

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Exploring Graph Databases

David Blanquett



Mestrado em Engenharia de Software

Orientador: Rui Maranhão Abreu

13 de setembro de 2022

Exploring Graph Databases

David Blanquett

Mestrado em Engenharia de Software

Aprovado em provas públicas pelo Júri:

Presidente: Prof. Nuno Flores

Arguente: Prof. Luis Cruz

Vogal Orientador: Prof. Rui Maranhão

13 de setembro de 2022

Resumo

Nos últimos anos a indústria tem enfrentado desafios sem precedentes no que toca à concretização da visão da Indústria 4.0. À medida que caminhamos, enquanto Humanidade, para a concretização desta visão os requisitos de dados têm aumentando, ao nível de volume e ao nível de velocidade de processamento, bem como na necessidade de extrair valor dos mesmos.

Devido à natureza do domínio industrial, onde os dados são altamente interligados e coesos, esta alteração de requisitos de dados representa uma ameaça para os sistemas de informação de suporte à indústria, que muitas vezes são baseados em sistemas relacionais — notáveis pela sua robustez e completude, mas apresentam défices no processamento de informação em ambientes altamente interligados. Como alternativa aos sistemas relacionais e estratégia de disrupção as *graph databases* podem ser exploradas, pois são descritas na literatura como ideais para esta categoria de domínios, onde a relação entre os dados cresce tanto na sua importância como no seu volume. A abordagem recomendada pela literatura para escolher uma determinada *graph database* consiste em empregar um processo *exploration based*, que se presa pela sua completude, mas também pela sua morosidade.

Como forma de, simultaneamente, tentar otimizar os processos *exploration based*, mas também de otimizar o contexto de *manufacturing* com o uso de *graph databases* foi proposto um processo *exploration based* modificado. Este processo exploratório permitiu correlacionar as propriedades do modelo de dados, como a densidade do mesmo, com aquilo que é o sucesso de aplicação das *graph databases* e, paralelamente, suportar a escolha de uma *graph database*.

As evidências geradas por este processo *exploration based* modificado permitiram, em paralelo, propor uma heurística de otimização de processo baseada na densidade do grafo, que nos permite, a jusante, deduzir que quanto maior for a densidade do grafo maior será a adequação das *graph databases* para o contexto do problema, bem como otimizar operações fulcrais para o domínio de *manufacturing* ao nível de *performance*, otimizando assim o domínio de *manufacturing*.

Keywords: NoSQL, Graph Databases, Data Engineering, Manufacturing Execution Systems

Abstract

In recent years the manufacturing industry has been facing unprecedented challenges when realizing the vision of Industry 4.0. As Humanity moves towards the realization of this vision, data requirements have increased, both in terms of volume, processing speed and as well in the need of extracting value from them.

Due to the nature of the manufacturing domain, where data is highly interconnected and cohesive, this change in data requirements poses a threat to information systems that support the industry, which are often based on relational systems — notable for their robustness and completeness, but highly inadequate in processing in highly interconnected environments. As an alternative to relational systems and as a disruption strategy, *graph databases* can be explored, since they are described in the literature as ideal for this category of domains where the relationship between data grows both in importance and volume. To consider the integration of *graph databases*, the approach recommended by the state of the art is to employ an exploration based process, which is bound by its completeness, but also by its lack of agility.

As a way of trying, simultaneously, to optimize the exploration based processes, but also optimizing the *manufacturing* context using *graph databases*, a modified exploration based process was proposed. This exploratory process made it possible to correlate the properties of the data model, such as its density, with the application success of graph databases and, in parallel, support the choice of a graph database.

The generated evidences by this modified exploration based process allowed, in parallel, the proposal of a process optimization heuristic based on the density of the graph, which will allow, beforehand, to deduce that the greater the density of the graph, the greater the suitability of *graph databases* is for the context of the problem, as well as optimizing core operations for the manufacturing domain performance-wise, thus optimizing the manufacturing domain.

Keywords: NoSQL, Graph Databases, Data Engineering, Manufacturing Execution Systems

Agradecimentos

A realização deste programa de mestrado e desta dissertação em particular não teria sido possível sem a contribuição e disponibilidade daqueles que, direta ou indiretamente, estiveram envolvidos ao longo de toda esta jornada. A todos quero expressar o meu reconhecimento e mais sincero agradecimento.

Em particular e sem especial ordem de importância, gostaria de expressar os seguintes agradecimentos:

Ao Professor Rui Maranhão Abreu, pela orientação científica do presente trabalho e pela disponibilidade que sempre demonstrou a responder às minhas questões.

À Critical Manufacturing, em especial à equipa de produto, pelo acolhimento, pelos desafios e pelo apoio demonstrado durante os últimos anos e, em particular, ao longo desta dissertação. Ao Luís Ponte pela disponibilidade e pela colaboração na criação de uma proposta de dissertação que me deu a conhecer uma área que tanto gosto.

Aos meus pais, ao meu irmão e à minha família mais próxima, por toda a força, pelo carinho que sempre demonstraram comigo e pelo apoio que me deram ao longo desta jornada.

Por fim, mas não menos importante, à minha namorada, pela força, pela compreensão e pelo carinho que tanto me ajudaram a concluir esta dissertação.

David Blanquett

*“If I have seen further than others,
it is by standing upon the shoulders of giants.”*

Isaac Newton

Conteúdo

1	Introdução	1
1.1	Contexto	1
1.2	Objectivos	2
1.3	Estrutura do documento	3
2	Revisão Bibliográfica	5
2.1	Industry 4.0	5
2.1.1	Contexto	5
2.1.2	Definição	6
2.1.3	Manufacturing Execution Systems	8
2.2	Graph Databases	10
2.2.1	Contexto	10
2.2.2	Definição	17
2.2.3	Processo de seleção	21
2.2.4	Comparação de implementações	25
2.2.5	Casos de Uso	28
2.3	Discussão	29
2.4	Conclusão	32
3	Descrição do Problema	35
3.1	Domínio	35
3.2	Arquitetura	37
3.2.1	Diagrama de Componentes	37
3.2.2	Diagrama de Implantação	38
3.3	Apresentação do problema	38
4	Desenvolvimento	41
4.1	Definição do Processo	41
4.2	Instanciação do Processo	43
4.2.1	Prólogo	43
4.2.2	Descrição dos Requisitos e respetivos indicadores	44
4.2.3	Escolha das implementações	46
4.2.4	Extração, Transformação e Carregamento dos dados	48
4.2.5	Benchmarking	55
4.2.6	Epílogo	65
4.2.7	Escolha Final	70

5	Integração da Solução	71
5.1	Processo de Integração	71
5.2	Arquitetura de Integração	73
5.3	Limitações	75
6	Conclusões e Trabalho Futuro	77
6.1	Discussão	77
6.2	Conclusão	79
6.3	Contribuições	80
6.4	Limitações	80
6.5	Trabalho Futuro	81
	Referências	83

Lista de Figuras

2.1	Resumo revolução Industrial. Fonte: World Economic Forum	6
2.2	Modelo da Indústria 4.0. Fonte: Critical Manufacturing	7
2.3	Framework de um Cyber-Physical System. Fonte: Griffor, E. , Greer, C. , Wollman, D. and Burns, M. (2017), Framework for Cyber-Physical Systems (NIST SP)"	8
2.4	Relações MES. Fonte: Critical Manufacturing [51]	9
2.5	<i>Schema</i> para o exemplo de informação modelada numa <i>Hierarchical Database</i>	11
2.6	Exemplo de informação modelada numa <i>Hierarchical Database</i>	11
2.7	<i>Schema</i> para o exemplo de informação modelada numa <i>Network Model Database</i>	12
2.8	Exemplo de informação modelada numa <i>Network Model Database</i>	12
2.9	ACM Search Engine — Número de Publicações com a keyword "Graph Database" entre 2000 e 2022	16
2.10	<i>Schema</i> para o exemplo de informação modelada numa <i>Graph Database</i>	18
2.11	Exemplo de informação modelada numa <i>Graph Database</i>	18
2.12	Representações do modelo tradicional. Adaptado de: Coimbra, M.E., Francisco, A.P., Veiga, L. "An analysis of the graph processing landscape"	19
2.13	Exemplo de informação modelada numa <i>Graph Database</i> — Modelo RDF	20
2.14	Fluxograma para escolher Graph Databases. Adaptado de: D. G. Denise and B. Matthias, The Practitioner's Guide to Graph Data [Book]	23
2.15	Distribuição dos modelos com base no par — Volume-Complexidade dos dados [42]	24
3.1	Modelo domínio — Manufacturing	36
3.2	Granularidade de Sistema: Diagrama de Componentes	37
3.3	Granularidade de Sistema: Diagrama de Implantação	38
4.1	Geração Script via Copilot	50
4.2	Comparação Schema Origem com Schemas Alvo — Exemplo FlowStep	51
4.3	Criação de Relações — Exemplo FlowStep	51
4.4	Tempo de execução <i>Material History</i> em milissegundos no Dataset 1 — escala logarítmica base 10	56
4.5	Tempo de execução <i>Material Geneology</i> em milissegundos no Dataset 1 — escala logarítmica base 10	57
4.6	Tempo de execução <i>Material History</i> em milissegundos no Dataset 2 — escala logarítmica base 10	57
4.7	Tempo de execução <i>Material Geneology</i> em milissegundos no Dataset 2 — escala logarítmica base 10	58
4.8	Tempo de execução <i>Material History</i> em milissegundos no Dataset 3 — escala logarítmica base 10	58

4.9	Tempo de execução <i>Material Geneology</i> em milissegundos no Dataset 3 — escala logarítmica base 10	59
4.10	Tempo de execução médio <i>Material History</i> em milissegundos no Dataset 1 — escala logarítmica base 10	59
4.11	Tempo de execução médio <i>Material Geneology</i> em milissegundos no Dataset 1 — escala logarítmica base 10	60
4.12	Tempo de execução médio <i>Material History</i> em milissegundos no Dataset 2 — escala logarítmica base 10	60
4.13	Tempo de execução médio <i>Material Geneology</i> em milissegundos no Dataset 2 — escala logarítmica base 10	60
4.14	Tempo de execução médio <i>Material History</i> em milissegundos no Dataset 3 — escala logarítmica base 10	61
4.15	Tempo de execução médio <i>Material Geneology</i> em milissegundos no Dataset 3 — escala logarítmica base 10	61
4.16	Diferença de desempenho dos diferentes tipos de modelação em função do tamanho de dados — Neo4J	64
4.17	Diferença de desempenho dos diferentes tipos de modelação em função do tamanho de dados — RedisGraph	64
4.18	Diferença de desempenho dos diferentes tipos de modelação em função do tamanho de dados — ArangoDB	64
4.19	Heatmap de correlação	68
5.1	Arquitetura de Integração — Diagrama de Componentes	73
5.2	Mensagens do MES Events — Operação sobre relação Flow-Step	73
5.3	Configuração do <i>Neo4j Streams Sink</i> — Operação sobre relação Flow-Step	74
5.4	Grafo Step-Area — Descrição da Area antes da replicação	75
5.5	Grafo Step-Area — Descrição da Area depois da replicação	75

