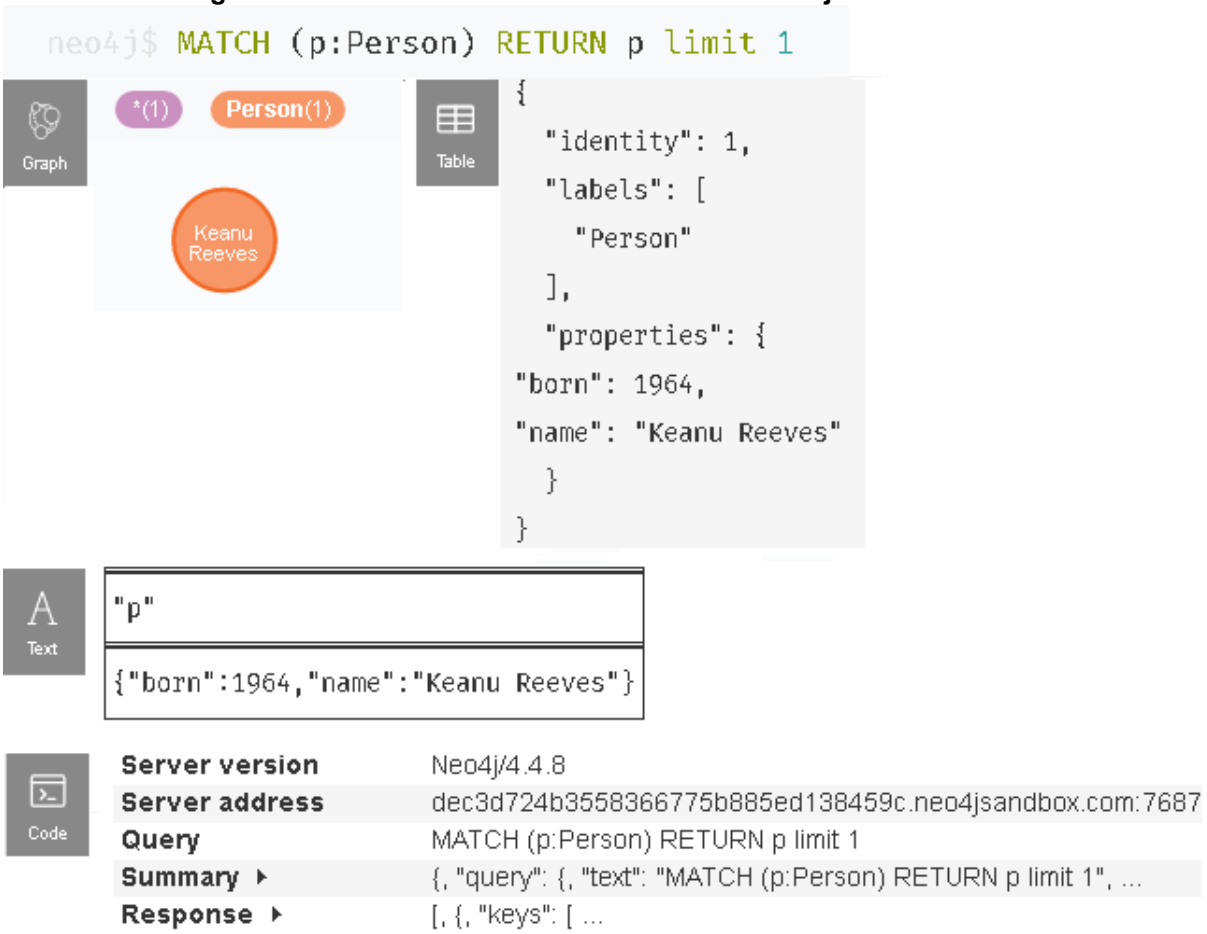
Figura 13 – Direcionamento nas consultas em Cypher



Fonte: Autoria própria (2022)

Sempre que um comando é executado no terminal do Neo4j, é possível acessar os dados referentes às consultas a partir de uma área de retorno de comandos disponível no SGBD. A opção padrão de retorno é o grafo formado pela consulta, mas também existem as opções de visualização da tabela, do texto e da documentação do código dos nós consultados. A Figura 14 apresenta o retorno do grafo (*Graph*), tabela (*Table*), texto (*Text*) e a documentação (*Code*) de uma consulta no Neo4j.

Figura 14 – Área de retorno de comandos do Neo4j



Fonte: Autoria própria (2022)

Existem procedimentos e funções que são nativas da linguagem Cypher como, por exemplo, a função de Dijkstra que calcula a menor distancia entre dois nós. Uma vez que as bases de dados no Neo4j podem conter milhares/milhões de nós, utilizar funções nativas de agregação como "*COUNT*" para saber quantos nós estão sendo consultados ou "*AVG*" para saber a média entre determinada propriedade numérica dos nós, facilita o entendimento das saídas visuais produzidas. É possível também utilizar operadores para organizar as tabelas (*table*), como por exemplo, ordenar os nós com o comando "*ORDER BY*". A Figura 15 apresenta exemplos do emprego das funções "*COUNT*", "*AVG*" e do operador "*ORDER BY*" respectivamente.

Figura 15 – Exemplos de comandos suportados pelo Neo4j

Fonte: Autoria própria (2022)

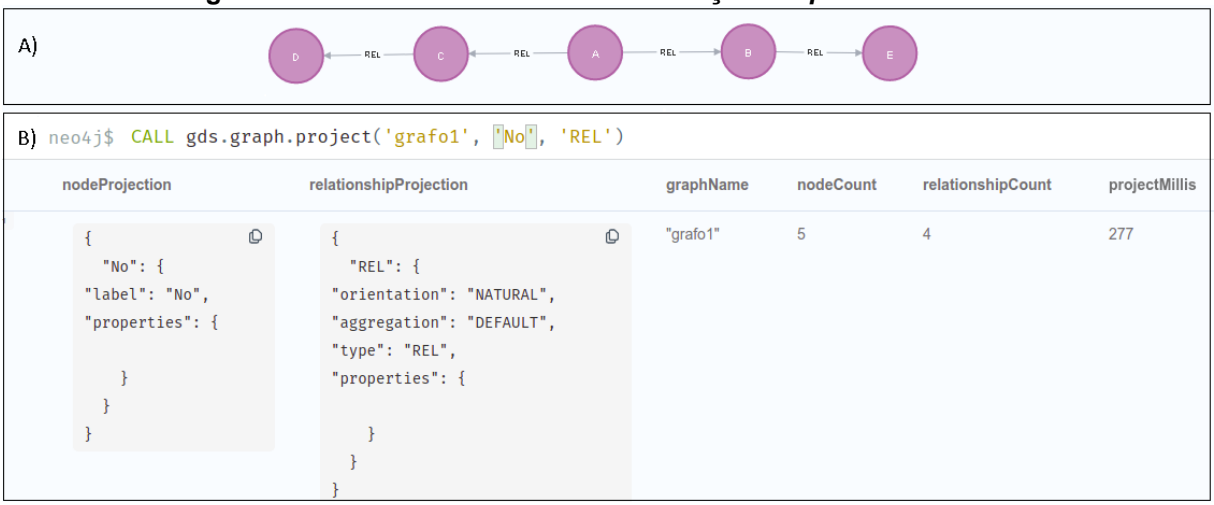
A linguagem Cypher também possibilita a criação de procedimentos e funções para automatizar a execução de comandos. Chamadas definidas pelo usuário (*User-defined*), são escritas na linguagem Java e implantadas na base de dados, podendo ser chamadas a qualquer momento, assim como as nativas da linguagem. Há duas maneiras de criar funções, as escalares (*Scalar*), que recebem parâmetros e realizam um retorno, e as agregações (*Aggregating*), que consomem linhas de comandos e produzem um resultado formado por agregações. Para criá-las é necessário compilar um arquivo *".jar"* e colocá-lo no diretório "*plugin*" da base de dados. Um exemplo de chamada de uma função definida como "*join*" dentro de um

diretório “exemplo” é dado da seguinte forma: “*MATCH (p: Pessoa) WHERE p.idade = 36 RETURN org.neo4j.exemplo.join(collect(p.nomes))*”.

Para realizar consultas é necessário navegar entre os nós através dos relacionamentos. Este conceito é conhecido como travessia, uma operação exclusiva para SGBDs orientados a grafos (VUKOTIC, 2015), sendo o que promove o bom desempenho em consultas neste tipo de base de dados, pois não será necessário buscar por algum identificador para encontrar os dados relacionados. Utilizando um nó como referencia, é possível realizar a travessia em vários níveis, percorrendo todos os nós conectados de acordo com a profundidade determinada na consulta. Como por exemplo, procurar todos os nós conectados ao nó referenciado com nível de profundidade 2. O resultado da consulta será todos os nós conectados diretamente a referência e suas adjacências. O Neo4j possui uma API para melhorar o desempenho de consultas deste tipo, já que travessias são muito utilizadas por aplicações com dados conectados, como sistemas de recomendações, redes sociais e investigações em crimes cibernéticos.

A Figura 16 representa uma travessia realizada com a função nativa do Neo4j “*Depth First Search*” da biblioteca GDS (*Graph Data Science*), onde, dado um nó, a travessia será feita o mais profundo possível a partir dele. Os três parâmetros de entrada para sua execução são: uma *String* representando o nome do grafo, que é escolhido pelo usuário, o nome da variável utilizada na criação do nó, e o nome da variável utilizada na criação do relacionamento. A saída será os dados relacionados ao nó e ao relacionamento passados por parâmetro, o nome do grafo, a contagem de nós e relacionamentos encontrados, e o tempo em milissegundos da execução da travessia. A Figura 16 A é um exemplo de um grafo criado, e a Figura 16 B é a saída da execução da função “*Depth First Search*”.

Figura 16 – Travessia realizada com a função "*Depth First Search*"



Fonte: Autoria própria (2022)

Uma das principais vantagens dos SGBDs NoSQL é poder dividir as bases de dados em servidores diferentes conectados em *cluster.* No Neo4j, a clusterização prove três principais recursos: a proteção dos dados (*Safety*), sendo tolerante a erros nos servidores, escalabilidade (*Scalability*), a partir de replicações, e consistência casual (*Eventual Consistency*), garantindo que os servidores leiam, no mínimo, suas próprias escritas. Há dois diferentes papeis para um servidor do *cluster*: ser um servidor primário, onde serão feitos leituras e escritas, ou um servidor secundário, que será utilizado para realizar replicações, podendo até mesmo atuar como os dois tipos ao mesmo tempo.

No Neo4j as propriedades ACID são respeitadas dentro de um servidor do *cluster* (SADALAGE; FOWLER, 2019), assim como é na maioria dos SGBDs orientados a grafos. Todas as transações serão concluídas apenas se não houver nenhuma falha nos blocos de comandos executados. Os dados manipulados dentro de um servidor do *cluster* passarão de um estado consistente para outro estado consistente. Porém isto só é possível ocorrer dentro do servidor do *cluster* que está executando à transação, não podendo garantir que os dados contidos nos outros servidores do *cluster* estejam consistentes. Há um mecanismo semelhante ao dos bancos relacionais que isola os dados que estão sendo manipulados para garantir a consistência. Os dados alterados por operações de escrita (*set*, *create, delete*) permanecerão intactos até que outra transação ocorra para alterá-los. A consistência eventual (BASE) está presente pela distribuição dos nós em vários servidores, já que não é possível garantir a consistência dos dados enquanto eles estão sendo sincronizados a partir da replicação.

3.2 Considerações

O Capítulo 3 justificou a escolha do SGBD orientado a grafos Neo4j. Os comandos apresentados compõem a carga de trabalho que será executada no ambiente experimental. O Capitulo 4 irá detalhar tal ambiente, juntamente com a base de dados escolhida. No SGBD serão aplicadas técnicas de *tuning*para diminuir o tempo de resposta aos comandos submetidos à base de dados.

4 TUNING NO SGBD NEO4J

Para melhorar o desempenho, técnicas de *tuning* será empregadas no SGBD Neo4j. A definição de *tuning,* segundo Tramontina (2008), é o ajuste fino dos parâmetros de configuração em diversos aspectos, envolvendo desde o SO, a configuração do SGBD e até mudança no esquema da base de dados. Sendo assim, o DBA deve possuir alto conhecimento e experiência utilizando o SGBD, já que a implementação de técnicas de *tuning* pode ser difícil, demandando alto custo e tempo às organizações (BINI, 2014).

