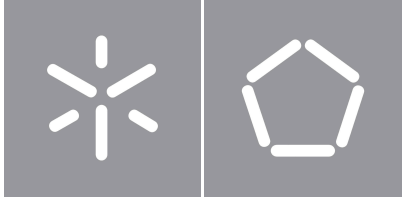
Universidade do Minho
Escola de Engenharia
Departamento de Informática

Carolina Alves da Cunha

Graph Databases for HR Relationships

outubro 2022



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Carolina Alves da Cunha

Graph Databases for HR Relationships

Dissertação de Mestrado
Mestrado Integrado em Engenharia Informática

Trabalho realizado sob a orientação do
Professor Doutor António Carlos da Silva Abelha

outubro 2022

Agradecimentos

Desejo exprimir os meus agradecimentos a todos aqueles que tornaram possível a realização da presente dissertação.

Em primeiro lugar, quero agradecer ao Professor Doutor Carlos Abelha, por ter aceite ser o orientador deste projeto, pela orientação prestada, pelo seu incentivo, disponibilidade e apoio que sempre demonstrou.

Agradeço, de igual forma, a todos os colaboradores da Konkconsulting - Consultoria Informática, S.A., por me permitirem e auxiliarem na abordagem a este tema. Agradeço, em especial, ao José Silva e Diogo Ferraz, por me terem acompanhado e guiado durante todo o percurso desta dissertação e pela forma amiga e generosa com que sempre me incentivaram e ajudaram.

À minha família, pelo apoio e motivação incondicionais que me prestaram durante todo o percurso da minha vida académica, e pela força que me deram para a elaboração da presente dissertação.

Ao João, por me ter acompanhado na aventura de um contexto empresarial, pela paciência e permanente incentivo que demonstrou durante todo este percurso.

Por último, agradeço aos meus amigos que me auxiliaram no decorrer da minha vida académica, pela paciência, apoio e conforto que me prestaram neste percurso.

Declaração de integridade

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

Resumo

Graph databases for HR relationships

As bases de dados relacionais não foram desenhadas para tratar de dados interligados, em oposição às tecnologias de grafos. A modelação de Recursos Humanos trabalha com estruturas altamente relacionadas entre si, pelo que a substituição de bases de dados relacionais por bases de dados de grafos neste contexto poderia melhorar a robustez e desempenho das aplicações.

Na presente dissertação, pretendeu-se comparar as tecnologias existentes (relacionais, não relacionais e grafos) e, dentro da tecnologia de grafos, avaliar qual a mais adequada em contexto de Recursos Humanos.

A revisão de literatura revelou que as bases de dados de grafos são mais eficientes para dados interligados do que as bases de dados relacionais e não relacionais. De todos os modelos de grafos analisados, o Neo4j foi o sistema de gestão de bases de dados que reuniu, num âmbito geral, as melhores características e, por este motivo, foi utilizado como prova de conceito. Foram realizadas três interrogações (duas com obtenção de dados de diferentes relações e uma com obtenção de dados de uma única relação), tendo-se obtido tempos de resposta de 41.48, 18.58 e 62.14ms vs. 804.68, 103.08 e 318.42ms entre bases de dados de grafos e relacionais, respetivamente.

Os resultados obtidos revelaram melhor desempenho do Neo4j na maioria das situações avaliadas. Em situações sem junções entre diferentes relações num ambiente relacional, o desempenho do SQL foi superior ao do Neo4j. Adicionalmente, verificou-se quebra significativa de desempenho do Neo4j quando foi analisado mais de metade do grafo.

As bases de dados de grafos apresentaram um melhor desempenho no tratamento de bases de dados altamente relacionadas.

Palavras-chave Bases de dados de grafos, Bases de dados relacionais, *Master Data Management*, Recursos Humanos

Abstract

Graph databases for HR relationships

Relational databases were not designed to handle linked data, as opposed to graph technologies. Human Resource Modelling works with highly interrelated structures. Therefore, replacing relational databases with graph databases in this context could improve applications' robustness and performance.

The aim of this dissertation was to 1) compare existing technologies (relational, non-relational and graphs) and 2) evaluate which is the most appropriate graph technology in a Human Resources context.

The literature review showed that graph databases are more efficient for interconnected data than relational and non-relational databases. Of all graph models analysed, Neo4j was the database management system that gathered, overall, the best features and therefore was used as proof of concept. Three queries were performed (two with data obtained from different relations and one with data obtained from a single relation), having obtained response times of 41.48, 18.58 and 62.14ms vs. 804.68, 103.08 and 318.42ms between graph and relational databases, respectively.

Results showed better performance of Neo4j in most of the evaluated situations. In a relational environment without joins between different relations, SQL outperformed Neo4j. Additionally, there was a significant drop in Neo4j's performance when more than half of the graph was analysed.

In conclusion, graph databases performed better in processing highly related databases.

Keywords Graph databases, Human Resources, Master Data Management, Relational databases

Índice

Acrónimos

ACID Atomicidade, Consistência, Isolamento, Durabilidade. viii, 4, 7, 11, 27, 30, 32, 40

API *Application Programming Interface*. viii, 8, 17, 29, 30, 39, 40, 45, 46, 51, 54, 56

AQL *ArangoDB Query Language*. viii, 30

BASE Basically Available, Soft-State, Eventually Consistent. viii, 11

BI Business Intelligence. viii

CAP *Consistency, Availability, Partition Tolerance*. viii, 4

CRUD *Create, Read, Update, Delete*. viii, 7, 18, 20

CSV Comma-Separated Values. viii, 44, 45

DI *Dependency Injection*. viii, 47

IoT *Internet of Things*. viii, 7, 40

JVM *Java Virtual Machine*. viii, 33

LCD Linguagem de Controlo de Dados. viii, 9, 30

LDD Linguagem de Definição de Dados. viii, 9, 30

LMD Linguagem de Manipulação de Dados. viii, 9, 30

MDM *Master Data Management*. viii, 19

MVC Model-View-Controller. viii, 46, 47

MVCC *Multiversion Concurrency Control*. viii, 32, 33

OLAP *Online Analytical Processing*. viii, 7

OLTP *Online Transaction Processing*. viii, 7, 18, 27

RDF *Resourse Description Framework*. viii, 16, 17, 38–40

REST *Representational State Transfer*. viii, 46

RH Recursos Humanos. viii, 1, 2, 19, 56

RU *Response Units*. viii, 30, 41

SGBD Sistema de Gestão de Base de Dados. viii, 1, 4, 8, 9, 26, 27, 32, 35, 39, 40, 56

SPARQL *SPARQL Protocol and RDF Query Language*. viii, 8, 38

SQL *Structured Query Language*. viii, 6, 8, 9, 19, 20, 22, 29, 30, 32, 34, 36, 41, 57

URI *Uniform Resource Identifiers*. viii, 16, 17

W3C *World Wide Web Consortium*. viii, 16

Índice de Figuras

Índice de Tabelas

Lista de Algoritmos

CAPÍTULO 1

Introdução

Neste capítulo, serão apresentados a motivação e os objetivos que proporcionaram o desenvolvimento desta dissertação.

1.1 Motivação

Os dados constituem a forma mais primitiva de representar factos ou afirmações, sendo o fundamento das aplicações modernas. Estes dados têm vindo a apresentar um rápido crescimento em volume, criação e variedade, conduzindo ao aumento da sua complexidade e consequente taxa de crescimento das inter-relações entre si (Neo4j, 2016). Uma base de dados é uma coleção de informação estruturada relacionada entre si, que persiste durante um determinado período de tempo (Gouveia, 2014). Um Sistema de Gestão de Base de Dados (SGBD) permite criar e gerir uma base de dados, facilitando o processo de definição, construção, manipulação e manutenção destes. A capacidade de responder de forma imediata e flexível às questões colocadas pelas aplicações está dependente do SGBD utilizado.

Em contexto de Recursos Humanos (RH), são tipicamente utilizadas bases de dados relacionais, caracterizadas pelo controlo de redundância e consistência de dados, assim como o reforço da segurança e integridade dos mesmos. No entanto, a modelação de dados de RH lida intensamente com estruturas altamente relacionadas entre si, modelo esse que encaixa perfeitamente no caso de uso de tecnologias de grafos. Por este motivo, a substituição de bases de dados relacionais por bases de dados de grafos neste contexto e a sua conexão a aplicações pode ser o caminho a seguir para o futuro, tornando estas aplicações mais robustas e com melhor desempenho.

1.2 Objetivos

A presente dissertação tem como objetivo principal o estudo de bases de dados de grafos, aprofundando o conhecimento sobre o seu funcionamento e métodos de interligação com aplicações *web*. Ademais, procura-se estabelecer uma analogia com um paradigma relacional, visto serem habitualmente utilizadas em contexto de RH.

Desta forma, foram definidos os seguintes objetivos específicos:

- Explorar as tecnologias atualmente existentes e as suas propriedades diferenciadoras.
- Analisar a melhor metodologia a adotar para desenvolver modelos de bases de dados de grafos em contexto de RH.
- Avaliar os diferentes casos de uso e as suas limitações técnicas.
- Definir um sistema para utilização como prova de conceito.
- Utilizar amostras de bases de dados de RH sobre a prova de conceito, de modo a aferir a realização de consultas mais céleres.

1.3 Questões de Investigação

Com base nos objetivos enumerados, foram formuladas algumas questões de investigação que se pretendem ver respondidas com esta dissertação.

- Poderá a utilização de bases de dados de grafos, num contexto de RH, melhorar o desempenho das aplicações?
- Para o caso de estudo, qual é a melhor metodologia a adotar para desenvolver modelos de bases de dados de grafos em contexto de RH?

1.4 Estrutura da Dissertação

Esta dissertação encontra-se dividida em cinco capítulos. No capítulo atual, são apresentados a motivação que conduziu à elaboração da presente dissertação e os objetivos que se pretende atingir. No capítulo que se segue, são descritos os fundamentos teóricos que contextualizam os temas abordados no documento. O capítulo 3 apresenta e discute as motivações para utilizar bases de dados de grafos, sendo sumariamente descritas as diferentes metodologias aplicáveis no problema. No capítulo 4, são apresentados os detalhes da metodologia implementada como prova de conceito, sendo interpretados os resultados obtidos. No último capítulo, são expostas as conclusões obtidas da elaboração do projeto.

CAPÍTULO 2

Fundamentos Teóricos

O armazenamento de informação existe desde o início da Humanidade. No decurso dos tempos foram desenvolvidos elaborados sistemas de bases de dados por escritórios do governo, bibliotecas, hospitais e organizações empresariais, e alguns dos princípios básicos desses sistemas são ainda atualmente utilizados. No entanto, com a crescente quantidade de registos, surgiu a necessidade de melhorar a organização e disponibilidade dos dados para facilitar a sua consulta posterior.

Nos primeiros anos da criação dos computadores, os "cartões perfurados" eram usados para entrada, saída e armazenamento de dados. Estes cartões ofereciam uma forma rápida de inserir e recuperar dados. Em 1890, Herman Hollerith criou uma máquina para a leitura, contagem e classificação dos cartões de censos, cujos orifícios representavam as informações recolhidas.

Durante a década de 1960, a maior capacidade de processamento e maior armazenamento local e externo proporcionado pelos computadores permitiu a transição do modo como os dados eram tratados, obtendo-se um meio que possibilitava poupar espaço, reduzir os custos de armazenamento e facilitar a consulta destes dados (Berg et al., 2013).

Em 1970, Edgar Codd, no artigo "*A Relational Model of Data for Large Shared Data Banks*", descreveu uma nova forma de estruturar dados em tabelas. Neste, evidenciava-se uma visão relacional onde o esquema da base de dados é desconectado do armazenamento de informações físico, conferindo uma maior independência dos dados em relação aos programas.

Este modelo inovou o modo de gerir a informação numa base de dados. Migrou-se de sistemas hierárquicos baseados em ficheiros para uma base de dados relacional, com tabelas que contêm os registos da informação, o que facilita a sua gestão.

Definiu-se como transação o conjunto de operações de leitura e escrita, individuais e distintas, que são tratadas

como uma unidade de trabalho indivisível. Estas transações são geridas pelo SGBD, que deve garantir a consistência dos dados, respeitando as características Atomicidade, Consistência, Isolamento, Durabilidade (ACID) e aplicando métodos de recuperação. Estas características asseguram que qualquer alteração realizada a uma base de dados é atómica, consistente e persistente. Para além disso, as transações devem ser isoladas, isto é, as operações de uma transação não devem afetar as operações de outra transação que esteja a decorrer concorrentemente.

A crescente complexidade e abundância de dados criou a necessidade de distribuição das bases de dados. Num ambiente distribuído, as bases de dados têm como princípio o Teorema *Consistency, Availability, Partition Tolerance* (CAP). Este teorema dita que um sistema de bases de dados apenas consegue, na melhor das hipóteses, garantir duas das três propriedades que se seguem:

- Consistência: Os dados ficam imediatamente disponíveis em todas as instâncias de um sistema distribuído;
- Disponibilidade (*Availability*): Os dados encontram-se sempre disponíveis para consulta e manipulação;
- Tolerância no particionamento (*Partition tolerance*): O sistema permanece operacional quando na ocorrência de uma falha de rede.

2.1 Bases de dados relacionais

O modelo relacional apresentado por Edgar Codd é baseado no conceito matemático de relação, que é fisicamente representado como uma tabela. Por este motivo, nas bases de dados relacionais, existe uma clara distinção entre relação e relacionamento, este último designando a ligação entre duas tabelas de dados.

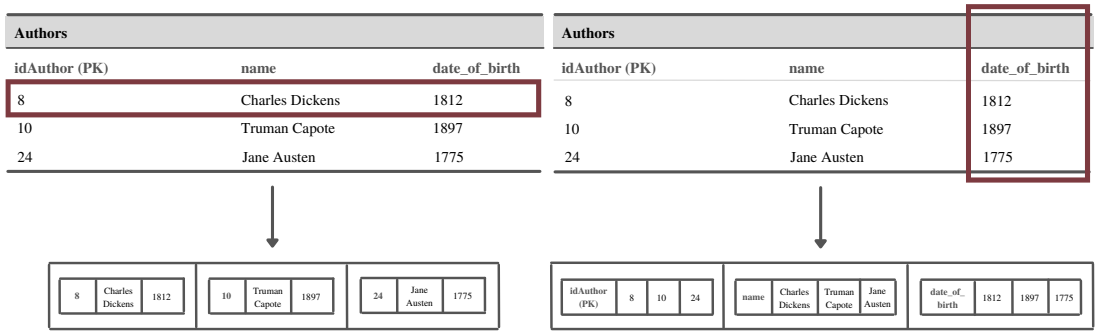
Numa base de dados relacional, os dados são organizados em tabelas bidimensionais que podem partilhar dados entre si. Esta especificidade permite, numa única interrogação, a criação de uma nova tabela com informação relevante das tabelas iniciais (IBM, 2019).

2.1.1 Armazenamento físico dos dados

Os sistemas de bases de dados envolvem sempre memória secundária, como discos, capazes de armazenar grandes quantidades de dados que persistem no tempo.

Existem diversas abordagens para o armazenamento físico dos dados de uma base de dados. O *layout* em linha (Figura 1a) consiste no armazenamento em memória secundária de registos de comprimento fixo, cuja inserção é realizada através da junção de novos registos no final da linha. Quando é necessária uma operação de consulta ou atualização de um registo, o bloco do disco (linha) é movido na sua totalidade para a memória principal. Esta abordagem apresenta adversidades, caso o armazenamento em memória não seja suficiente para os dados. Adicionalmente, a remoção de registos ou a não utilização do espaço reservado para cada registo levam ao desperdício de espaço em memória.

Figura 1: Armazenamento físico dos dados em *layout* em linha (a) e *layout* em coluna (b)



Esta dificuldade pode ultrapassar-se com a implementação de uma *offset table*, onde são guardados os apontadores para os registos pertencentes ao bloco. Nesta estratégia, a tabela cresce a partir do início do bloco e os novos registos são colocados a partir do final do mesmo bloco, recuperando o espaço ocupado por registos eliminados (i.e., não existe fragmentação). O crescimento dos registos não tem interferência com as referências da tabela e os registos podem ser migrados, deixando o endereço de reencaminhamento.

Quando um ou mais registos são de comprimento variável, devem conter informação suficiente para serem encontrados no bloco. Uma metodologia criada para encontrar os registos é a colocação dos campos de tamanho fixo no início do registo, seguidos dos campos de tamanho variável. No cabeçalho de cada registo são colocados o tamanho do registo e os apontadores para o início dos campos de tamanho variável.

Em alternativa ao *layout* em linha, existe o *layout* em coluna (Figura 1b), onde se guarda cada coluna como um registo. Ao armazenar os dados em colunas, obtém-se uma consulta sequencial eficiente, minimizando a transferência de dados e permitindo a sua compressão. Deste modo, é possível reconstruir a relação a partir dos valores das diferentes colunas. Adicionalmente, a inserção dos dados é eficiente, dado que é realizada uma concatenação no final da coluna.

2.1.2 Indexação

Um índice é uma estrutura de otimização do desempenho de uma base de dados que permite minimizar os acessos ao disco durante o processamento de uma interrogação. A utilização de um índice é apenas vantajosa quando o número total de registos da tabela que vai ser processada é consideravelmente menor que o tamanho da tabela total. Isto porque, caso sejam lidos muitos registos em relação ao tamanho da tabela, e devido à ordenação da estrutura de apontadores do índice, a mesma página pode necessitar de ser carregada para memória várias vezes, prejudicando o seu desempenho.

Algumas bases de dados suportam diversos tipos de indexação dos dados, sendo as *B-Tree* a estrutura de

indexação base para a maioria dos mecanismos de armazenamento.

Nesta estrutura, os valores são ordenados de forma semelhante às árvores binárias de procura, e a procura é realizada do mesmo modo. As *B-Tree* são, no entanto, árvores balanceadas, pelo que todos os ramos possuem o mesmo comprimento. Nestas, cada nó corresponde a uma página. A inserção e remoção de elementos são realizados maioritariamente dentro das páginas existentes, pelo que as modificações são realizadas numa única página e a divisão de nós é pouco habitual. No que diz respeito às consultas, é verificado o intervalo onde se encontram os dados pretendidos; posteriormente, são lidas sequencialmente as linhas correspondentes aos apontadores no intervalo. A facilidade em alternar entre as diferentes páginas do índice garante uma procura eficiente. Este tipo de índices suporta *identity lookup*, interrogações de intervalos, travessia ordenada e atualizações. Apesar da eficiência na procura, a travessia entre os diferentes níveis de uma tabela condiciona a leitura repetida da mesma página, uma vez que a ordem dos índices não é necessariamente a ordem das linhas da tabela.

Os índices agrupados (*clustered indexes*) surgiram na tentativa de ordenar as linhas de uma tabela conforme a ordem das linhas correspondentes no índice. Deste modo, quando se pretende uma pesquisa na tabela, para um mesmo parâmetro, vão ser lidas as linhas necessárias, presentes em páginas consecutivas do índice (Garcia-Molina et al., 2014).

Para além destas, podem ser utilizadas outras estruturas de indexação, como índices em *hash*, índices invertidos, índices *bitmap*, entre outros.

2.1.3 Junção de dados

Nestas bases de dados, a resposta a consultas de relacionamentos entre relações envolve operações de junção, que apresentam um custo exponencial à medida que aumentam as consultas e relações. Estas operações são utilizadas quando se pretende combinar registos de uma ou mais relações, tendo por base uma condição de junção.

Se não existir qualquer condição de junção, estas consultas requerem o produto cartesiano dos índices das relações. Assim sendo, cada linha da primeira relação é combinada com as linhas da segunda relação. No entanto, a utilização do produto cartesiano consome computação e memória em grande escala, uma vez que a operação $R_1 \times R_2$ origina um número elevado de linhas (Bhanuprakash et al., 2014).

2.1.4 Casos-tipo

Os diferentes tipos de bases de dados adequam-se a situações distintas, sendo representadas por casos-tipo.