Lista de Tabelas

4.1	Tabela de Hipóteses	42
4.2	Número de entidades nós diferentes datasets	44
4.3	Indicadores de desempenho do processo de seleção	46
4.4	Catálogo de <i>Graph Databases</i>	48
4.5	Informações relacionados com o carregamento dos dados	52
4.6	Dados estatísticos relativos à estrutura de dados no dataset QA	53
4.7	Dados estatísticos relativos à estrutura de dados no dataset CD	54
4.8	Dados estatísticos relativos à estrutura de dados no dataset PROD	54
4.9	Propriedades dos diferentes Datasets	55
4.10	Matriz de correlação	68
4.11	<i>Cohen's Standard</i> para a interpretação de coeficientes de correlação	69
5.1	Prós e Contras: Integrações Kafka [46]	72

Abreviaturas e Símbolos

API	Application Programming Interface
BFS	Breadth-First Search
CPS	Cyber-Physical System
CSV	Comma-separated Values
EISM	Enterprise Information Systems in Manufacturing
GDB	Graph Database
GDBMS	Graph Database Management System
I4.0	Indústria 4.0
JSON	JavaScript Object Notation
LPG	Labeled-Property Graph
MES	Manufacturing Execution System
OLAP	Online Analytics Processing
OLTP	Online Transaction Processing
PR	Pull-Request
RDBMS	Relational Database Management System
RDF	Resource Description Framework
SGBD	Sistema de Gestão de Bases de Dados
SQL	Structured Query Language
SUT	System Under Test
W3C	Wide Web Consortium

Capítulo 1

Introdução

1.1 Contexto

Devido à disrupção tecnológica que o mundo digital tem presenciado nos últimos anos, um novo provérbio foi formulado pela comunidade tecnológica: “*Data is the new Oil*” [25], que significa que o valor dos dados se tornou mais influente do que o valor do petróleo, recurso que teve uma importância fulcral no passado, em especial na Era Industrial. Este valor dos dados está relacionado, primeiramente, com a análise e extração de significado dos mesmos, mas também com o suporte de requisitos como o Volume, a Velocidade, a Variedade, a Veracidade, a Variabilidade e acréscimo Valor, conotados como os 5 V’s de *Big Data* [59].

Esta mudança de paradigma de dados é homogênea com o advento da Indústria 4.0, que introduz conceitos como *Smart Products*, *Smart Equipements* e *Smart Factories*, que prevê uma capacidade de organização, comunicação e interação autónoma entre estas entidades, o que implica que sistemas de suporte à indústria têm de suportar novos requisitos de dados nunca antes vistos — relacionados com o volume e com a velocidade de processamento. Como tal os sistemas de suporte à indústria, como os Manufacturing Execution Systems (MES), tem de se adaptar ou deixarão de ser viáveis.

Recentemente, na comunidade científica dos Database Systems, tem-se notado um crescente interesse no que toca a Graph Databases — um tipo de base de dados NoSQL. Este interesse está relacionado com a importância que os dados e a análise dos mesmos têm vindo a ganhar nas organizações, mas também com a alegada capacidade que estes sistemas têm de suportar dados altamente coesos, ao nível de relações, com excelentes resultados. Algo que se tem notado no estado da prática é que para ambientes complexos, como a Indústria e o sector Biomédico, a informação é altamente coesa e correlacionada [83]. Como resposta a esta tendência, onde a relação entre os dados cresce tanto na sua importância como no seu volume, foram propostas pela comunidade os *Graph Database Management Systems* (GDBMS). Este modelo surge porque até então estes domínios altamente complexos eram representados, normalmente por grafos, em sistemas relacionais — Relational Database Management Systems (RDBMS) — notáveis pela sua robustez

e riqueza de ecossistema, no entanto, para este tipo de domínios complexos, a informação é, devido à sua natureza bastante coesa, representada por um grande número de entidades com relações de muitos para muitos, cuja relação é suportada por múltiplas operações de *JOIN*, o que têm um grande impacto na performance [20], bem como naquilo que é a facilidade de manipulação de dados, podendo pôr em causa os novos requisitos da Indústria 4.0 descritos anteriormente. Devido às suas propriedades, as GDBMS tratam as relações entre os dados como "*first-class citizens*" otimizando estes cenários [67], tanto ao nível de performance, como ao nível de manutenção e de manipulação dos dados.

Tendo em conta o intuito de um sistema de MES, que consiste em modelar e agregar as operações de uma fábrica, e a estrutura dos seus dados podemos afirmar que o domínio do mesmo é bastante denso ao nível de relações com as entidades. Devido a este facto, operações cruciais estão a ter degradação de performance à medida que o sistema evolui ao nível de volume de dados. Tendo em conta o contexto de Manufacturing, o advento da Indústria 4.0 e as características das Graph Databases faz sentido perceber como é que estas podem responder a este problema.

Como forma de restringir o escopo da dissertação para perceber se o contexto de Manufacturing pode ser, de facto, otimizado com as Graph Databases a análise do mesmo terá em conta as duas principais operações que têm visto degradação de performance — A *Material History*, relacionada com o histórico de operações e relações de um material, e a *Material Genealogy*, relacionada com a estrutura hierárquica do mesmo.

Para resolver este tipo de problema, a abordagem recomendada pelo estado da arte é seguir um processo *exploration based* [71], que consiste em fazer uma análise exaustiva das diferentes soluções de bases de dados a considerar — isto porque existe a premissa que para diferentes problemas devemos utilizar diferentes soluções. Os processos *exploration based*, devido à sua completude, são bastante morosos e podem não garantir uma solução ideal. A acrescentar a este facto podemos ainda argumentar que da mesma forma que existem trabalhos publicados que indicam o sucesso dos GDBMS, como, por exemplo, [14], [62], também existem trabalhos que dizem o oposto, como, por exemplo, [27], [79]. Não existindo entendimento nem diretivas generalizáveis para todos os problemas. Tendo em conta a complexidade e morosidade do processo, existe a necessidade de perceber se é possível otimizar o mesmo — algo que é proposto por este documento — e permitindo assim, simultaneamente, otimizar o contexto de Manufacturing.

1.2 Objectivos

Tendo em conta o contexto descrito anteriormente, podemos afirmar que o propósito desta dissertação consiste em endereçar os seguintes objetivos:

- (i) Desenvolver um processo, *exploration based*, para suportar a escolha de um novo GDBMS
- (ii) Criar uma heurística de otimização de processos *exploration based* que consideram GDBMS

- (iii) Otimizar o contexto de *Manufacturing* com o uso de GDBMS

1.3 Estrutura do documento

No capítulo 2, é elaborado um levantamento no que toca ao contexto de Indústria 4.0 e ao estado da arte e da prática no que toca ao enquadramento do tópico de *Graph Databases*, bem como é apresentada uma prescritiva crítica sobre o mesmo. Este capítulo surge como ponto de ligação entre o conceito de Indústria 4.0 e *Graph Databases* e tem o objetivo elucidar o leitor com o conhecimento necessário relativamente ao estado da arte e o estado da prática no que toca a contexto e técnicas relacionadas com *graph databases* e o domínio de *manufacturing*, de forma a evidenciar o *knowledge gap* existente entre estes dois conceitos e a sua aplicação, dando suporte ao restante corpo da dissertação. A seguir, no capítulo 3, é descrito o problema que esta dissertação se propõe a resolver. É apresentada uma descrição do domínio de *manufacturing*, bem como uma vista arquitetural do sistema vigente. Como resultado deste capítulo são formuladas quatro questões que terão de ser respondidas nos seguintes capítulos. No capítulo 4 é definido, na secção 4.1, e aplicado, na secção 4.2, um processo que permite responder às questões levantadas no capítulo anterior. Como resultado deste capítulo são propostas possíveis respostas para as questões levantadas anteriormente e que, em paralelo, suportam a escolha de um SGBD (Sistema de Gestão de Bases de Dados) que se propõe a otimizar o contexto de *manufacturing*. Seguidamente, no capítulo 5, é apresentada uma sugestão de integração do SGBD escolhido, resultante do capítulo anterior, com o sistema *Manufacturing Execution System* (MES) da Critical Manufacturing. Aqui são apresentadas as decisões de integração, bem como uma prescritiva arquitetural da nova solução. Por fim, no capítulo 6, é feito um sumário da dissertação onde é apresentada uma visão global do trabalho desenvolvido, bem como as conclusões que tiveram origem ao longo do corpo de texto da dissertação. Por fim é feita uma retrospectiva ao nível do processo empregado e das implicações das conclusões obtidas, bem como são apresentadas limitações do trabalho desenvolvido e aquilo que poderão ser diretrizes de suporte para trabalhos futuros.

Capítulo 2

Revisão Bibliográfica

Neste capítulo é feito inicialmente, na secção 2.1, um sumário da visão da Indústria 4.0 (I4.0), bem como as principais mudanças que advêm a jusante deste evento disruptivo. Também aqui é discutido como este evento disruptivo poderá ser potenciado pelos sistemas MES (Manufacturing Execution Systems). A seguir, na secção 2.2, é inicialmente feita uma contextualização das *graph databases*, como também é apresentada uma possível explicação para o crescente interesse relativamente ao assunto nos últimos anos. Nesta secção está também presente uma sistematização dos conceitos relativos a *graph databases*, bem como o seu estado da prática, a avaliação, os seus prós e os seus contras. Logo depois, na secção 2.3 é apresentada uma perspetiva crítica sobre o estado da arte e aquilo que são, na opinião do autor, os principais pontos em aberto relativamente ao estado da arte. Por fim, na secção 2.4 são apresentados os comentários finais sobre o estado da arte e apresentadas as diretivas para trabalho futuro, que servem de suporte para o trabalho desenvolvido nesta dissertação.

2.1 Industry 4.0

Nesta secção é discutido o conceito de Indústria 4.0. Inicialmente na secção 2.1.1 é explicado o contexto da Indústria 4.0. De seguida na secção 2.1.2 é explicado como foi cunhado este termo, quais é que foram as principais motivações e quais os paradigmas associados. Por fim, na secção 2.1.3 é feito uma associação entre os conceitos de Indústria 4.0 e os *Manufacturing Execution Systems* (MES) e como estes são essenciais para a concretização desta visão.

2.1.1 Contexto

A Indústria é um sector que beneficia diretamente daquilo que são as evoluções tecnológicas. Algo que a história nos mostra é que esta evolução não acontece gradualmente, mas sim abruptamente, um acontecimento denominado por revolução Industrial. Reparando naquilo que são as tecnologias emergentes, como *Cyber-Physical Systems*, *Internet of Things*, *Cloud Computing* e *Artificial Intelligence*, e na maneira que elas afetam a nossa vida, podemos afirmar que estamos nos primórdios de uma nova revolução Industrial — Indústria 4.0 [19].