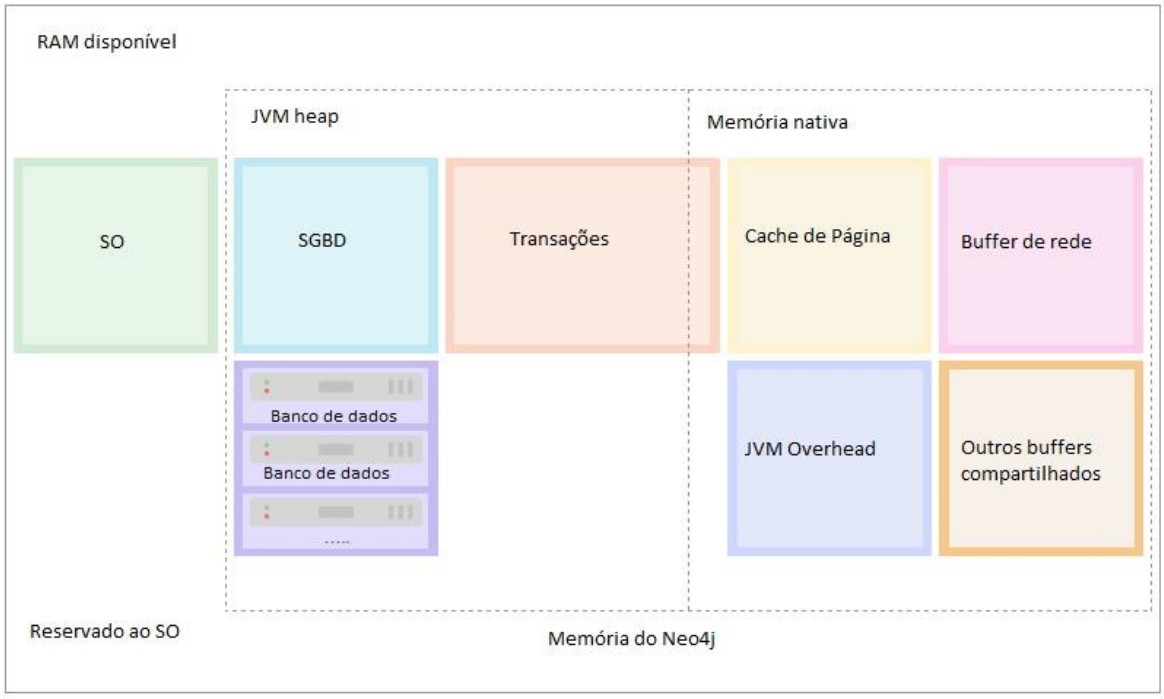
O levantamento de técnicas de *tuning* empregadas neste trabalho foi realizado a partir da leitura de materiais relacionados ao Neo4j, buscando principalmente conteúdos sobre diminuição do tempo de resposta a cargas de trabalho submetidas. No capítulo 13 do manual do Neo4j (NEO4J TEAM, 2019), há uma série de configurações que buscam aperfeiçoar o desempenho do SGBD. O capítulo é dividido em 8 tópicos, apresentando como configurar parâmetros relacionados tanto ao SGBD (índices por exemplo), quanto ao SO (sistema de arquivos do Linux por exemplo).

As técnicas que foram aplicadas para melhorar o desempenho do SGBD Neo4j estão relacionadas à configuração do uso da memória RAM (*Random Access Memory)* e ao uso de índices. A justificativa para escolha está relacionada à ênfase dada nestes componentes, pelos desenvolvedores do Neo4J em seu manual. Pesquisas relacionadas sobre ao tema deste trabalho, como "*Sharding the* LDBC *Social Network*" (NEO4J DOCS, 2022), também avaliam seus resultados considerando memória RAM e o emprego de índices.

4.1 Configuração da memória RAM

O uso da memória RAM pelo Neo4j é distribuído conforme a Figura 17: (1) A memória destinada ao SO, reservada para a execução de suas tarefas; (2) *heap*, relacionado ao JVM (*Java Virtual Machine*), separada para alocar objetos Java; e (3) memórias nativas, que são utilizadas para processos não diretamente relacionados ao *heap*, porém necessários para o funcionamento do mesmo, e diretamente alocado pelo Neo4j. A Figura 17 também exemplifica como é distribuído o uso da memória RAM pelo SGBD.

Figura 17 – Distribuição do uso da memória RAM pelo SGBD Neo4j



Fonte: Adaptado de Neo4j Docs (2022)

Entre as memórias nativas estão o *Cache de Página*, os *Buffers de rede*, o *JVMOverhead* e alguns *outrosbuffers compartilhados* necessários para o funcionamento do Neo4j. O *Cachede Página* é utilizado para armazenar temporariamente dados do próprio SGBD e dos índices. Desta forma, evitam-se custos relacionados a acesso ao disco, melhorando o desempenho. O *Buffer de rede* armazena dados a serem enviados e recebidos antes de criar uma cópia em disco. E o *JVMOverhead* é uma memória alocada para o JVM funcionar corretamente.

Com relação ao *heap (JVM heap)*, a memória é alocada para o funcionamento do SGBD, para os *Bancos de dados*, e para as *Transações*. A porção de memória destinada para o SGBD é utilizada para o armazenamento de componentes globais do Neo4j, como por exemplo, servidores *bolts*, relacionados à conexão com os bancos de dados, serviços de *login* e monitoria. Em cada *Transação*, o Neo4j aloca memória para manter dados ainda não salvos em disco, como o resultado de transações, e estados intermediários dos comandos na memória.

O *heap* é utilizado na execução de comandos, já que seu objetivo é armazenar objetos Java, e seu tamanho é o que determina diretamente o comportamento do *garbage collector*. Este, por sua vez, localiza objetos não

utilizados, para compactá-los e retirá-los da memória destinada ao *heap* (JONES; HOSKING; MOSS, 2016).

A coleta de lixo (*garbage collector*) ocorre em ciclos, onde os objetos criados por comandos executados recentemente são armazenados na nova geração (*young generation*). Se o objeto ainda continua sendo utilizado após o final de um ciclo, ele será destinado à velha geração (*old generation*). Caso o objeto não seja mais utilizado, ele é descartado. Este é um processo custoso, portanto, para um bom desempenho, é necessário alocar apenas a quantidade de memória suficiente para executar qualquer tipo de consulta.

Para alocar memória destinada ao *heap* é necessário analisar a carga de trabalho, pois caso não seja suficiente, os ciclos do *garbage collector* serão finalizados precocemente, ocasionando perda de desempenho na transição da nova para velha geração, pelo descarte de objetos que poderiam ser utilizados novamente. Caso seja alocada memória em excesso, muitos objetos serão armazenados na nova e velha geração, já que os ciclos se tornarão mais longos, também resultando em perda de desempenho, já que o armazenamento de objetos no *heap* é custoso.

Ao criar uma base de dados, antes de conectá-la, o Neo4j permite modificar manualmente o uso de algumas partes da memória, como o tamanho de *heap* e *cache,* e outras configurações relacionadas a portas de conexões *(bolt, HTTP, cluster)*. A recomendação inicial é ajustá-las explicitamente no arquivo de configuração (*.conf*) localizado na pasta da base de dados, pois a configuração padrão do SGBD pode não ser ideal para obter o melhor desempenho possível. Segundo o manual (NEO4J DOCS, 2022), as configurações de memória relacionadas ao SO, *buffers,* e banco de dados não podem ser alteradas, pois o SGBD não disponibiliza comandos de configuração para a alocação de memória RAM nestes casos, sendo responsabilidade do próprio SGBD. Desta forma, as principais recomendações de configurações de *tuning* do Neo4j estão relacionadas ao tamanho do *Cache de Página* e ao tamanho do *heap*.

O primeiro passo para configurar o uso da memória é determinar a disponibilidade de memória RAM do servidor. Também, o quanto de memória de disco está sendo utilizado para armazenar dados dos bancos e índices. Para isso, o Neo4j disponibiliza o comando “*memrec*”, que pode ser digitado no terminal do

diretório “*bin*” de qualquer base de dados. A Figura 18 apresenta o resultado de tal comando, com dicas e recomendações para a configuração inicial do SGBD.

Figura 18 – Saída do comando “*memrec*” do Neo4j

```
vinicius@vinicius-VirtualBox:~/.config/Neo4j Desktop/Application/relate-data/dbm
ss/dbms-ff57b53a-5ae0-4585-8db4-32b6c205bb46$ bin/neo4j-admin memrec
Selecting JVM - Version:11.0.14.1+1-LTS, Name:OpenJDK 64-Bit Server VM, Vendor:Azul Systems, Inc.
# Memory settings recommendation from neo4j-admin memrec:
#
# Assuming the system is dedicated to running Neo4j and has 3.832GiB of memory,
# we recommend a heap size of around 1900m, and a page cache of around 524600k,
# and that about 1500m is left for the operating system, and the native memory
# needed by Lucene and Netty.
#
# Tip: If the indexing storage use is high, e.g. there are many indexes or most
# data indexed, then it might advantageous to leave more memory for the
# operating system.
#
# Tip: Depending on the workload type you may want to increase the amount
# of off-heap memory available for storing transaction state.
# For instance, in case of large write-intensive transactions
# increasing it can lower GC overhead and thus improve performance.
# On the other hand, if vast majority of transactions are small or read-only
# then you can decrease it and increase page cache instead.
#
# Tip: The more concurrent transactions your workload has and the more updates
# they do, the more heap memory you will need. However, don't allocate more
# than 31g of heap, since this will disable pointer compression, also known as
# "compressed oops", in the JVM and make less effective use of the heap.
#
# Tip: Setting the initial and the max heap size to the same value means the
# JVM will never need to change the heap size. Changing the heap size otherwise
# involves a full GC, which is desirable to avoid.
#
# Based on the above, the following memory settings are recommended:
dbms.memory.heap.initial_size=1900m
dbms.memory.heap.max_size=1900m
dbms.memory.pagecache.size=524600k
#
# It is also recommended turning out-of-memory errors into full crashes,
# instead of allowing a partially crashed database to continue running:
```

Fonte: Autoria própria (2022)

O comando “*memrec*” (Figura 18) faz a sugestão de configuração dos parâmetros de configuração “*dbms.memory.heap.initial_size*” (tamanho inicial do *heap*), “*dbms.memory.heap.max_size*” (tamanho máximo do *heap*), e “*dbms.memory.pagecache.size*” (tamanho do *Cache de Página*). Porém é necessário analisar as dicas apresentadas pelo comando, pois a configuração sugerida pode não ser a ideal. De acordo com a Figura 18, dependendo do tipo de carga de trabalho, se houver mais operações de leituras que escritas, é recomendado alocar mais memória ao *Cache de Página.* Se ocorrer elevada execução concorrente de comandos ou diversas atualizações de dados na carga de trabalho enviadas ao SGBD, o ideal é alocar mais memória RAM ao tamanho do *heap.* É sugerido também configurar o tamanho inicial e máximo do *heap* com o mesmo valor, para evitar que a JVM precise alterar o tamanho do *heap.*

4.2 Uso de índices

Para obter melhorias de desempenho na execução de consultas é recomendada a utilização de índices (NEO4J DOCS, 2022), ou seja, criar cópia de dados no banco para propositalmente gerar redundância. O objetivo é tornar as buscas mais eficientes, já que terá mais formas de procurar por dados que estão indexados. Em contrapartida, o desempenho na escrita diminuirá devido ao custo de armazenamento e de manutenção dos índices.

No Neo4j é possível indexar uma ou mais propriedades de um nó ou relacionamento. Na versão 4.4 do SGBD, utilizada neste trabalho, existem três tipos de índices: *btree* (padrão), *text* e *lookup*. Os tipos *btree (Balanced tree)* e *text* indexam propriedades, podendo ser uma propriedade (*single-property index*), ou várias propriedades (*composite index*). A diferença entre esses dois tipos de índice é que o tipo *text* apenas é aplicável se a propriedade for um texto (*String*), enquanto os tipos *btree,* podem armazenar todo tipo de valor. O tipo *lookup* indexa apenas rótulos que poderão ser utilizados em nós ou relacionamentos inteiros, necessitando para sua criação, como parâmetro(s), o(s) nome(s) do(s) rótulo(s), ao contrário dos outros dois tipos, que utilizam como parâmetro de criação o(s) nome(s) de propriedades. Desde a versão 4.3 do Neo4j, índices do tipo *lookup* estarão presentes por padrão nas bases de dados assim que elas são criadas (NEO4J DOCS, 2022). O Quadro 4 exemplifica todos os comandos relacionados a índices. O texto entre colchetes é opcional.