A utilização das bases de dados relacionais deve ter em consideração as seguintes características que as definem:

- Utilizam a linguagem padrão *Structured Query Language* (SQL), que apresenta uma curva de aprendizagem

pequena e permite uma execução fácil e simples de tarefas (Connolly and Begg, 2015).

- As regras de normalização aplicadas aos dados de uma base de dados relacional garantem a eliminação de informação redundante, impedindo a sua inconsistência e reduzindo o espaço de armazenamento da mesma.
- Seguem o modelo transacional ACID, assegurando elevada confiabilidade das transações.
- São construídas com base num esquema bem definido e organizado, pelo que as linhas de uma coluna de uma tabela terão sempre a mesma estrutura.

Assim sendo, as bases de dados relacionais são adequadas para aplicações *Online Transaction Processing* (OLTP), uma vez que oferecem suporte a operações *Create, Read, Update, Delete* (CRUD) e rápidos tempos de resposta para pequenas quantidades de dados e permitem o acesso por parte de um grande número de utilizadores. Por este motivo, são bases de dados ideais quando a atomicidade é uma prioridade, como é o caso de transferências bancárias.

As soluções de *Internet of Things* (IoT) exigem velocidade e a capacidade para recolher e processar dados de dispositivos. As bases de dados relacionais permitem a incorporação em dispositivos de *gateway* e a gestão de dados de série temporal gerados por dispositivos IoT.

Além disto, em ambiente de *Data Warehouse*, as bases de dados relacionais podem ser otimizadas para *Online Analytical Processing* (OLAP), onde os dados históricos são analisados para *Business Intelligence*. A modelação dimensional é uma técnica de estruturação de dados otimizada para o armazenamento de dados num *Data Warehouse*, consistindo em tabelas de factos e de dimensão. A utilização de uma abordagem dimensional permite facilitar as consultas de um elevado número de registos e resumir os dados de diversas formas (IBM, 2021).

As bases de dados relacionais apresentam, contudo, algumas desvantagens. O armazenamento dos dados em tabela pode provocar um aumento considerável da complexidade do processo, caso os dados não consigam ser facilmente encapsulados numa tabela, isto é, em caso de ausência de capacidade de armazenar a totalidade dos dados (Jatana et al., 2012). Adicionalmente, estas bases de dados dependem de escalabilidade vertical, o que representa custos muito elevados quando existe necessidade de adicionar novos recursos. Quando a escalabilidade atinge o seu limite máximo, os dados necessitam de ser distribuídos por diferentes servidores, aumentando significativamente a complexidade da junção de tabelas e o desempenho destas operações (Mohamed et al., 2014). As propriedades ACID, apesar de garantirem um ambiente seguro para operar sobre dados, requerem diversos métodos de *locking*, podendo provocar indisponibilidade lógica.

Os esquemas rígidos e modelação complexa não são apropriados para suportar a grande variedade de dados (Neo4j, 2016): a inflexibilidade do esquema das bases de dados relacionais obriga à utilização de colunas com valores nulos (*sparse tables*) para operações que requeiram maior flexibilidade, abordagem que exige um incremento

considerável da quantidade de código utilizado e maior número de exceções neste. Adicionalmente, estas bases de dados não suportam facilmente interrogações recursivas; a complexidade destas interrogações apresenta grande impacto na sintaxe e desempenho, com desnormalização excessiva (Bechberger and Perryman, 2020).

As bases de dados relacionais apresentam, portanto, e paradoxalmente ao que o nome poderia indicar, limitações importantes que condicionam a sua utilização para lidar com dados relacionados, pelo que, nestes casos, diferentes SGBD devem ser considerados.

2.1.5 Linguagens de Interrogação

Um SGBD deve fornecer linguagens e interfaces apropriadas para a realização de consultas e manipulação dos dados por parte dos utilizadores. Neste sentido, existem dois paradigmas das linguagens de interrogação que se aplicam às bases de dados: as linguagens imperativas e as linguagens declarativas.

Linguagens Imperativas Estas linguagens são utilizadas para descrever minuciosamente o modo como se pretende realizar alguma operação, existindo, para o efeito, um controlo explícito sobre todos os passos a realizar.

Apesar de não existirem linguagens interrogativas integralmente imperativas, o *Gremlin* e a *Application Programming Interface* (API) Java do *Neo4j* são dois exemplos que incorporam características imperativas. No entanto, as linguagens imperativas são limitantes e não são *user-friendly*, exigindo bastante conhecimento sobre o seu funcionamento e implementação física (Sasaki et al., 2018).

Linguagens Declarativas A adoção dos SGBD baseados no "Modelo Relacional" criou dificuldades em escrever consultas em álgebra ou cálculo relacional.

A conceção da linguagem declarativa, como por exemplo o SQL, permitiu a colmatação desta dificuldade, através da elaboração de consultas em inglês sem perder, no entanto, o poder expressivo da álgebra relacional (Gouveia, 2014), simplificando o desenvolvimento das bases de dados.

As linguagens declarativas permitem que os utilizadores expressem o padrão de dados que pretendem obter (que condições devem reunir e como a informação deve ser transformada), permitindo que o otimizador do sistema da base de dados seja responsável pela decisão dos métodos e índices utilizados para a sua obtenção, e em que ordem executará as várias partes da interrogação. Assim, contrariamente às linguagens imperativas, estas requerem apenas instruções gerais sobre a tarefa a realizar, ao invés dos detalhes da sua resolução.

Por este motivo, são linguagens de fácil compreensão e utilização (Sasaki et al., 2018). O SQL, Cypher, *SPARQL Protocol and RDF Query Language* (SPARQL) e Gremlin são algumas das linguagens de interrogação declarativas frequentemente utilizadas.

SQL

As instruções SQL podem ser caracterizadas de acordo com as suas funcionalidades, sendo subdivididas em três categorias de instruções. A Linguagem de Definição de Dados (LDD) permite criar, modificar e apagar objetos da base de dados, tendo associadas as instruções *create*, *alter* e *drop*. Por sua vez, a Linguagem de Manipulação de Dados (LMD) permite definir comandos para realizar consultas, modificações, inserções e remoções dos dados armazenados. Nesta categoria, encontram-se instruções como *select*, *insert*, *update* e *delete*. Por fim, a Linguagem de Controlo de Dados (LCD) permite gerir o controlo de acesso a objetos específicos na base de dados, restringindo ou garantindo o mesmo, através dos comandos *revoke* e *grant*, respetivamente.

O processamento de interrogações SQL é realizado de forma semelhante por grande parte dos SGBD. Para o processamento de uma declaração *SELECT*, os passos base incluem um *parser* que verifica a instrução e a divide em unidades lógicas, como palavras-chave, expressão, operadores e identificadores. Uma árvore de consulta é posteriormente construída, descrevendo as etapas lógicas necessárias para transformar os dados de origem no formato exigido pelos resultados. O *query optimizer* analisa, então, o melhor método de acesso às tabelas de origem. A árvore de consulta é atualizada para a versão otimizada das etapas lógicas e chamada pelo plano de execução. De seguida, o mecanismo relacional efetua o plano de execução, solicitando ao mecanismo de armazenamento os dados dos conjuntos de linhas das tabelas processadas e retornando o conjunto de resultados ao cliente (Microsoft, 2021a).

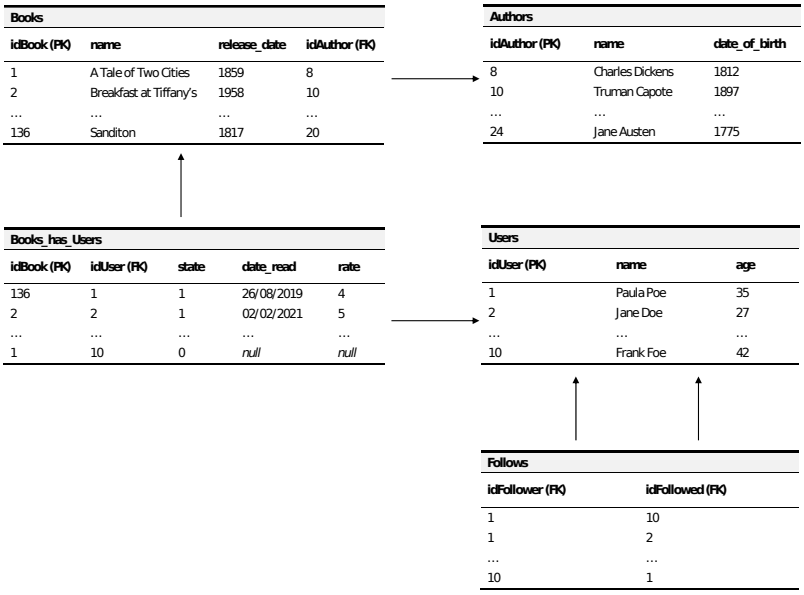
2.1.6 Exemplo

De modo a visualizar a dissemelhança entre as diferentes bases de dados, foi desenvolvido um exemplo prático que visa simular uma plataforma de partilha da lista de leitura dos seus utilizadores. Para este fim, foram consideradas as entidades:

1. **Books**, representando um livro pertencente à lista de leitura de um utilizador, com nome e data de lançamento original, e identificador referente ao escritor;
2. **Author**, com o nome e ano de nascimento de um autor;
3. **User**, que designa um utilizador da plataforma, contendo o seu nome e idade;
4. **Book_has_User**, entidade que indica os livros presentes na lista de leitura de cada utilizador, com uma *flag* que especifica se este livro já foi lido ou se pertence aos livros que o utilizador pretende ler;
5. **Follow**, correspondente à lista de seguidores de um utilizador.

A incapacidade de lidar com dados interligados implica a necessidade de criação de chaves estrangeiras nas relações que estabelecem associações entre si. Assim, a relação *Books* exige a chave estrangeira do seu respetivo autor, garantindo o relacionamento entre estes.

Figura 2: Modelação da plataforma de partilha da lista de leitura numa base de dados relacional



A Figura 2 representa a modelação deste exemplo prático numa base de dados relacional.

2.2 Bases de dados não relacionais

No decorrer da última década, tem sido notório o crescimento dos dados em volume, dinâmica de alteração dos mesmos, e sua variedade.

Nos modelos relacionais, o aumento do volume de dados implica um maior tempo de execução das interrogações devido ao acréscimo do tamanho das tabelas e do número de junções. Para além disso, a rápida modificação dos dados e respetivas estruturas requerem níveis elevados de edições (*write loads*) e de picos de atividade, provocando um grande impacto no desempenho da base de dados.

De modo a combater os desafios encontrados, as bases de dados não relacionais foram desenvolvidas com modelos de dados flexíveis e otimizadas para muitos *write loads*, sendo desenhadas para se ajustarem a diferentes tipos de dados (Sasaki et al., 2018).

As bases de dados não relacionais (*Not Only SQL*) são bases de dados não tabulares que armazenam os dados de forma distinta das tabelas relacionais. Podem dividir-se em duas categorias principais: as bases de dados orientadas à agregação, e as bases de dados não orientadas à agregação. As primeiras englobam *key-value*, documentos e família de colunas.

As bases de dados não relacionais não foram, no seu grosso conjunto, desenhadas para armazenar conjuntos de dados interligados. Uma estratégia utilizada para contornar este problema é a inclusão do identificador de um agregador no campo pertencente a outro agregador (chave estrangeira). Esta estratégia é, no entanto, dispendiosa, uma vez que requer a junção de agregadores (Sasaki et al., 2018).

As bases de dados não relacionais orientadas à agregação têm principal foco na disponibilidade e

desempenho das transações, pelo que abdicam da consistência e isolamento das transações que caraterizam as propriedades ACID e que as tornam altamente complexas. Esta abordagem designa-se por Basically Available, Soft-State, Eventually Consistent (BASE); nesta, a consistência dos dados é da responsabilidade do programador e não da própria base de dados (Chandra, 2015). As bases de dados não orientadas à agregação obedecem, por sua vez, às propriedades ACID.

2.2.1 Key Value

As bases de dados *key value* são estruturas que armazenam valores identificados por uma chave única (Figura 3b). Quando se pretende implementar uma base de dados distribuída, estes valores são repartidos por grupos por diferentes *buckets*. De modo a garantir tolerância a faltas e permitir balanceamento de carga, cada *bucket* é, posteriormente, replicado para diversas máquinas. Por este motivo, oferecem elevada disponibilidade e escalabilidade.

Estas bases de dados utilizam estruturas de indexação compactas e eficientes, permitindo uma procura eficiente e rápida de valores por chave. São, por este motivo, ideais para sistemas que exigem a consulta de dados em tempo constante (MongoDB, 2021).

Nestas bases de dados, os dados são opacos, pelo que a obtenção de informação proveniente de diferentes registos exige a utilização de métodos externos, como o MapReduce, revertendo resultados com elevada latência (Robinson et al., 2015).

Apesar de estas bases de dados garantirem armazenamento e obtenção eficientes de dados, também frequentemente fornecem o total dos valores, sendo necessários filtros de informação para coleta selecionada dos dados desejados, o que torna as interrogações ineficientes (Sasaki et al., 2018).

As bases de dados *key value* são comumente utilizadas para armazenamento de dados em memória cache para reduzir a necessidade de leituras e escritas em sistemas sediados em disco mais lentos (Hazelcast, 2021a). Por este motivo, as aplicações *web* recorrem frequentemente a estas bases de dados para armazenar informações e preferências da sessão do utilizador, permitindo leituras e escritas rápidas.

Estas bases de dados são também indicadas para recomendações e anúncios em tempo real, uma vez que permitem o rápido acesso e apresentação de novas recomendações (ou anúncios) conforme o utilizador que visita o sítio.

2.2.2 Documentos

Em bases de dados de documentos, os dados são armazenados em documentos de identificador único, funcionando como estruturas *key-value* (Figura 3a). Estes documentos incluem habitualmente listas e *maps*, permitindo a existência de hierarquias naturais (Sasaki et al., 2018).

Nos documentos e nos campos das diferentes hierarquias são frequentemente utilizados índices, abdicando do desempenho da escrita, de forma a melhorar o desempenho da leitura, facilitando assim o acesso aos documentos de acordo com os seus atributos. Por conseguinte, a obtenção de dados requer a leitura de um menor número de registos da base de dados. Em caso (pouco habitual) de ausência de índices, e uma vez que este fenómeno cursa com aumento a latência das interrogações realizadas, e recurso a *frameworks* de computação paralela externas ajuda a ultrapassar esta latência.

A não interligação do modelo de dados destas bases de dados permite atribuir-lhes uma escalabilidade horizontal. Esta característica facilita a criação de mecanismos de controlo de escritas concorrentes no mesmo documento. Estes mecanismos diferem entre diferentes bases de dados de documentos, podendo 1. distinguir e conciliar escritas em diferentes zonas do documento, 2. recorrer a *timestamps* para combinar diferentes escritas no mesmo documento independentemente da zona de escrita, ou 3. replicar automaticamente o estado do documento concorrentemente acedido em instâncias.

Em oposição às bases de dados *key value* e família de colunas, estas bases de dados necessitam, frequentemente, do planeamento de *sharding* para garantir a escalabilidade horizontal (Robinson et al., 2015).

Dada a flexibilidade dos seus esquemas, estas bases de dados são capazes de armazenar documentos com diferentes atributos e valores. Por este motivo, são uma solução prática e eficiente para perfis *online*, facilitando o armazenamento das informações específicas de cada utilizador. Simplificam, também, a gestão de conteúdo através da criação e incorporação de novos tipos de conteúdo, como imagens, comentários, vídeos, entre outros.

A capacidade de extrair informação operacional em tempo real é fundamental em ambientes de negócio. A utilização de bases de dados de documentos permite armazenar e gerir dados provenientes de qualquer fonte, e alimentar, simultaneamente, os dados para o mecanismo de *Business Intelligence* selecionado (DocumentDB, 2021).

2.2.3 Família de colunas

As bases de dados de família de colunas consistem numa tabela esparsa (possui elevada quantidade de elementos com valores nulos), cujas linhas são compostas por um número arbitrário de colunas (par chave-valor). Nestas bases de dados, a chave de cada linha concede indexação natural à tabela (Sasaki et al., 2018). Por sua vez, cada linha fornece uma estrutura de *hashmap* aninhada, sobre a qual se desnormalizam os dados (Figura 3c). Esta desnormalização permite aumentar o desempenho de leitura dos dados, agrupando dados que são frequentemente usados em conjunto (Sullivan, 2015). Assim, apresentam um reduzido *overhead* I/O e a memória é utilizada de forma mais eficiente (Chandra, 2015).

Estas bases de dados apresentam um modelo de dados mais expressivo do que as bases de dados de documentos e *key value*. No entanto, são também orientadas à agregação, pelo que a realização de interrogações que requerem junções exige um processamento por parte de infraestruturas externas (Robinson et al., 2015).

As bases de dados de família de colunas são apropriadas para o *deployment* de bases de dados em grande escala, onde são indispensáveis elevados níveis de desempenho de escritas e elevado número de servidores ou disponibilidade de múltiplos centros de dados (Sullivan, 2015).

Figura 3: Exemplo prático aplicado a bases de dados de documentos (a), *key-value* (b) e família de colunas (c)

2.3 Grafos

O clássico "problema das sete pontes de *Königsberg*" pretendia determinar, dada qualquer configuração de um rio e dos ramos em que este se pode dividir, bem como qualquer número de pontes, a possibilidade de atravessar cada ponte exatamente uma vez. Em 1736, Leonhard Euler resolveu este conceito matemático através da representação das áreas terrestres como vértices e as pontes como arestas, dando origem à teoria dos grafos (Euler, 1953).

Desde então, define-se grafo como o conjunto de entidades, representadas por nodos ou vértices, e as relações que as conectam (ou arestas).

Existem duas grandes propriedades nas bases de dados de grafos. O *mecanismo de processamento de grafos* e o *armazenamento de grafos*.

Processamento de grafos

O modo mais eficiente de processamento de dados em grafos é realizado através do *index-free adjacency* (processamento nativo de grafos), em que os nodos apontam fisicamente para os seus nodos vizinhos. Com esta propriedade, o tempo de execução de cada interrogação é proporcional à porção de grafo analisada, não aumentando com o tamanho global dos dados. O modo de armazenamento das relações nas bases de dados de grafos facilita a travessia em qualquer direção com processamento nativo de grafos, maximizando a eficiência das travessias (Sasaki et al., 2018).

O processamento não nativo de grafos recorre, por sua vez, a índices globais para estabelecer relações entre os nodos, resultando em problemas de desempenho e escalabilidade.

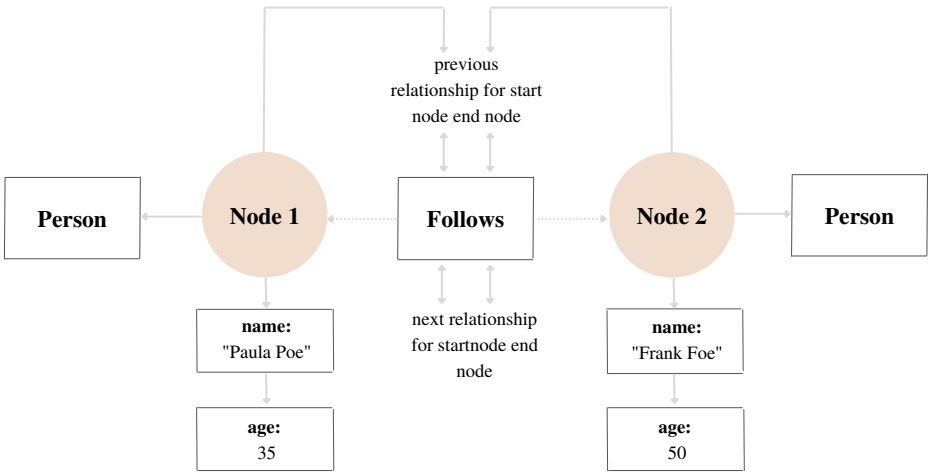
Armazenamento de grafos

No que diz respeito ao armazenamento de dados, algumas bases de dados utilizam um armazenamento nativo, desenvolvido para armazenar e gerir grafos de forma eficiente para conjuntos de dados altamente interligados. Por outro lado, o armazenamento não nativo de grafos recorre a bases de dados relacionais, a bases de dados orientadas aos objetos, entre outras, para armazenar os dados, podendo criar uma elevada latência, o que é especialmente notório com o crescimento do volume de dados e da complexidade das interrogações.