Navigating the next industrial revolution

Revolution	Year	Information
	1	1784 Steam, water, mechanical production equipment
	2	1870 Division of labour, electricity, mass production
	3	1969 Electronics, IT, automated production
	4	? Cyber-physical systems

Figura 2.1: Resumo revolução Industrial. Fonte: World Economic Forum

A primeira revolução Industrial é reconhecida como a mudança de paradigma no que toca a fonte de energia, passamos de depender do esforço humano ou animal para o uso de combustíveis fósseis, e como consequência tivemos acesso à máquina a vapor e aos caminhos-de-ferro, o que levou a um aumento brutal de produtividade. A segunda revolução Industrial tem início no fim do século XIX e as principais mudanças de paradigma foram a distribuição de eletricidade, comunicação por cabo e *wireless*, a síntese de amónia e novas formas de geração de energia, foi aqui que criamos o conceito de produção em massa e de linha de montagem e onde teve origem a celebre frase de Henry Ford, “Qualquer um pode comprar um Ford Model T, na cor que desejar. Desde que ele seja preto” que tinha que ver com o facto de o preto ser a tinta que secava mais rápido, aumentado a produção e por consequência baixando os preços. A terceira revolução Industrial teve início por volta de 1950, e é onde nos encontramos hoje, com o desenvolvimento dos sistemas digitais, o aumento do poder de computação, que permitiu novas maneiras de gerar, processar e partilhar informação, esta revolução está associada com a automação e com o universo da eletrónica. A próxima revolução será a quarta revolução Industrial, descrita como *the advent of cyber-physical systems*. A principal ideia é construir um novo paradigma, utilizando as tecnologias e infraestrutura da terceira revolução Industrial, para permitir que a tecnologia seja embebida na sociedade e a interação com o ser humano suceda de forma fluida. Exemplos incluem: novos paradigmas de *machine intelligence* e estratégias de gestão baseados em métodos distribuídos e criptográficos como a *blockchain* [19].

2.1.2 Definição

O termo *Industry 4.0* foi cunhado na Alemanha, por um consórcio público-privado, em 2013, pelo documento "Recommendations for implementing the strategic initiative INDUSTRIE 4.0"[36] que foi apresentado na Feira de Hannover. A Feira de Hannover é, ainda hoje, um foco mundial da Indústria 4.0. O documento começa por afirmar que a Alemanha é atualmente uma das Indústrias de manufatura mais avançadas no mundo, em especial na Indústria automóvel e de maquinaria,



Figura 2.2: Modelo da Indústria 4.0. Fonte: Critical Manufacturing

e para o continuar a ser os Cyber-Physical Systems (CPS) surgem como aposta para a estratégia da Indústria no país. Dado o mote de partida pela Alemanha, todos os países decidiram ir atrás, no caso de Portugal temos o programa Indústria 4.0 [64], que tinha o intuito de digitalizar a economia portuguesa usando técnicas e tecnologias da Indústria 4.0. A Indústria 4.0, como referido anteriormente, utiliza aquilo que foram os avanços anteriores, nomeadamente da automação, da digitalização da produção para alcançar a sua visão [40]. A Indústria 4.0 é um ecossistema tecnológico com muitas inovações associadas, também referidas como *Technologies 4.0*, das quais: robótica, automação, 3D Printing, cobots (robots colaborativos), *Cloud Computing* e *Artificial Intelligence*. São estas tecnologias que vão estar presentes nas fábricas do futuro, ou como referidas na bibliografia *Smart Factories* [74]. De uma maneira geral, podemos definir o modelo da Indústria 4.0 com base na figura 2.2. Atendendo à figura, tudo começa com o modelo de oferta-procura, onde os consumidores procuram cada vez mais produtos mais personalizados, e a resposta a esta necessidade é dada pelos CPS. A ideia principal consiste em transformar o processo de manufacturing como um *marketplace* de capacidade de processamento que através dos CPS vão gerir, autonomamente, a linha de produção de fábrica e a respetiva *supply-chain*, organizando todo o processo como um MaaS (*Manufacturing as a Service*) e funcionando assim como um sistema automatizado multiagente. Analisando esta ideia podemos dizer que passamos de um paradigma de “produção em massa” para um paradigma “customização em massa”. Para responder a este requisito, é preciso a criação *Smart Products* e *Smart Equipements*, o conceito de *smart* evidencia, entre outras coisas, que as entidades têm a capacidade de comunicar entre si, de se organizarem e tomarem decisões autonomamente — a este fenómeno de comunicação e organização chamamos de *shop-floor marketplace*.

Esta ideia está assente nos Cyber-Physical Systems (CPS), representado na figura 2.3. Uma das principais características e funcionalidades dos CPS é a informatização/digitalização do mundo físico e é definido como um *smart system* que inclui redes de componentes físicos e componentes computacionais. Estes componentes altamente integrados fornecem novas possibilidades para melhorar a qualidade de vida e avanços tecnológicos em áreas críticas, como a saúde, a indústria, entre outros. Um CPS pode ser formalmente definido como um sistema que integra redes de

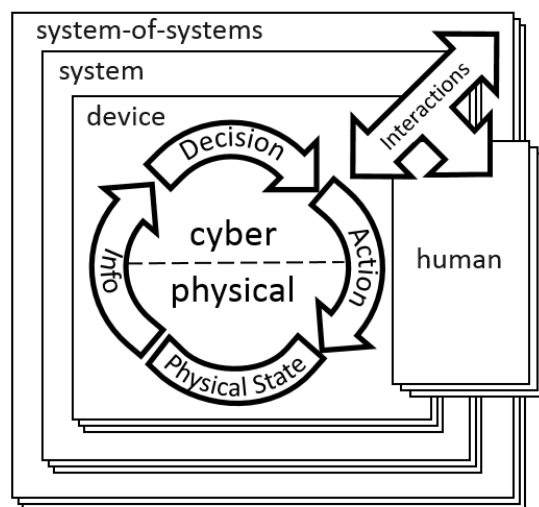


Figura 2.3: Framework de um Cyber-Physical System. Fonte: Griffor, E. , Greer, C. , Wollman, D. and Burns, M. (2017), Framework for Cyber-Physical Systems (NIST SP)"

computação, de comunicação, de sensores e atuação com sistemas físicos para concretizar operações *time-sensitive*, com vários níveis de interação com o ambiente, incluído a interação humana [31]. Estas são as características que permitem que estes sistemas altamente desenvolvidos sejam capazes de tomar decisões autonomamente, suportando o cenário de gestão autónoma descrita anteriormente.

Ao contrário das revoluções anteriores, a Indústria 4.0 é a primeira revolução que está a ser antevista pelos especialistas, o que permite a que as empresas se transformem para tornar esta visão uma realidade. Isto apresenta uma grande oportunidade, mas também grande desafio para Enterprise Information Systems in Manufacturing (EISM) descritos por Mantravadi et al. [50], dos quais se incluem os Manufacturing Execution Systems (MES), os sistemas de Supply-chain Management (SCM) e de Enterprise Resource Planning (ERP).

Devido aos requisitos da I4.0, como a descentralização, a necessidade de comunicação entre entidades, ao suporte de métodos analíticos altamente avançados estes sistemas, que hoje suportam o ecossistema de manufacturing, têm de se adaptar, ou deixarão de existir [2].

2.1.3 Manufacturing Execution Systems

A Critical Manufacturing descreve um sistema MES como “um sistema de informação responsável pelas operações de Manufacturing” [51]. Na realidade o conceito de MES tem origem nos anos 90, quando uma organização *non-profit* de soluções de *manufacturing*, a AMR-Research mais tarde adquirida pela Gartner, estruturou os sistemas EISM em três níveis, *Planning*, *Execution* e *Control* [76]. A camada de *planning* é representada pelos sistemas corporativos, como os ERPs. A camada de execução pelos sistemas de MES. A camada de *Control*, responsável pela automação das máquinas. Com o passar do tempo os sistemas de *MES* aumentaram a sua *coverage* de funcionalidades e agora conseguem suportar a de execução processo de manufatura, na sua

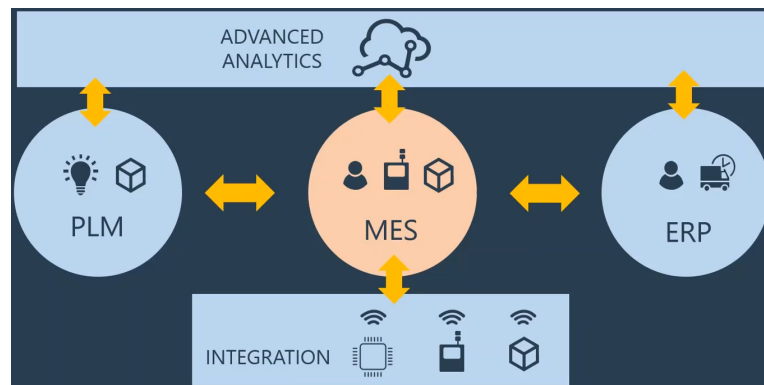


Figura 2.4: Relações MES. Fonte: Critical Manufacturing [51]

totalidade, como podemos verificar pela seguinte revisão de literatura [73]. De um ponto de vista sucinto, podemos afirmar que os sistemas MES são sistemas de informação usados em *manufacturing*, para fazer *tracking* e documentar a transformação de *raw materials* em produtos finais. O sistema MES fornece informação que ajuda a tomar decisões, por exemplo, fornece informação de como está o chão de fábrica atualmente e como pode ser melhorado a sua produção. O sistema MES funciona em tempo real para permitir o controlo de múltiplos elementos de produção, como máquinas, operários, serviços de suporte, entre outros. O sistema de MES tem ainda a informação como é produzido um material, que *workflows* de produção poderá usar, permite a recolha de dados, permite a otimização do processamento, e funciona como garantia de qualidade no sentido em que garante o cumprimento de determinados *standards* internacionais de produção [54].

Devido à sua especificação, o MES funciona como *sistema operativo da fábrica* e como intermediário naquilo que são mecanismos de *Advanced Analytics* necessários na I4.0. Em suma podemos afirmar que um sistema MES tem as seguintes vantagens [76]:

- Aumento de produtividade: Através de funcionalidades de escalonamento pode sugerir qual o melhor *workflow* de processamento para um determinado material
- Controlo de Qualidade: Oferece funcionalidades que garantem o controlo de qualidade
- Flexibilidade: Melhora o *time to produce* de um produto.

Por outro lado, podemos enumerar as seguintes desvantagens:

- Treino: É um *software* altamente especializado, que implica que os trabalhadores de fábricas tenham treino prévio.
- Sistema “Crítico”: funciona como a espinha dorsal de uma fábrica, por ser tão essencial para as operações de fábrica, tem de considerar requisitos como baixa latência, segurança, entre outros.

Devido às especificações deste sistema conseguimos perceber o quão fundamental é para concretizar a visão da I4.0 e como este se relaciona com os CPS. Por um lado, os sistemas MES têm

o propósito de digitalizar o domínio de *manufacturing* para diversos fins, uma etapa essencial no caminho para a I4.0. Por outro lado, à medida que caminhamos para a I4.0, os sistemas MES vão ter de se adaptar para continuarem relevantes [2], no sentido em que têm de suportar requisitos ainda mais exigentes relativamente aos dados, como requisitos de Volume, Velocidade, Variedade, Veracidade, Variabilidade e acréscimo Valor (5 V's de *Big Data* [59]).