Quadro 4 – Comandos de manipulação de índices no Neo4j

Comando	Descrição
CREATE INDEX [nome_indice] [IF NOT EXISTS] **FOR (n:nomeRotulo)** **ON (n.propriedade)** **[OPTIONS "{" opcao: valor[, ...] "}"]**	**Criação de um índice do tipo BTREE para apenas uma propriedade de um nó. (*single-property index*)**
CREATE INDEX [nome_indice] [IF NOT EXISTS] **FOR ()-"["r:nome_relacionamento"]"-()** **ON (n.propriedade)** **[OPTIONS "{" opcao: valor[, ...] "}"]**	**Criação de um índice do tipo BTREE para uma propriedade de um relacionamento.**
CREATE INDEX [nome_indice] [IF NOT EXISTS] **FOR (n:nomeRotulo)** **ON (n. propriedade_1,** **n. propriedade_2,** **…** **n. propriedade_n)** **[OPTIONS "{" opcao: valor[, ...] "}"]**	**Criação de um índice do tipo BTREE para várias propriedades de um nó. (*composite index*)**
CREATE TEXT INDEX [nome_indice] [IF NOT EXISTS] **FOR (n:nomeRotulo)** **ON (n.propriedade)** **[OPTIONS "{" opcao: valor[, ...] "}"]**	**Criação de um índice do tipo TEXT para apenas uma propriedade de um nó. (*single-property index*)**
CREATE INDEX [nome_indice] [IF NOT EXISTS] **FOR ()-"["r:nome_relacionamento"]"-()** **ON (n.propriedade)**	**Criação de um índice do tipo TEXT para uma propriedade de um relacionamento.**

[OPTIONS "{" opcao: valor[, ...] "}"]	
CREATE LOOKUP INDEX [nome_indice] [IF NOT EXISTS] **FOR (n)** **ON EACH labels(n)** **[OPTIONS "{" opcao: valor[, ...] "}"]**	**Criação de um índice do tipo LOOKUP para nós.**
CREATE LOOKUP INDEX [nome_indice] [IF NOT EXISTS] **FOR ()-"["r"]"-()** **ON [EACH] type(r)** **[OPTIONS "{" option: value[, ...] "}"]**	**Criação de um índice do tipo LOOKUP para relacionamentos.**
DROP INDEX nome_indice [IF EXISTS]	**Comando para apagar qualquer índice.**

Fonte: Adaptado de Neo4j Docs (2022)

Os índices melhoram o desempenho dos seguintes tipos de consultas: igualdade (n.propriedade = valor); verificar se a propriedade pertence à determinada lista (n.propriedade Is listaNome); existência (n.propriedade *IS NOT NULL*); alcance (n.propriedade > valor); prefixo (*STARTS WITH*); sufixo (*ENDS WITH*); substring (*CONTAINS*). Para tornar o uso de índices ainda mais eficientes, tanto o manual do Neo4j (NEO4J DOCS, 2022), quanto a saída do comando "*memrec*" (Figura 18) recomendam que nem toda a memória RAM do servidor seja destinada ao Neo4j, deixando-a livre para o SO, pois se não há memória RAM suficiente, o SO começará a fazer *swap*, afetando o desempenho do SGBD.

4.3 Considerações

O Capítulo 4 detalhou as técnicas de *tuning* expressivamente citadas na literatura, como o manual do Neo4j, pesquisas relacionadas e referências bibliográficas (livros). A importância da aplicação de tais técnicas, explicando como configurar o uso da memória RAM para obter melhor desempenho, com ênfase no tamanho do *cache* e tamanho do *heap.* Por fim, foram apresentados os tipos de

índices (*btree, text* e *lookup*) e como eles influenciam no desempenho em operações de leituras.

O Capítulo 5 irá descrever o ambiente experimental utilizado no presente trabalho, composto pelo *hardware* (descrevendo o computador empregado), e pelo *benchmark* responsável por padronizar os experimentos no SGBD Neo4j. Então serão comentados os resultados obtidos com a aplicação das técnicas de *tuning,* devidamente detalhadas.

5 AMBIENTE EXPERIMENTAL

Para realização dos experimentos foi empregado um computador com processador Intel Core I7-3770K de 3.90 GHz, com 8 MB de memória *cache* e 8 GB de memória RAM. Quanto a memória secundária, foi utilizado 1 disco SATA de 1 TB e 7200 RPM (Rotações Por Minuto). O SO instalado era o Linux Ubuntu 20.04.5 LTS com Kernel 5.15.0-56-generic. O SGBD orientado a grafos utilizado foi o Neo4j versão 4.4.12.

Para padronizar os resultados, tornando-os mais confiáveis e passíveis de reproduzir os experimentos por futuros pesquisadores, foi utilizada uma ferramenta de *benchmark* chamada LDBC (*Linked Data Benchmark Council*) *Social Network Benchmark (SNB)* versão 2.2.1 *(*ANGLES, 2020*)*, cujo objetivo é avaliar SGBDs orientados a grafos. O desenvolvimento do LDBC deve-se a ferramentas de *benchmark* como os fornecidos pelo TPC (*Transaction Processing Performance Council*) não avaliarem apropriadamente SGBDs orientados a grafos, pois suas bases de dados e tipos de consultas não foram desenvolvidas no contexto de dados conectados.

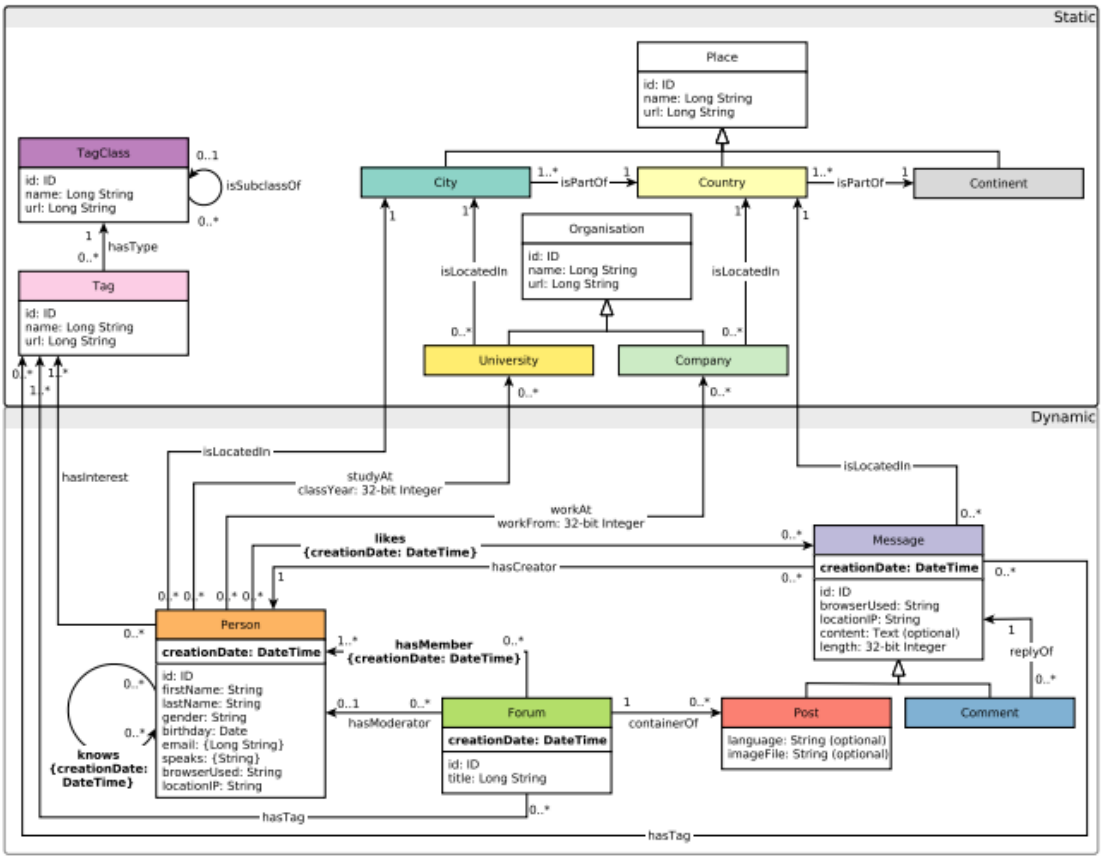
O LDBC SNB disponibiliza para uso dois tipos de cargas de trabalho, a *Interative Implementation* e a *Business Inteligence (BI).* A *Interative Implementation* contém operações de leituras e escritas, separadas em 14 "*complex reads*", 7 "*short reads*", 8 inserções e 8 remoções. Apesar de conter leituras, o grande foco da *Interative Implementation* são operações do tipo OLTP (*Online Transaction Processing*), caracterizadas por operações de escrita. Logo a BI executa 20 tipos de consultas diferentes somente após realizar inserções, remoções e atualizações, com foco em avaliar operações do tipo OLAP, que se caracterizam por leituras. As tabelas criadas pela BI destacam intencionalmente o tempo de execução das consultas, já que elas são separadas do tempo total do *benchmark* com as inserções e atualizações de dados.

Os dados contidos nas bases utilizadas pelo *benchmark* são gerados sinteticamente, e os comandos submetidos pelas duas cargas de trabalho simulam atividades em uma rede social durante um determinado período de tempo, iniciado no dia 29 de novembro de 2012 até 30 de dezembro de 2012. A quantidade de nós, relacionamentos e propriedades contidas nas bases de dados depende do fator de escala (SF), que determina o tamanho da base de dados, escalando de forma

crescente, obedecendo a um conjunto de valores pré-determinado (1, 3, 10, 30, 100, 300, 1000, 3000, 10000). O tamanho da base de dados criada geralmente segue o seguinte padrão: SF1 = 1 GiB (*Giga Binary Byte*), SF100 = 100 GiB e SF10000= 10000GiB.

As bases de dados utilizadas pelo LDBC SNB *benchmark* são criadas seguindo o esquema do diagrama de classe da Figura 19. As principais características das bases de dados são: Realismo, pois os dados simulam características encontradas em redes sociais reais; Escalabilidade, pela capacidade de criação de base de dados com tamanhos diferentes dependendo de um fator de escala (SF); Determinismo, garantindo que todos os testes serão os mesmos, independente do ambiente utilizado; Usabilidade, desenvolvida para não haver dificuldades no uso por pesquisadores.

Figura 19 – Diagrama de classe do esquema de geração de grafos pelo LDBC SNB



Fonte: Angles (2020)

A carga de trabalho OLAP (BI) composta por 20 consultas[1], foi empregada nos experimentos uma vez que as técnicas de *tuning* descritas no Capitulo 4 são mais eficientes neste ambiente. A descrição das consultas executadas pela carga de trabalho encontra-se no Quadro 5.