Em bases de dados com armazenamento nativo, os dados são armazenados em ficheiros, cada um dos quais contém dados para uma parte específica do grafo, como nodos, relações, etiquetas e propriedades. Esta divisão de armazenamento possibilita travessias no grafo com alto desempenho. Nestas bases de dados, o registo de um nodo tem como principal propósito apontar para listas de relações, etiquetas e propriedades, tornando-o leve (Neo4j, 2016).

Cada registo de um nodo contém um apontador para a primeira propriedade e primeira relação numa cadeia de relações desse nodo (Figura 4). A leitura das propriedades de um nodo é realizada através da travessia de uma estrutura em lista ligada, com início no apontador para a primeira propriedade. Da mesma forma se percorre a lista duplamente ligada de relações para encontrar uma relação para um nodo. Assim que identificado o registo para a relação pretendida, as suas propriedades podem ser lidas através da mesma estrutura de lista ligada. Podem também ser analisados os nodos que se encontram conectados pela relação, através dos seus identificadores. Estes identificadores (posição do nodo), multiplicados pelo tamanho do registo do nodo, permitem obter a posição de cada nodo no ficheiro de armazenamento de nodos (Robinson et al., 2015).

Figura 4: Armazenamento físico de um grafo [Adaptado de Robinson et al. (2015)]



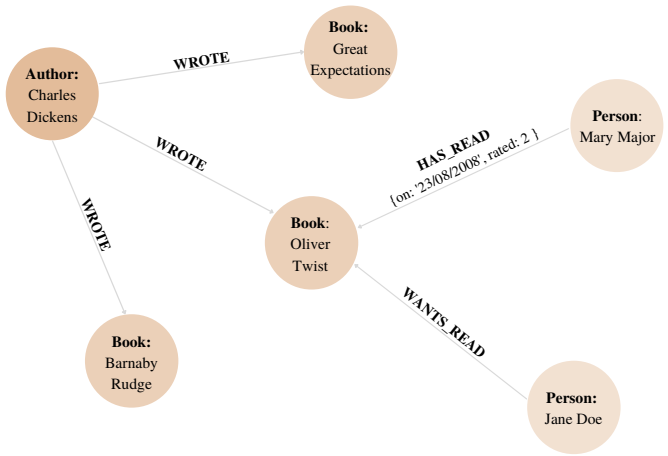
2.3.1 Modelação de dados

A modelação de dados é um meio de organização e definição dos dados e das relações existentes entre estes, atribuindo-lhes uma estrutura. É, por isso, a tradução da visão conceptual dos dados para um modelo lógico.

Grafos de Propriedade

No modelo de grafos de propriedade (*property graphs*), os dados encontram-se organizados em nodos, relações e propriedades (pares chave-valor armazenados nos anteriores). Nestes grafos, cada nodo é nomeado e pode conter várias etiquetas (*labels*), e as relações são direcionadas, com apenas um nodo inicial e um nodo final (Robinson et al., 2015). Uma etiqueta é uma construção de grafos nomeados utilizada para agrupar nodos, pelo que todos os nodos com a mesma etiqueta pertencem a um só conjunto. Deste modo, parte das interrogações à base de dados podem ser realizadas sobre esses conjuntos de dados, em vez de ser utilizado o grafo na sua totalidade, resultando em interrogações mais simples e eficientes. Estes grafos permitem visualizar o modo como os dados estão modelados em diferentes bases de dados e a forma como os meta-dados estão relacionados (Figura 5). Para além disto, fornecem grande flexibilidade e expressividade, permitindo o armazenamento adicional de atributos arbitrários (Pokorný, 2015).

Figura 5: Exemplo de um grafo de propriedades

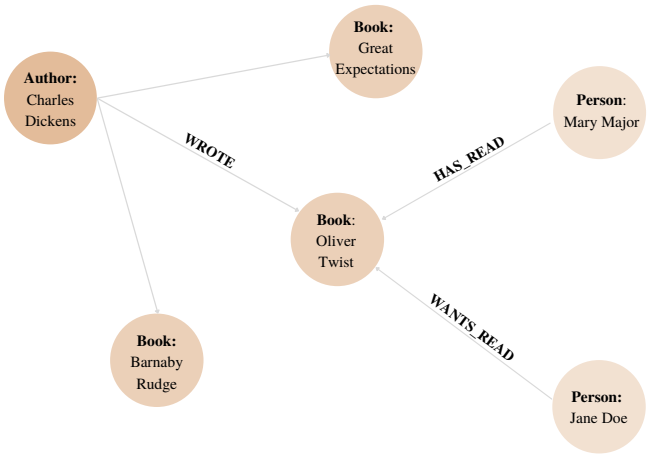


Hipergrafos

Num contexto generalizado encontram-se os hipergrafos, um modelo de grafos onde uma relação (hiperaresta) pode estar conectada a um número variado de nodos. Este modelo de dados é especialmente benéfico em relações muitos-para-muitos. Em teoria, os hipergrafos deveriam produzir modelos de dados precisos e ricos em informação. No entanto, existe grande perda de detalhes durante a modelação (Figura 6).

Apesar dos hipergrafos reduzirem o número de relações existentes no grafo, os grafos de propriedade oferecem

Figura 6: Exemplo de um hipergrafo



uma técnica de modelação de dados mais familiar e explícita, reduzindo problemas subjacentes. Além disto, os hipergrafos não permitem ajustar o modelo com propriedades para as diferentes relações (Sasaki et al., 2018).

RDF

O *Resourse Description Framework* (RDF) é um modelo de dados *World Wide Web Consortium* (W3C) para publicação e troca de dados na *web*. O RDF oferece um método de especificar grafos etiquetados direcionados, em que cada relação é representada por um triplo.

Um triplo *RDF* é uma declaração no formato $(SUJEITO) \rightarrow [PREDICADO] \rightarrow (OBJETO)$ e permite, através de qualquer sujeito, conectá-lo a qualquer objeto, recorrendo a um predicado (verbo) para representar o tipo de relação entre estes. Os recursos de informação são comumente identificados por literais ou por *Uniform Resource Identifiers* (URI) (Barrasa, 2016). A vantagem da sua utilização está na forma globalmente inequívoca de referenciar recursos, não requerendo uma autoridade centralizada para os fornecer. Uma coleção de triplos RDF pode ser representada visualmente com os sujeitos e objetos como nodos e os predicados como arestas (Renzo et al., 2019). A linguagem padrão adotada é o *SPARQL*, que retorna informação de um grafo com base em reconhecimento de padrões. O predicado de uma declaração é uma referência URI que representa uma propriedade, cujo valor corresponde ao objeto da declaração. Este valor pode ser um recurso ou um valor literal (*string* de um determinado tipo de dados que só pode ocorrer como objeto de uma declaração). A utilização de URI para a definição de nodos e relações não permite a criação de várias instâncias da mesma relação. Por este motivo, não é possível qualificar estas instâncias ou conferir-lhes atributos. Por outro lado, os grafos de propriedades não são capazes de lidar com propriedades multi-valor. Existem, no entanto, soluções aplicáveis em ambas as situações (Donkers and Yang, 2020), como a utilização de *arrays* para as propriedades multi-valor.

Um *triplestore* RDF é uma base de dados de grafos que armazena factos semânticos como uma rede de objetos com relações materializadas entre eles. É uma base de dados capaz de lidar com interrogações semânticas e de

recorrer à inferência para reconhecer novas informações a partir das relações existentes.

Em oposição a outros tipos de bases de dados de grafos, os mecanismos dos *triplestores* suportam modelos de esquemas opcionais (ontologias). As ontologias permitem uma descrição formal dos dados e especificam classes de objeto e propriedades de relações, bem como a sua ordem hierárquica (Ontotext, 2021a). No entanto, a sua utilização exige um nível de acordo entre as partes interessadas e limita a liberdade de representação de entidades.

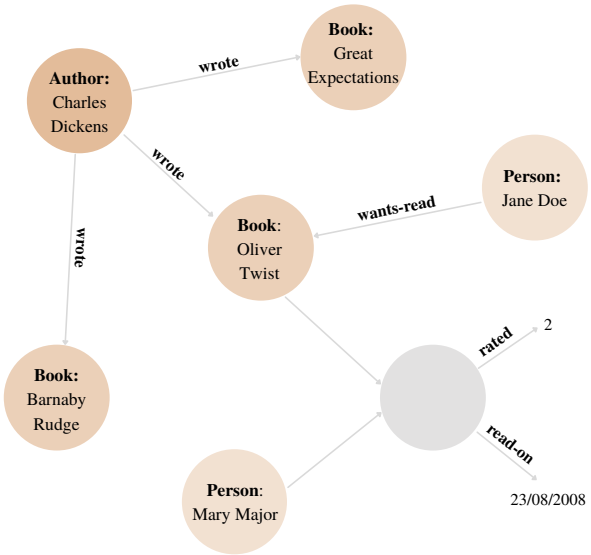
Devido à inexistência de estrutura interna, as propriedades dos nodos em RDF só podem ser descritas adicionando novos nodos ou literais, resultando numa decomposição atómica das entidades (Figura 7). Em contrapartida, os grafos de propriedade apresentam uma estrutura mais compacta, permitindo o armazenamento das propriedades dentro dos nodos ou arestas.

Os *triplestores* são fortemente baseados em índices, não sendo adequados para interrogações que requerem a análise de travessias no grafo. Assim sendo, não devem ser aplicados em casos de uso operacionais e transacionais (Barrasa, 2016).

No que diz respeito à segurança, e uma vez que o RDF logra de URI HTTP, o proprietário do domínio pode implementar restrições de segurança para os URI. Outras bases de dados, como o *Neo4j*, possuem um sistema baseado em funções para utilizadores, permitindo que os administradores determinem os nodos pelos quais um cliente pode passar. Adicionalmente, oferecem opções de *whitelisting* e restrições para as API externas.

Donkers and Yang (2020) concluíram que, quantitativamente, os grafos de propriedade nativos superam o RDF na totalidade das medidas de desempenho, sobretudo nas operações em tempo real. Por outro lado, o RDF pode ter mais interesse em ambientes de múltiplas partes interessadas, uma vez que permite a partilha do conhecimento do domínio entre estas e a realização de inferências com base neste conhecimento.

Figura 7: Exemplo de um grafo RDF



RDF*

O RDF* é um modelo em estudo que pretende combinar as vantagens dos modelos RDF e grafos de propriedade, atribuindo ao RDF a possibilidade de adicionar atributos a nodos e relações (Donkers and Yang, 2020).

Para responder a interrogações semânticas, uma base de dados de grafos utiliza estruturas de dados altamente interligados com nodos, relações e propriedades para representar e armazenar dados (Webber and Bruggen, 2020). Nestas bases de dados, as relações entre entidades são tão importantes quanto as entidades em si (Bechberger and Perryman, 2020). Por este motivo, são habitualmente utilizadas ligações de memória diretas entre nodos adjacentes, não requerendo junções ou chaves estrangeiras (Obispo, 2012). Desta forma, torna-se possível a representação da relação entre objetos e o reconhecimento de padrões baseados nestas relações (Bechberger and Perryman, 2020).

Construídas para a utilização com sistemas OLTP transacionais, as bases de dados de grafos recorrem a métodos CRUD e são otimizadas para integridade e desempenho transacional (Neo4j, 2016), devido às seguintes características:

1. **Declaração de restrições:** As bases de dados de grafos apresentam a propriedade *schema-less*, possuindo, por este motivo uma fluidez de esquema que garante facilidade de acrescentar ou remover informação, conferindo diferentes graus de liberdade a diferentes partes do grafo (Webber and Bruggen, 2020).

2. **Naturalmente aditivas:** Estas bases de dados permitem adicionar relações, nodos, subgrafos, entre outros à estrutura existente sem interferir com a base de dados atualmente implementada (Robinson et al., 2015).

3. **Index-free adjacency:** Cada nodo tem armazenadas referências diretas para os seus nodos adjacentes, funcionando como um índice dos seus nodos vizinhos (Pokorný, 2015). Esta propriedade permite eliminar junções complexas, garantindo um alto desempenho independente do tamanho do conjunto de dados.

4. **Flexibilidade**: Permite reduzir os riscos e *overhead* de manutenção da base de dados, uma vez que não existe a necessidade de modelar o domínio de forma exaustiva desde o primeiro momento.

5. **Deteção de dependências**: Facilitam a visualização de dependências, permitindo conhecer de que forma dois objetos estão interligados (Bechberger and Perryman, 2020).

Apesar das suas vantagens, as bases de dados de grafos apresentam também limitações consideráveis. A grande interligação entre os dados armazenados dificulta a partição e distribuição dos grafos por diferentes máquinas, crucial para a escala destas bases de dados. Uma vez que estas bases de dados permitem diferentes graus de liberdade para diferentes partes do grafo, pode ser facilmente introduzida inconsistência de dados.

2.3.2 Casos-tipo

A resolução do "problema das sete pontes de *Königsberg*" é considerado o primeiro caso de uso dos grafos. As aplicações geoespaciais das bases de dados de grafos partiram deste conceito e abrangem o cálculo de rotas entre diferentes localizações e operações espaciais, como encontrar todos os pontos de interesse de uma determinada área, encontrar o centro de uma região e calcular a interseção entre duas ou mais regiões.

Nas redes sociais, a utilização de grafos permite identificar as relações diretas e indiretas entre pessoas, grupos e as suas interações, permitindo encontrar com facilidade outros utilizadores e interesses.

O armazenamento e interrogação de dados em tecnologias de grafos permite obter resultados em tempo real para sistemas de recomendação, apresentando uma grande vantagem para empresas no retalho, logística, recrutamento, entre outras. A facilidade de reconhecimento de relações complexas entre dados com desempenho de interrogações em tempo real constitui uma grande vantagem contra a lavagem de dinheiro e apropriação indébita. Adicionalmente, estas bases de dados oferecem métodos de identificação de fraude em tempo real com um elevado nível de exatidão, através de técnicas de análise de dados.

A capacidade de armazenar estruturas conectadas complexas e densas permite que os grafos sejam utilizados para soluções de autorização e controlo de acesso, onde informação sobre partes e recursos são armazenados, bem como as regras de acesso para esses recursos. Estas soluções são aplicáveis sobretudo em áreas de gestão de conteúdo, serviços de autorização, preferências de redes sociais

O *Master Data Management* (MDM) é uma técnica de identificação, limpeza, armazenamento e administração de dados. Tem como principais objetivos a gestão das alterações realizadas na estrutura organizacional, incorporar novas fontes de dados, suplementar dados existentes com fontes externas e versionamento de dados (Webber, 2021). Num contexto de RH, o MDM engloba informação relativa a clientes e funcionários de uma empresa e respetivas posições, contratos, tarefas, unidades organizacionais, entre outras.

2.3.3 Linguagens Declarativas

Em bases de dados de grafos, é imprescindível a utilização de uma linguagem cujo desempenho consiga acompanhar o elevado crescimento da relação entre os dados. Em interrogações onde existem muitos dados interligados, a extensão das interrogações SQL tem um grande impacto no seu desempenho e facilita o aparecimento de erros humanos. Em contrapartida, a pequena extensão das interrogações destas linguagens facilita a compreensão e manutenção das mesmas.

Cypher

O *Cypher* é uma linguagem construída com o intuito de possibilitar uma fácil aprendizagem e utilização para consulta de dados em grafos, incorporando o poder e a funcionalidade de outras linguagens de acesso a dados.

A sintaxe do *Cypher* fornece uma forma visual e lógica de combinar padrões de nodos e relações no grafo. Esta linguagem permite declarar o que se pretende selecionar, inserir, atualizar ou excluir dos dados sem que seja necessária uma descrição do modo como o realizar. Deste modo, é possível implementar interrogações expressivas e eficientes para lidar com as funcionalidades CRUD (Neo4j, 2021b). No entanto, no modelo de dados do Cypher, não existe a noção de relações não direcionadas.

Gremlin

O *Gremlin* é uma linguagem funcional de travessia de grafos que permite a expressão sucinta de travessias ou consultas complexas no grafo de propriedade. Cada travessia é composta por uma sequência de etapas potencialmente aninhadas. Cada etapa executa uma operação atómica no fluxo de dados. Estas etapas podem ser *map-steps*, transformando os objetos do fluxo; *filter-step*, removendo objetos do fluxo; ou *sideEffect-step*, calculando estatísticas sobre o fluxo.

Uma travessia no *Gremlin* pode ser escrita de modo imperativo (procedimental), declarativo (descritivo), ou de forma híbrida. Uma travessia imperativa informa como proceder em cada etapa da travessia. Por outro lado, uma travessia declarativa permite selecionar um padrão de execução de entre uma coleção de padrões possivelmente aninhados.

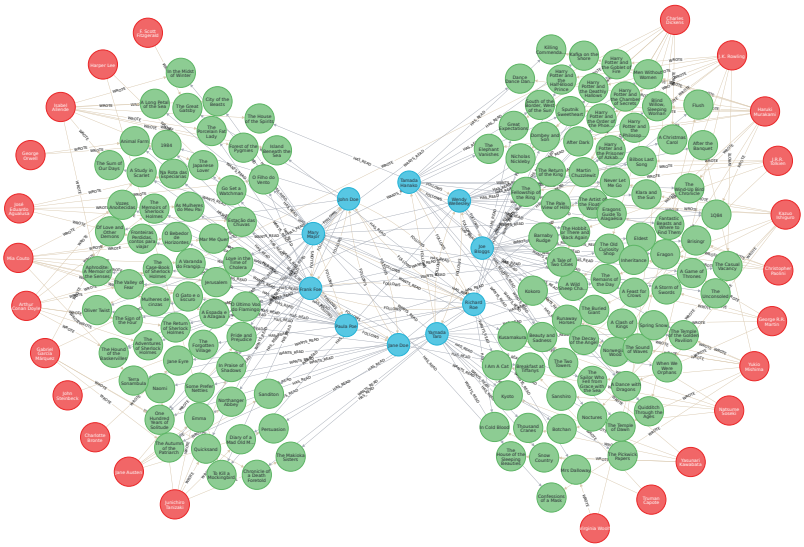
Contrariamente às linguagens de consulta de dados, como o SQL, que foram desenvolvidas intrinsecamente diferentes das linguagens de programação, o *Gremlin* unificou as linguagens de consulta e de programação, permitindo que as travessias possam ser escritas em qualquer linguagem de programação que ofereça suporte à composição e aninhamento de funções. Desta forma, existem várias variantes da linguagem *Gremlin*, como o *Gremlin-Java*, *Gremlin-Groovy*, *Gremlin-Python*, *Gremlin-Scala*, entre outros (TinkerPop, 2021).

2.3.4 Exemplo

A representação dos dados em estruturas de grafos permite observar diretamente a relação que estes estabelecem entre si, facilitando a sua gestão e manipulação.

A Figura 8 representa o exemplo apresentado na secção 2.1.6. Nesta, verifica-se a existência de 170 nodos, dos tipos *AUTHOR*, *BOOK* e *PERSON*, e 311 relações estabelecidas entre estes.

Figura 8: Exemplo prático realizado no Neo4j.



2.4 Sharding

O elevado conjunto de dados e o débito de um sistema de base de dados podem tornar-se um desafio para a capacidade de um único servidor. De forma a garantir maiores níveis de disponibilidade e desempenho, foram desenvolvidos métodos de *sharding* e *partitioning*, que consistem na divisão de um grande conjunto de dados em subconjuntos mais pequenos. A diferença entre *sharding* e *partitioning* reside no modo como os dados são distribuídos, pelo que em *sharding* a distribuição dos dados é realizada sobre diversas máquinas. Por outro lado, o *partitioning* consiste em agrupar subconjuntos de dados numa única instância da base de dados.

No método de *sharding*, as linhas ou colunas de uma tabela são separadas em tabelas mais pequenas (*shards*). Desta forma, é possível armazenar novos fragmentos de dados (*logical shards*) sobre múltiplos nodos (*physical shards*), obtendo escalabilidade horizontal e, consequentemente, maior desempenho.