2.2 Graph Databases

Esta secção tem o objetivo de sintetizar, com base na bibliografia, a área das *graph databases* e perceber como elas podem ser utilizadas, bem como as suas principais vantagens e desvantagens. Na secção 2.2.1 é efetuada uma contextualização histórica relativamente aquilo que foram os problemas, as principais soluções e o seu respetivo racional em matéria de persistência de dados em sistemas complexos. O objetivo desta secção é perceber quais é que foram os principais design principles ao longo do tempo e como é que eles tiveram impacto naquilo que são as soluções atuais. A seguir, na secção 2.2.2 é formalizado o conceito de *graph system* e *graph database*, desde a forma como podem ser caracterizados, bem como estratégias de modelação e tipos de *storage*. Posteriormente na secção 2.2.3 são apresentadas as estratégias presentes na bibliografia para escolher um sistema de base de dados. Seguidamente na secção 2.2.4 é focado o processo de *Benchmark* bem como as estratégias utilizadas, e resultados apresentados na bibliografia. Por fim, na secção 2.2.5 são apresentados alguns dos principais casos de uso das *graph databases*.

2.2.1 Contexto

É de conhecimento geral que as bases de dados relacionais dominam o mercado já há vários anos, mas nem sempre foi assim. A verdade é que a proposta de modelos de bases de dados existe desde do fim dos anos 60 [78], e desde aí podemos agrupar o tempo em aproximadamente 4 eras [22], a era *Pre-Relational Databases*, a era *Relational Databases*, a era *NoSQL Databases* e, por mim aquilo que na minha opinião será a próxima era, a era de *Graph Systems*. Relativamente à divisão podemos dizer que as 3 primeiras já são reconhecidas pela comunidade científica. Cada era é caracterizada por tentar concretizar requisitos diferentes, o que, por conseguinte, leva a criação de abordagens diferentes para problemas semelhantes relacionados com a persistência e gestão de dados, como podemos perceber pelas seguintes subsecções.

2.2.1.1 Pre-Relational Databases - 1960 – 1980

A era das Pre-Relational Databases é caracterizada pelo desenvolvimento de dois modelos em simultâneo, o modelo de *Hierarchical Databases* e de *Network model Databases*.

A literatura atual emprega o conceito de *Hierarchical Databases* e *Navigational Databases* de forma semelhante, o conceito de então era de organizar a informação em formato de árvore. As *Hierarchical Databases* têm início com o IMS System [38], fundado por volta de 1968. Este sistema já implementava a noção de *record type* como sendo uma lista de campos associados com

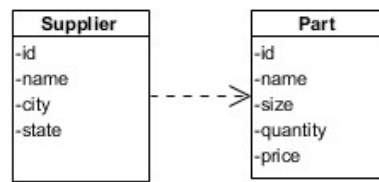


Figura 2.5: *Schema* para o exemplo de informação modelada numa *Hierarchical Database*

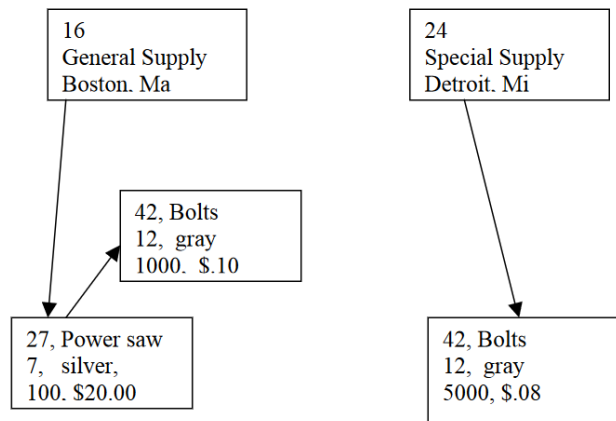


Figura 2.6: Exemplo de informação modelada numa *Hierarchical Database*

um determinado tipo de dados, ainda mais, cada conjunto de campos especificava uma *key*, como acontece com os sistemas atuais, por fim, estes sistemas foram desenhados com uma estrutura em árvore, onde cada *record* era responsável por apontar para um único *parent record type*. Um exemplo pode ser visualizado na imagem 2.6.

Esta estratégia de resolução tinha duas propriedades que não eram ideais:

- A informação era persistida de forma **redundante**. Como podemos perceber pela modelação da figura 2.6 o *record* dos bolts está duplicado, isto deve-se ao facto de ao nível de design um nó-filho só pode fazer referência a 1 nó-pai.
- A informação existe de forma dependente. Como podemos perceber pela modelação da figura 2.6 o *record* dos bolts é apenas acessível através do nó pai. Esta dependência entre *records* faz com a modelação seja menos flexível, o que leva todas as soluções que utilizem este tipo de sistemas percam agilidade.

Nesta era, em simultâneo, outro modelo foi proposto na *Conference on Data Systems Languages* (CODASYL), o *netwrok model* que permitia uma relação de muitos-para-muitos entre a informação, ao contrário da anterior [60]. Isto foi possível porque esta estratégia foi desenhada usando por base a estrutura de grafo, isto resolvia problemas relacionados, por exemplo, com a independência dos dados, enquanto que no modelo hierárquico era impossível remover nós-pai, aqui era mais fácil. Uma outra informação importante é que foi neste contexto que foi proposta/tentativa de criar um *standard* daquilo que seria o acesso a bases de dados.

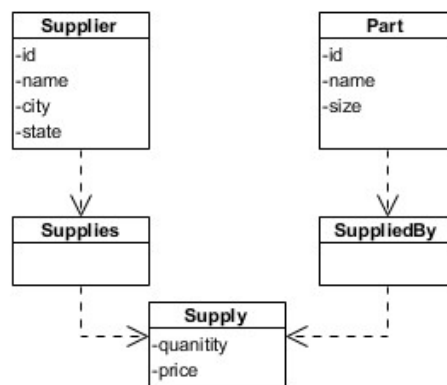


Figura 2.7: Schema para o exemplo de informação modelada numa *Network Model Database*

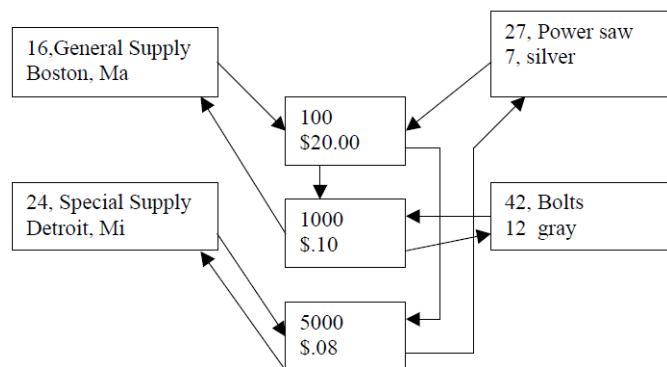


Figura 2.8: Exemplo de informação modelada numa *Network Model Database*

Um exemplo pode ser visualizado na imagem 2.8.

Analisando a figura 2.8 e comparando-a com o modelo hierárquico podemos tirar, por um lado, as seguintes ilações:

- O facto de a informação ser modelada em grafo permite que um determinado nó-filho faça referência a vários nós-pai, o que reduz a redundância quando comparado com o modelo hierárquico
- O facto de a informação ser modelada em grafo permite os dados sejam de alguma forma mais independentes, porque não é obrigatório apontar para um nó-pai

Por outro lado, esta estratégia de modelação também tem os seguintes problemas:

- O facto de a informação ser modelada em grafo implica que quando queremos migrar os dados de um sistema para um novo, o **processo de upload seja mais demorado**
- O facto de a informação ser modelada em grafo onde os nós podem ter qualquer tipo, garante mais flexibilidade por um lado, mas por outro faz com que o **esquema de dados seja duplicado**, como podemos analisar no exemplo onde é feita a referência aos preços.

Podemos concluir que ambos os paradigmas eram mapeados em estruturas de dados diferentes, e tinham por isso funcionamento diferente, no entanto, o seu principal calcanhar de Aquiles está relacionado com a forma com que a informação era consultada. Como não existia separação lógica dos dados, a informação era consultada por *queries* bastante complexas, que consistiam em percorrer todas as entradas utilizando sucessivos *loops*, e quando o *data schema* era alterado, todo o código desenvolvido tinha de ser revisto.

2.2.1.2 Relational Databases - 1980 – 2000

Em 1970, Ted Codd propõe a ideia disruptiva de criar uma abstração entre organização da informação do sistema — modelo físico — e a sua representação — modelo conceptual — [16], o que deu início a um novo ciclo de inovação. Segundo o autor, a principal razão que levou ao desenvolvimento da sua investigação foi o facto de o sistema IMS ter difícil manutenção quando mudanças físicas aconteciam, a alteração do *schema* levava à alteração de tudo o que eram implementações dos sistemas cliente [78]. Neste modelo a informação é estruturada em forma de relações (ou conjunto de tuplos) que era mapeada em forma de tabela. O modelo combinava ideias anteriores, como, por exemplo, as *record keys* naquilo era a pesquisa de informação, com um forte background científico de álgebra relacional. Esta implementação também tinha uma novidade de conter *metadata* da definição das tabelas, o que potenciava a separação lógica dos dados. Nos sistemas relacionais é possível relacionar entidades diferentes, mapeadas em diferentes tabelas. Para criar relações entre entidades é preciso criar mais tabelas, que têm informação relativamente a *record key* da tabela origem e da tabela destino.

Esta estratégia de abstração da modelação por ser recente e diferente do paradigma anterior demorou a ser aceite pela comunidade, repare-se que desde publicação de Ted Codd em 1970 até à primeira implementação foram precisos 10 anos, isto levou a uma grande discussão na comunidade científica que acabou por ser resolvida pelo facto de o modelo relacional ter sido proliferado no mercado, deixando as outras implementações anteriores com pouco uso. Um outro fator que levou a proliferação do modelo relacional foi o facto de ser mais eficiente em termos de memória, e menos redundante na forma como guardara a informação, o que pode ser comprovado, por exemplo, pela terceira forma normal (3FN) [41], que previne a redundância introduzindo tabelas intermédias em cada relação. Fator este que era bastante importante na época, devido ao valor do recurso, e mais resistente a falhas de hardware [78]. Foi graças a esta estratégia de modelação que hoje, quando queremos modelar algum problema, vamos abordar este problema segundo uma estratégia tabular.

Durante esta era foi também proposto o modelo orientado a objetos, mas não teve muita tração. O racional que levou ao desenvolvimento do mesmo estava relacionado com o facto de o paradigma de programação vigente ser o orientado a objetos e existia um questionável *overhead* naquilo que era o mapeamento Objeto, com uma estrutura semelhante ao JSON, para a forma tabular. Apesar do racional por trás desta proposta ser legítimo e fazer sentido de um ponto de vista de engenharia, o modelo não teve tração porque a verdade é que o sistema relacional resolvia todos os problemas de então.