Quadro 5 – Todas as consultas realizadas pela carga Business Inteligence

Consulta	Descrição
Q1 – Resumo de publicações	Encontra todas as mensagens criadas a partir de uma data ($datetime). A saída desta consulta será: Ano, Tipo (comentário ou publicação), Tamanho (curto, uma linha, tweet, longo).
Q2 – Variação de tópicos	Compara se *tag* ($tagClass) utilizadas durante 100 dias continuam sendo relevantes nos 100 dias seguintes. A entrada é *tag* e uma data ($date), e a saída é o nome da *tag*, uma variável contendo a contagem da primeira janela de tempo (os primeiros 100 dias), e outra variável com a segunda contagem de tempo (os seguintes 100 dias), e uma variável contendo a diferença entre a primeira e segunda janela.
Q3 – *Tags* populares por país	Dada uma *tag,* ($tagClass) e um país ($country), encontra todos os fóruns que contenham mensagens com a *tag*. O país é determinado pela nacionalidade do autor do fórum.
Q4 – Maiores criadores de mensagens por país	Encontra os maiores fóruns por país. A métrica para o cálculo é a quantidade de membros. Se for popular em mais de um país ele é calculado apenas uma vez (sendo escolhido o maior). Em caso de empate, o menor id é escolhido. Na saída, além dos dados da pessoa que criou (*person.id, person.firstName, person.lastName, person.creationTime*), está contido o número de mensagens postadas no fórum.
Q5 – Pessoas mais ativas em um Fórum	Dado um tag ($tag), encontra pessoas que criaram mensagens. A saída é a contagem de mensagens, de *likes*, comentários, o id da pessoa e sua pontuação.
Q6 – Pessoa mais autoritária	O cálculo está relacionado com a criação de mensagens em uma *tag* e sua contagem de likes. *Authoritative score* é a soma da popularidade. Esta é a quantidade de *likes* que uma pessoa recebe em seus comentários. Após o cálculo, são comparadas duas pessoas para saber qual é a mais autoritária.
Q7 – Tópicos relacionados	Procura todas as mensagens com uma *tag* passada como parâmetro. Encontra todas as mensagens que replicaram o conteúdo contendo a *tag* buscada.
Q8 – Pessoa central de um *tag*	Encontra todas as pessoas que tiveram interesse em uma *tag*, ou seja, que escreveram alguma mensagem ou *post*. Então é calculada uma

[1] Para verificar o código das 20 consultas do LDBC *benchmark* com a carga de trabalho BI: **LDBC SNB Business Intelligence (BI) workload implementations**, 2022: Disponível em: https://github.com/ldbc/ldbc_snb_bi/tree/main/cypher/queries

	pontuação por pessoa: - Pontuação 100, caso a pessoa tem interesse. Caso contrário a pessoa recebe 0. - O retorno da função será a pontuação da pessoa, o id dela, e a pontuação dos amigos relacionados.
Q9 – *Top* iniciador de *thread*	Para cada pessoa, contam quantas mensagens o usuário criou em um período de tempo, e quantas árvores de replays as mensagens receberam. O retorno é cada pessoa encontrada, número de *posts* criados por elas, e o número de mensagens nas árvores de replays.
Q10 – *Expert* em círculos sociais	Dada uma pessoa, encontra todos os amigos relacionados a partir de uma função de travessia. O parâmetro para a travessia é determinado na entrada ($minPathDistance, $maxPathDistance). Então é realizado o caminho mínimo da pessoa selecionada. Na saída estarão contidas todas as mensagens que contém uma *tag* que será pré-determinada na entrada.
Q11 – Triângulo de amigos	Dado um país, conta todos os triângulos de amizades encontrados. Um triângulo segue as seguintes condições: - *pessoaA* é amigo de *pessoaB*; - *pessoaB* é amigo de *pessoaC*; - *pessoaC* é amigo de *pessoaA*. E essas amizades devem ter sido criadas em um determinado período de tempo.
Q12 – Quantas pessoas tem uma quantidade de mensagens	Conta quantas mensagens cada pessoa enviou seguindo as instruções: - Conteúdo não vazio; - Data após a variável $startDate; - Tamanho da mensagem abaixo da variável $lengthThreshold; - Em qualquer idioma que esteja na variável $languages. Então, é realizada a contagem de quantas mensagens do mesmo tamanho existem.
Q13 – Zumbis por país	Encontra zumbis por países. A definição de zumbi é: pessoa que escreve uma média de mensagens entre 0 e 1. O retorno inclui também o *zombieScore*, que é calculado da seguinte forma: - zombieLikeCount: likes vindos de outros zumbis; - totalLikeCount: total de likes recebidos; - zombieScore: zombieLikeCount / totalLikeCount, se o zombieLikeCount for 0, então o zombieScore também deve ser 0
Q14 – Dialogo internacional	Considerando um par de pessoas, sendo elas de nacionalidade diferente, calcula um *score* com base nas seguintes regras: - *Score* inicial = 0; - pessoa1 respondeu a algum comentário da pessoa2: score += 4; - pessoa1 criou alguma mensagem que pessoa2 respondeu: score +=1;

	- pessoa1 deu *like* em alguma mensagem da pessoa2: score +=10; - pessoa1 criou alguma mensagem que recebeu *like* da pessoa2: score+=1;
Q15 – Caminhos de conexões confiáveis entre fóruns em um dado período de tempo	Dada duas pessoas, calcula o custo da largura do menor caminho em um sub-grafo induzido formado pelo seu relacionamento "conhece". O peso da aresta "conhece" é calculado da seguinte forma: - Réplicas diretas à Post: 1 ponto; - Réplica a um comentário: 0.5 pontos. Considera apenas *post* criados em um período de tempo pré-determinado. Se não existe um caminho entre os nós formados pelas pessoas escolhidas, então o retorno é -1.
Q16 – Detector de *fake* news	Dado um par de datas e *tags* $tagA/$dateA$tagB/$dateB), para cada $tagX/$dateX: - Cria um sub-grafo induzido entre cada par de pessoas, as duas criaram uma mensagem no dia $dataX com a $tagX; - No sub-grafo induzido, somente será mantido os pares de pessoas com o maior *maxKnowsLimit* amigos; - Para estas pessoas, conta o número de mensagens criadas na $dataX com a $tagX. Retorna pessoas que tenham pelo menos uma das mensagens para $tagA/$dateA e $tagB/$dateB, ranqueadas pelo total de número de mensagens (descendente).
Q17 – Análise de propagação da informação	Analisa a submissão de mensagens com determinada *tag* em um fórum. Verifica se a mensagem foi lida por outras pessoas e se houve discussão sobre a *tag*. O conceito de discussão é caso uma pessoa2 envia uma mensagem2 e é diretamente replicada por uma pessoa3. O retorno será a pessoa1 que criou a mensagem no fórum com o número de interações.
Q18 – Recomendações de amizades	Dada um *tag*, para cada pessoa interessada, recomenda uma possível amizade seguindo os seguintes critérios: - Ainda não se conhecem; - Tem pelo menos um amigo em comum; - Tem interesse pela *tag* dada. Ranqueia pessoas com base no número de amigos em comum.
Q19 – Caminhos de interações entre cidades	Encontra o menor "caminho de interação" entre duas pessoas que moram em cidades diferentes. Se há mais de um par de pessoas com o menor caminho, então o retorno será todos os pares. Uma interação é qualquer tipo de aresta criada no grafo, podendo ser, pessoa1 conhece pessoa2, réplicas, comentários, mensagens diretas.
Q20 – Recrutamento	Dada uma empresa e uma pessoa2 que não está trabalhando e nunca trabalhou na mesma, encontra outra pessoa1 que tenha sido, ou é empregada. O critério para retornar esta

	pessoa1 é que o nó dela seja alcançado pela pessoa2, ou seja, se houve algum contato na vida real entre elas, como por exemplo, estudado na mesma escola. Se há mais de uma pessoa candidata a ser a pessoa1, então retorna a que tenha o menor caminho. Caso várias pessoas tenham o menor caminho, retorna todas elas.

Fonte: Adaptado de Angles (2020)

Como resultado da execução da carga de trabalho BI do LDBC SNB são gerados 4 tabelas no formato "csv" contendo: (1) o tempo total de execução do *benchmark*; (2) o tempo de carregamento dos dados; (3) o *log* das consultas; (4) o tempo de execução das consultas. A medida de tempo utilizada é exibida em segundos.

Os testes realizados em cada execução do *benchmark* são divididos em "*power test*" e "*throughput test*", com a finalidade de criar 4 métricas para quantificar o desempenho (*scores*), sendo elas: *power score*, *throughput score*, e suas variações de preços (*price-adjusted*). Todas elas incluem o fator de escala, denominado por "@SF". O preço (@SF/$) está relacionado com o custo da máquina, custos de licença de *software* e custos de manutenção por 3 anos, que é calculado pela multiplicação do @SF por 1000 e dividido pelo custos citados (*total cost of ownership*) (TCO). O preço não será abordado neste trabalho, pois se optou por utilizar a versão sem custo dos *softwares,* e não é relevante saber o quanto a Universidade Tecnológica Federal do Paraná gastou para adquirir o *hardware* onde os experimentos foram conduzidos.

O *power test* da carga de trabalho BI, a exemplo do *benchmark* TPC-H (TPC, 2022), utiliza a média geométrica das 20 consultas realizadas, com o objetivo de avaliar o conjunto de execução de todas elas. A execução é unitária, carregando apenas os dados do dia 29 de novembro de 2012 e executando as consultas. O *score* para o *power test* é calculado com a Equação 1 (ANGLES, 2020):

$$\text{power@SF} = \frac{3600}{\sqrt[29]{w \times q1 \times q2a \times q2b \times \ldots \times q18 \times q19a \times q20a \times q20}} \times SF \qquad (1)$$

Logo a fórmula para calcular o preço do *power test* é representado na Equação 2 (ANGLES, 2020):

$$\text{power@SF/\$} = \text{power@SF} \times \frac{1000}{TCO} \qquad (2)$$

O *throughput test* da carga de trabalho BI avalia o tempo de carregamento de dados na base, sendo medido em horas. Portanto para realizar o teste, o *benchmark* carrega dados do primeiro dia (29 de novembro de 2012) e executa as consultas, assim como no *power test*. Caso a operação supere uma hora, o teste é finalizado. Caso isso não ocorra, a base será atualizada com os dados sinteticamente criados simulando um dia seguinte na rede social até completar pelo menos uma hora, ou executar todos os 30 dias. A Equação 3 (ANGLES, 2020) representa a fórmula para o cálculo do *score* do *throughput test*:

$$throughput@SF = (24\ horas - tempo\ de\ carregamento) \times \frac{tempo\ das\ consultas}{dias} \times SF \qquad (3)$$

Já o preço do *throughput test* é calculado com a Equação 4 (ANGLES, 2020):

$$\text{thoughput@SF/\$} = \text{thoughput@SF x } \frac{1000}{TCO} \qquad (4)$$

5.1 Resultados

A presente Subseção detalha os resultados obtidos a partir da aplicação das técnicas de *tuning* citadas no Capitulo 4, tanto na base de dados quanto no SGBD. Antes de executar o *benchmark* LDBC SNB, foi realizada a limpeza do *cache* do SO, executando o comando "*sync; echo 3 > /procs/sys/vm/drop_caches*", para não haver qualquer interferência, pois os dados resultantes de comandos executados recentemente no SO são mantidos em *cache* para melhorar o desempenho (*cache* quente). Durante a realização dos experimentos não havia execução de processos concorrentes. Os testes utilizaram o fator de escala 1 "SF=1", pois o *hardware* disponível não suportou a execução do SF=10. Além disso, o SF=3 obteve tempo de execução maior que 24 horas, tendo seus resultados desprezados.

Nas subseções 5.1.1, 5.1.2 e 5.1.3 serão comparados os tempos de execução de cada consulta da carga de trabalho BI do LDBC, considerando aplicação ou não de *tuning* em índices e memória RAM. Na base de dados foi alterado o arquivo relacionado a importação de índice (arquivo "*indices.cypher*", localizado no caminho "*cypher/dll*", a partir do diretório principal). No SGBD foram

aplicadas diferentes configurações para a memória RAM destinadas ao *cache* e *heap*.

Para a obtenção dos dados de resultado foram executadas duas vezes o *benchmark* completamente, considerando quatro saídas (dois *power test* e dois *throughput test*), devido ao tempo limitado disponível pararealizar todos os experimentos. O *throughput test* superou uma hora de execução, considerando apenas o dia 29 de Novembro de 2012 em todos os testes. Desta forma, esta data foi utilizada para o cálculo das médias apresentadas, já que os mesmos dados são empregados no *power test e throughput test*.