O *sharding* divide-se em duas vertentes: O *sharding* horizontal é utilizado quando as interrogações retornam maioritariamente subconjuntos de linhas que são habitualmente agrupadas, uma vez que estas interrogações serão limitadas a um subconjunto dos servidores. Por sua vez, o *sharding* vertical é ideal para interrogações que retornem apenas um subconjunto de colunas dos dados, permitindo fragmentar as diferentes colunas em diversos servidores.

Com estas abordagens, caso ocorra uma falha nos dados, apenas se tornam inutilizáveis as porções da aplicação que dependam dos fragmentos de dados em falta. No entanto, é habitual a replicação dos fragmentos para *backup* em nodos adicionais, mitigando o impacto provocado por possíveis falhas (Hazelcast, 2021b).

2.4.1 Arquiteturas de sharding

Existem diversas arquiteturas que podem ser implementadas para a aplicação de *sharding* nos sistemas. Entre estas, as mais habituais são *Key-based*, *Range-based* e *Geo-partitioning*.

Na arquitetura *Key-based*, é gerado um valor de *hash* para cada registo dos dados, que determinará em que fragmento serão armazenados. De forma a garantir a correta inserção dos dados em cada fragmento, são utilizados sempre os valores da mesma coluna para calcular a função *hash* (*shard key*). Nesta estratégia, é improvável que dados com uma chave próxima sejam guardados no mesmo fragmento. Apesar de comum, esta arquitetura pode complicar-se quando se pretende adicionar ou remover dinamicamente servidores à base de dados. Para este fim, os registos da base de dados são novamente mapeados para um novo valor de *hash* e migrados para o servidor indicado. Durante o rebalanceamento dos dados, a aplicação pode apresentar tempo de inatividade, com impossibilidade de escrita de novos dados no servidor (Drake, 2019).

A arquitetura *Range-based* consiste em dividir os dados com base em intervalos de valores. Contrariamente à arquitetura anterior, as *shard keys* com valores próximos serão geralmente armazenados no mesmo fragmento.

Na arquitetura *Geo-partitioning*, os dados são divididos de acordo com uma coluna determinada pelo utilizador, pelo que os fragmentos serão mapeados para regiões específicas. Por sua vez, e para cada região, os dados serão fragmentados utilizando *hash* ou*range sharding* (Choudhury, 2019).

2.4.2 Sharding em bases de dados relacionais

As bases de dados monolíticas (como o *Oracle*, *MySQL* e *PostgreSQL*, encontrando-se este último em fase de desenvolvimento) e algumas bases de dados SQL distribuídas (por exemplo, a Amazon Aurora) não oferecem suporte para *sharding* automático. A necessidade de utilização de *sharding* manual implica um acréscimo considerável na complexidade do desenvolvimento da base de dados. Nesta abordagem, é necessário introduzir lógica adicional de *sharding*, definindo o modo como os dados são distribuídos, o método de *sharding* a adotar, o número de fragmentos a criar e de que forma se irá aceder aos dados (Choudhury, 2019).

Existem, no entanto, bases de dados relacionais que incorporam *sharding* automático. O *YugabyteDB* é um exemplo de uma base de dados SQL distribuída que suporta as arquiteturas *Key-based* e *Range-based*.

2.4.3 Sharding em bases de dados não relacionais

As bases de dados não relacionais foram desenhadas com a escalabilidade como propriedade, pelo que o *sharding* ocorre de forma quase transparente ao utilizador. Por este motivo, o *sharding* é um método quase nativo em bases de dados não relacionais, sendo implementado de forma automática na maioria destas.

Para a aplicação deste método, é frequente a utilização das arquiteturas *range-based* e *key-based*, podendo ser aplicado, por exemplo, com base nas coleções.

2.4.4 Sharding em bases de dados de grafos

Uma possível abordagem para a aplicação de *sharding* em bases de dados de grafos é o *Random Sharding*. Trata-se de uma seleção aleatória de nodos para distribuição por diferentes *shards*. Sendo uma estrutura de dados altamente interligada, esta abordagem implica a perda destas conexões e introduz aleatoriedade na travessia do grafo. Numa interrogação, esta aleatoriedade implica a necessidade de realizar vários *hops* na rede entre servidores para uma travessia, resultando em problemas de latência. O *sharding* por entidades é, por sua vez, uma abordagem que introduz problemas de *fan-out* (número de funções que são chamadas por uma outra função).

De forma a contornar estas adversidades, algumas bases de dados, como o *DGraph*, distribuem os dados por diferentes *shards* com base nas suas relações, isto é, dados com as mesmas relações são colocados num mesmo *shard*. Este método permite diminuir consideravelmente a latência em sistemas do mundo real e permite junções de profundidade arbitrária, sendo flexível e com desempenho previsível (Sakib, 2021).

2.5 Comparação de paradigmas

Num estudo realizado por Čerešňák and Kvet (2019), foram selecionadas algumas bases de dados relacionais para comparar o seu desempenho em relação a bases de dados não relacionais. No primeiro modelo de avaliação de desempenho, foi utilizada uma pequena porção de dados para verificar a velocidade de operação e carregamento para a tabela. Um segundo modelo foi desenvolvido para avaliar a velocidade e de que forma evolui o tamanho de armazenamento. Para este efeito, este modelo foi desenvolvido com um elevado número de registos.

A tabela 1 apresenta os resultados obtidos no segundo modelo desenvolvido neste estudo. Verifica-se que operações de escrita e leitura são mais eficientes em bases de dados não relacionais. Esta diferença de eficiência prende-se no modo como os dados são armazenados e acedidos em paradigmas relacionais e paradigmas não relacionais.

Tabela 1: Desempenho de operações CRUD em base de dados com 100 000 registos (em milissegundos). [Adaptado de Čerešňák and Kvet (2019)]

Type/Operation	Oracle	MySQL	MsSQL	Mongo	Redis	GraphQL	Cassandra
Insert	0.091	0.038	0.093	0.005	0.010	0.008	0.011
Update	0.092	0.068	0.075	0.009	0.013	0.012	0.014
Delete	0.119	0.047	0.171	0.015	0.021	0.018	0.019
Select	0.062	0.067	0.060	0.009	0.015	0.011	0.014

2.6 Síntese

A rápida evolução da tecnologia está a provocar um crescimento exponencial no volume, variedade e velocidade dos dados. De forma a armazenar, gerir e manipular estes dados em grande escala, é crucial avaliar o sistema de bases de dados que melhor se enquadra no problema em causa, considerando e comparando as especificidades inerentes a cada base de dados.

Em esquemas onde se requer uma modificação constante dos dados, a melhor abordagem será recorrer a uma base de dados *schema-less*, como por exemplo as bases de dados não relacionais.

Uma vez que apenas as bases de dados relacionais e de grafos são capazes de modelar relações com algum nível de sofisticação, estas devem ser consideradas quando na presença de um sistema com dados altamente interligados. As relações nas bases de dados relacionais são chaves estrangeiras (apontadores para chaves primárias de outras tabelas). Estes apontadores são de difícil observação e manipulação. Por outro lado, a propriedade de *index-free adjacency* dos grafos confere-lhes grande eficiência a percorrer relações entre diferentes entidades. Assim, para sistemas que necessitem de responder a questões que dependem fortemente de relações entre os dados, as bases de dados em grafos apresentam melhor desempenho comparado com os outros tipos de bases de dados.

2.7 Human Capital Management

O termo *capital humano*, cujo conceito é remetido para 1776 por Adam Smith, refere-se ao estudo dos recursos humanos e é utilizado para designar atributos pessoais benéficos no processo de produção, medido através das capacidades, valores e conjuntos de competências individuais (Diebolt and Haupert, 2014). Na década de 1960, Gary Becker e Theodore Schultz sublinharam que a educação e o treino dos seus colaboradores são investimentos que permitem o crescimento da produtividade e eficiência de uma organização (Schultz, 1961).

O conceito de *gestão de capital humano* (HCM em inglês - *Human Capital Management*) é uma filosofia de negócios que se foca no recrutamento, treino, gestão e manutenção de colaboradores mediante as necessidades e falhas encontradas no sistema. Aplicando estratégias que visam encontrar e desenvolver as potencialidades dos colaboradores que são mais necessárias, consegue-se otimizar os talentos de cada colaborador e garantir um investimento e compromisso dos colaboradores para com a empresa a longo prazo (Oracle, 2022).

Os departamentos de recursos humanos são essenciais para as empresas, uma vez que são o elo de ligação entre o conhecimento individual e os objetivos de cada instituição. No entanto, um aspeto do HCM, que muitas vezes é menosprezado pelas instituições, é a importância e o valor de compreender as relações interpessoais, o seu conhecimento e a sua experiência. Essas relações são vitais para o sucesso organizacional, mas tirar proveito da sua crescente complexidade é um desafio significativo para a maioria das instituições (Burnett and Frelas, 2020).

Neste contexto, cada indivíduo na instituição deixa de ser um nó na hierarquia, mas um nó numa rede,

conectada com muitas outras pessoas, projetos e informação. As bases de dados de grafos simplificam a criação de modelos sobre o modo como as pessoas trabalham nas suas redes, como as pessoas pesquisam por informação e objetos e como as pessoas comunicam e constroem diferentes tipos de relações (entre colegas, entre equipa, com os chefes e subordinados) (Bersin, 2021).

CAPÍTULO 3

Estado da Arte

3.1 Sistemas de Gestão de Bases de Dados de Grafos

Os SGBD de grafos surgiram para colmatar as falhas existentes no mercado, apresentando soluções para o armazenamento e gestão de grande volume de dados. A crescente popularidade das bases de dados de grafos conduziu ao aparecimento de novas implementações para a sua aplicação (Kolomičenko et al., 2013). A escolha da implementação mais adequada deve considerar as suas características, vantagens e limitações.

Em solid IT (2022), é possível verificar a classificação atual dos diferentes SGBD existentes com base nas suas pesquisas no Google, frequência de menção em discussões *online* e artigos, relevância nas redes sociais e ofertas de emprego. A Tabela 2 apresenta os cinco SGBD com melhor classificação em janeiro de 2022.

Tabela 2: Classificação de Sistemas de Gestão de Bases de Dados em janeiro de 2022 [Adaptado de (solid IT, 2022)]

Classificação			**SGBD**	**Modelo de base de dados**	**Pontuação**		
Jan 2022	Dez 2021	Jan 2021			Jan 2022	Dez 2021	Jan 2021
1.	1.	1.	Neo4j	Grafos	58.03	0.00	+4.25
2.	2.	2.	Microsoft Azure Cosmos DB	Multi-modelo	40.04	+0.33	+7.07
3.	3.	7.	Virtuoso	Multi-modelo	5.37	+0.31	+3.23
4.	4.	4.	ArangoDB	Multi-modelo	4.73	-0.02	-0.56
5.	5.	3.	OrientDB	Multi-modelo	4.56	+0.16	-0.77

3.1.1 Neo4j

O Neo4j é uma base de dados *open source* desenvolvida em *Java*. Inicialmente, era utilizado como uma biblioteca de grafos que podia ser incorporado em código para lidar facilmente com estruturas de dados interligados. Encontrava-se sobre sistemas tradicionais de gestão de bases de dados relacionais, como o MySQL, criando uma camada de abstração de grafos.

Posteriormente, a infraestrutura do Neo4j foi otimizada para suportar dados em grafos, interligando-os conforme o seu armazenamento. Esta evolução possibilitou evidentes melhorias na complexidade das interrogações e desempenho da base de dados face a outros SGBD (Bruggen, 2014).

O Neo4j é uma base de dados em grafos nativa e robusta, uma vez que é compatível com as propriedades ACID, e inclui *logs write-ahead* e recuperação após falha súbita, garantindo a persistência dos dados enviados para a base de dados (Neo4j, 2016).

As características referidas são vantajosas para sistemas onde é necessária a obtenção de dados do SGBD num sistema online (OLTP). Nestes sistemas, pretende-se que as interrogações colocadas a um SGBD sejam respondidas num curto intervalo de tempo (na ordem dos milissegundos) entre o pedido *web* e a resposta *web* (Bruggen, 2014).

Esta base de dados apresenta um modelo de grafos de propriedade, pelo que os nodos e as relações podem conter propriedades na forma chave-valor. Adicionalmente, a possibilidade de introduzir etiquetas nos nodos e relações, que permitem a inclusão de meta-dados adicionais, propicia um melhor desempenho das interrogações e a consistência dos dados. Estas etiquetas viabilizam a declaração de uma propriedade indexada automaticamente, uma vez que providenciam meta-informação de indexação. Nesta base de dados, são permitidos índices *single value-based*, *composite value-based*, *range-based* ou geoespaciais.

No *Neo4j*, os nodos e relações possuem identificadores internos únicos que podem ser utilizados para a pesquisa de dados. No entanto, os nodos não podem fazer referência a si mesmos diretamente.

O *Neo4j* pode ser executado no modo incorporado, ou no modo servidor. O modo incorporado age como uma instância que armazena todas as informações no disco, e não como uma base de dados em memória, que trabalha exclusivamente com a memória interna da máquina. Este modo é ideal para dispositivos de *hardware*, aplicações de *desktop* e aplicações integradas em servidores. Em oposição, no Neo4j em modo servidor, em cada servidor existe uma instância incorporada do *Neo4j* (Fernandes and Bernardino, 2018). Este método de implementação da base de dados é atualmente mais utilizado.

Replicação de Dados

A replicação dos dados no *Neo4j* é realizada através de uma estratégia de *cluster* composto por servidores de duas características distintas: *Cores* e *Read Replicas* (réplicas de leitura).

Os servidores *Core* oferecem três recursos principais:

- **Segurança**: Oferecem uma plataforma tolerante a falhas para o processamento de transações, que estará disponível enquanto a maioria destes servidores estiver *online*.
- **Escala**: As réplicas de leitura fornecem uma plataforma altamente escalonável para a realização de interrogações de grafos, permitindo repartir a carga de trabalho por uma topologia amplamente distribuída.
- **Consistência casual**: Uma aplicação cliente tem a garantia de que consegue, pelo menos, ler as suas próprias operações de escrita.

Do ponto de vista operacional, o *cluster* é composto por servidores primários e secundários. Consoante a sua modalidade de instância, os servidores primários podem operar com redundância por adição dos servidores secundários, garantindo consistência casual mas sem segurança dos dados; ou replicar todas as transações através do protocolo RAFT, permitindo operações de escrita e leitura e garantindo a segurança dos dados. O requisito de segurança tem um impacto na latência das operações de escrita, devido à necessidade de reconhecimento pela maioria das instâncias.

Se o *cluster* sofrer falhas de determinado número de servidores primários, deixa de ser possível processar operações de escrita, realizando nesta altura apenas operações de leitura para preservar a segurança.

Os servidores secundários (réplicas de leitura) são construídos de forma assíncrona a partir dos servidores principais através do envio de *logs* de transações. Estes servidores visam expandir a carga de trabalho de leitura, agindo como caches para os dados do grafo. Deste modo, são capazes de executar interrogações e procedimentos arbitrários de leitura (Neo4j, 2021a).

O *Neo4j Fabric* é a solução adotada pelo *Neo4j* para aplicar *sharding* a bases de dados em grafos, repartindo os dados por grafos mais pequenos e armazenando-os em bases de dados distintas. Para grafos altamente conectados, o *sharding* implica a introdução de redundância de dados, de forma a preservar as relações entre as entidades (Neo4j, 2022a).

Controlo Transacional

As interrogações *Cypher* são sempre executadas em transações. As alterações realizadas pela atualização de interrogações são armazenadas em memória pela transação até que esta seja confirmada (*committed*). Assim que confirmada, as alterações são persistidas em disco e ficam visíveis para as restantes transações. Em caso de erro durante a avaliação da interrogação ou confirmação da mesma, a transação é automaticamente revertida (*rolled back*), e nenhuma alteração é persistida no grafo.

As transações no *Neo4j* utilizam um nível de isolamento de *read-committed*, que torna os dados visíveis assim que confirmados, e não visíveis caso as transações não forem confirmadas. Este tipo de isolamento oferece vantagens de desempenho significativas, sendo suficiente para a grande maioria dos casos (Neo4j, 2022c).

Encriptação de Dados

Todos os dados armazenados no *Neo4j AuraDB* são encriptados através da encriptação intra-*cluster* entre os diferentes nodos que integram a base de dados. Além disto, são encriptados *at-rest*, recorrendo a um mecanismo de encriptação do fornecedor da *cloud* subjacente (Neo4j, 2021c).

Licenciamento

O *Neo4j Community* é uma edição *open-source* distribuída pelo *GPL v3*, podendo ser utilizada gratuitamente em *cloud* ou sob um *firewall*. Por sua vez, a edição *Enterprise* do *Neo4j* é desenhada para implementações comerciais onde são cruciais a escalabilidade e disponibilidade das aplicações. Esta edição oferece escalabilidade horizontal com replicação e *sharding* ilimitada e elevada disponibilidade (entre outras).

A implementação do *Neo4j* em *cloud*, através do *Neo4j AuraDB*, pode ser realizada de forma gratuita (para projetos de pequena dimensão, aprendizagem, experimentação e prototipagem); ou através de uma subscrição com um custo de 0.09$/hora (65$/mês) e tem como base a alocação de memória (RAM). Além disto, o *Neo4j AuraDB* é provido de uma edição *Enterprise*, para aplicações em grande escala que requerem suporte e segurança avançados.

3.1.2 Microsoft Azure Cosmos DB

O *Microsoft Azure Cosmos DB* é uma base de dados não relacional utilizada para o desenvolvimento de aplicações modernas.

O *Azure Cosmos DB* oferece diferentes modelos de consistência desde consistência presente em SQL até à eventual consistência do NoSQL. Por este motivo, permite selecionar o melhor modelo para a representação dos dados, suportando dados em documentos, grafos, família de colunas e chave-valor. Além disto, oferece suporte para várias API de acesso a dados, como DocumentDB, Core (SQL), MongoDB, Apache Cassandra, Graph e Table. Para as bases de dados em grafos, o *Azure Cosmos DB* fornece um serviço concebido para qualquer escala através da API do *Gremlin*.

O *Azure Cosmos DB* providencia distribuição global, ou seja, as bases de dados podem ser distribuídas em diferentes regiões do Microsoft Azure, garantindo armazenamento e acessibilidade mais perto dos clientes. Esta funcionalidade possui um elevado grau de automatização e desempenho, pelo que não é necessário lidar com configurações complexas, tempo de inatividade de replicação, latência elevada ou questões de segurança (J.R, 2018). Além disso, todas as propriedades nos documentos do *Azure Cosmos DB* são automaticamente indexadas. Por este motivo, contrariamente ao *Neo4j*, o *Azure Cosmos DB* não permite a definição de índices por parte do utilizador.

Esta base de dados é elástica e com grande capacidade de crescimento, oferecendo tempos de resposta de

milissegundos e escalabilidade automática e instantânea. A API do Gremlin combina as vantagens de desempenho dos algoritmos de base de dados de grafos com infraestruturas altamente escaláveis e geridas para fornecer uma solução única e flexível para os problemas de dados mais comuns associados à falta de flexibilidade e abordagens relacionais (Microsoft, 2022).

Controlo Transacional

O *Microsoft Azure Cosmos DB* implementa o controlo transacional fornecido pela *framework Apache TinkerPop*. O *TinkerPop* permite flexibilidade no modo como as transações são realizadas dependendo da implementação de grafos utilizada, pelo que o seu comportamento e compatibilidade com as propriedades ACID podem variar (Microsoft, 2022). A documentação do *Azure Cosmos DB* com a API do Gremlin para o controlo transacional não é clara quanto à sua compatibilidade com as propriedades ACID.

Encriptação dos dados

O *Azure Cosmos DB* utiliza uma encriptação *AES-256* em todos os documentos e *backups* armazenados em disco, sem afetar o seu desempenho, latência, disponibilidade e funcionalidade. Os dados armazenados em disco são encriptados automaticamente com um sistema de chaves geridas pela Microsoft, podendo ser adicionada uma segunda camada de encriptação com chaves de gestão própria (Microsoft, 2022).