2.2.1.3 NoSQL Databases - 2000 – 2020

No fim dos anos 90, os requisitos para sistemas de base de dados estavam bastante estanques, o que levou a uma grande consolidação do modelo relacional. Com o passar do tempo o hardware também ficou mais barato, tal como previsto pela lei de Moore [11], portanto era possível afirmar que o problema das eras anteriores — *storage inefficiency* — estava na sua maior parte mitigado. No início dos anos 2000 deu-se o início da Web 2.0, o que levou a uma massificação da web e, por conseguinte, a uma quantidade enorme de dados com requisitos de *high-availability* bastante elevados, este evento disruptivo fez com que aquilo que eram os requisitos para os sistemas de base de dados fossem alterados [20]. Passamos de um problema de *storage efficiency* para um problema de acessos concorrentes, *high-availability* e *scalability*, o que fez com que as bases de dados relacionais deixassem de ser a *silver bullet* para todos os problemas. Para responder a estes requisitos, as grandes empresas tecnológicas de então tomaram a decisão de colocar em causa alguma das premissas das bases de dados relacionais, decidiram trocar propriedades como transações, suporte de uma interface universal, como o SQL, por propriedades de *high-availability* e *scalability* [84]. A diversidade de aplicações também aumentou, e para isso os responsáveis de arquitetura tinham a necessidade de tratar a informação com diferentes formas e volumes para aplicações num ambiente ágil, o que deu origem a bases de dados NoSQL, como key-value, wide-column, document, and graph stores [20]. Definir as bases de dados NoSQL é algo complicado, mas podemos identificar algumas características, como, por exemplo [84]:

- Foram construídas para serem executadas em *clusters* e ambientes altamente distribuídos, com a exceção das bases de dados em grafo.
- Não precisam de um *schema* predefinido — *schemaless* — o que confere uma agilidade adicional quando comparado com outros modelos
- Open source
- Consistência eventual de dados — não garante a consistência de dados, como o caso do modelo relacional — via regras de integridade.

Principalmente o movimento NoSQL foi o responsável por massificar a mensagem que para diferentes problemas, diferentes abordagens — bases de dados — deviam ser utilizadas [72]. Mas isso levanta a questão: quais os problemas e que sistemas usar para os diferentes problemas?

De acordo com Roy-Hubara et al. [71] existem três estratégias para escolher um modelo de base de dados:

- *(i) Agenda based*: O arquiteto de soluções escolhe conforme a sua vontade de explorar/implementar um determinado sistema;
- *(ii) Knowledge based*: O arquiteto de soluções escolhe conforme o conhecimento e experiência da equipa;

- **(iii) *Exploration based*:** uma abordagem mais científica, onde são considerados as características dos dados, bem como os requisitos funcionais e requisitos não funcionais do sistema, infelizmente esta última é a menos comum, devido a restrições de recursos e tempo.

Relativamente a abordagem *Exploration based*, aquela que é mais passível de trabalho científico, existem alguns trabalhos que a aplicam, como, por exemplo [80] onde é feita uma comparação de *performance*, no caso, entre 5 das mais populares bases de dados NoSQL, excluindo o modelo em grafo, onde se tira a conclusão que o Redis — uma base de dados key-value — é a mais indicada para *workloads* com dados simples, com poucas interdependências, onde a capacidade de leitura/escrita é bastante elevada (como, por exemplo, sistemas de *caching*) as bases de dados em documentos — como o MongoDB — são mais eficientes ao nível de *performance* e escalabilidade por suporta um maior número de operações por segundo.

Existe também estudos relacionados com outras características que não a *performance*, por exemplo, Bernardino J. et. al [28] elabora uma comparação de bases de dados em grafo, relativamente o suporte a determinadas *query languages*, o suporte de vários modelos, a possibilidade de *sharding*, *scalability*. No caso foi concluído, por exemplo, que a escolha de uma determinada base de dados em grafo depende do problema que temos em mãos. De grosso modo o Neo4J e a ArangoDB respondem à maior parte dos problemas de grafos, no entanto, se estivermos perante um problema de sistemas altamente distribuídos pode não ser a melhor solução por não possuir a capacidade de *sharding*.

Apesar de este tipo de estudos ajudar na escolha de um modelo de bases de dados, a verdade é que há muitos pontos de variação para fazer comparações, um dos que, a meu ver, é mais importante é, por exemplo, a qualidade com a que a informação é modelada num determinado modelo. Podemos argumentar que quanto mais experiência o autor tem com uma determinada tecnologia/modelo melhor será a sua implementação/modelação. Segundo o Monika Shah Et al. [75] podemos perceber a influência que o *Schema design* tem sobre a *performance* de um sistema, bem como que não existe uma estratégia predefinida que me garanta resultados.

Durante esta era, também existiram esforços de continuar com o modelo relacional e responder a questões de *high-availability* e *scalability*, como é o caso do NewSQL¹, o principal problema é que a maior parte dos sistemas desenvolvidos eram *closed-source*, o que fez com que a adoção pela comunidade fosse muito lenta [63].

2.2.1.4 Graph Systems — 2020-?

Como foi referido anteriormente, as *Graph Databases* tiveram origem na era anterior, mas nunca tiveram tanta tração como as outras, como podemos verificar pelo artigo [32] onde é desenvolvido um *index of interest* de bases de dados NoSQL entre os anos 2008 e 2017, baseado em questões abertas no stackoverflow, verificamos que as bases de dados em grafo são as que existem em menos número de implementações e que o seu *index of interest* é menor, no entanto, se

¹<https://www.cs.cmu.edu/pavlo/slides/hydra-newsql2021.pdf>

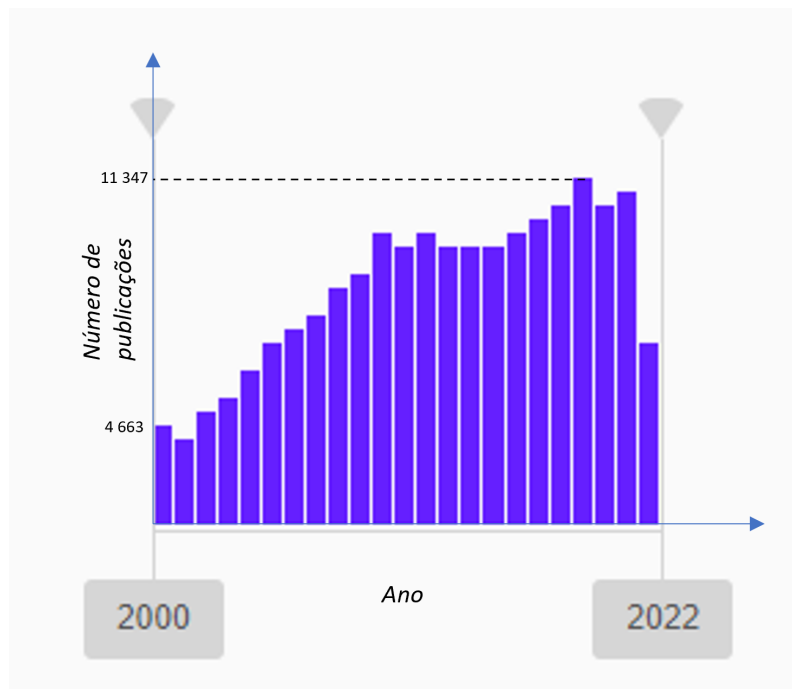


Figura 2.9: ACM Search Engine — Número de Publicações com a keyword "Graph Database" entre 2000 e 2022

no motor de pesquisa da ACM, onde existem as mais prestigiadas conferências de SGBD, podemos verificar que o número de publicações tem um aumento bastante acentuado nos últimos anos, como podemos verificar pela figura 2.9.

A figura 2.9 demonstra o crescente interesse nos últimos anos, com um pico de cerca de 11.347 entradas em 2019. E isto leva-nos à questão: porquê agora? O primeiro ponto está relacionado com o atrasado relativo ao ciclo de inovação [69], algo que já aconteceu no passado com o modelo relacional, repare-se que existiu um atraso de 10 anos entre a publicação que descrevia o modelo relacional e a primeira implementação, a história descrita nas secções anteriores mostra exatamente isso, todas as tecnologias começaram como sendo um nicho de mercado até serem massificadas. O segundo ponto está relacionado com a crescente importância que os dados estão a ganhar nas organizações, como referido pelo *The Economist*, com a célebre frase “Data is the new Oil” [25]. No processo de transformação digital da economia a informação e a sua qualidade passa de ser uma consequência do facto de termos um sistema digital, para ser um *asset* estratégico naquilo que são as empresas e o seu sucesso, o que leva com que os requisitos para os dados agora tenham uma grande ênfase no valor, em vez da sua velocidade e custo de storage. Devido a este mindset-shift, onde passamos de representar uma determinada entidade como um conjunto de atributos (modelo relacional) para minerar o valor dos dados está cada vez mais relacionado com perceber onde é que uma determinada entidade está inserida num determinado sistema e como é que ela se relaciona com entidades terceiras. É neste contexto em que os *graph systems* ganham uma importância diferenciada, algo que podemos entender melhor na próxima secção.

2.2.2 Definição

A área dos *graph systems* pode ser analisada em 3 subdomínios [21]:

- **Application:** Corresponde à produtização dos *graph systems*, são aplicações cujo domínio é um grafo — como, por exemplo, uma rede social ou uma rede de estradas num sistema de GPS.
- **Computation:** Corresponde ao desenvolvimento de modelos e algoritmos para processar eficientemente os grafos. Aqui podemos subdividir em 2 subdomínios, tendo em conta o tipo de processamento, (i) *offline* and (ii) *online graph analytics*.
- **Storage:** Corresponde às estratégias usadas para guardar os grafos para rápido acesso — como, por exemplo, uma base de dados em grafo.

Relativamente ao domínio **Computation**, como referido, existem 2 tipos de processamento, o processamento *online* e *offline*, ou como é referido na bibliografia *Offline Graph Analytics* e *Online Graph Analytics*, respetivamente. Relativamente *Offline Graph Analytics* a computação é feita em *batch jobs*. Isto significa que o modelo de dados é lido da memória de forma estática e são efetuadas operações bastante pesadas sobre este, exemplos presentes na bibliografia podem ser Pegasus², Hama³ e Giraph⁴. Relativamente *Online Graph Analytics* o processamento dos grafos é feito dinamicamente, pois a estrutura dos grafos é alterada por operações de delete, update ou insert, um exemplo disso é, por exemplo, as conexões de um utilizador no LinkedIn, a mesma query utilizada para identificar quais as conexões de um determinado utilizador tem de ser executada várias vezes, devido à constante alteração da rede do utilizador, portanto, neste tipo de aplicações critérios como a latência de resposta é muito importante, exemplo destas aplicações são, por exemplo, Acacia⁵, Neo4j⁶, OrientDB⁷, Titan⁸, TigerGraph⁹, dois quais o Neo4j, Acacia e o TigerGraph são sistemas nativos de storage em grafo — *native graph database*. Ainda sobre *Online Graph Analytics* podemos dividir em suporte para Online Transaction Processing (OLTP) e Online Analytics Processing (OLAP). Os sistemas OLTP tem foco em operações transacionais sobre grafos de menor dimensão, enquanto os sistemas OLAP são otimizados para operações mais caras, do ponto de vista computacional, com uma perspetiva global dos grafos, e para grafos de maior dimensão [83]. Ainda sobre os sistemas OLAP podemos afirmar que os sistemas têm melhor preparação para suportar escalabilidade tanto horizontal como vertical [33].