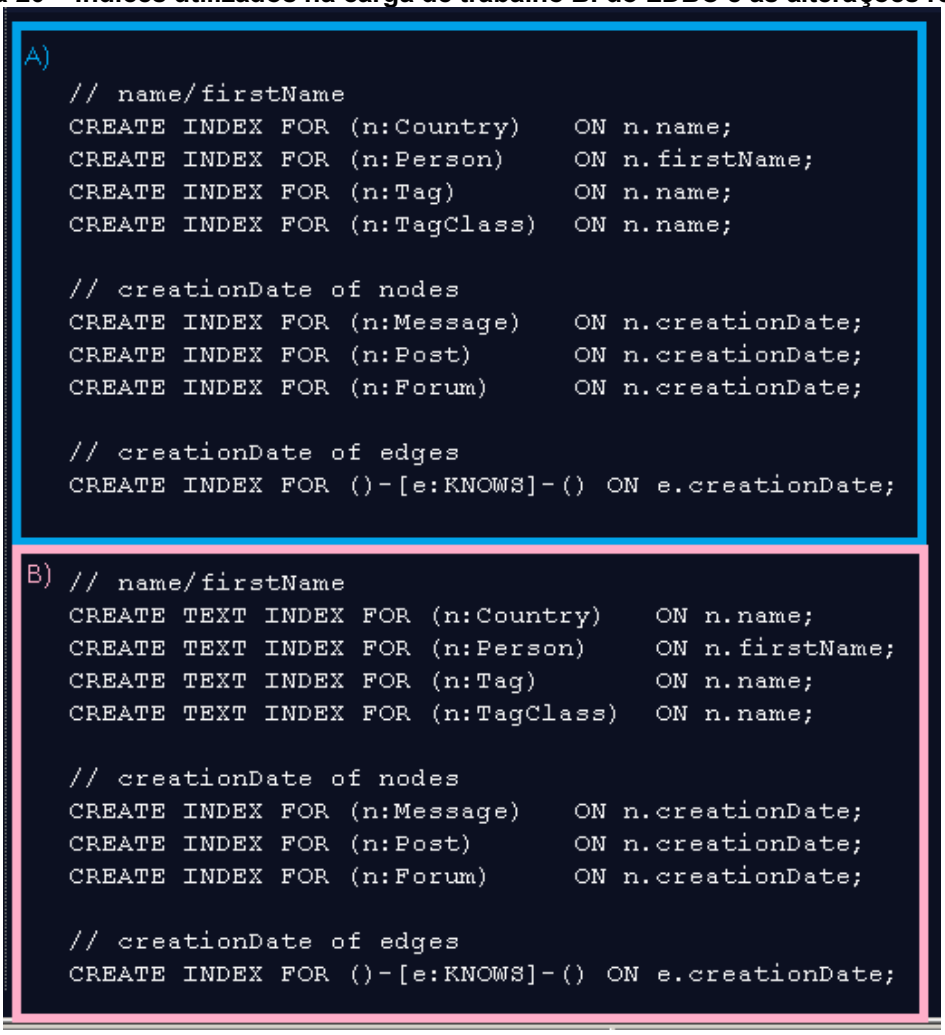
O *thoughput score* não pode ser calculado nos experimentos realizados. Ao executar o arquivo "*scores-full.sh*" do diretório "*scripts*", que calcula esta métrica, a variável dias (*days*) recebe o valor "0". Assim não podendo continuar o cálculo por causa de divisão por zero. Devido à forma como os desenvolvedores do *benchmark* programaram o cálculo do *thoughput score*, o dia que supera uma hora de execução do *thoughput test* não é considerado. Como apenas um dia foi executado no *thoughput test*, os valores deste dia não podem ser utilizados no cálculo. Portanto o processo não pode ser continuado.

5.1.1 Uso de indices

A presente subseção tem por objetivo apresentar o impacto no desempenho de consultas causado pelo emprego de índices. Primeiramente será verificado se há ganho ou perda de desempenho ao utilizar índices do tipo *btree* e *text,* já que o índice do tipo *lookup* criado na base de dados por padrão, indexa todos os rótulos utilizados pelo *benchmark*. No arquivo de importação de índices da carga de trabalho BI do LDBC *benchmark* "*indices.cypher*" os 4 primeiros índices são criados em propriedades textuais. Os três seguintes, em propriedades do tipo *creationDate*. Também foi criado um índice em uma propriedade do tipo *creationDate* de um relacionamento. O *benchmark* foi executado com as seguintes instruções no arquivo de importação de índices: (1) retirar as linhas de comando relacionadas à criação de índices; (2) execução padrão do *benchmark* (índices do tipo *btree*); (3) alteração dos índices em propriedades textuais para o tipo *text.* A Figura 20 apresenta as alterações realizadas, sendo a Figura 20 (A) os comandos originais do *benchmark,* e a Figura 20 (B) representando as alterações realizadas no arquivo de importação de índices. Os desenvolvedores do Neo4j (NEO4J DOCS, 2022) sugerem empregar

índices do tipo *text* para os predicados "*STARTS WITH*", "*END WITH*" e "*CONTAINS*", portanto também será verificado se nas consultas da carga de trabalho BI haverá diferença no tempo de execução com a mudança de índices para o tipo *text*.

Figura 20 – Índices utilizados na carga de trabalho BI do LDBC e as alterações realizadas



Fonte: Autoria própria (2022)

O Quadro 6 apresenta a média dos resultados em segundos das execuções do *benchmark* LDBC com a carga de trabalho BI não utilizando índices do arquivo "*indices.cypher*" (Apenas *lookup*), com índices do tipo *btree (Btree* padrão*),* e com índices do tipo *text*. Nenhuma alteração nos parâmetros de configuração do uso de memória foi empregada nestes experimentos. Conforme o Quadro 6, observa-se que algumas consultas da carga de trabalho BI possuem versão "b", que de acordo com Angles (2020) é uma versão menos custosa da versão "a" da mesma consulta.

Tabela 1 – Desempenho considerando índices do arquivo "índices.cypher", assim como as modificações aplicadas

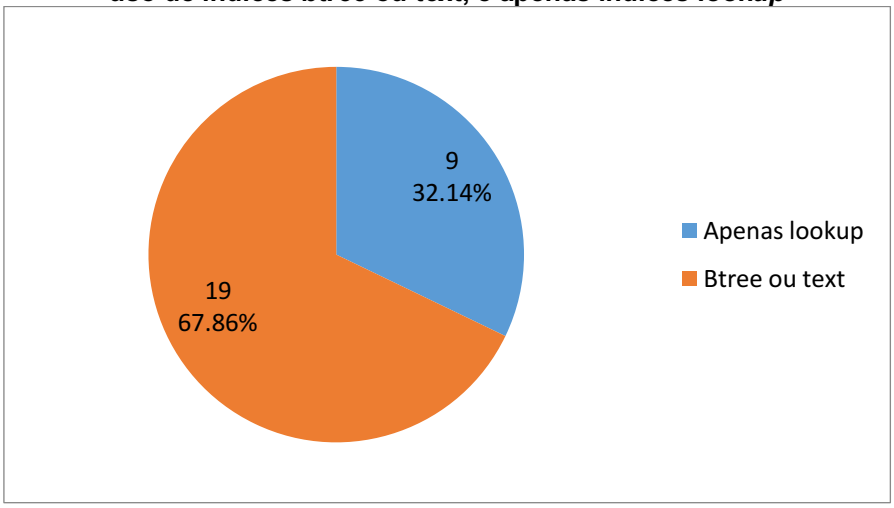
Consulta	Apenas *lookup*	*Btree* (padrão)	*Text*
Q1	142,48	6,54	6,64
Q2a	88,60	122,40	96,25
Q2b	5,66	8,19	6,24
Q3	196,73	176,66	183,54
Q4	4092,70	5281,41	4461,49
Q5	2,15	1,89	1,01
Q6	141,30	161,27	142,71
Q7	5,00	4,17	4,35
Q8a	1209,94	1293,68	1249,69
Q8b	30,43	31,35	27,31
Q9	402,64	314,10	287,38
Q10a	14,27	13,40	14,54
Q10b	9,14	8,58	10,73
Q11	87,63	103,34	139,17
Q12	140,22	999,53	761,65
Q13	64,50	80,01	56,20
Q14a	282,22	279,74	215,82
Q14b	0,99	0,86	0,74
Q15a	5179,14	2836,46	2521,33
Q15b	5097,64	2946,75	2611,36
Q16a	1097,37	1190,23	1045,15
Q16b	2,11	0,40	0,44
Q17	103,74	76,10	150,75

Q18	99,98	78,01	94,95
Q19a	45,57	43,64	46,92
Q19b	37,28	39,64	38,15
Q20a	0,72	0,79	0,77
Q20b	0,33	0,35	0,34
Total	18580,63	16099,61	14175,75

Fonte: Autoria própria (2022)

Dados da Tabela 1 demonstram que retirar os índices do arquivo "*indices.cypher*" da base de dados provocou perda de desempenho em 67,86% das consultas. O Gráfico 1 apresenta graficamente esta porcentagem. Empregando-se índices do tipo *btree* o SGBD foi 2.481,08 segundos, ou seja, mais de 41 minutos mais eficiente comparado a execução utilizando apenas índices *lookup*. Logo, utilizando-se de índices do tipo *text* o SGBD foi 4.404,88 (72 minutos aproximadamente) mais rápido para executar o mesmo conjunto de consultas comparado à execução utilizando apenas índices *lookup*. Quando comparado somente os resultados que fizeram uso de índices *btree* e *text*, verifica-se que o uso do índice *text* foi mais eficiente em 1.923 segundos (mais de 32 minutos) comparado ao índice *btree*.

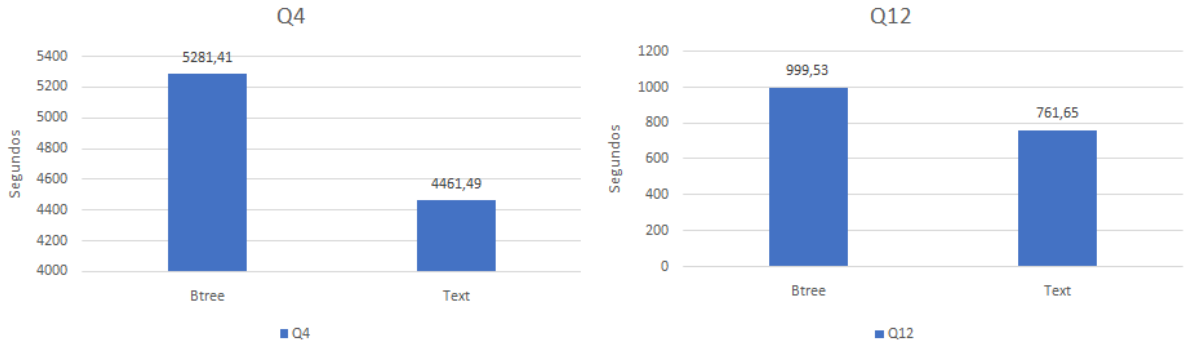
Gráfico 1 - Quantidade de consultas que obtiveram tempo de execução menor, considerando o uso de índices *btree* ou *text*, e apenas índices *lookup*



Fonte: Autoria própria (2022)

É possível observar na Tabela 1 que houve ganho de desempenho total ao utilizar índices do tipo *text* em relação ao tipo *btree* conforme já mencionado. Ao comparar as consultas que utilizam os índices modificados (Q4 e Q12, por exemplo), a redução do tempo de execução foi de 1.057,8 segundos. O Gráfico 2 demonstra graficamente a diferença nos tempos de execução das consultas Q4 e Q12.

Gráfico 2 – Comparação do tempo de execução das consultas Q4 e Q12 com índices *btree* e *text*



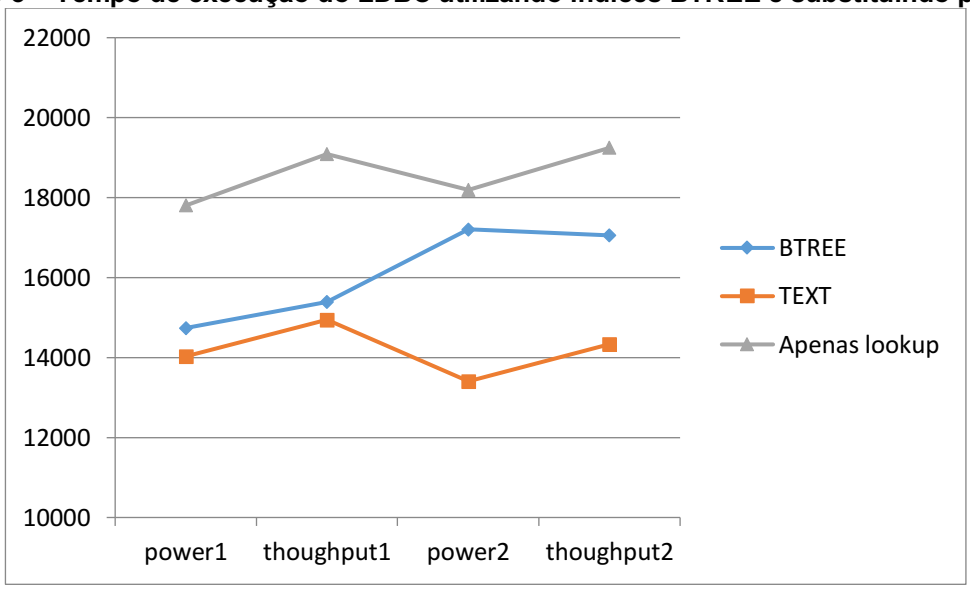
Fonte: Autoria própria (2022)

O ganho de desempenho apresentado pelo Gráfico 2 acontece principalmente quando há comparação entre *Strings*, como as que possuem o predicado "*IN*" (Q4), e o predicado "*not null*" (Q12). A explicação para isso é que o manual do Neo4j recomenda utilizar índices do tipo *text* com o predicado "*IN*" em listas de *Strings*, mesmo não sendo sua principal aplicabilidade. Desta forma, é possível que o tempo de execução seja menor que o do tipo *btree*, já que existe semelhança entre o predicado "*IN*" e "*CONTAINS*", onde os dois procuram por um texto dentro de uma entrada. A diferença está em qual parte do texto que a busca será realizada. O "*not null*" também possuí semelhança com "*CONTAINS*", pois o predicado é utilizado para verificar se existe algum *caracter* em uma *String*.