Licenciamento

O *Azure Cosmos DB* não oferece qualquer licença gratuita, pelo que os custos da sua implementação estão dependentes do número e tipo de recursos utilizados, bem como as *Response Units* (RU) utilizadas para cada operação de base de dados.

3.1.3 ArangoDB

O *ArangoDB* é uma base de dados *open-source* de multimodelo nativo com modelos de dados flexíveis para documentos, grafos e *key-value*. Nesta, os dados podem ser acedidos e modificados com a linguagem declarativa *ArangoDB Query Language* (AQL). Assim como o SQL, o AQL é uma LMD, que permite a leitura e modificação dos dados armazenados. No entanto, esta linguagem não tem suporte para operações de definição de dados (LDD) ou de controlo de dados (LCD) (ArangoDB, 2022a).

Esta base de dados multimodelos oferece uma escala de desempenho simplificada, maior flexibilidade e a possibilidade de tolerância a falhas num ambiente distribuído, com uma elevada capacidade de armazenamento em memória e custos reduzidos (Fernandes and Bernardino, 2018). O *ArangoDB* suporta transações ACID, oferecendo a consistência e isolamento das operações transacionais para instâncias singulares e operações atómicas quando

em modo de *cluster*. A arquitetura de *cluster* do *ArangoDB* é um modelo CP (Teorema de CAP) mestre/mestre sem ponto único de falha, pelo que, na presença de uma partição de rede, a base de dados opta por manter a sua consistência interna em detrimento da disponibilidade. Assim, os clientes terão a mesma visão da base de dados, independentemente do nodo ao qual se conectam, e o *cluster* pode continuar a responder a pedidos mesmo em caso de falha de um nodo. O *ArangoDB* permite, ainda, configurar a base de dados numa arquitetura *Leader/Follower* para replicação dos dados, ou dividir grandes conjuntos de dados por diversos servidores (*sharding*) (ArangoDB, 2022a).

Replicação de Dados

O *ArangoDB* oferece dois mecanismos de replicação: síncrona e assíncrona. A replicação síncrona é utilizada entre servidores de bases de dados de um *cluster* do *ArangoDB*. Por sua vez, a replicação assíncrona é utilizada entre o *Leader* e o *Follower* de uma configuração *Leader/Follower* do *ArangoDB*; entre o *Leader* e o *Follower* de uma configuração de *failover* ativo do *ArangoDB*; e entre vários centros de dados do *ArangoDB* (dentro do mesmo centro de dados, a replicação é síncrona).

A replicação síncrona é habitualmente utilizada em dados de natureza crítica que devem estar sempre acessíveis. Nesta replicação, é armazenada uma cópia dos dados de um *shard* num servidor distinto, que é constantemente sincronizado. Para o seu armazenamento, o *cluster* aguarda pela escrita dos dados nas diferentes réplicas da base de dados previamente à operação de escrita no cliente, o que aumenta a latência devido à necessidade de um salto extra na rede para cada escrita. Esta abordagem garante, no entanto, a transição imediata para uma réplica na presença de uma falha no *cluster*.

Na replicação assíncrona, qualquer operação de escrita é registada no *write-ahead log*. Com esta replicação, os nodos *Follower* conectam-se ao nodo *Leader* e aplicam localmente todos os eventos do *log* na mesma ordem. Deste modo, os nodos *Follower* terão o mesmo estado de dados que o *Leader* (ArangoDB, 2022a).

Controlo Transacional

O *ArangoDB* não oferece comandos transacionais explícitos, pelo que uma transação é iniciada com a sua descrição na função $db._executeTransaction$. Uma nova transação é iniciada automaticamente com esta função, onde serão executadas todas as operações de recuperação e/ou modificação de dados. No final desta função, a transação será automaticamente confirmada. Se ocorrer um erro durante a execução da transação, esta é abortada e todas as alterações revertidas (ArangoDB, 2022a).

Encriptação dos dados

O *ArangoDB* usufrui da encriptação para segurança dos dados previamente à sua gravação em disco. O recurso de encriptação é suportado por todos os modos de implementação do *ArangoDB*, mas apenas na versão *Enterprise*.

Os dados são encriptados com *AES-256-CTR*, um algoritmo de encriptação forte e adequado para ambientes com múltiplos processadores, garantindo uma base de dados segura e rápida, mesmo sob carga.

A encriptação do *ArangoDB* tem algumas limitações, uma vez que não permite a encriptação de uma única coleção; não é possível ativar a encriptação em tempo de execução (na existência de dados, é necessário realizar um *backup*, ativar a encriptação, iniciar o servidor numa diretoria de dados vazia e, posteriormente, restaurar o *backup*); e o recurso de encriptação requer o mecanismo de armazenamento *RocksDB* [1] (ArangoDB, 2022a).

Licenciamento

No que diz respeito ao licenciamento, o *ArangoDB* oferece uma versão *Community open-source* e gratuita. No entanto, existe uma edição *Enterprise*, não gratuita, que providencia uma vasta coleção de opções de segurança e escalabilidade. Contrariamente à versão *Community*, esta edição permite o controlo de encriptação e uma encriptação *at rest*.

3.1.4 OrientDB

O *OrientDB* é um SGBD não relacional *open-source* implementado em Java e suporta modelos de dados de documentos, grafos, *key-value* e objetos, cuja manipulação pode ser realizada em Java, SQL ou Gremlin. Para além de uma abordagem *schema-less*, o *OrientDB* suporta a aplicação de um esquema rígido no modelo de dados, com uma solução *schema-full* ou híbrida.

Esta base de dados é construída sobre uma base em documentos com funcionalidades orientadas aos objetos para o armazenamento de vértices físicos utilizando, no entanto, um processamento nativo de grafos através do *index-free adjacency*.

As transações no *OrientDB* são compatíveis com as propriedades ACID, devido à utilização de um modelo *Multiversion Concurrency Control* (MVCC), que permite leituras e escritas concorrentes no mesmo registo, sem necessidade de implementar mecanismos de *locking* na base de dados (Kolomičenko et al., 2013).

Este SGBD suporta uma arquitetura distribuída sobre diversos servidores num *cluster* (*sharding*), recorrendo ao *Hazelcast* [2] para gestão do mesmo, ou uma arquitetura distribuída com replicação.

O *OrientDB* é frequentemente utilizado para deteção de fraude, operações de rede/*IT*, procura em grafos, mecanismos de recomendações, *Master Data Management*, gestão de identidades e análise forense (Fernandes

[1] http://rocksdb.org/

[2] https://hazelcast.com/

and Bernardino, 2018).

Replicação de Dados

O *OrientDB* suporta a replicação *Multi Master* numa arquitetura distribuída. Nesta replicação, todos os nodos do *cluster* são mestre e podem realizar operações de leitura e escrita na base de dados, permitindo um escalonamento horizontal, sem riscos de *bottleneck*. O *sharding* é suportado a nível de classe, utilizando diversos *clusters* por classe, onde cada *cluster* tem a sua própria lista de servidores em que os dados são replicados (OrientDB, 2022).

Controlo Transacional

O *OrientDB* recorre ao modelo MVCC para garantir consistência. A diferença entre a gestão do MVCC em casos transacionais e não transacionais está numa exceção realizar o *rollback* da transação antes de ser levantada pela aplicação. No *OrientDB*, pode ser utilizada uma transação otimista. Esta recorre ao MVCC, permitindo múltiplas leituras e escritas nos mesmos registos. A verificação de integridade é realizada no *commit*. Se um registo for guardado por outra transação nesse ínterim, é aberta uma exceção, pelo que a aplicação poderá optar por repetir a transação ou abortá-la (OrientDB, 2022).

Encriptação dos dados

O *OrientDB* permite a encriptação dos registos da base de dados, evitando o seu acesso por parte de utilizadores não autorizados. A chave de encriptação não é, no entanto, armazenada na base de dados, pelo que deve ser fornecida em tempo de execução.

A encriptação no *OrientDB* opera ao nível da base de dados, pelo que é possível existir múltiplas bases de dados no mesmo servidor (ou *Java Virtual Machine* (JVM)) com diferentes *interfaces* de encriptação. Assim sendo, pode utilizar-se configurações globais para definir regras de encriptação em todas as bases de dados na mesma JVM. A partir da versão 3.x, o *OrientDB* cessou o seu suporte para encriptação de um *cluster* (OrientDB, 2022).

Licenciamento

O *OrientDB* oferece, também, uma edição *community* gratuita, e uma edição *Enterprise*, onde se acrescem maior segurança, configurações de *clustering* distribuído, monitorização em tempo real, *backups* incrementais, entre outras.

3.2 Comparação entre Sistemas de Gestão de Bases de Dados

Um estudo realizado por Fernandes and Bernardino (2018) procurou comparar diferentes bases de dados de grafos, entre estas o *Neo4j*, *ArangoDB* e *OrientDB*, selecionando características importantes a considerar no processo de escolha de uma base de dados e atribuída uma classificação, segundo a escala de *Likert*, a cada. Consideraram as seguintes características:

- Possuir um esquema flexível;
- Utilizar uma linguagem para manipulação dos dados de um grafo;
- Permitir métodos de *sharding*;
- Oferecer funções para planeamento, execução e recuperação de *backups* da base de dados;
- Possibilitar o desenvolvimento de uma base com dados não estruturados através de uma base de dados multimodelo;
- Facultar diferentes abordagens arquiteturais;
- Permitir a escalabilidade da base de dados;
- Possibilitar uma implementação em *cloud*.

Neste estudo, a classificação mais elevada foi atribuída ao *Neo4j*, por apresentar um esquema flexível, permitir a escalabilidade da base de dados e possuir uma linguagem para manipulação de dados (Cypher). Adicionalmente, o *Neo4j* oferece uma ferramenta para planeamento, execução e recuperação de *backups* e dois modos arquiteturais, podendo ser executado no modo incorporado, ou no modo servidor. O *Neo4j* pode, ainda, ser implementado num serviço *cloud*, através do *Neo4j AuraDB*. De notar que, posteriormente à realização deste estudo, o *Neo4j* introduziu o *Neo4j Fabric*, possibilitando o armazenamento e consulta de dados num ambiente de múltiplas bases de dados (*sharding*).

Em segundo lugar encontrava-se o *ArangoDB*, uma base de dados multimodelo que oferece um esquema e modelo de dados flexíveis. Similarmente ao *Neo4j*, o *ArangoDB* tem uma linguagem declarativa própria, o AQL, que possibilita a manipulação dos dados presentes na base de dados, não permitindo, no entanto, operações de definição de dados, como a criação ou remoção de bases de dados, coleções ou índices, nem a gestão de controlo de acesso a objetos específicos. Esta base de dados possibilita a utilização de métodos de replicação e de *sharding*, e pode ser implementada em serviços *cloud*.

O *OrientDB* é também uma base de dados multimodelo que oferece um esquema de dados flexível, podendo utilizar uma solução *schema-less*, *schema-full* ou *schema-mixed*. Implementado em Java, a travessia de grafos pode ser realizada através das linguagens SQL e Gremlin. O *OrientDB* pode ser implementado em *cloud*, suportando

métodos de *sharding* e funções para planeamento, execução e recuperação de *backups* da base de dados. Não obstante, utiliza um armazenamento não nativo de dados, que pode provocar um aumento da latência quando deparado com um elevado volume de dados.

O *Microsoft Azure Cosmos DB* não foi avaliado neste estudo, no entanto é uma base de dados multimodelo globalmente distribuída e oferece um esquema de dados flexível com elevada escalabilidade horizontal. Tratando-se de uma base de dados para implementação em *cloud*, oferece métodos de replicação e *sharding* automáticos. Além disto, esta base de dados permite a definição de métodos *Map/Reduce* através da integração com o *Hadoop*. Por outro lado, não permite a definição de índices pelos utilizadores.

No que diz respeito a *benchmarks* para avaliação do desempenho entre diferentes bases de dados de grafos, Jouili and Vansteenberghe (2013) realizaram uma comparação empírica entre as bases de dados Neo4j, Titan, OrientDB e DEX, focada em operações de grafos. Para as interrogações realizadas, o Neo4j é o SGBD que apresentou, globalmente, melhores resultados. Assim, verificam-se os resultados mais satisfatórios para cargas de trabalho *read-only*, como carregamento os dados, procura dos caminhos mais curtos, procura *breadth-first* da vizinhança e obtenção de nodos por propriedade. Por outro lado, é visível um decréscimo acentuado do seu desempenho em operações *read-write*. Ciglan et al. (2012) avaliaram cinco SGBD, entre estes o Neo4j e o OrientDB, recorrendo a um *benchmarking* de operações de travessia em grafos. Neste estudo, o Neo4j apresentou melhores resultados que o OrientDB. Matyjaszczyk et al. (2020) desenvolveram um *benchmark* com cinco casos-tipo reais sobre o qual realizaram uma avaliação de desempenho de cinco SGBD *open-source* (entre as quais o OrientDB e o ArangoDB). Os resultados revelam um melhor desempenho do ArangoDB comparativamente com o OrientDB.

3.3 Linguagens de Interrogação

A evolução e crescente diversidade de SGBD de grafos deu origem a uma panóplia de linguagens de interrogação, algumas das quais padronizadas, ou exclusivas de cada base de dados, que podem ser aplicadas conforme o modelo de dados utilizado. Nesta secção, são apresentadas algumas das linguagens de interrogação mais frequentes no desenvolvimento de bases de dados de grafos.

3.3.1 Cypher

O *Cypher* recorre a uma sintaxe *ASCII-art* para a correspondência de padrões de nodos e relações da forma

$$(NODE\_A) - [: CONNECT\_TO] \rightarrow (NODE\_B)$$

em que *CONNECT_TO* é a relação que interliga ambos os nodos. Esta sintaxe encontra-se presente na cláusula *MATCH*, que introduz novas linhas com ligação às instâncias correspondentes ao padrão no grafo consultado.

Uma interrogação em *Cypher* recebe como entrada um grafo de propriedades e gera uma tabela. O *Cypher* estrutura as interrogações linearmente, permitindo que os utilizadores analisem o processamento da interrogação de forma linear desde o início. Numa interrogação, cada cláusula é uma função com uma tabela de entrada que produz uma tabela com expansão do número de campos ou adição de novos tuplos. Esta ordem linear das cláusulas permite, no entanto, a livre reordenação da execução das cláusulas, desde que a semântica da interrogação se mantenha inalterada. Assim, a projeção é declarada no final da interrogação através do *RETURN*, em oposição ao SQL, cuja projeção é realizada no início de cada interrogação com a instrução *SELECT*.

O *Cypher* tem suporte de uma complexa linguagem de *update* para manipulação do grafo. As cláusulas básicas para a manipulação dos dados de um grafo incluem o *CREATE*, para a criação de novos nodos e relações; *DELETE*, para a remoção de entidades; e *SET*, para a atualização de propriedades. Adicionalmente, o *Cypher* integra a cláusula *MERGE*, que procura estabelecer uma correspondência ao padrão fornecido, dando origem a um novo padrão caso nenhuma correspondência seja encontrada. Uma implementação pode recorrer a primitivas de sincronização, como o *locking*, para garantir que os padrões de *MERGE* são exclusivos na base de dados (Francis et al., 2018).

O Algoritmo 1 representa o esqueleto para uma interrogação básica em *Cypher*, que consiste numa cláusula que é seguida por um padrão, restrição ou expressão.

Algoritmo 1 Esqueleto de uma interrogação base em Cypher

```
1 MATCH pattern
2 [WHERE restriction]
3 RETURN expression | pattern
4 [ORDER BY pattern [ASC/DESC]]
5 [LIMIT value]
```

Neste esqueleto, as cláusulas encontram-se identificadas por palavras em letra maiúscula, e as cláusulas opcionais encontram-se entre []. Um *pattern* corresponde a um conjunto de símbolos utilizados para a representação dos nodos e relações de um grafo. Por sua vez, uma *restriction* é uma restrição aplicada ao sub-grafo identificado pela interrogação (por exemplo, uma consulta de acordo com um conjunto de propriedades). A interrogação pode obter um resultado da forma de variável, correspondente à *expression* (Balaghan, 2019).

O Algoritmo 2 representa uma interrogação colocada sobre o grafo apresentado na Figura 8. Esta interrogação procura *conhecer, para cada utilizador, o número de livros da sua lista de leitura com data de lançamento inferior a 1900 e respetivo autor*.

Algoritmo 2 Interrogação em Cypher com base no exemplo prático

```
1 MATCH (a:Author)-[:WROTE]->(b:Book)
2 MATCH (p:Person)-[:HAS_READ]->(b)
3 WITH a,b,p
4 WHERE b.released < 1900
5 RETURN p.name AS Person, a.name AS Author, count(b) AS Books_Read
6 ORDER BY Books_Read DESC
```

Esta interrogação inicia-se com uma cláusula *MATCH* seguida por um *pattern* $(a) - [: WROTE] \rightarrow (b)$. Neste padrão, a e b representam um nodo e *WROTE* uma relação entre a e b. A cláusula *WITH* permite limitar as variáveis que são enviadas para os passos seguintes da interrogação (Neo4j, 2021b). A utilização da cláusula opcional *WHERE* restringe os dados que serão retornados no final da interrogação, através da restrição $b.released < 1900$, onde *released* representa o ano de lançamento de um livro *b*. Na cláusula de retorno da interrogação, são retornadas as propriedades *name* e a variável *Books_Read*, ordenadas pela variável *Books_Read* de forma decrescente.

3.3.2 Gremlin

O *Gremlin* é uma linguagem de travessia de grafos distribuída e é utilizado para percorrer grafos de grande escala distribuídos por diferentes nodos de um *cluster*, *cloud* ou *grid*. O *Gremlin* não é uma linguagem empregue para expressar grafos e, por este motivo, suporta diferentes tipos de representação de grafos, podendo ser utilizado para consultas de diferentes arquiteturas distribuídas (Kremer-Herman, 2022).

A simplicidade da gramática do *Gremlin* permite que as suas interrogações sejam incorporadas em diferentes linguagens de programação, como Ruby, Python, Groovy, JavaScript e PHP.

Sendo uma linguagem de travessia, o *Gremlin* é composto pelo conjunto de dados sob a forma de um grafo (G), uma travessia (T) e um conjunto de pontos de travessia (*read/write heads* P).

Esta linguagem opera sobre um grafo de propriedades multi-relacional $G = (N, R, \lambda)$, onde N é um conjunto de nodos, $R \subseteq (N \times N)$ é um multi-conjunto de relações binárias direcionais e $\lambda : ((N \cup R) \times \Sigma* \rightarrow ((U \setminus (N \cup R))$ é uma função parcial que mapeia um par elemento/ *string* para um objeto no conjunto universal U, excluindo nodos e arestas como valores de propriedade. Dado λ, cada nodo e relação pode conter um número arbitrário de propriedades, estabelecendo o conjunto U (Rodriguez, 2015).

As interrogações em *Gremlin* são construídas de acordo com uma sintaxe de fluxo de dados funcional, e executadas na consola do *Gremlin*. Estas interrogações atravessam condicionalmente o grafo através da análise dos nodos e relações que o constituem. A sintaxe utilizada pelo *Gremlin* facilita a compreensão do resultado das interrogações, uma vez que cada chamada de função é construída a partir da anterior, como se verifica no *Scala* e *Javascript* (Kremer-Herman, 2022).

Algoritmo 3 Interrogação em Gremlin com base no exemplo prático

```
g.V().hasLabel('Person').as('p')
    .out('HAS_READ').as('b')
    .has('released',lt(1900))
    .in('WROTE').as('c')
    .groupCount()
    .by('name')
```

3.3.3 SPARQL

O SPARQL é uma linguagem e protocolo de interrogação para *Linked Open Data* na *web* ou para *triplestores* RDF, e permite a consulta de informação de uma base de dados ou qualquer fonte de dados que possa ser mapeada para RDF. Trata-se de uma linguagem expressiva que suporta diversas operações da álgebra relacional, como junções, projeções, seleções, uniões, entre outras. O protocolo SPARQL permite transmitir consultas e resultados SPARQL entre um cliente e um mecanismo SPARQL via HTTP (Ontotext, 2021b).