Uma **Graph Database** pode ser definida, formalmente, como um SGBD onde os modelos de dados definidos, o seu schema e as instâncias são modeladas em forma de grafo, ou generalizações

²<https://db.cs.cmu.edu/projects/pegasus/>

³<https://hama.apache.org/>

⁴<https://giraph.apache.org/>

⁵<https://github.com/miyurud/Acacia>

⁶<https://github.com/neo4j/neo4j>

⁷<https://www.orientdb.org/docs/3.0.x/misc/Overview.html>

⁸<https://titan.thinkaurelius.com/>

⁹<https://www.tigergraph.com/>



Figura 2.10: Schema para o exemplo de informação modelada numa *Graph Database*

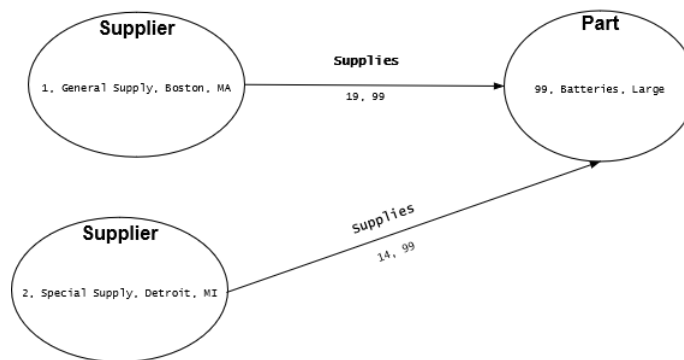


Figura 2.11: Exemplo de informação modelada numa *Graph Database*

do mesmo, e a manipulação dos dados é expressa em operações orientadas a grafos [5]. Um conceito chave do sistema é a estrutura de dados “grafo”. O grafo relaciona a informação de instâncias em *nodes* e *edges*. Os *nodes* representam as entidades do sistema, enquanto que os *edges* a relação entre estas. Um exemplo pode ser verificado pela figura 2.11.

Com o auxílio da figura 2.11 podemos definir uma *graph database*, de onde podemos retirar três principais características:

- **Schema em grafo:** A informação é representada via grafos, ou generalizações do mesmo (*hypergraph* ou *hypernodes*);
- **Data manipulation via operações sobre grafos:** A informação é manipulada via conceitos como *Path*, *neighborhoods*, *sub-graphs*, *connectivity*, *diameter* e *centrality*;
- **Consistência e integridade do modelo:** Labels com nomes únicos.

Para garantir estas propriedades existem três modelos de grafos. O **modelo tradicional**, onde o grafo é modelado com formalismos matemáticos, e mapeado para memória para uma *Adjacency matrix* (AJ), *Adjacency List* (AL) ou *Compressed Sparse Row* (CSR), cuja utilização é dependente do caso de uso [15] [17]. Um exemplo de representação pode ser verificado pela figura 2.12

Este tipo de abordagem não é a mais utilizada nas implementações de sistemas de bases de dados, pois não tem funcionalidades necessárias para suportar cenários do mundo real, ligados à Indústria, [7], com a célebre exceção de fazer *queries* baseados na centralidade do grafo [9]. O **Labeled-Property Graph (LPG)**, referido na bibliografia também por *property graph model*, onde o grafo é representado por um conjunto de *nodes*, *relations*, *properties* e *labels*. Tanto os *nodes* como as *relations* podem ter de agregar informação, um exemplo pode ser o representado

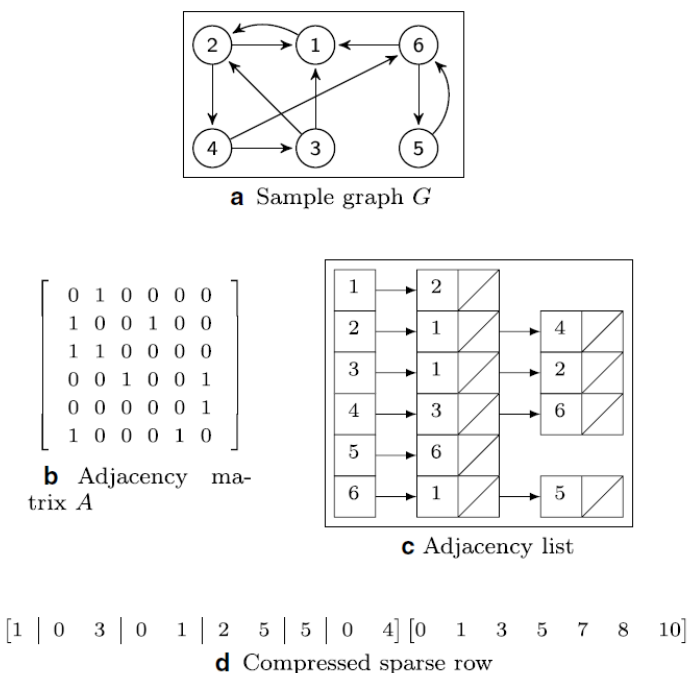


Figura 2.12: Representações do modelo tradicional. Adaptado de: Coimbra, M.E., Francisco, A.P., Veiga, L. "An analysis of the graph processing landscape"

pela figura 2.11. O **Resource Description Framework (RDF)** [44] uma recomendação da World Wide Web Consortium (W3C), com origens na forma de representar meta-data na web, cuja última versão foi atualizada em 2014 [18]. O objetivo da RDF é formalizar um mecanismo para descrever recursos, permitindo a troca de informação entre formatos diferentes de dados. Uma das vantagens é de criar relações entre recursos de forma modelar, criando assim uma estrutura em grafo. Uma expressão RDF cria um tuplo de três entradas, um *object*, um *subject* e a relação entre os dois — a *relation* [5]. Um exemplo de uma modelação RDF pode ser observado na figura 2.13. Relativamente aos 2 últimos modelos, aquele que é mais comum nas implementações é o LPG [65].

Relativamente ao domínio **Storage** existem duas estratégias de implementação de *graph databases*, as *native graph databases* e as *non-native graph databases*. As **native graph databases**, cujo design foi efetuado exclusivamente para manter e efetuar query de grafos, podem ser baseadas no modelo LGP ou RDF, referidos anteriormente. Uma característica importante das *native graph databases* é o facto de possuírem a propriedade de *index-free adjacency*. Esta propriedade é o que lhes permite serem mais eficientes a representar aquilo que são as relações entre as entidades [65] [67], quando comparada com outras bases de dados. A propriedade de *index-free adjacency* é definida pelo facto de cada *node* ter referência direta dos *nodes* adjacentes, ou uma *adjacency list* como vimos anteriormente. Por referência direta não estamos a falar de chaves de procura, como em outros modelos, mas sim por referências a espaços de memória dos *adjacent nodes*, funcionando assim como um *index* para *nodes* terceiros. Esta propriedade leva a que o tempo de

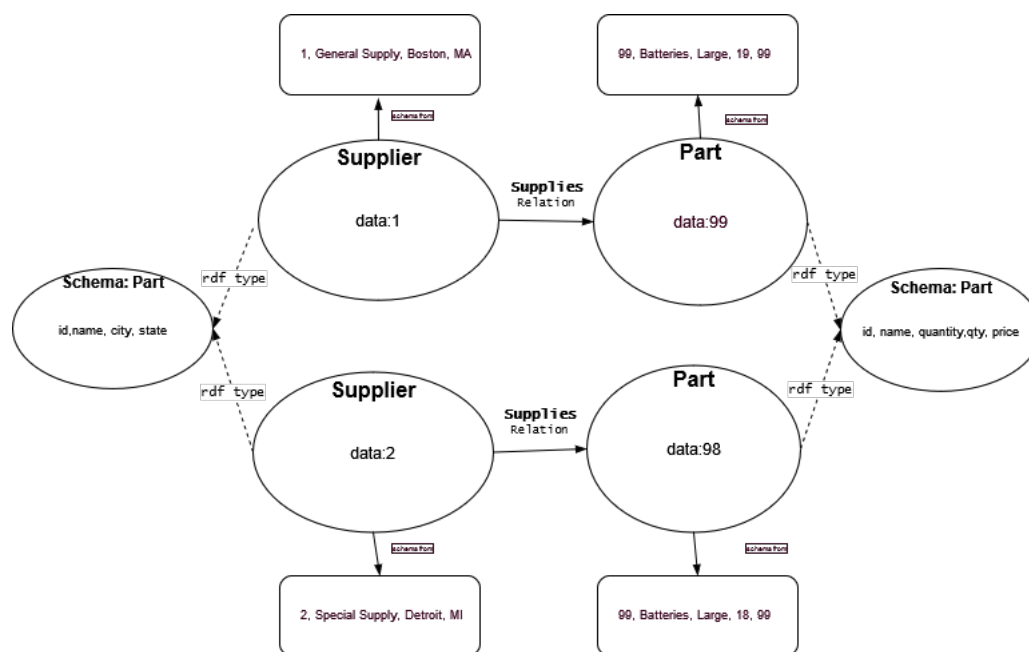


Figura 2.13: Exemplo de informação modelada numa *Graph Database* — Modelo RDF

query seja, praticamente, independente do tamanho do grafo [68]. Isto faz com que operações equivalentes ao “*JOIN*” no modelo relacional fiquem, a nível computacional, mais baratas [20]. Este é o principal argumento para o uso *native graph databases*, no entanto, isto levanta problemas relativamente a propriedades de distribuição do sistema, bem como existem trabalhos recentes que afirmam que esta estratégia apenas faz sentido para quando estamos a pesquisar informação de forma regional, *regional graph traversing*, e não global [45]. As *non-native graph databases* são caracterizados pela característica do sistema de *storage* não ser nativamente um grafo, apenas a interação com a mesma (query) e a modelação dos dados é baseado em grafos, como, por exemplo, sistemas de key-value pairs, documents e relacional [7]. O racional destes sistemas pode ser analisando por várias Surveys, como, por exemplo, [72] [13]. Tendo isto em conta, o processo de discussão destes sistemas será omissos, e vamos apenas analisar os pontos-chave de cada estratégia e como é que o modelo de grafo é estruturado em diferentes implementações, com base em [7]:

- **Key-Value Pairs:** Neste tipo de implementações a informação é armazenada via pares key-value. Esta estratégia é reconhecida por ser bastante performant e eficiente na procura de keys [72]. Existem os seguintes sistemas a usar esta estratégia: DGraph ¹⁰ e o HyperGraphDB ¹¹. Neste caso, o *node* do grafo seria representado por uma key e a vizinhança do mesmo pelo value.
- **Documents:** Uma document database é constituída por documentos — unidade base do sistema. Este tipo de sistema é uma extensão das key-value database onde o value é um

¹⁰<https://dgraph.io/>

¹¹<http://www.hypergraphdb.org/>

documento, normalmente estruturado em JSON [72]. Um exemplo é a ArangoDB ¹². Neste caso, o *node* do grafo é guardado num documento de *nodes*, enquanto os *edges* podem ser guardados, ou no documento do *node* origem, ou no *node* destino.