Pela variação no tempo de execução, algumas consultas que não utilizavam índices *text*, como, por exemplo, a Q15a e Q15b, também houve diminuição no tempo de execução, onde somadas, o ganho foi de 650,52 segundos em relação ao tipo *btree*. Portanto, a Subseção 5.1.3 foi desenvolvida para uma análise mais precisa na mudança de índices. Além dos dados contidos na presente Subseção sobre os índices do tipo *text*, também serão comparados os resultados utilizando *tuning* na memória RAM, para uma análise mais detalhada.

Como já mencionado nesta Seção, o *benchmark* LDBC BI foi executado duas vezes por completo para a obtenção dos dados. A linha “Total” da Tabela 1 é a média do tempo de execução de dois *power tests* e dois *thoughput tests.* Para o cálculo do *power score (power@SF*) foram utilizados os dados dos dois *power tests*. A comparação dos tempos de execução total tanto dos *powers tests (power1* e *power2),* quanto *thoughput tests (thoughput1* e *thoughput2)* são ilustrados no Gráfico 3.

Gráfico 3 – Tempo de execução do LDBC utilizando índices BTREE e substituindo por TEXT



Fonte: Autoria própria (2022)

Para a métrica do *benchmark* (*scores*), a média dos dois *power scores* (*power$SF*) obtidos foi 61,53 pontos para índices do tipo *btree*, 64,81 pontos para índices do tipo *text*, e 56,98 pontos sem os índices do arquivo “*indices.cypher*”. Assim, sendo avaliado o conjunto das 28 consultas (contando as que possuem versão “a” e “b”), o *benchmark* obteve pontuação mais baixa, quando não foram empregados índices *btree* e *text*.

5.1.2 Configuração da memória RAM

As configurações de memória analisadas foram a padrão do SGBD Neo4j (o mínimo possível de memória RAM configurada para a execução do SGBD), a configuração de memória sugerida pelo comando "*memrec*", e a configuração padrão do *benchmark*, que também possui uma forma de distribuição do total de memória RAM disponível. O *benchmark* permite que antes de sua execução, os parâmetros de configuração de memória RAM destinadas ao tamanho do *cache* e do

heap sejam alterados. Caso não seja configurado, já que não é obrigatório, por padrão, a ferramenta adota a metade da memória RAM disponível, como o tamanho do *cache,* e a outra metade como o tamanho do *heap*.

Para o *hardware* empregado nos experimentos, o *benchmark* alocou por padrão (Padrão BI) aproximadamente 2 GB (1,932 GiB) para o tamanho do *cache* e do *heap.* Na versão *desktop* do Neo4j, a configuração padrão (Padrão Neo4j) destina 500 MB para o tamanho do *cache* e o tamanho do *heap* varia entre 500 MB à 1 GB. Como no LDBC *benchmark* não é possível configurar menos de 1 GiB destinado ao tamanho do *cache* e *heap*, então este valor foi empregado em ambos (*cache* e *heap*). A última configuração de memória testada foi a sugerida pelo comando “*memrec*”, distribuindo 1,9 GB ao tamanho do *cache* e 3,5 GB ao tamanho do *heap*. A Tabela 2 apresenta a média dos resultados em segundos obtidos após a execução dos experimentos com as configurações de memória citadas.

Tabela 2 – Desempenho em consultas da carga de trabalho BI do *benchmark* LDBC com diferentes configurações de memória

Consulta	Padrão Neo4j	Padrão BI	Memrec
Q1	7,79	6,54	6,45
Q2a	122,87	122,40	102,91
Q2b	8,55	8,19	7,09
Q3	180,99	176,66	146,59
Q4	7045,06	5281,41	4480,51
Q5	1,55	1,89	1,22
Q6	205,88	161,27	147,42
Q7	6,14	4,17	3,99
Q8a	1259,29	1293,68	1297,07
Q8b	31,82	31,35	29,33
Q9	310,94	314,10	303,83
Q10a	14,33	13,40	16,43
Q10b	9,43	8,58	10,96
Q11	126,47	103,34	144,94
Q12	824,59	999,53	824,03
Q13	81,35	80,01	74,95
Q14a	280,86	279,74	256,34
Q14b	0,94	0,86	0,88
Q15a	3522,10	2836,46	2584,79
Q15b	3836,33	2946,75	2584,47
Q16a	1062,58	1190,23	1138,36
Q16b	0,39	0,40	1,45
Q17	91,64	76,10	79,79

Q18	96,68	78,01	83,85
Q19a	46,97	43,64	44,50
Q19b	38,94	39,64	36,67
Q20a	0,73	0,79	0,77
Q20b	0,33	0,35	0,33
Total	19215,68	16099,61	14410,07

Fonte: Autoria própria (2022)

É possível observar na Tabela 2 a queda no tempo de execução provocada pela mudança no tamanho do *cache* e *heap*, onde 57,14% das consultas executaram mais rápido com a configuração de memória "*memrec*" em relação às outras duas configurações empregadas. Grande parte acontece pelo ganho de desempenho nas consultas mais custosas (Q4 e Q15), que somadas, diminuíram o tempo de execução em 4.753,71 segundos (79,22 minutos) em relação a configuração padrão do Neo4j, e 1.414,85 segundos (23,58 minutos) em relação a configuração padrão do *benchmark*. A representação da diferença no tempo de execução das consultas Q4 e Q15 esta representada no Gráfico 4.

Gráfico 4 – Comparação do tempo de execução das consultas Q4 e Q15 com as diferentes configurações de memórias apresentadas

Fonte: Autoria própria (2022)

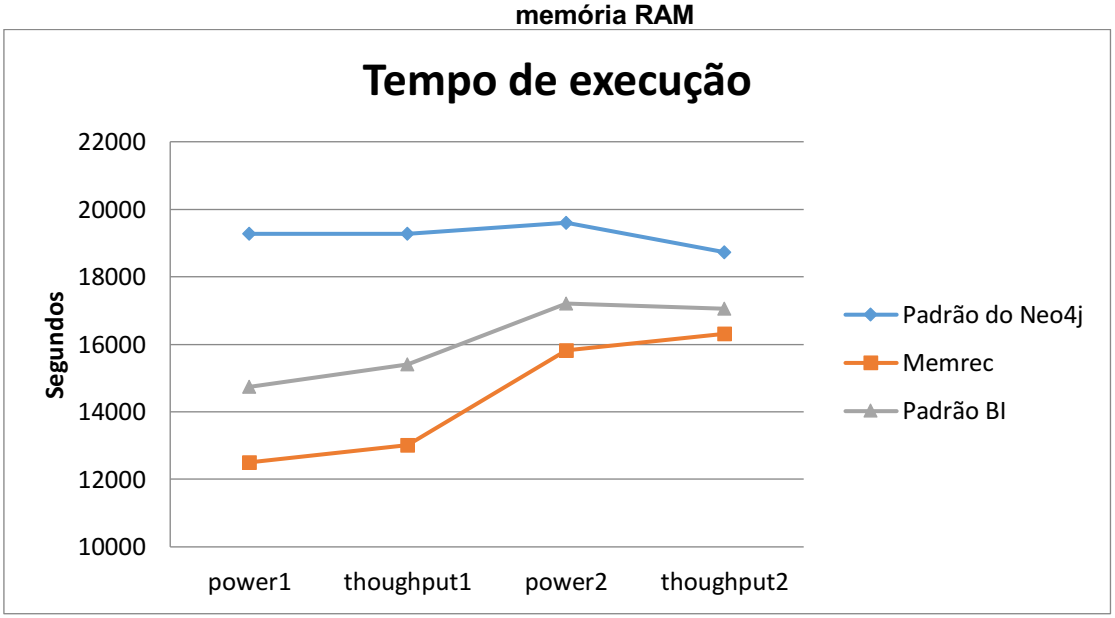
As consultas Q4 e Q15 possuem algumas características em comum, que as fazem criar um volumoso número de objetos na execução de seus comandos, como por exemplo, em sua composição, um dos principais gargalos é o reuso de objetos criados por comandos executados (ANGLES, 2020). Neste caso aumentar o valor do *heap* na configuração "*memrec*" foi efetivo sobre o desempenho das consultas. Isto porque, com a criação de uma quantidade maior de objetos, a melhor solução foi um *heap* grande o suficiente para armazená-los, também adiando o término do ciclo de

coleta de lixo, causando impacto positivo no desempenho das consultas que precisam reusar o resultado de comandos.

Também é importante mencionar que o comando "*memrec*" sugere uma distribuição de memória que aloca mais memória RAM para o *cache* e o *heap* que as outras duas configurações de memória da Tabela 2, deixando uma menor quantidade para o SO. Isto explica como algumas consultas se tornaram mais lentas em relação às outras duas configurações de memória (Q8a e Q11, por exemplo). O ocorrido é exemplo do que acontece quando o SO possuí pouca memória RAM para manipular índices. Além disso, o tamanho do *heap* configurado pode não ter sido o ideal para executar este tipo de consulta.

Da mesma maneira que na Subseção 5.1.1, o *power score (power@SF)* contido na presente Subseção é a média do cálculo realizado a partir de dois *power tests (power1* e *power2).* O Gráfico 5 os apresenta, em conjunto com os *thoughput tests (thoughput1 e thoughput2)*. Além disso, a média dos valores do Gráfico 5 compõe a linha "Total" da Tabela 2.

Gráfico 5 – Tempo de execução do LDBC com diferentes tipos de configuração do uso de memória RAM



Fonte: Autoria própria (2022)

Para a métrica utilizada pelo LDBC *benchmark* (BI), assim como no tempo de execução, as formas de distribuição da memória RAM foram avaliadas da seguinte maneira no *power score (power@SF)*: 55,10 pontos para a configuração padrão do Neo4j, 61,53 pontos para a configuração padrão do *benchmark,* e 68,71 pontos para a configuração *"memrec".* Portanto, as mudanças no tamanho do *heap*

e *cache*, não apenas tornaram a execução das consultas mais rápidas, como também a ferramenta pontua melhor o tempo de execução do conjunto das 28 consultas.

5.1.3 Índice *text*

O último experimento realizado teve como objetivo verificar se o emprego de índices do tipo *text* realmente diminuí o tempo de execução do *benchmark* LDBC quando comparado a utilização de índices *btree*, pois os testes contidos na Subseção 5.1.1 o tempo de execução obtidos com o uso de índices *text* foi 1.923,86 segundos (30,06 minutos) menor. Nos experimentos da presente Subseção, foram empregados os dois tipos de índices novamente (*btree* e *text*). Porém com a aplicação de *tuning* na memória sugerida pelo comando "*memrec*", pois foi a configuração que obteve o menor tempo de execução (1.689,54 segundos menor) conforme a Subseção 5.1.2. Desta forma, além da comparação de desempenho entre os tipos de índices, buscou-se também diminuir ainda mais o tempo de execução nas consultas da carga de trabalho BI. A Tabela 3 apresenta as médias em segundos da execução do *benchmark* com aplicação de *tuning* na distribuição de memória (*memrec*), utilizando índices do tipo *btree* e *text*.