Uma interrogação SPARQL consiste num conjunto de triplos, em que cada elemento pode ser uma variável. Estas interrogações são baseadas no conceito de *pattern matching* de grafos.

O SPARQL abarca quatro cláusulas de retorno padrão (Semantics, 2022):

- *SELECT*: Retorna dados que correspondam a algumas condições, representados numa tabela simples, onde cada resultado correspondente é uma linha e cada coluna é o valor de uma variável específica.
- *CONSTRUCT*: Cláusula utilizada para transformar dados RDF, com base num modelo especificado como parte da própria interrogação.
- *ASK*: Verifica se existe, pelo menos, um resultado para um padrão de consulta, cujo resultado é $true$ ou $false$.
- *DESCRIBE*: Retorna um grafo RDF que descreve um recurso específico.

O Algoritmo 4 representa o esqueleto para uma interrogação básica em SPARQL.

Algoritmo 4 Esqueleto de uma interrogação base em SPARQL

```
PREFIX kw: <URI>
SELECT ?attr
FROM <dataset>
WHERE { pattern }
ORDER BY ?attr
```

Neste esqueleto, as cláusulas encontram-se identificadas por palavras em letra maiúscula. Em *PREFIX*, a palavra-chave (*kw*) descreve uma declaração para abreviar o *URI*, para futura referenciação. O *dataset* permite definir o conjunto de dados RDF sobre o qual é realizada a interrogação. Por sua vez, o *pattern* especifica o padrão de consulta do grafo a ser correspondido.

3.4 Ferramentas

A implementação de uma base de dados num projeto implica o estudo prévio das ferramentas que possui, de forma a averiguar se lhe permitem concretizar o que lhes é suposto.

Uma vez que a prova de conceito será aplicada sobre uma aplicação em *.NET*, é crucial analisar as ferramentas que permitam a conexão entre a base de dados e a aplicação.

3.4.1 Neo4jDotNetDriver

O *driver .NET Neo4j* é oficialmente suportado pelo *Neo4j* e permite a conexão do projeto à base de dados através de um protocolo binário.

A partir da versão 4.0 do *Neo4j*, a configuração de encriptação padrão encontra-se desativada e o *Neo4j* não gera certificados autoassinados. Esta configuração aplica-se a instalações padrão, por meio de imagens *Docker* e *Neo4j Desktop* (Neo4j, 2021c).

3.4.2 Gremlin.Net

O *Gremlin.NET* é um *driver* do *TinkerPop* para realizar a conexão a um servidor com um SGBD de grafos. Este *driver* implementa o *Gremlin* dentro do *.NET*, e a sintaxe *C#* tem as mesmas construções que Java, incluindo a notação de ponto para encadeamento de funções. Além disso, existem algumas construções adicionadas ao *Gremlin.NET* que tornam as travessias mais simples e sucintas (NuGet, 2022).

3.4.3 Arangodb-net-standard

O *ArangoDB* não oferece nenhum *driver* oficial para a conexão e manipulação direta da base de dados através do *.NET*. Existem, no entanto, diversos *drivers* comunitários, como o *arangodb-net-standard* (ArangoDB, 2022b), um driver consistente, abrangente e mínimo para a *API REST* do *ArangoDB*.

Este *driver* é construído através de uma *interface HttpClient* para a realização de pedidos HTTP, bem como Json.NET para a de(serialização) de/para tipos CLI. Além disto, oferece suporte para operações *async/await*; compatibilidade com as convenções de nomenclatura do C#; e uma abordagem consistente para cada *endpoint* da API (Actify-Inc, 2021).

3.4.4 OrientDB-NET.binary

O *OrientDB* permite o acesso à base de dados por uma aplicação C#/.NET através do *driver OrientDB-NET.binary*. Este *driver* fornece suporte ao *driver* por meio de um protocolo binário de rede, possibilitando a gestão dos servidores através de diversas API e *drivers*.

A documentação deste *driver* fornece um manual para a construção de interrogações, bem como o método de aplicação das *interfaces OServer* (para gerir a configuração ao nível do servidor, bem como criar, remover e listar as suas bases de dados), *ODatabase* (para gestão de *clusters*, realizar operações em registos e emitir *scripts*, interrogações e comandos) e *OTransaction* (para operações à base de dados por meio de transações) (OrientDB, 2022).

3.5 Frameworks

Uma *framework* é uma ferramenta de *software* que auxilia a implementação e execução de uma aplicação, facilitando e reduzindo o tempo de desenvolvimento da mesma.

3.5.1 .NET

O *.NET* é uma *open source cross-platform* framework para a construção de diferentes tipos de aplicações. Esta plataforma pode ser utilizada com várias linguagens (C#, F# e Visual Basic), editores e bibliotecas para a construção de aplicações *web*, *mobile*, *desktop*, jogos e IoT.

A velocidade do *.NET* permite melhor desempenho e requer menor poder de computação nas diferentes aplicações (Microsoft, 2021b).

3.6 Síntese

O Neo4j é o SGBD que reúne melhores características num âmbito geral: apresenta um esquema flexível e oferece uma ferramenta para planeamento, execução e recuperação de *backups*; permite a escalabilidade e replicação dos dados; oferece dois modos arquiteturais de desenvolvimento e pode ser implementado num serviço *cloud*; e apresenta compatibilidade com as propriedades ACID. O Neo4j foi inclusivamente considerada a melhor base de dados na classificação de SGBD de janeiro de 2022 (tabela 2). No entanto, permite apenas a representação dos dados através de um grafo de propriedades, pelo que não fornece a melhor flexibilidade. De igual modo, o CosmosDB e o ArangoDB permitem apenas uma modelação de grafos de propriedades, existindo, no entanto, soluções para mapear dados RDF para o ArangoDB. Por outro lado, o OrientDB é a base de dados que apresenta maior flexibilidade, uma vez que os dados podem ser representados através de um grafo de propriedades e o *TinkerPop* fornece suporte para utilizar o OrientDB como um *triplestore* RDF.

Apesar de não ter sido encontrada qualquer bibliografia com o estudo do desempenho do *Azure Cosmos DB* em relação às restantes bases de dados analisadas, é de esperar que este apresente um desempenho elevado, uma vez que possui um elevado grau de automatização, com uma grande capacidade de crescimento e escalabilidade automática e instantânea. Além do *Azure Cosmos DB*, o melhor desempenho é conseguido pelo *Neo4j*. Embora este estudo tenha sido realizado com base em *benchmarks* de longa data, a carência de novos testes de desempenho realizados pressupõe a invariabilidade dos resultados apresentados.

O *Neo4j* padece, no entanto, na encriptação de dados, uma vez que esta é possível apenas em dados armazenados no *Neo4j AuraDB*. Por outro lado, as restantes bases de dados utilizam métodos de encriptação de dados, sendo relevante notar que o *OrientDB* cessou o suporte para encriptação de um *cluster*.

Relativamente ao licenciamento, as bases de dados *Neo4j*, *OrientDB* e *ArangoDB* oferecem uma edição *community* gratuita, bem como uma edição *enterprise*, não gratuita, com um acréscimo de funcionalidades. Em contrapartida, o *Azure Cosmos DB* não oferece licenças gratuitas, estando os custos dependentes dos recursos e RU utilizadas. Desta forma, o *Neo4j* apresenta-se como a base de dados mais indicada para utilizar como prova de conceito.

No que concerne à linguagem de interrogação a utilizar, o *Gremlin* apresenta-se num cariz imperativo com uma sintaxe compacta, representando um entrave na aprendizagem e leitura da linguagem. Por outro lado, a sintaxe inspirada no SQL do *Cypher* confere-lhe uma transição mais simples do paradigma relacional para uma base de dados em grafos. Ademais, em oposição ao *Cypher*, o *Gremlin* não permite a definição de múltiplas etiquetas por nodo, permitindo, no entanto, propriedades multi-valor. O modelo de dados do *Gremlin* e o seu *bag semantics* baseado em *homomorfismo* concede-lhe uma menor capacidade de realizar travessias de grafos de forma eficiente quando em analogia com o *Cypher*, podendo uma interrogação retornar uma travessia interminável (Klampfer, 2020). Assim sendo, a linguagem interrogativa a utilizar será o *Cypher*.

CAPÍTULO 4

Detalhes de Implementação e Resultados

4.1 Levantamento de Requisitos

O levantamento de requisitos é um processo de compreensão e identificação das necessidades que o cliente espera serem satisfeitas pelo sistema que será desenvolvido. Para a implementação de uma solução em grafos, foi conduzido um levantamento de requisitos funcionais e não funcionais.

4.1.1 Requisitos funcionais

A solução implementada deve:

- Permitir uma pesquisa simples sobre os *employees*, idêntica à solução atualmente implementada.
- Permitir o cálculo da segurança numa base de dados relacional.
- Realizar o cálculo da segurança numa base de dados em grafos, partindo da informação fornecida pela base de dados relacional.
- Possibilitar a análise do desempenho em ambas as soluções.
- Facultar a configuração de regras de segurança na solução implementada numa base de dados em grafos.
- Possibilitar a configuração da segurança sobre os utilizadores e as suas propriedades.
- Permitir herdar regras de segurança de perfis de terceiros (*secondary scopes*).

4.1.2 Requisitos não funcionais

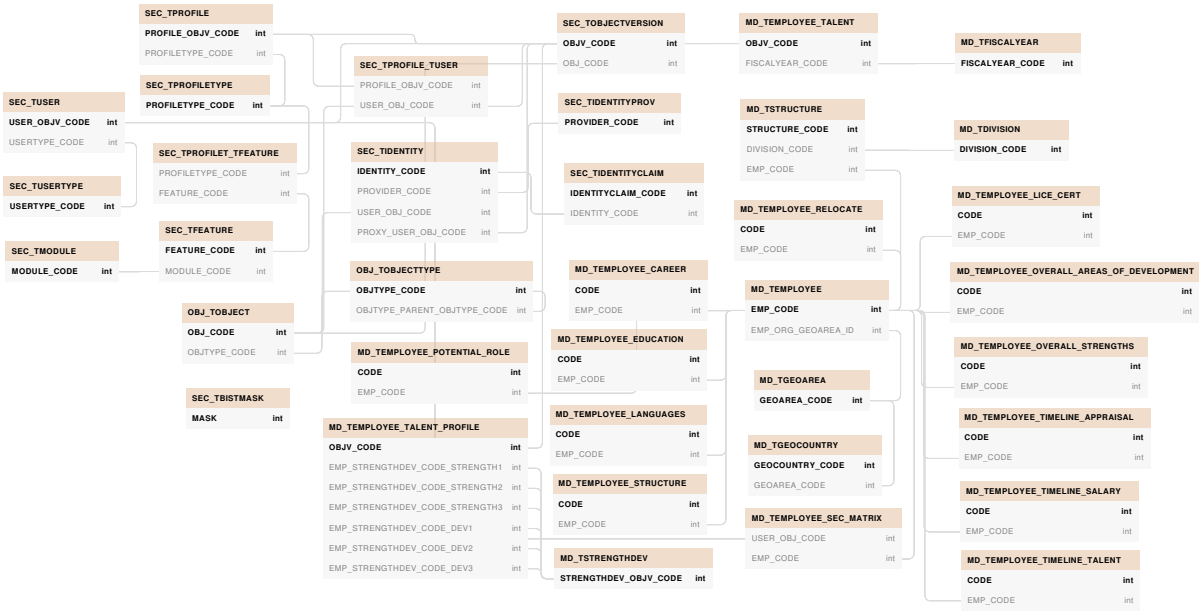
No que respeita aos requisitos não funcionais, a solução implementada deve reger-se pelas melhores práticas e padrões definidos para a sua implementação.

Uma vez se trata de uma prova de conceito tecnológica, não foram identificados requisitos não funcionais adicionais.

4.2 Modelação da base de dados em grafos

A modelação da base de dados em grafos teve por base o modelo lógico da solução implementada num ambiente relacional (Figura 9). No modelo relacional apresentado, foram desconsiderados os atributos não-chave, para simplificação visual.

Figura 9: Modelo lógico da solução implementada num ambiente relacional



A tradução do modelo relacional para um modelo de grafos baseou-se nos pontos de referência que se seguem (Neo4j, 2022b):

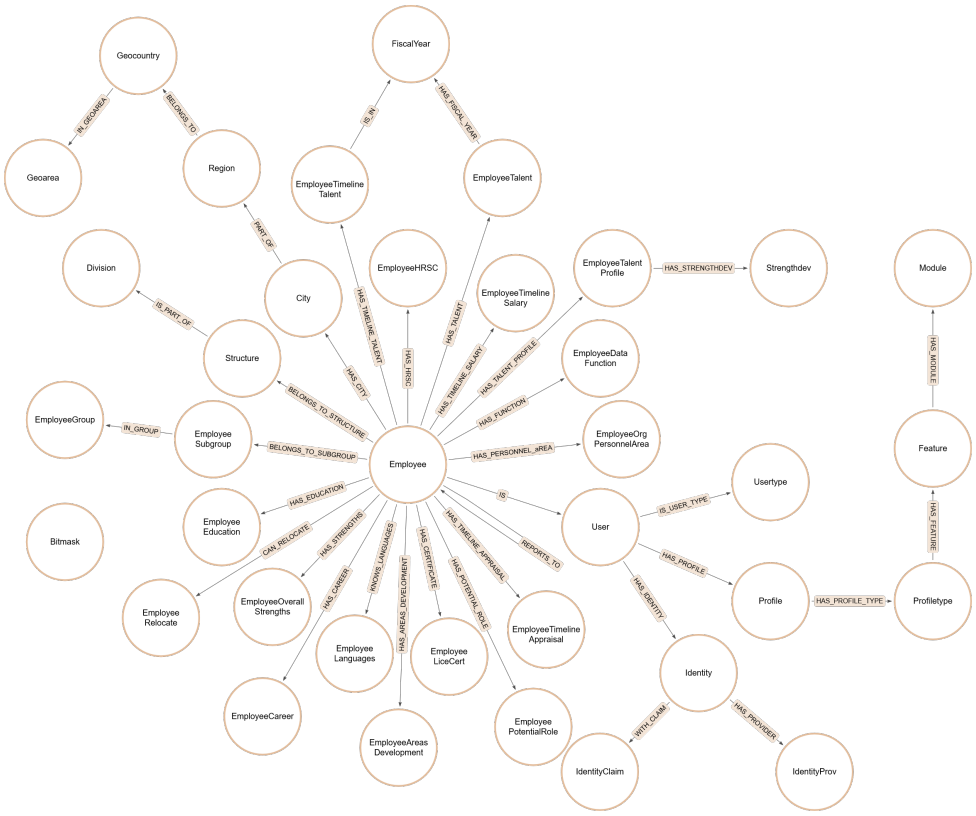
- Cada tabela de entidade no modelo relacional é traduzida para uma etiqueta de nodo no modelo de grafos.
- Cada linha numa tabela de entidade relacional corresponde a um nodo no grafo.
- Os atributos das tabelas relacionais traduzem-se em propriedades de um nodo no grafo.
- As chaves estrangeiras são substituídas por relações entre nodos.
- Todos os dados com valores padrão são removidos.

- Os nomes de colunas indexadas (como email1, email2, email3) podem indicar uma propriedade *array*.
- As tabelas de junção traduzem-se em relações, cujos atributos correspondem a propriedades da relação.

Além da aplicação destes pontos de referência, foi realizada a desnormalização dos dados cujos valores se encontram presentes na filtragem da pesquisa.

O modelo da base de dados em grafos resultante (Figura 10) rege-se pelas regras de nomenclatura do Neo4j.

Figura 10: Modelo em ambiente de grafos



4.3 Povoamento das bases de dados

O povoamento das bases de dados iniciou-se com a execução de *scripts* fornecidos para a geração de dados *dummy* com base nos dados existentes no *data lake* da plataforma em estudo. Posteriormente, foram executadas as *background queues* para a realização do cálculo de segurança.

De modo a garantir a legitimidade da analogia entre as soluções relacional e em grafos, os dados das relações relevantes foram extraídos do SQL Server Management Studio para ficheiros Comma-Separated Values (CSV) e importados para o Neo4j Desktop.

Para a importação dos dados para a base de dados em grafos, foi executado um conjunto de interrogações que permitiram a leitura dos ficheiros CSV e a extração da sua informação para nodos e relações.

O Algoritmo 5 exemplifica a criação de novos nodos a partir de ficheiros CSV. A cláusula *LOAD CSV* permite a leitura, por linha, de ficheiros CSV, locais ou remotos. A inclusão da opção *WITH HEADERS* permite a associação de cada coluna à respetiva nominação na primeira linha.

A criação dos diferentes nodos é conseguida através da cláusula *MERGE*, que combina as cláusulas *MATCH* e *CREATE*. No entanto, e em oposição ao *CREATE*, a cláusula *MERGE* cria um novo nodo apenas se este não estiver presente na base de dados, reduzindo a possibilidade de duplicação de dados. Dado que todos os dados são lidos em formato *string*, foi necessária a utilização de funções como *toInteger()* e*date()* para a correta conversão dos valores. Foi ocasionalmente utilizada a opção *ON CREATE* da cláusula *MERGE*, que permite a definição de propriedades aquando da criação de um novo nodo.

Algoritmo 5 Interrogação em Cypher para a criação do nodo Geoarea

```
1 LOAD CSV WITH HEADERS
2 FROM "file:///MD_TGEOAREA.csv" AS row
3 MERGE (a:Geoarea {
4     geoarea_code: toInteger(row.GEOAREA_CODE),
5     geoarea_id: row.GEOAREA_ID })
6 ON CREATE
7     SET a.geoarea_name = CASE row.GEOAREA_NAME WHEN "NULL"
8         THEN null ELSE row.GEOAREA_NAME END
```

O Algoritmo 6 representa uma nova relação entre os nodos Geocountry e Geoarea. Para este fim, é lido o ficheiro CSV e, para cada linha, são correspondidos os nodos Geocountry e Geoarea aos *row.GEOCOUNTRY_CODE* e *row.GEOAREA_CODE* do ficheiro CSV, respetivamente. Através da cláusula *MERGE*, é criada uma relação *:IN_GEOAREA* entre os respetivos nodos.

Algoritmo 6 Interrogação em Cypher para a criação da relação IN_GEOAREA

```
1 LOAD CSV WITH HEADERS
2 FROM "file:///MD_TGEOCOUNTRY.csv" AS row
3 MATCH (g: Geocountry {geocountry_code: toInteger(row.GEOCOUNTRY_CODE)})
4 MATCH (ga: Geoarea {geoarea_code: toInteger(row.GEOAREA_CODE)})
5 MERGE (g)-[:IN_GEOAREA]->(ga)
```

4.4 Estruturação da API

De forma a garantir que a solução implementada se rege pelas melhores práticas e padrões definidos, foram estudadas diferentes abordagens para a construção da API. Entre estas, foi estudada a viabilidade da utilização do GraphQL na estruturação da API.

O GraphQL é uma linguagem de interrogação e um ambiente de execução para API, desenvolvido com o intuito de tornar as API mais rápidas, flexíveis e intuitivas. Contrariamente às API *Representational State Transfer* (REST), a descrição intensiva dos dados permite a criação de interrogações que acedem às propriedades dos recursos e estabelecem ligações entre eles. Deste modo, torna-se possível a obtenção de todos os dados necessários através de um único endpoint, não existindo cenários de *overfetching* e *underfetching* (obtenção de mais ou menos dados, respetivamente, do que os necessários) (Foundation, 2022). O GraphQL permite a adição e remoção de parâmetros à API sem impacto nas interrogações previamente existentes, facilitando a evolução da API e propiciando um código mais limpo e de fácil manutenção.

Apesar das suas vantagens, o GraphQL possui uma curva de aprendizagem elevada, apresentando um acréscimo na complexidade do armazenamento em *cache* e na carga de trabalho na consulta de dados para o servidor. Por este motivo, foi avaliada a utilização de uma API REST.