- **Relacional:** O sistema relacional é constituído por tabelas e relações entre as mesmas. Cada tabela é constituída por *rows* e *column*. As tabelas representam as entidades do sistema, as *rows* as instâncias de cada entidade e as *columns* uma determinada propriedade dessa instância. Existem várias implementações a usar esta estratégia, especialmente através de *syntatic layers*, descritos na secção 2.2.3. Um grafo pode ser representado como 2 tabelas, uma de *nodes* e outra de *columns*, cada uma com uma Primary Key (PK). Os *edges* fazem referência entre os *nodes* origem e destino, pelas suas PK.

Ao contrário do modelo relacional que utiliza o SQL como interface para acesso e manipulação de dados, os *graph systems* não possuem uma interface estandardizada existem, portanto, várias implementações, das quais as mais populares são [14]:

- **Cypher:** É uma linguagem declarativa para grafos com o modelo LPG, criada pelo Neo4J. Por ser declarativa, permite a existência de um otimizador que escolhe a melhor estratégia para otimizar a operação. Atualmente encontra-se implementada em outros sistemas como: SAP HANA Graph, Redis Graph, Agens Graph e Memgraph. Isto implica que seja a linguagem mais popular [30].
- **SPARQL:** É uma linguagem declarativa inspirada em SQL criada pela W3C para modelos RDF.
- **Gremlin:** É uma DSL (Domain Specific Language) para manipulação de grafos, que tanto pode ser imperativa como declarativa.

O facto de não existir um standard torna o mundo das GDB (Graph Databases) mais complexo, como tal, existiu um esforço em 2018 de duas organizações, o International Organization for Standardization (ISO) e o International Electrotechnical Commission (IEC), a para criar um standart o Graph Query Language (GQL) [4] [23]. GQL é inspirado no Cypher, no PGQL, da Oracle, e no GCORE, do Linked Data Benchmark Council [LDBC], apesar disto a sua utilização ainda não é uniforme nos sistemas GDB.

2.2.3 Processo de seleção

Explorando com mais profundidade a ideia descrita em secções anteriores de “Para diferentes problemas diferentes soluções” como é que sei se estou perante um problema de grafos? Por um lado, é verdade que qualquer domínio de problema pode ser representado por um grafo, por outro lado, é possível que existam estratégias mais adequadas. Respondida a esta pergunta, que processos — *Exploration based* — existem para escolher o SGBD?

¹²<https://www.arangodb.com/>

Para responder a este problema existem várias abordagens, com passos diferentes, mas com características muito semelhantes. Denise et al [22] tenta responder a esta pergunta propondo um conjunto de passos para tomar esta decisão:

- **Does your problem need graph data?:** A primeira questão está relacionada com o formato da informação que queremos modelar. Se para concretizar o objectivo do sistema só quero saber uma determinada propriedade de uma determinada instância, por exemplo, o nome e a idade de uma determinada pessoa, devo utilizar o modelo relacional. Se só quero pesquisar a secção e página neste documento que me fala sobre os tipos de *graph databases*, devo utilizar uma document database. Se, por outro lado, estou interessado em analisar uma determinada instância e o seu contexto/relações o processo de *graph thinking* é iniciado.
- **What are you going to do with the relationships in your data?:** A segunda questão está relacionada com a finalidade do sistema e do uso do grafo. Utilizarei o sistema para realizar análise de dados? ou para gerir os dados? Se for para gerir os dados, devo utilizar uma *graph database*. Se for para analisar dados devo prosseguir para o seguinte ponto.
- **What do you need the results for?:** A terceira questão está relacionado com a finalidade dos dados gerados pelo sistema de análise. A finalidade é construir reports? É realizar um pré-processamento do grafo? Dependendo do caso terei usos diferentes.

Loureço et al. [48], faz um estudo interessante no processo de escolha de bases de dados NoSQL, catalogando as bases de dados segundo o *CAP Theorem*, *consistency*, *durability*, *querying*, *concurrency control* e *partitioning scheme*, mas deixa de fora as *native graph databases*. Hajoui et al. [35] faz uma comparação entre diferentes tipos de bases de dados, usando um método de classificação pesado que considera critérios como *performance*, *integrity*, *reliability*, *interoperability*, *cloud support*, *query complexity* and *security*.

Para Khazaei et al. [42] estrutura o processo de escolha em três passos preliminares e um final. Os três passos preliminares consistem em ir reduzindo sucessivamente o número de sistemas a considerar. Estes três passos consistem em: (i) *Data Model and access pattern*, (ii) *Query Requirements* e (iii) *Non-functional properties*. O passo final consiste em realizar *Benchmark* dos sistemas restantes e tomar a decisão sobre o sistema mais indicado — (iv) *Performance Evaluation*.

- **(i) Data Model and access pattern:** O primeiro passo é perceber como está estruturada a informação, se a informação tiver características relacionais deverá ser empregue o modelo relacional, no entanto, se propriedades como *horizontal scalability* forem fundamentais, devemos optar por um sistema NewSQL. Relativamente a outro tipo de dados, como Column oriented, document-like, graph, object, multi-values, JSON e BLOBS, devem ser consideradas soluções NoSQL. Para escolher um determinado sistema NoSQL, temos de considerar características como *volume*, *complexity*, *schema flexibility*, *durability* e *data access* para diminuir os sistemas concorrentes, de modo geral podemos seguir o gráfico da Figura 2.15

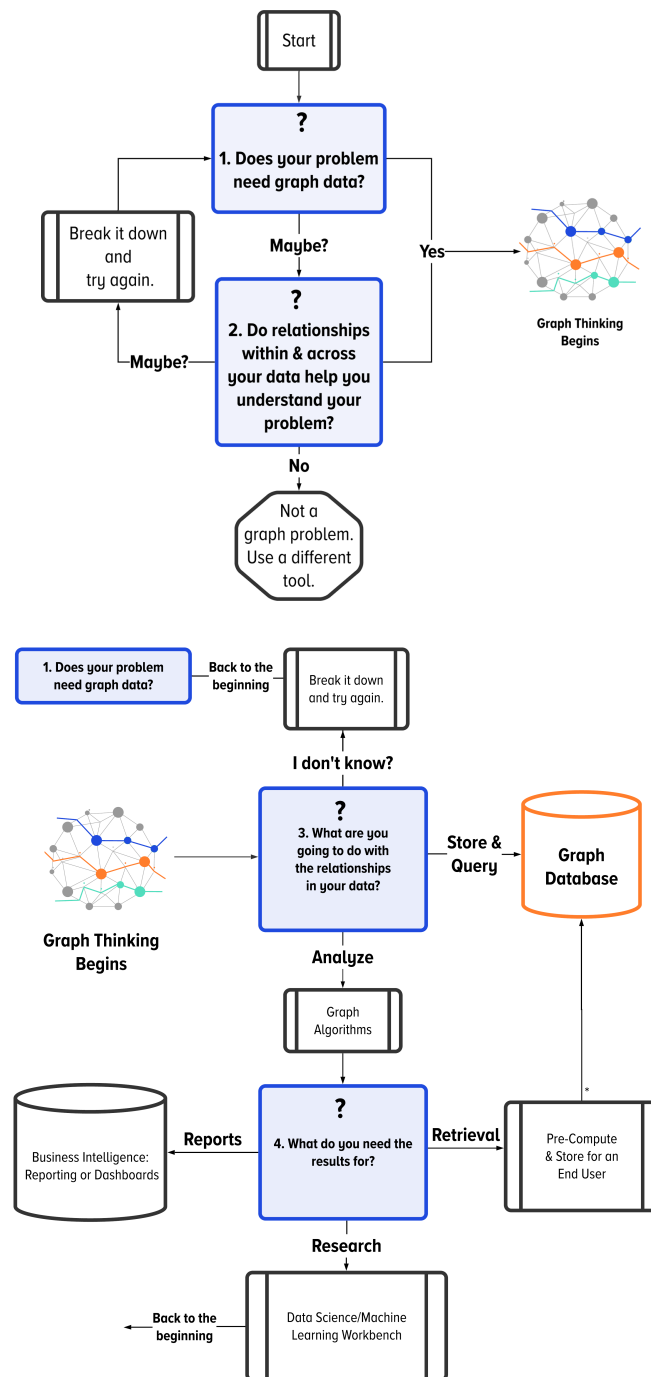


Figura 2.14: Fluxograma para escolher Graph Databases. Adaptado de: D. G. Denise and B. Matthias, The Practitioner's Guide to Graph Data [Book]

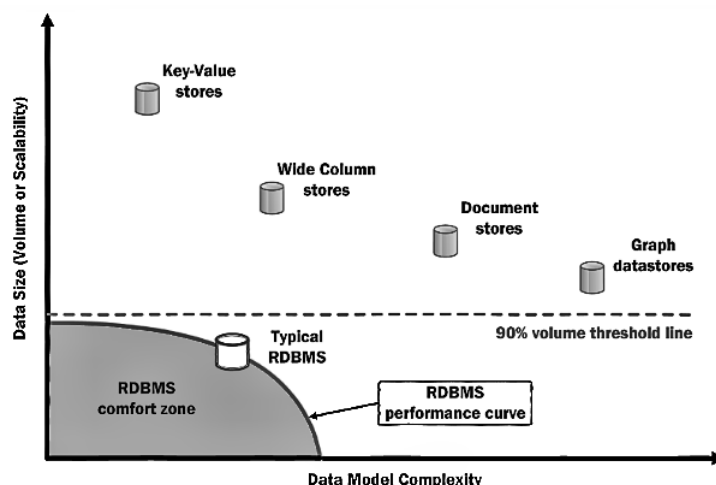


Figura 2.15: Distribuição dos modelos com base no par — Volume-Complexidade dos dados [42]

- **(ii) Query Requirements:** Outro facto a considerar são os requisitos de *query*. Que tipo de questões queremos fazer ao sistema? Que tipo de perfil terá a interação com o sistema? Que tipo de interface quero que o sistema tenha?
- **(iii) Non-functional properties:** Aqui são considerados um grande número de requisitos, dos quais o mais importante é a desempenho do sistema, traduzido em latência (de *read*, *write* e *update*) e *throughput* — número de operações por unidade de tempo. O desempenho depende de outras propriedades não funcionais, como, por exemplo, o *partitioning*, *replication*, *horizontal scalability*, *load balancing*, *auto-scalability* e *concurrency mechanisms*.
- **(iv) Performance Evaluation:** Aqui é onde se toma a decisão final. Com o conjunto de passos anteriores fomos reduzindo o número de tecnologias a considerar, neste passo temos de aplicar as tecnologias que sobram e aplicar ao problema que temos em mãos. Neste exemplo foram considerados os seguintes indicadores: latência das operações de *read*, *write*, *update* consoante o *throughput*.

Roy-Hubara et al. [70], [71] propõe uma estratégia semelhante, com a análise de requisitos funcionais e não funcionais, com a diferença que suporta a sua decisão com métodos de apoio à decisão como o Analytic Hierarchy Process.