Tabela 3 – Resultados utilizando *tuning* na memória e utilizando índices do tipo *text* pelo LDBC *benchmark*

Consulta	Memrec + Btree	Memrec + Text
Q1	6,45	6,54
Q2a	102,91	66,00
Q2b	7,09	4,54
Q3	146,59	156,89
Q4	4480,51	4059,14
Q5	1,22	0,87
Q6	147,42	131,85
Q7	3,99	4,05
Q8a	1297,07	1150,38
Q8b	29,33	25,99
Q9	303,83	372,05
Q10a	16,43	16,31
Q10b	10,96	11,02
Q11	144,94	157,50
Q12	824,03	179,87
Q13	74,95	94,48
Q14a	256,34	246,15
Q14b	0,88	0,79
Q15a	2584,79	2782,23
Q15b	2584,47	2789,05
Q16a	1138,36	1050,41
Q16b	1,45	1,81
Q17	79,79	4,63

Q18	83,85	67,95
Q19a	44,50	40,11
Q19b	36,67	40,16
Q20a	0,77	0,86
Q20b	0,33	0,41
Total	14410,07	13590,84

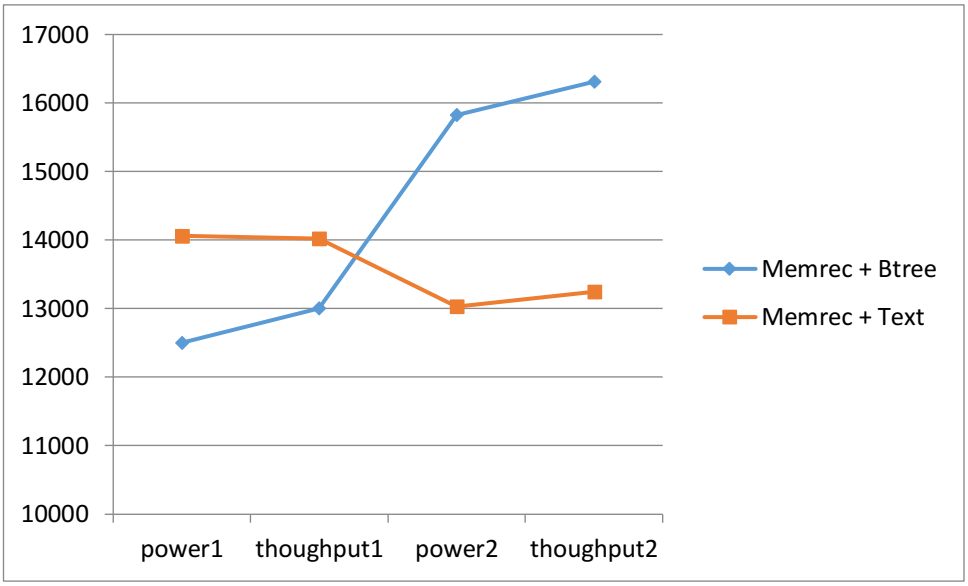
Fonte: Autoria própria (2022)

Conforme a Tabela 3, apesar do tempo médio de execução das consultas ter sido 819,23 segundos menor considerando índices *text,* apenas 53,57% das consultas obtiveram redução nos seus tempos de execução quando comparado ao emprego dos índices *btree*. Na configuração de memória padrão do *benchmark* LDBC para a carga de trabalho BI com índices do tipo *text*, a exemplo da Q15 (a e b), o tempo de execução foi 650,52 segundos (10,84 minutos) menor que índices *btree* (Tabela 1 da Subseção 5.1.1), onde as propriedades utilizadas estão principalmente relacionadas a datas. Isto já não ocorre quando utilizado a configuração de memória sugerida pelo comando "*memrec*" (Tabela 3), onde o tempo de execução das duas Q15 somadas com índices *text* foram 402,02 segundo (6,70 minutos) maior se comparados com os índices *btree*, provando que a alteração no tipo de índice não foi responsável pelo desempenho superior nos resultados dessas consultas na Subseção 5.1.1.

O foco do *benchmark* LDBC com a carga de trabalho BI não é a comparação de tipos de índices, porém observa-se que existem consultas em que o tempo de execução foi menor ao utilizar índices do tipo *text* sob o *btree* nas duas configurações de memória testadas (Padrão BI da Subseção 5.1.1 e *memrec da Subseção 5.1.3*). São exemplos, as consultas que realizam comparações entre *Strings*, como as possuem o predicado "IN" (Q4), ou a Q12, que possuía o predicado "*not null*", tiveram tempo de execução menor em ambas as configurações de memória (1.057,8 segundos, Tabela 1). A soma do tempo de execução dessas duas consultas tornou o tempo de execução 1.065.53 segundos (17,76 minutos) mais rápido na presente Subseção, sendo um diferencial no tempo de execução total do *benchmark (*Total da Tabela 3*).* Este, que é média dos valores dos *power tests*

(*power1 e power2*) e dos *thoughput tests* (*thoughput1* e *thoughput2*) conforme ilustrado no Gráfico 6.

Gráfico 6 – Tempo de execução do LDBC com diferentes índices e a configuração de memória "memrec"



Fonte: Autoria própria (2022)

O *power score (power@SF)*, que é a média do cálculo realizado a partir do *power1* e *power2* do Gráfico 6, apresenta uma pontuação melhor para índices do tipo *btree* (68,71 pontos), contra 65,36 pontos com o tipo *text*. Mesmo com um tempo de execução menor na maioria das consultas e no tempo total de execução, tendo uma diferença de 819,23 segundos (Tabela 3), de acordo com as métricas empregadas pelo LDBC *benchmark* com a carga de trabalho BI, o tempo de execução do conjunto das 28 consultas utilizando índices do tipo *text* não foi melhor avaliado que os índices do tipo *btree*.

6 CONCLUSÃO

O diferencial do modelo de armazenamento orientado a Grafos é que os dados possuem conexões (relacionamentos), o que o torna ideal em contextos, como redes sociais, investigações em crimes cibernéticos, e sistemas de recomendação em *e-commerce*. Além de não precisar abdicar totalmente das propriedades ACID, sendo aplicado dentro dos membros de um *cluster*, o tempo de execução de consultas por meio da navegação entre os dados se torna menor, quando a navegação pelos nós através dos relacionamentos precisa ser empregada.

O Neo4j é o SGBD mais popular atualmente na área de dados conectados, conforme citado no Capítulo 3 deste trabalho. Grande parte do sucesso se deve as características do SGBD que tornam as buscas mais eficientes, como a adjacência livre, podendo navegar entre os nós utilizando os relacionamentos, e utilizar grafos com atributos, onde os nós e arestas podem armazenar agregados de dados como valores.

Apesar do SGBD Neo4j ser ideal para realizar consultas com dados conectados, diminuindo o tempo de execução em consultas desse tipo, buscar por melhorias de desempenho é recomendado, tanto para a comunidade científica, quanto no meio empresarial. Neste contexto, técnicas de *tuning* relatadas na literatura, como o manual do SGBD e livros sobre a ferramenta, foram aplicadas neste trabalho.

Tanto no manual do Neo4j quanto na literatura, as técnicas de *tuning* mais citadas estão relacionadas aos parâmetros de configuração que consideram a distribuição de memória RAM destinados ao tamanho do *cache* e tamanho do *heap*. Com isso, é possível torná-los adequados para a carga de trabalho aplicada ao banco de dados. Com relação às consultas, o uso de índices é a principal recomendação, pois ao criar redundância na base de dados a partir de cópias das propriedades de nós e arestas facilitam na busca das mesmas.

Com os resultados obtidos nos experimentos realizados, melhorias de desempenho foram alcançadas aplicando técnicas de *tuning* nos seguintes casos: (1) A partir do comando "*memrec*", aumentar o tamanho do *cache* em relação à configuração padrão do Neo4j, armazenando dados de comandos recentemente utilizados, para reutilizá-los evitando acesso ao disco (*cache* quente); (2) Aumentar o tamanho do *heap*, tornando-o adequado para evitar que objetos sejam descartados

precocemente após o fim de um ciclo de coleta; (3) Também foi apresentado o impacto do uso de índices, em que após a retirada dos índices criados pelo *benchmark* para a base de dados, o tempo de execução aumentou; (4) Alteração no tipo do índice de *btree* para *text* também trouxe mudança no tempo de execução em várias consultas, melhorando o desempenho quando havia comparação entre *Strings*.

Nos três experimentos analisados, era de domínio qual a carga de trabalho aplicada à base de dados. Sendo assim, tanto as mudanças no tamanho da memória RAM destinadas ao tamanho do *heap* e tamanho do *cache*, quanto as mudanças nos tipos de índices apenas surtiram efeito porque tinha-se conhecimento sobre as consultas a serem executadas. Desta maneira, os desenvolvedores do Neo4j recomendam sempre checar as características da carga de trabalho que são executadas nas bases. Para isto, existe no Neo4j uma ferramenta chamada Halin (NEO4J LABS, 2022). Em suas funcionalidades estão contidos o monitoramento dos servidores do *cluster,* podendo ser verificados o uso do *cache* e do *heap*, e quais comandos estão sendo executados no SGBD no momento, além de disponibilizar um *log* da carga de trabalho. O objetivo é auxiliar o DBA a administrar melhor sua base de dados, principalmente na criação de índices, sendo recomendada a criação nas propriedades mais requisitadas. Também a distribuição de memória RAM, destinando uma quantidade grande o suficiente para o tamanho do *cache* e tamanho do *heap* para evitar acesso ao disco em consultas, e tornar adequado os ciclos de coleta de lixo do *garbage collector*.

6.1 Trabalhos futuros

No presente trabalho foram aplicadas técnicas de *tuning* na configuração da memória RAM e utilizados os diferentes tipos de índices da versão 4.4.12 do Neo4j em apenas um *hardware*. Uma sugestão de trabalhos futuros é realizar os mesmos experimentos, porém em servidores conectados em *cluster,* com a versão mais recente do SGBD empregado. Assim verificando se alterar o tamanho do *heap* e o tamanho do *cache* resultam em melhoria proporcional de desempenho quando comparado à execução em um único servidor. Também analisar os diferentes índices que estão inseridos na versão 5 do Neo4j, como o “*range*” e o “*point*”.

Outra sugestão de trabalho futuro é, por meio de um *cluster*, aplicar outras técnicas de *tuning*, como “fragmentar” o esquema da base de dados em partes

menores, colocando cada fragmento do esquema nos membros do *cluster* (*sharding*). Esta técnica foi empregada na pesquisa "*Sharding the LDBC Social Network*" (NEO4J DOCS, 2022), em que um grupo de desenvolvedores da ferramenta conseguiu diminuir o tempo de execução nas consultas do LDBC *benchmark* com a carga de trabalho *Interative Implementation*.

Utilizar a ferramenta Halin para monitorar as atividades no *cache* e *heap* pode obter explicações mais detalhadas do provável ganho de desempenho com o *tuning* aplicado. Como é possível verificar quais objetos estão armazenados no *heap* durante o ciclo de coleta de lixo, utilizar o Halin deixará evidente quais foram as razões para a diferença de desempenho. Além da possibilidade de verificar qual índice estava sendo empregado no plano de execução de cada consulta, apresentando com precisão qual o impacto dos tipos de índices.

Também analisar a aplicação de técnicas de *tuning* considerando outros modelos de armazenamento de dados *NoSQL,* como chave-valor ou família de colunas, podem trazer benefícios relacionados ao desempenho das soluções.

REFERÊNCIAS

ANGLES, Renzo. **A comparison of current graph database models**. In: 2012 IEEE 28th International Conference on Data Engineering Workshops. IEEE, 2012. p. 171-177.

ANGLES, Renzo et al. **The LDBC social network benchmark**. arXiv preprint arXiv:2001.02299, 2020.

BINI, Tarcizio Alexandre. **Análise da aplicabilidade das regras de ouro ao tuning de sistemas gerenciadores de bancos de dados relacionais em ambientes de computação em nuvem**. Dissertação (Doutorado em Ciência da Computação) – Programa de Pós-Graduação em Informática, Setor de Ciências Exatas da Universidade Federal do Paraná, 2014.

BRITO, Ricardo W. **Bancosde dados NoSQL x SGBDs relacionais**: análise comparativa. Faculdade Farias Brito e Universidade de Fortaleza, 2010.

CASSANDRA, 2022. Disponível em: <https://cassandra.apache.org/_/index.html/> Acesso em: 03 Abr. 2022.

CHANDRA, Deka Ganesh. **Base analysis of nosql database**. Future Generation Computer Systems, Elsevier, v. 52, p. 13–21, 2015.

CHAO, Joy. **Graph Databases for Beginners**: Native vs. Non-Native Graph Technology. Neo4j Blog, 2018. Disponível em <https://neo4j.com/blog/native-vs-non-native-graph-technology/> Acesso em: 23 Mar. 2022.

CODD, E. F. **A relational model of data for large shared data banks**. 1970. MD computing: computers in medical practice, v. 15, n. 3, p. 162-166, 1998.

COUCHBASE, 2022. Disponível em: <https://www.couchbase.com/> Acesso em: 03 Abr. 2022.

DB-Engines Ranking. Disponível em: <https://db-engines.com/en/ranking>. Acesso em: 03 Abr. 2022

DE SOUZA, Alexandre Morais. **Critérios para Seleção de SGBD NoSQL**: o Ponto de Vista de Especialistas com base na Literatura. In: Anais do X Simpósio Brasileiro de Sistemas de Informação. SBC, 2014. p. 149-160.

DOMINICO, Simone. **Tuning**: um estudo sobre a otimização de desempenho de sistemas gerenciadores de banco de dados relacionais sob carga de trabalho de suporte a decisão. Monografia (Tecnologia em Sistemas para Internet) – Curso Superior de Tecnologia em Sistemas Para a Internet, Universidade Tecnológica Federal do Paraná, 2013. Disponível em: <http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/7287/1/GP_COINT_2013_2_02.pdf>. Acesso em: 03 Abr. 2022.