A REST é uma arquitetura de software que impõe condições sobre o funcionamento de uma API. Uma API REST é uma API que permite a interação com serviços web RESTful e está em conformidade com as restrições do estilo arquitetónico REST: Os recursos devem ser identificáveis por meio de um único URL, e através de métodos subjacentes ao protocolo de rede, como DELETE, PUT e GET com HTTP; deve existir um acoplamento solto entre o cliente e o servidor, permitindo o seu desenvolvimento de forma independente; as operações servidor-cliente devem ser *stateless*, todos os recursos devem permitir *caching*, salvo indicações contrárias explícitas. A arquitetura pode ser composta por múltiplas camadas de servidores (S. Gillis, 2020). Estas restrições garantem a escalabilidade eficiente e flexibilidade dos sistemas. Adicionalmente, estas API são independentes da tecnologia utilizada, pelo que a alteração destas tecnologias não tem qualquer impacto na comunicação entre o cliente-servidor ou na estrutura da API (AWS, 2022).

A ASP.NET Web API é uma *framework* para a construção de serviços baseados em HTTP que auxilia na implementação de serviços web RESTful. Na API implementada, a Web API age como infraestrutura base que permite a criação dos *endpoits* REST.

A API desenvolvida rege-se pelo padrão arquitetural Model-View-Controller (MVC), dividindo a aplicação em três componentes: o modelo, a vista e o controlador. Neste padrão, o modelo define esquemas que compõem a estrutura de dados; a vista renderiza o *JSON* proveniente do controlador; o controlador atua como um componente mediador entre o modelo e a vista, processando a lógica de negócio e os pedidos recebidos.

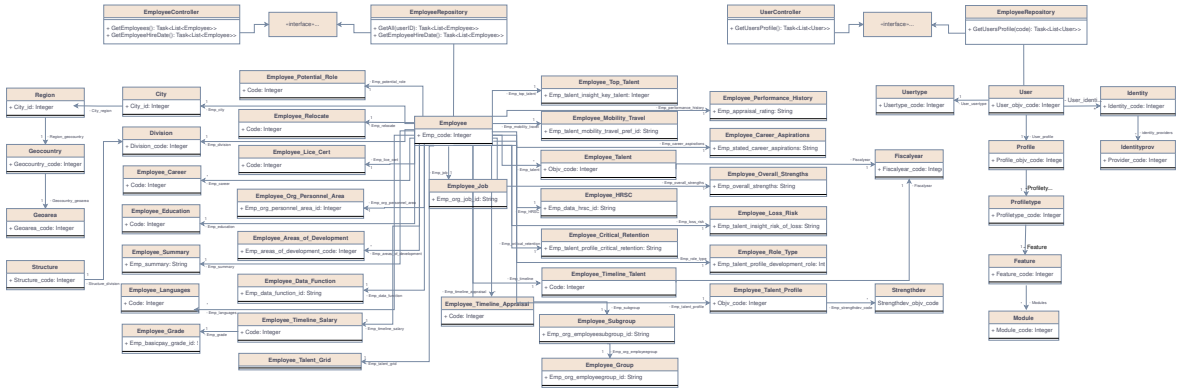
A API recorre ao padrão *Repository*. Um repositório é uma classe ou um componente que encapsula a lógica necessária para aceder às fontes de dados, centralizando a funcionalidade comum de acesso aos dados (de la Torre et al., 2022). O padrão *Repository* permite então criar uma camada de abstração entre as camadas da lógica de negócio e de acesso aos dados (Pragimtech, 2020). A sua utilização proporciona uma melhor manutenção e desacoplamento do sistema utilizado para aceder à base de dados, e simplifica a realização de testes unitários.

A API encontra-se, assim, estruturada em *Models*, *Controllers* e *Repositories*. Uma vez que os modelos têm

como propósito a representação dos dados, foi definida uma classe para cada entidade representada na Figura 10.

O diagrama de classes (Figura 11) descreve a estrutura dos objetos no sistema, traduzindo a sua identidade, os seus atributos e as suas operações, bem como o relacionamento com outros objetos.

Figura 11: Diagrama de classe do sistema.



A implementação dos controladores, *EmployeeController* e *UserController*, permite a interação com o repositório para aceder ao modelo de negócio da aplicação. Com recurso à *Dependency Injection* (DI), o repositório pode ser injetado no construtor do controlador (*Constructor Injection*), coincidindo com o padrão *Repository*.

Algoritmo 7 Definição do controlador EmployeeController

```
1  public class EmployeeController : ControllerBase {
2      public readonly IEmployeeRepository _employeeRepository;
3      private readonly ILogger<EmployeeController> _logger;
4
5      public EmployeeController(IEmployeeRepository employeeRepository,
6      ILogger<EmployeeController> logger){
7          _logger = logger;
8          _employeeRepository = employeeRepository;
9      }
10 }
```

O padrão arquitetural DI é uma técnica utilizada na construção de aplicações, no qual um objeto declara as dependências sobre outro. Esta técnica pretende diminuir as junções entre classes, garantindo uma evolução mais suave do *software*. Uma dependência é representada como um objeto, sendo este usado, por exemplo, como uma funcionalidade ou serviço de uma aplicação. Este padrão permite a redução das dependências e a separação das responsabilidades (criação e utilização de objetos), ampliando a fiabilidade e reutilização de código (Fernandes, 2020). O ASP.NET Core integra suporte para o padrão DI, dado que os controladores, pelo padrão MVC, solicitam dependências explicitamente através de construtores (Namrouti et al., 2022). A injeção do serviço é realizada da seguinte forma:

Algoritmo 8 Injeção do serviço EmployeeRepository

```
1 public void ConfigureServices(IServiceCollection services) {
2     services.AddScoped<IEmployeeRepository, EmployeeRepository>(); }
```

No que respeita aos repositórios, foram criadas duas classes *EmployeeRepository* e *UserRepository* e respetivas interfaces *IEmployeeRepository* e *IUserRepository*, necessárias para a comunicação entre as camadas de negócio e de acesso a dados. Nestes repositórios, são implementadas todas as operações a realizar sobre a base de dados existente.

Algoritmo 9 Definição do repositório EmployeeRepository

```
1  public interface IEmployeeRepository {
2      Task<List<Employee>> GetAll(int userid);
3  }
4
5  public class EmployeeRepository : IEmployeeRepository {
6      private readonly IDriver _driver;
7
8      public EmployeeRepository(IDriver driver) {
9          _driver = driver;
10     }
11 }
```

Para a solução em grafos, optou-se pela utilização de uma estrutura semelhante à da solução previamente existente, tendo sido definido um esqueleto base para o acesso aos dados necessários.

4.5 Importação da segurança para grafos

Numa primeira fase, avaliou-se a viabilidade da utilização da base de dados em grafos, importando o cálculo de segurança da base de dados relacional.

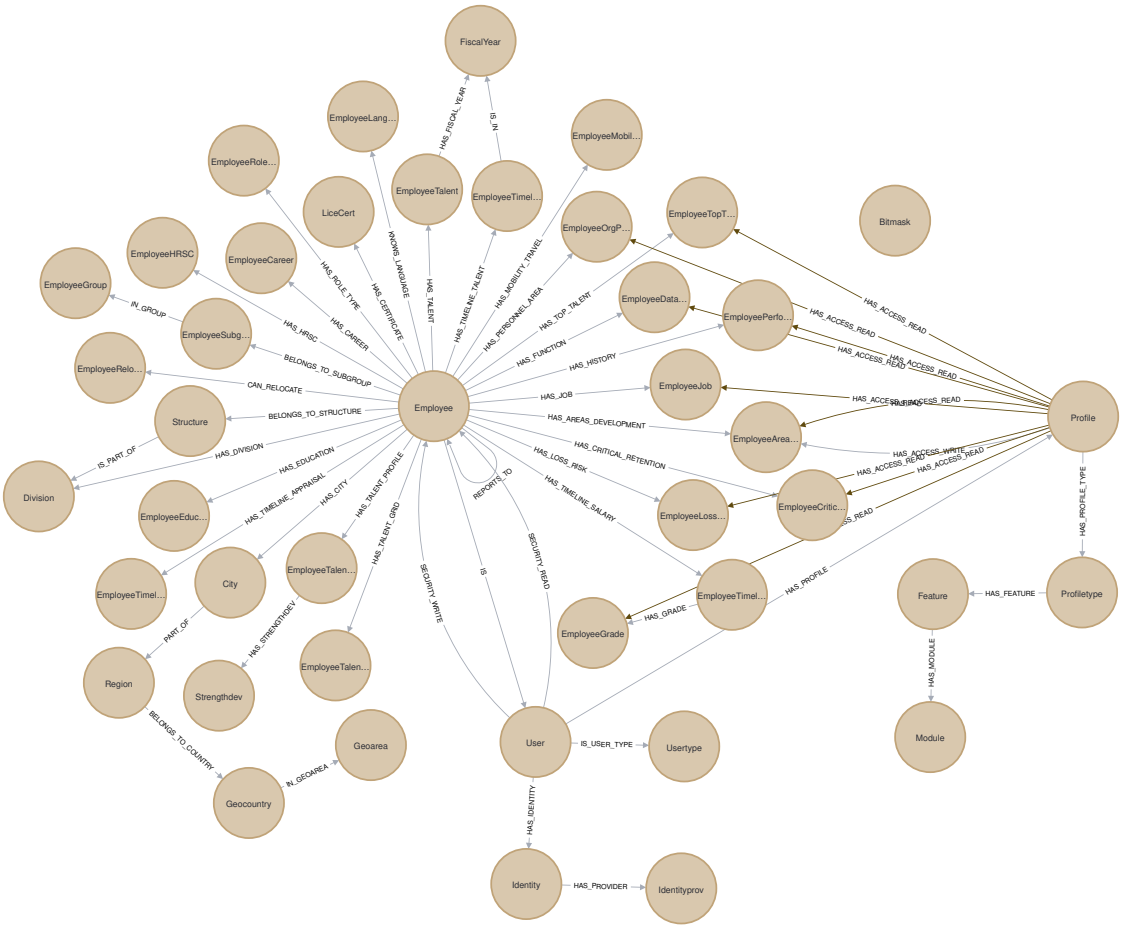
Para este fim, as propriedades sobre as quais um perfil poderia ter acesso foram diferenciadas em entidades e foram definidas duas relações para restringir o acesso aos dados:

- **SECURITY_READ**: Estabelece as permissões de leitura de um *user* sobre um *employee*.
- **SECURITY_WRITE**: Estabelece as permissões de escrita de um *user* sobre um *employee*.

Sobre estas relações foram definidas duas outras, **HAS_ACCESS_READ** e **HAS_ACCESS_WRITE**, que definem as propriedades sobre as quais um perfil tem acesso de leitura e/ou escrita.

A Figura 12 representa a estrutura do modelo em grafos com a importação da segurança. Neste primeiro modelo, não foi discriminada a herança de regras de segurança sobre perfis terceiros (*secondary scopes*).

Figura 12: Estrutura do modelo em grafos na importação da segurança



4.6 Cálculo da segurança em grafos

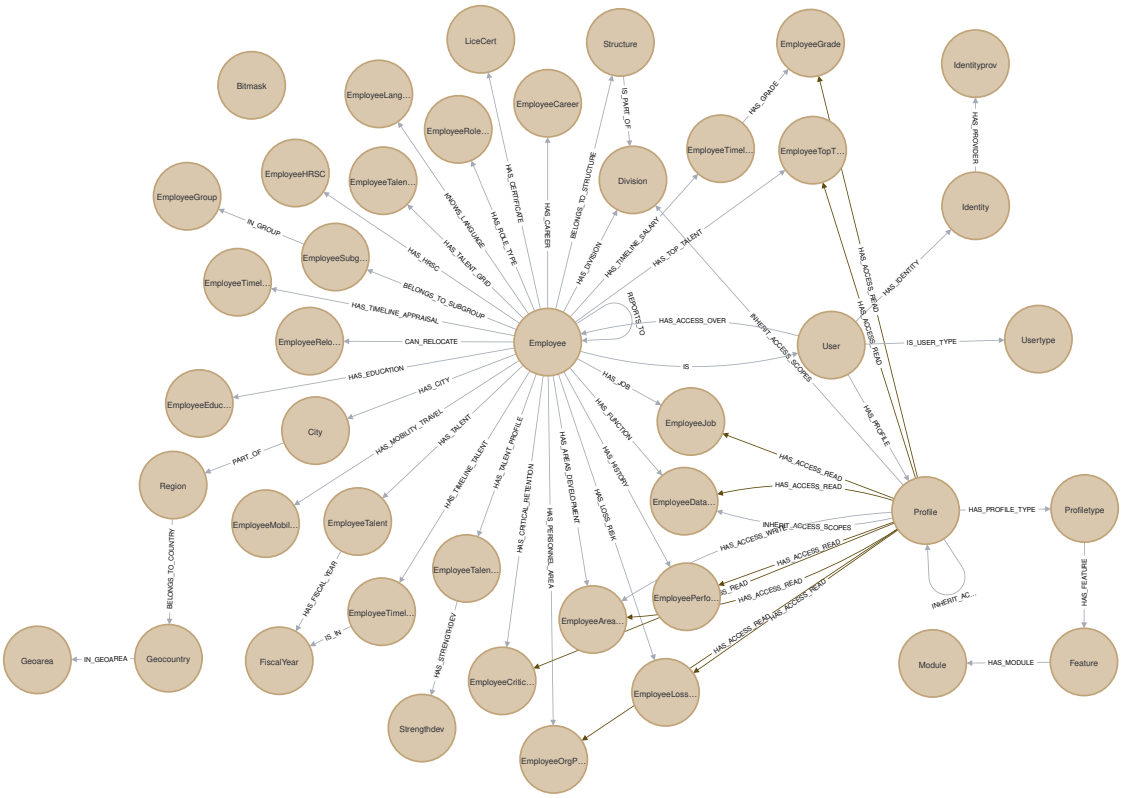
Para a realização do cálculo da segurança na base de dados em grafos, partindo da informação fornecida pela base de dados relacional, foi necessária uma nova modelação dos dados (Figura 13).

Para esta modelação, foram considerados os *secondary scopes* e a configuração da segurança sobre os utilizadores e as suas propriedades.

Neste modelo de dados com controlo de acesso, optou-se pela utilização de relações *fine-grained* em vez de relações *coarse-grained* qualificadas por propriedades, pela sua maior eficiência (Bastani, 2022). Foram criadas quatro regras de acesso:

- **INHERIT_ACCESS**: conecta dois perfis, permitindo que um perfil herde as regras de segurança do segundo perfil.
- **INHERIT_ACCESS_SCOPES**: conecta um perfil a uma divisão ou função, possibilitando o seu acesso aos colaboradores pertencentes às respetivas divisão e/ou função.

Figura 13: Modelo em ambiente de grafos com o cálculo de segurança



- **HAS_ACCESS_READ\WRITE**: define as propriedades sobre as quais um perfil tem acesso de leitura e/ou escrita.

O cálculo da segurança foi traduzido numa relação, **HAS_ACCESS_OVER**, após a verificação dos *primary* e *secondary scopes* (Algoritmo 10). O cálculo da segurança em grafos foi concluído em 14152ms, apresentando uma melhoria face ao cálculo da segurança num ambiente relacional, com uma duração de 17291ms.

Algoritmo 10 Cálculo da segurança em grafos

```
MATCH (u: User)<-[:IS]-(e: Employee)
OPTIONAL MATCH (e)-[rt:REPORTS_TO]->()
WITH e, u, rt.emp_level AS level
OPTIONAL MATCH (e)<-[:REPORTS_TO*]-(c: Employee)
MATCH (u)-[:HAS_PROFILE]->(p)
OPTIONAL MATCH (p)-[:INHERIT_ACCESS]->(pr: Profile)
OPTIONAL MATCH (p)-[:INHERIT_ACCESS_SCOPES]->(ownscope)
OPTIONAL MATCH (pr)-[:INHERIT_ACCESS_SCOPES]->(scopes)
WITH ownscope, scopes, c, e, p, level, u
OPTIONAL MATCH (emps)-[:BELONGS_TO]->(ownscope)
OPTIONAL MATCH (emp)-[:BELONGS_TO]->(scopes)
WITH collect(DISTINCT emps) AS employeeScopes,
collect(DISTINCT c) AS employeeReports,
```

```
14 collect(DISTINCT scopes) AS employeeSScopes, e, p, level, u
15 WITH apoc.coll.union(employeeScopes, employeeReports) as employees, e, p,
16 employeeSScopes, level, u
17 WITH apoc.coll.union(employees, employeeSScopes) AS employees2, e, p, level, u
18 UNWIND employees2 as employees
19 OPTIONAL MATCH path = (employees)-[:REPORTS_TO]->()
20 WITH *, relationships(path) AS rel
21 WHERE ALL(reports in rel WHERE reports.emp_level > level)
22 MERGE (u)-[:HAS_ACCESS_OVER]->(employees)
```

4.7 Avaliação de métricas

Para avaliar o desempenho das soluções implementadas, foram realizadas métricas entre ambas. Nesta avaliação, realizaram-se testes sobre as API de cada base de dados (relacional e grafos), para simular o desempenho das implementações numa situação real.

Para este efeito, recorreu-se ao *Apache JMeter*, uma ferramenta que permite testar o desempenho em recursos estáticos e dinâmicos, simulando várias situações de pressão (sobrecarga num servidor, grupo de servidores, rede ou objeto) para então analisar o seu desempenho geral (JMeter, 2020).

Na realização das métricas, são relevantes as seguintes propriedades:

- *Number of threads*: Número de utilizadores concorrentes num grupo de *threads*.
- *Ramp-up period*: Período de *warm-up* de cache do sistema. O *JMteter* não irá contabilizar este período para o desempenho calculado.
- *Loop count*: Número de iterações que serão executadas por *thread*.

Para estes testes, assumiu-se o valor de um para os *ramp-up period* e *number of threads*, e 200 para o valor de *loop count*, com um número variável de *employees*.

De modo a avaliar analogamente os limites de desempenho das soluções implementadas, foram utilizadas interrogações com uma elevada carga de trabalho sobre as bases de dados, o que significa, no contexto relacional, a travessia do maior número de relações possível e, no contexto de grafos, a travessia pelo maior número de nodos e relações. Após a realização dos primeiros testes, construíram-se *scripts*, em java, para a geração aleatória de dados para o povoamento das bases de dados com o dobro da informação inicial.

Interrogação 1 A primeira interrogação consiste na procura multi-filtro de todos os colaboradores cujo perfil definido tem acesso de leitura, com base nos filtros estipulados:

- **Employment Group and Subgroup**: (Contingent Worker) CW - SubGroup 1;

- **Willingness to Relocate**: Willing to relocate within the same country;
- **Region(s), Country(ies), State(s)**: Europe, Middle East, Africa;
- **Job Grade**: G01.

Tabela 3: Resultados da primeira interrogação: procura multi-filtro de todos os colaboradores cujo perfil definido tem acesso de leitura, com base em filtros estipulados

API	**Tempo de Resposta (ms)**			
	Média	Máx.	Mín.	Mediana
Relacional	804.68	7738	424	529.50
Grafos[a]	41.48	1344	17	32.00
Grafos[b]	212.51	2091	164	229.80

[a] Implementação com importação da segurança para grafos; [b] Implementação com o cálculo da segurança na base de dados em grafos

Tabela 4: Resultados da primeira interrogação utilizando o dobro dos dados iniciais.

API	**Tempo de Resposta (ms)**			
	Média	Máx.	Mín.	Mediana
Relacional	1971.93	34497	148	155.50
Grafos[a]	225.49	5398	142	177.50
Grafos[b]	4249.93	5642	3711	4199.50

[a] Implementação com importação da segurança para grafos; [b] Implementação com o cálculo da segurança na base de dados em grafos

Analisando os resultados (Tabelas 3 e 4), verifica-se uma discrepância significativa entre os tempos (em milissegundos) de resposta médios das soluções relacional e grafos. Esta disparidade deve-se à eficiência das relações entre nodos, num ambiente de grafos, face à crescente carga computacional das junções entre os dados num ambiente relacional.

O intervalo entre os tempos de resposta mínimos e máximos é consequência das definições padrão de *cache* de ambas as bases de dados utilizadas, não tendo sido estes valores propositadamente descartados para permitir uma maior aproximação à realidade.