ERVEN, Gustavo C. Galvão van: **Mdg-nosql**: modelo de dados para bancos nosql baseados em grafos. Em Dissertação (Mestrado Profissional em Computação Aplicada) - Universidade de Brasília. Universidade de Brasília, 2015. 7, 8

GREMLIN, 2022 . Disponível em: <https://tinkerpop.apache.org/gremlin.html/> Acesso em: 07 Jun. 2022.

HBASE, 2022. Disponível em: <https://hbase.apache.org/> Acesso em: 03 Abr. 2022.

HIBERNATE, 2022. Disponível em: <https://hibernate.org/> Acesso em: 07 Abr. 2022.

IBATIS, 2022. Disponível em: <https://ibatis.apache.org/> Acesso em: 07 Abr. 2022.

JONES, Richard; HOSKING, Antony; MOSS, Eliot. **The garbage collection handbook**: the art of automatic memory management. CRC Press, 2016.

KHAN, Wisal, and Waseem SHAHZAD. **Predictive performance comparison analysis of relational & nosql graph databases**. Int. J. Adv. Comput. Sci. Appl 8.5. 2017: 523-530.

KHASAWNEH, Tariq N.; AL-SAHLEE, Mahmoud H.; SAFIA, Ali A. **Sql, newsql, and nosql databases**: A comparative survey. In: 2020 11th International Conference on Information and Communication Systems (ICICS). IEEE, 2020. p. 013-021.

LÓSCIO, F; OLIVEIRA, D; PONTES, S. **Nosql no desenvolvimento de aplicações web colaborativas VIII Simpósio Brasileiro de Sistemas Colaborativos, Brasil**. 2011. Disponível em: <www.addlabs.uff.br/sbsc_site/SBSC2011_NoSQL.pdf> Acesso: 22/08/2019.

MICROSOFT DOCS. **Entender os modelos de armazenamento de dados**, 2022. Disponível em: <https://docs.microsoft.com/pt-br/azure/architecture/guide/technology-choices/data-store-overview> Acesso em: 03 Jun. 2022.

MICROSOFT SQL SERVER, 2022. Disponível em: <https://www.microsoft.com/pt-br/sql-server/sql-server-downloads> Acesso 25 Mai. 2022.

MONGODB, 2022. Disponível em: <www.mongodb.com/> Acesso em: 03 Abr. 2022.

MUS, MAM. **Comparison between SQL and NoSQL databases and their relationship with big data analytics**. 2019.

NEO4J, 2022. Disponível em: <www.neo4j.com/product/?ref=home-banner/> Acesso 03 Abr. 2022.

NEO4J CASE STUDIES. Graph Database Case Studies. Disponível em: <https://neo4j.com/case-studies/>. Acesso em: 15 Abr. 2022.

NEO4J DOCS. **Neo4j documentation**. 2022. Disponível em: <https://neo4j.com/docs>. Acesso em: 06 Abr. 2022.

NEO4J LABS. 2022. Disponível em: <https://neo4j.com/labs>. Acesso em: 15 Abr. 2022.

NEO4J TEAM. The Neo4j Operations Manual v3.5. San Mateo, 2019. 368 p. Disponível em: <https://neo4j.com/docs/pdf/neo4j-operations-manual-3.5.pdf>. Acesso em: 15 Abr. 2022.

ORACLE, 2022. Disponível em: <https://www.oracle.com/br/database/> Acesso 25 Mai. 2022.

PI, A. Turu, O. KOROGLU, E. Zimanyi. **Graph Databases and Neo4J**. University libre de Bruxelles, 2017. Disponível em: <https://cs.ulb.ac.be/public/_media/teaching/neo4jj_2017.pdf>. Acesso em: 12 Abr. 2022.

POSTGRESQL, 2022. Disponível em: < https://www.postgresql.org/> Acesso 25 Mai. 2022.

REDIS, 2022. Disponível em: <www.redis.io/> Acesso em: 03 Abr. 2022.

RIAK, 2022. Disponível em: <www.riak.com/> Acesso em: 03 Abr. 2022.

SADALAGE, Pramod J.; FOWLER, Martin. **NoSQL Essencial**: um guia conciso para o mundo emergente da persistência poliglota. Novatec Editora, 2019.

SAHATQIJA, Kosovare. **Comparison between relational and NOSQL databases**. In: 2018 41st international convention on information and communication technology, electronics and microelectronics (MIPRO). IEEE, 2018. p. 0216-0221.

SILBERSCHATZ, Abraham. **Sistema de banco de dados** - Rio de Janeiro: GEN LTC, 2020.

STROZZI, Carlo. **NoSQL**: a non-SQL RDBMS. Online verfügbar unter: http://www. strozzi. it/cgi-bin/CSA/tw7/I/en US/nosql/Home% 20Page, abgerufen am, v. 25, p. 2012, 1998.

TRAMONTINA, Gregório Baggio. **Database Tuning**: Configurando o Interbase e o PostgreSQL. 2008. Disponível em: <http://www.ic.unicamp.br/~geovane/mo410-091/Ch20-ConfigInterbasePosgres-art.pdf>. Acesso em: 16 Abr. 2022.

TPC, 2022. Disponível em: <https://www.tpc.org/tpch/default5.asp/> Acesso em: 10 Out. 2022.

VUKOTIC, Aleksa. **Neo4j in action**. Vol. 22. Shelter Island: Manning, 2015.

WANZELLER, D. **Investigando o Uso de Bancos de Dados Orientados a Documentos para Gerenciar Informações da Administração Pública**. Dissertação (Graduação em Computação - Licenciatura) – Universidade de Brasília – UnB. 2013.

ZDEPSKI, Cristofer. **PDDM**: um método de projeto de banco de dados aplicado à persistência poliglota. 2019. Dissertação de Mestrado. Universidade Tecnológica Federal do Paraná.

ZDEPSKI, C., BINI, T. A., MATOS, S. N. **An Approach for Modeling Polyglot Persistence**. In: ICEIS (1). 2018. p. 120-126.

APÊNDICE A – PASSOS PARA A REPLICAÇÃO DOS EXPERIMENTOS

Hardware:

- Processador Intel Core I7-3770K de 3.90 GHz;
- 8 MB de memória *cache;*
- 8 GB de memória RAM;
- 1 disco SATA de 1 TB e 7200 RPM;
- SO Linux Ubuntu 20.04.5 LTS com Kernel 5.15.0-56-generic;
- SGBD Neo4j versão 4.4.12.

Software:

- LDBC *benchmark* na versão 2.2.1;
- Carga de trabalho BI na versão 1.0.3;
- Base de dados com SF=1 sem cabeçalhos (*headers*).

Instruções para a execução do *benchmark*:

- Verificar quais índices serão importados para a base de dados no arquivo "*indices.cypher*", encontrado no diretório "*cypher/ddl*";
- No terminal do Linux:
  - Definição do fator de escala: "*export SF=1*";
  - Definição do diretório da base de dados: "*export NEO4J_CSV_DIR ="caminho absoluto para o diretório requisitado da base de dados""*;
  - Definição do tamanho do *heap* e *cache*: "*export NEO4J_END_VARS="${NEO4J_END_VARS-} –env NEO4J_dbms_memory_pagecache_size=1G –env NEO4J_dbms_memory_heap_max__size=1G"".*
- Executar o arquivo "*run-benchmark.sh*" do diretório "*cypher/scripts*";
- Verificar o resultado da execução do *benchmark* no diretório "*cypher/output*";
- Para o cálculo dos *Scores* (*power score* e *thoughput score*):
  - Executar o arquivo "*score-full.sh*" do diretório "*scripts*" seguindo o exemplo: "*scripts/score-full.sh cypher 1*".

ANEXO A - Lei n. 9.610, de 19 de fevereiro de 1998

Presidência da República
Casa Civil
Subchefia para Assuntos Jurídicos

LEI Nº 9.610, DE 19 DE FEVEREIRO DE 1998[2].

Altera, atualiza e consolida a legislação sobre direitos autorais e dá outras providências.

O PRESIDENTE DA REPÚBLICA Faço saber que o Congresso Nacional decreta e eu sanciono a seguinte Lei:

Título I - Disposições Preliminares

Art. 1º Esta Lei regula os direitos autorais, entendendo-se sob esta denominação os direitos de autor e os que lhes são conexos.

Art. 2º Os estrangeiros domiciliados no exterior gozarão da proteção assegurada nos acordos, convenções e tratados em vigor no Brasil.

Parágrafo único. Aplica-se o disposto nesta Lei aos nacionais ou pessoas domiciliadas em país que assegure aos brasileiros ou pessoas domiciliadas no Brasil a reciprocidade na proteção aos direitos autorais ou equivalentes.

Art. 3º Os direitos autorais reputam-se, para os efeitos legais, bens móveis.

Art. 4º Interpretam-se restritivamente os negócios jurídicos sobre os direitos autorais.

Art. 5º Para os efeitos desta Lei, considera-se:

I - publicação - o oferecimento de obra literária, artística ou científica ao conhecimento do público, com o consentimento do autor, ou de qualquer outro titular de direito de autor, por qualquer forma ou processo;

II - transmissão ou emissão - a difusão de sons ou de sons e imagens, por meio de ondas radioelétricas; sinais de satélite; fio, cabo ou outro condutor; meios óticos ou qualquer outro processo eletromagnético;

III - retransmissão - a emissão simultânea da transmissão de uma empresa por outra;

IV - distribuição - a colocação à disposição do público do original ou cópia de obras literárias, artísticas ou científicas, interpretações ou execuções fixadas e fonogramas, mediante a venda, locação ou qualquer outra forma de transferência de propriedade ou posse;

V - comunicação ao público - ato mediante o qual a obra é colocada ao alcance do público, por qualquer meio ou procedimento e que não consista na distribuição de exemplares;

VI - reprodução - a cópia de um ou vários exemplares de uma obra literária, artística ou científica ou de um fonograma, de qualquer forma tangível, incluindo qualquer armazenamento permanente ou temporário por meios eletrônicos ou qualquer outro meio de fixação que venha a ser desenvolvido;

VII - contrafação - a reprodução não autorizada;

VIII - obra:

a) em co-autoria - quando é criada em comum, por dois ou mais autores;

b) anônima - quando não se indica o nome do autor, por sua vontade ou por ser desconhecido;

c) pseudônima - quando o autor se oculta sob nome suposto;

d) inédita - a que não haja sido objeto de publicação;

e) póstuma - a que se publique após a morte do autor;

f) originária - a criação primígena;

g) derivada - a que, constituindo criação intelectual nova, resulta da transformação de obra originária;

h) coletiva - a criada por iniciativa, organização e responsabilidade de uma pessoa física ou jurídica, que a publica sob seu nome ou marca e que é constituída pela participação de diferentes autores, cujas contribuições se fundem numa criação autônoma;

i) audiovisual - a que resulta da fixação de imagens com ou sem som, que tenha a finalidade de criar, por meio de sua reprodução, a impressão de movimento, independentemente dos processos de sua captação, do suporte usado inicial ou posteriormente para fixá-lo, bem como dos meios utilizados para sua veiculação;

IX - fonograma - toda fixação de sons de uma execução ou interpretação ou de outros sons, ou de uma representação de sons que não seja uma fixação incluída em uma obra audiovisual;

X - editor - a pessoa física ou jurídica à qual se atribui o direito exclusivo de reprodução da obra e o dever de divulgá-la, nos limites previstos no contrato de edição;

XI - produtor - a pessoa física ou jurídica que toma a iniciativa e tem a responsabilidade econômica da primeira fixação do fonograma ou da obra audiovisual, qualquer que seja a natureza do suporte utilizado;

XII - radiodifusão - a transmissão sem fio, inclusive por satélites, de sons ou imagens e sons ou das representações desses, para recepção ao público e a transmissão de sinais codificados, quando os meios de decodificação sejam oferecidos ao público pelo organismo de radiodifusão ou com seu consentimento;

XIII - artistas intérpretes ou executantes - todos os atores, cantores, músicos, bailarinos ou outras pessoas que representem um papel, cantem, recitem, declamem, interpretem ou executem em qualquer forma obras literárias ou artísticas ou expressões do folclore.

Art. 6º Não serão de domínio da União, dos Estados, do Distrito Federal ou dos Municípios as obras por eles simplesmente subvencionadas.

---

[2] Disponível em: http://www.planalto.gov.br/ccivil_03/leis/l9610.htm.