Entre as duas implementações em grafos, verifica-se um acréscimo significativo do tempo médio de resposta obtido nos diferentes testes realizados. Este resultado deve-se à realização do cálculo de segurança na própria base de dados, em oposição à migração do cálculo de segurança realizado na primeira avaliação de métricas. Adicionalmente, foram considerados os *secondary scopes*, de forma explícita, apenas na segunda implementação, contribuindo para o aumento do tempo médio de resposta. A quebra de desempenho verificada deve-se à travessia de uma parte significativa do grafo, que resulta no aumento proporcional do tempo médio de resposta. De salientar, não obstante, que o desempenho na solução em grafos[a] permaneceu notoriamente superior ao da solução relacional.

Interrogação 2 Nesta interrogação, pretende-se visualizar a lista de utilizadores de um dado tipo de perfil. Para este fim, foi estipulado o filtro **Profiletype**: HRMANAGER. Esta interrogação, apesar de realizada sobre uma amostra pouco significativa, é determinante para corroborar a melhoria de desempenho com a travessia de nodos e relações, comparativamente à travessia entre relações num ambiente relacional.

Tabela 5: Resultados da segunda interrogação: lista de utilizadores de um dado tipo de perfil

API	**Tempo de Resposta (ms)**			
	Média	Máx.	Mín.	Mediana
Relacional	103.08	1270	69	86.00
Grafos[a]	18.58	2460	3	5.00
Grafos[b]	12.72	1604	5	7.00

[a] Implementação com importação da segurança para grafos; [b] Implementação com o cálculo da segurança na base de dados em grafos

Tabela 6: Resultados da segunda interrogação utilizando o dobro dos dados iniciais.

API	**Tempo de Resposta (ms)**			
	Média	Máx.	Mín.	Mediana
Relacional	139.33	1382	117	124.00
Grafos[a]	21.67	1634	6	10.00
Grafos[b]	18.31	1399	6	10.00

[a] Implementação com importação da segurança para grafos; [b] Implementação com o cálculo da segurança na base de dados em grafos

Com os resultados apresentados, observa-se a otimização de desempenho entre as bases de dados em grafos face à base de dados relacional, quando realizadas travessias entre dados altamente interligados.

A realização desta interrogação na API relacional implica uma travessia pelas relações *SEC_TUSER*, *SEC_TPROFILE_TUSER*, *SEC_TPROFILE* e *SEC_TPROFILETYPE*, envolvendo um número considerável de operações de junção, provocando uma quebra do desempenho da interrogação. Por outro lado, numa API em grafos, é necessário percorrer os nodos *User*, *Profile* e *Profiletype*. Uma vez que o Neo4j é uma base de dados com armazenamento nativo, a travessia no grafo é de alto desempenho, pelo que o tempo médio de resposta é consideravelmente menor (Tabelas 5 e 6).

Interrogação 3 Esta interrogação pretende obter a lista de colaboradores cujo perfil definido tem acesso de leitura, com base no filtro **Country Seniority**: Greater than or equal to 20 years. Com esta interrogação, procurou-se ampliar a complexidade lógica ao introduzir o cálculo da diferença temporal.

Tabela 7: Resultados da terceira interrogação: procura de todos os colaboradores cujo perfil definido tem acesso de leitura, com base no filtro estipulado

API	**Tempo de Resposta (ms)**			
	Média	Máx.	Mín.	Mediana
Relacional	318.42	5557	164	279.50
Grafos[a]	62.14	3384	28	42.00
Grafos[b]	198.90	1374	183	188.00

[a] Implementação com importação da segurança para grafos; [b] Implementação com o cálculo da segurança na base de dados em grafos

Tabela 8: Resultados da terceira interrogação utilizando o dobro dos dados iniciais.

API	**Tempo de Resposta (ms)**			
	Média	Máx.	Mín.	Mediana
Relacional	181.09	1266	157	167.50
Grafos[a]	260.17	1387	231	240.00
Grafos[b]	1037	5120	852	989.00

[a] Implementação com importação da segurança para grafos; [b] Implementação com o cálculo da segurança na base de dados em grafos

Os testes realizados sobre os dados iniciais apresentam um tempo de resposta inferior para ambas as implementações com bases de dados em grafos. Verifica-se, no entanto, uma redução significativa do tempo de resposta da base de dados relacional aquando da introdução do dobro dos dados.

No ambiente relacional, esta interrogação é de reduzida complexidade, uma vez que a totalidade dos dados pretendidos se encontra numa única relação, pelo que não sofre quebras de desempenho com a presença de operações de junção. Por outro lado, num ambiente de grafos, esta interrogação é composta por nodos e relações de múltiplas *labels*, com uma elevada porção do grafo a ser percorrida na implementação com o cálculo de segurança na base de dados.

CAPÍTULO 5

Conclusões e trabalho futuro

Na realização da presente dissertação, foram avaliadas propriedades que caracterizam e diferenciam as diversas bases de dados em estudo.

O primeiro fator determinante na seleção de um SGBD está na modelação dos dados que se pretende armazenar. Num contexto de RH, a estruturação hierárquica de cada indivíduo numa instituição, conectado com outros indivíduos, projetos e informação, requer uma abordagem que permita a modelação de relações entre dados altamente interligados, conferida pela utilização de uma base de dados em grafos. Dado que num ambiente de grafos existem apenas dois métodos de representação de dados (nodos e relações), a sua modelação é mais simples comparativamente com um ambiente relacional. De entre as diferentes bases de dados de grafos analisadas, o Neo4j apresenta-se como a mais indicada para utilização como prova de conceito.

No que respeita à construção e validação de interrogações, a sintaxe *ASCII-art* do *Cypher* para a correspondência de padrões de nodos e relações confere-lhe uma maior facilidade na visualização e compreensão dos dados. Além disso, a capacidade de visualizar e interagir com os dados permite uma fácil validação das interrogações realizadas, em oposição ao *SQL Server*, cujo formato tabular dos resultados dificulta a verificação dos dados obtidos. Deste modo, a melhor metodologia a adotar para o desenvolvimento de modelos de bases de dados de grafos em contexto de RH, no caso de estudo, é a utilização do Neo4j em conjunto com o Cypher.

O fator decisivo para a seleção de uma base de dados encontra-se no desempenho obtido nas avaliações de métricas. Foram realizadas três interrogações para este fim. Nas primeira e segunda interrogações, procurou-se atravessar intensivamente a estrutura das bases de dados. Nestas interrogações, verificou-se um custo exponencial com o aumento das consultas e relações na API relacional, existindo uma diferença significativa nos tempos de resposta entre os dois paradigmas. Dada a complexidade da primeira interrogação, a segunda implementação em grafos apresentou uma quebra relevante no seu desempenho. Este resultado indica que, para interrogações cuja

travessia percorra a maior parte do grafo, o SQL terá um desempenho superior ao *Neo4j*. Por outro lado, a terceira interrogação revela que em situações onde não existam junções entre diferentes relações num ambiente relacional, o SQL apresentará um desempenho superior ao *Neo4j*.

As principais limitações foram escassez de tempo e memória para realizar métricas com maior complexidades sobre os diferentes paradigmas. Apesar das limitações, os resultados obtidos demonstram que, para instituições com dados de recursos humanos altamente interligados, a utilização de um ambiente em grafos poderá ser mais vantajosa.

A avaliação de métricas permitiu averiguar que, apesar do cálculo de segurança ser mais rápido num ambiente em grafos, a sua utilização direta tem um impacto substancial na complexidade das interrogações. Por este motivo, deverá ser considerada a importação da segurança para grafos para posterior utilização.

A presente dissertação permitiu aprofundar o conhecimento sobre o funcionamento e métodos de interligação com aplicações web de bases de dados de grafos. Considera-se interessante como próximo passo o desenvolvimento de uma aplicação *web* que permita a verificação da funcionalidade da base de dados, através da consulta dos dados da mesma, e possibilite assim uma comparação visual entre os dois ambientes (relacional e grafos).

Bibliografia

Actify-Inc. arangodb-net-standard: The c#/.net standard api driver for arangodb, 2021. URL https://github.com/Actify-Inc/arangodb-net-standard. Acedido a 27/01/2022.

ArangoDB. Arangodb v3.8.4 documentation, 2022a. URL https://www.arangodb.com/docs/stable/index.html. Acedido a 17/01/2022.

ArangoDB. Arangodb v3.8.5 drivers documentation, 2022b. URL https://www.arangodb.com/docs/stable/drivers/. Acedido a 27/01/2022.

AWS. What is restful api?, 2022. URL https://aws.amazon.com/what-is/restful-api/. Acedido a 28/06/2022.

Phininder Balaghan. *An Exploration of Graph Algorithms and Graph Databases*. PhD thesis, University of Hull, 2019.

Jesús Barrasa. Rdf triple stores vs. labeled property graphs: What's the difference?, 2016. URL https://neo4j.com/blog/rdf-triple-store-vs-labeled-property-graph-difference/. Acedido a 07/02/2022.

Kenny Bastani. Entitlements and access control - graphgists, 2022. URL https://neo4j.com/graphgists/entitlements-and-access-control/. Acedido a 20/06/2022.

Dave Bechberger and Josh Perryman. *Graph Databases in Action*. Manning Publications Co., 2020. ISBN 978-1-617-29637-6.

Kristi Berg, Tom Seymour, and Richa Goel. History of databases. *International Journal of Management & Information Systems*, 07(01):29–30, 2013.

Josh Bersin. Hr technology 2021: The definitive guide, 2021. URL https://joshbersin.com/wp-content/uploads/2021/04/HR_TechMarket_2021_v7.pdf. Acedido a 22/05/2022.

C. Bhanuprakash, Y.S. Nijagunarya, and M.A. Jayaram. A simple approach to sql joins in a relational algebraic notation. *International Journal of Computer Applications*, 104(4):18–26, 2014. 10.5120/18190-9099.

Rik Van Bruggen. *Learning Neo4j*. Packt Publishing, 2014. ISBN 978-1-84951-716-4.

Simon Burnett and Michael Frelas. Finding buried treasure: The role of graph databases in hcm, 2020. URL https://www.tlnt.com/finding-buried-treasure-the-role-of-graph-databases-in-hcm/. Acedido a 22/05/2022.

Deka Ganesh Chandra. Base analysis of nosql database. *Future Generation Computer Systems*, 52:13–21, 11 2015.

Sid Choudhury. How data sharding works in a distributed sql database, 2019. URL https://blog.yugabyte.com/how-data-sharding-works-in-a-distributed-sql-database/. Acedido a 26/11/2021.

Marek Ciglan, Alex Averbuch, and Ladialav Hluchy. Benchmarking traversal operations over graph databases. *2012 IEEE 28th International Conference on Data Engineering Workshops*, 2012. 10.1109/icdew.2012.47.

Thomas Connolly and Carolyn Begg. *Database Systems A Practical Approach to Design, Implementation, and Management*. Pearson Education, 6 edition, 2015. ISBN 978-1-292-06118-4.

Cesar de la Torre, Bill Wagner, and Mike Rousos. *.NET Microservices: Architecture for Containerized .NET Applications*. Microsoft Corporation, 6 edition, 2022. URL https://raw.githubusercontent.com/dotnet-architecture/eBooks/main/current/microservices/NET-Microservices-Architecture-for-Containerized-NET-Applications.pdf. Acedido a 25/06/2022.

Claude Diebolt and Michael Haupert. *Handbook of Cliometrics*. Springer Berlin, Heidelberg, 2014. https://doi.org/10.1007/978-3-642-40458-0.

Amazon DocumentDB. Document database use cases, 2021. URL https://docs.aws.amazon.com/documentdb/latest/developerguide/document-database-use-cases.html. Acedido a 09/12/2021.

Alex Donkers and Dujuan Yang. Linked data for smart homes: Comparing rdf and labeled property graphs. *LDAC*, 06 2020.

Mark Drake. Understanding database sharding, 2019. URL https://www.digitalocean.com/community/tutorials/understanding-database-sharding. Acedido a 26/11/2021.

Leonhard Euler. Leonhard euler and the koenigsberg bridges. *Scientific American*, 189(1):66–70, 1953. 10.1038/scientificamerican0753-66.

Diogo Fernandes and Jorge Bernardino. Graph databases comparison: Allegrograph, arangodb, infinitegraph, neo4j, and orientdb. *In Proceedings of the 7th International Conference on Data Science, Technology and Applications*, pages 373–380, 2018. 10.5220/0006910203730380.

Tiago Fernandes. Dependency injection: definition, principles and uses, 2020. URL https://www.growin.com/blog/what-is-dependency-injection/. Acedido a 28/06/2022.

The GraphQL Foundation. Graphql, 2022. URL https://graphql.org/. Acedido a 27/06/2022.

Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, and Tobias Lindaaker. Cypher: An evolving query language for property graphs. *SIGMOD'18 Proceedings of the 2018 International Conference on Management of Data*, 06 2018. 10.1145/3183713.3190657.

Hector Garcia-Molina, Jeffrey David Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Pearson, 2014.

Feliz Gouveia. *Fundamentos de Bases de Dados*. FCA, 2014. ISBN 978-9-727-22799-0.

Hazelcast. What is a key-value store?, 2021a. URL https://hazelcast.com/glossary/key-value-store/. Acedido a 07/12/2021.

Hazelcast. An overview of sharding & partitioning, 2021b. URL https://hazelcast.com/glossary/sharding/. Acedido a 25/11/2021.

IBM. Relational databases. https://www.ibm.com/cloud/learn/relational-databases, 2019. Acedido a 02/11/2021.

IBM. What is a relational database?, 2021. URL https://www.ibm.com/analytics/relational-database. Acedido a 29/11/2021.

Nishtha Jatana, Sahil Puri, Mehak Ahuja, Ishita Kathuria, and Dishant Gosain. A survey and comparison of relational and non-relational database. *International Journal Of Engineering Research & Technology (IJERT)*, 1(6), 2012.

Apache JMeter. Apache jmeter, 2020. URL https://jmeter.apache.org/. Acedido a 22/05/2022.

Salim Jouili and Valentin Vansteenberghe. An empirical comparison of graph databases. *2013 International Conference on Social Computing*, 2013. 10.1109/socialcom.2013.106.

Guay Paz J.R. *Microsoft Azure Cosmos DB Revealed*. Apress, 2018. ISBN 978-1-4842-3350-4. https://doi.org/10.1007/978-1-4842-3351-1_1.

Martin Klampfer. *Analysis and Comparison of Common Graph Query Languages*. PhD thesis, Technische Universität Wien, 2020.

Vojtěch Kolomičenko, Martin Svoboda, and Irena Holubová. Experimental comparison of graph databases. *IIWAS*, 12 2013. 10.1145/2539150.2539155.

Nathaniel Kremer-Herman. Gremlin graph traversal language, 2022.

Piotr Matyjaszczyk, Przemyslaw Rosowski, and Robert Wrembel. Goodbye: a good graph database benchmark - an industry experience. *T 2020 Joint Conference*, 2020.

Microsoft. Query processing architecture guide, 2021a. URL https://docs.microsoft.com/en-us/sql/relational-databases/query-processing-architecture-guide?view=sql-server-ver15. Acedido a 14/12/2021.

Microsoft. Why choose .net?, 2021b. URL https://dotnet.microsoft.com/en-us/platform/why-choose-dotnet. Acedido a 30/12/2021.

Microsoft. Azure cosmos db documentation, 2022. URL https://docs.microsoft.com/en-us/azure/cosmos-db/. Acedido a 26/01/2022.

Mohamed Mohamed, Obay Altrafi, and Mohammed Ismail. Relational vs. nosql databases: A survey. *International Journal of Computer and Information Technology (IJCIT)*, 03:598–560, 05 2014.

MongoDB. Key-value databases. https://www.mongodb.com/databases/key-value-database, 2021. Acedido a 15/11/2021.

Shadi Namrouti, Rick Anderson, and Steve Smith, 2022. URL https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/dependency-injection?view=aspnetcore-6.0. Acedido a 14/07/2022.

Neo4j. *The Definitive Guide to Graph Databases for the RDBMS Developer*. Neo4j Graph Database Platform, 2016. Ebook.

Neo4j. Casual clustering, 2021a. URL https://neo4j.com/docs/operations-manual/current/clustering/introduction/. Acedido a 6/01/2022.

Neo4j. Cypher query language, 2021b. URL https://neo4j.com/developer/cypher/. Acedido a 29/11/2021.

Neo4j. The neo4j operations manual v4.4, 2021c. URL https://neo4j.com/docs/operations-manual/current/. Acedido a 26/01/2022.

Neo4j. Sharding graph data with neo4j fabric, 2022a. URL https://neo4j.com/developer/neo4j-fabric-sharding/. Acedido a 6/01/2022.

Neo4j. Model: Relational to graph, 2022b. URL https://neo4j.com/developer/relational-to-graph-modeling/. Acedido a 05/04/2022.

Neo4j. Neo4j chyper manual transactions, 2022c. URL https://neo4j.com/docs/cypher-manual/current/introduction/transactions/. Acedido a 20/01/2022.

NuGet. Gremlin.net, 2022. URL https://www.nuget.org/packages/Gremlin.Net/. Acedido a 27/01/2022.

Cal Obispo. The current state of graph databases, 2012.

Ontotext. What is an rdf triplestore?, 2021a. URL https://www.ontotext.com/knowledgehub/fundamentals/what-is-rdf-triplestore/. Acedido a 6/01/2022.

Ontotext. What is sparql?, 2021b. URL https://www.ontotext.com/knowledgehub/fundamentals/what-is-sparql/. Acedido a 26/01/2022.

Oracle. What is human capital management?, 2022. URL https://www.oracle.com/pt/human-capital-management/what-is-hcm/. Acedido a 17/05/2022.

OrientDB. Orientdb documentation, 2022. URL https://orientdb.org/docs/3.0.x/introduction/. Acedido a 17/01/2022.

Jaroslav Pokorný. Graph databases: Their power and limitations. *International Federation for Information Processing (IFIP)*, pages 58–69, 2015.

Pragimtech. Repository pattern in asp.net core rest api, 2020. URL https://www.pragimtech.com/blog/blazor/rest-api-repository-pattern/. Acedido a 25/06/2022.

Angles Renzo, Harsh Thakkar, and Dominik Tomaszuk. Rdf and property graphs interoperability: Status and issues, 2019.

Ian Robinson, Jim Webber, and Emil Eifrem. *Graph Databases*. O'Reilly Media, 2015. ISBN 978-1-491-93200-1.

Marko A. Rodriguez. The gremlin graph traversal machine and language (invited talk). *DBPL*, 10 2015. 10.1145/2815072.2815073.

Alexander S. Gillis. What is rest api (restful api)?, 2020. URL https://www.techtarget.com/searchapparchitecture/definition/RESTful-API. Acedido a 28/06/2022.

Abu Sakib. Database sharding: How to scale a graph database, 2021. URL https://dgraph.io/blog/post/db-sharding/. Acedido a 14/12/2021.

Bryce Merkl Sasaki, Joy Chao, and Rachel Howard. *Graph Databases for Beginners*. Neo4j Graph Database Platform, 2018. Ebook.

Theodore W. Schultz. Investment in human capital. *The American Economic Review*, 51(1):7–11, 03 1961.

Cambridge Semantics. Learn sparql, 2022. URL https://cambridgesemantics.com/blog/semantic-university/learn-sparql/. Acedido a 27/01/2022.

solid IT. Db-engines ranking of graph dbms, 2022. URL https://db-engines.com/en/ranking/graph+dbms. Acedido a 10/01/2022.

Dan Sullivan. *NoSQL for Mere Mortals*. Addison-Wesley, 2015. ISBN 978-0-13-402321-2.

Apache TinkerPop. The gremlin graph traversal machine and language, 2021. URL https://tinkerpop.apache.org/gremlin.html. Acedido a 07/12/2021.

Jim Webber. *The Top 10 Use Cases of Graph Database Technology*. Neo4j Graph Database Platform, 2021.

Jim Webber and Rik Van Bruggen. *Graph Databases for Dummies*. Neo4j, 2020.

Roman Čerešňák and Michal Kvet. Comparison of query performance in relational and non-relation databases. *Transportation Research Procedia*, 40:170–177, 2019. 10.1016/j.trpro.2019.07.027.