Guzman et al. [34] propõe uma técnica parecida com Denise et al [22], composta por 5 fases:

- **(i) Problem Analysis:** O primeiro ponto consiste em definir os objetivos do problema e se os SGBD satisfazem os mesmos, por exemplo, se as SGBD satisfazem os requisitos de volume de processamento, filtros e algoritmos.
- **(ii) Requirements Analysis:** Com base nos objetivos definidos, definir um conjunto de requisitos, qualitativos e quantitativos, que podem ser de 3 tipos: *performance*, *flexibility* e

agility. Para além disso, temos de definir métricas associadas aos requisitos. Para o caso foram escolhidos atributos de performance a executar determinados algoritmos e suporte de determinadas *query languages*.

- **(iii) GDB Pre-selection:** Este ponto consiste em consultar/construir um catálogo de bases de dados, e verificar quais é que respeitam as propriedades definidas no ponto anterior.
- **(iv) Benchmarking:** Este ponto consiste em relacionar as métricas escolhidas com as bases de dados seleccionadas.
- **(v) SGDB Selection:** Este ponto está relacionado com a tomada de decisão baseada nas avaliações feitas. O SGDB a escolher é a que obtiver a pontuação mais favorável.

2.2.4 Comparação de implementações

Independentemente do processo definido para escolher o melhor SGBD para um determinado problema, um passo inevitável é perceber como comparar diferentes implementações e perceber qual é a que mais se relaciona com o nosso problema. Existem benchmarks que se classificam como standard, mas não existe consenso sobre o standard. Exemplos disso são o Linked Data Benchmark Council (LDBC) [3] e o HPC Scalable Graph Analysis Benchmark [24].

Fernandes et al. [28], descreve de alto nível as tecnologias AllegroGraph, ArangoDB, InfiniteGraph, Neo4j, and OrientDB, quanto a propriedades como *Schema flexibility*, *Query Language*, *Sharding*, *Backups*, *Multi-model*, *Multi-architecture*, *Scalability*, e *Cloud Ready*. Para cada propriedade classificou conforme uma determinada escala. Os resultados finais reforçam a ideia que não existe um sistema melhor que os outros, apenas sistemas mais indicados para determinados problemas. De grosso modo o Neo4J e a ArangoDB respondem à maior parte dos problemas de grafos, no entanto, se estivermos perante um problema de sistemas altamente distribuídos pode não ser a melhor solução por não possuir a capacidade de sharding.

P. Cailliau et al. [12] faz um estudo de performance comparativo de diferentes tecnologias, das quais RedisGraph, TigerGraph, Neo4J, Neptune JanusGraph e ArangoDB. Para tal recorre a 2 datasets, um gerado a partir do Graph500 com 41.6 milhões de nós e outro com origem no Twitter com 2.4 milhões nós. Como resultado da investigação o autor observa que o RedisGraph é entre 0.8 a 2 vezes mais rápido que os restantes sistemas relativamente ao tempo de resposta numa query de travessia de vizinhança.

Jouili et al [39] faz uma comparação de performance propondo um sistema de benchmarking que implementa datasets populares para testar grafos, e faz uma análise de algoritmos, como o shortest path, usando as tecnologias: Neo4j, DEX, Titan (BerkeleyDB e Cassandra) e OrientDB. A conclusão deste estudo foi: (i) Para workloads de travessia de grafos, o Neo4j tem melhor performance; (ii) para workloads de Read-Only Neo4j, DEX, Titan-BerkeleyDB e OrientDB têm resultados semelhantes. (iii) Para workloads de escrita, as implementações Neo4j, Titan-BerkeleyDB e OrientDB a performance é pior, enquanto DEX e Cassandra é melhor.

Para além deste tipo de análise crivadas pelo processo científico, existem *benchmarks* fornecidos pelas empresas que vendem/representam as diferentes *graph databases*, que como é óbvio afirmam que a sua implementação é superior para a maior parte dos casos [66] [82] [86], [81], [85]. Apesar de explicarem o processo de *benchmarking* e apresentarem os *datasets* utilizados não passaram pelo processo de *peer-review*, no caso em específico [66] há comparações são pouco meticulosas em que se afirma que o tempo de *insert* é inferior no *DGraph* quando comparado com o *Neo4j* quando o dataset está no formato RDF que não é suportado nativamente pelo Neo4j, outro ponto a realçar é o facto da versão usada Neo4J não ser a mais atual, considerando a altura da experiência.

Macko et al. [49] propõe PIG (Performance Introspection for Graph Databases) uma metodologia de comparação de *graph databases* que considera operações primitivas de *graph databases*, como read, write e update, a que intitula de *microbenchmarks* e operações baseadas em algoritmos, a que intitula de *macrobenchmarks* para 3 grupos de datasets diferentes, baseado em domínios diferentes. Cada grupo é constituído por vários *datasets* de tamanhos diferentes. O processo é dividido em três níveis. O primeiro nível considera a latência de leitura escrita para operações-base, como GetVertex, GetEdge, AddVertex, AddEdge. O segundo nível está relacionado com operações fundamentais dos grafos, como, por exemplo, GetNeighbors ou um dataloader incremental/bulk. O nível 3 considera algoritmos de grafos, como o *ShortestPath*. Como demonstração são comparadas duas *graph databases*, DEX e Neo4J. Para o primeiro nível Neo4J tem excelentes resultados em *workflows* de leitura, especialmente quando o grafo tem uma dimensão que lhe permite encaixar na heap, por outro lado, por ser *ACID compliant*, os tempos de escrita são mais altos, quando comparado com o DEX. Relativamente ao nível dois, ao executar a operação *GetNeighbors* o Neo4J demonstra que para grafos pequenos (mil *nodes*) e que para grafos grandes (10 milhões de *nodes*) o tempo de reposta é idêntico, independentemente do tamanho da vizinhança. Ao nível de bulk/incremental insert DEX tem resultados bastante melhores, algo confirmado pelo nível anterior. No terceiro nível, para a execução do *ShortestPath*, para datasets menores o Neo4J tem melhor performance, para datasets maiores os resultados são equivalentes.

Chen et al. [14] afirma que quando o domínio do problema exige tratar um grande número de relações, num sistema relacional, por cada relação um JOIN terá de ser realizado e isto causa um grande *overhead*. Isto piora quando entramos em sistemas/domínios com entidades altamente relacionadas, onde as relações entre as entidades do sistema crescem exponencialmente. Como as *graph databases* são baseados no modelo em grafo, e não tabular, isto faz com que a integração em sistemas orientados a objetos seja mais fácil e mais intuitiva. As *graph databases* também têm menos restrições de *schema* e melhores níveis de performance em operações de *JOIN*. Neste artigo o autor compara a performance de um sistema relacional e de uma *native graph database* para um problema de grafos — uma rede de transportes. O autor utiliza 2 *datasets*, um com três mil *edges* e outro com cem mil *edges*. Ao nível de Load inicial da informação o sistema relacional foi mais rápido. Ao nível de execução de algoritmos o Neo4J foi mais rápido que o sistema relacional, na ordem dos 30%. Estes resultados permitiram ao autor concluir que para este problema as *native graph database* eram mais apropriadas. Estudos semelhantes a este são fáceis de encontrar,

pois, este é o principal argumento de venda das *native graph database*, outro exemplo poderá ser [1] compara o Neo4J com MySQL, e quantifica o tempo de resposta para diferentes *queries* em problemas com centenas de *nodes*, onde as *graph databases* também apresentam melhor performance.

Jing Fan et al. [27] propõe *Grail*, um *syntactic layer* para fazer queries em grafo, sobre sistemas relacionais. O autor tenta desconstruir ideias anteriores de que *graph systems* são benéficos em *workflows de analytics*, onde o propósito é executar algoritmos orientados a grafos. O principal argumento é que os sistemas relacionais são tão robustos, maduros, têm um ecossistema tão completo que não deve ser desconsiderado. Um outro argumento importante é que num ecossistema empresarial, quantas menos tecnologias forem suportadas, mais barato será o custo do sistema. *Grail* é uma linguagem que expõe uma interface orientada a grafos sobre SQL, que torna a manipulação de grafos mais eficiente e mais *developer friendly*. Ao nível de resultados podemos resumir que, existem casos onde os *specialized engines* são mais *performant*, mas para grandes *datasets* a *performance* é muito parecida. Para *Datasets* na ordem dos 100 milhões de *nodes* e 3.3 mil milhões de *edges* só o *Grail* é que conseguir reproduzir resultados a este nível de exigência ao nível de memória, o que mostra a robustez dos sistemas relacionais. Este artigo põe em causa a especialização de *online graph analytic engines*, como o Giraph e o GraphLab. Este é um dos artigos mais críticos sobre este tipo de sistema, mas não considera como é que o Giraph e o GraphLab se comportam em nível de escalabilidade. Guo et al. [33] demonstra que este tipo de plataformas, para grandes grafos, têm a possibilidade de escalar tanto em nível de *Cores* (verticalmente) como em nível de clusters (horizontalmente) o que lhes permite ter um ganho de performance significativo.

Sun et. al [79] defende um ponto bastante parecido. Afirma que os sistemas relacionais têm uma grande vantagem sobre os modelos NoSQL, no sentido em que são ACID compliant, têm eficácia comprovada a nível Industrial em questões como segurança, concorrência e otimização. Bornea et. al [8] mostra que é possível ter performance comparável com *native graph databases* usando um sistema relacional com a seguinte ideia: a lista de adjacência deve ser guardada na mesma row que o nó — algo parecido com a estratégia *index free-adjacency* das *native graph databases*. O autor compara a performance da sua implementação, SQLGraph, com a performance de duas graph databases, o Titan e o Neo4j. Usou duas estratégias de benchmarking DBPedia [57] um benchmark execução de determinados algoritmos para RDF graphs, LinkBench [6] uma benchmark proposta pela Meta para simular *workloads* de redes sociais, com *updates* frequentes. Para o primeiro dataset o SQLGraph tem resultados superiores quando comparados com as outras bases de dados. Para o segundo dataset o SQLGraph é melhor em todas as operações, como, por exemplo, adicionar/remover/update um novo *node*, retornar uma vizinhança de um nó. Um comentário que se pode fazer sobre este artigo é o facto de o Neo4J ter uma performance comprovada superior quando o modelo utilizado é o LGP e não o RDF [28] [62].

Pascaci et al. [62] propõe uma arquitetura para realizar *benchmarking* de processamento de transações em tempo real, que, ao contrário dos estudos anteriores, tenta representar melhor aquilo que é um *workflow* normal do uso de uma base de dados, usando um sistema distribuído de *event-*

streaming. Neste estudo são utilizados como SUT (*System under test*) uma *native graph database*, um *graph engine*, um sistema multimodelo, um sistema relacional e um sistema relacional com uma interface orientada a grafos, ao nível de modelação dos dados não é apresentada informação. O estudo é dividido em duas fases, a primeira fase consiste em avaliar o tempo de leitura dos sistemas, a segunda fase consiste em avaliar o número de operações suportadas quando o sistema está sobre carga com acessos concorrentes gerados pelo sistema distribuído de *event-streaming*. Numa primeira fase é avaliada a performance de leitura usando 4 tipos de *queries* a grafos: (i) *point lookups*, (ii) *one-hop traversal*, (iii) *two-hop traversal*, (iv) *shortest path*. Dos resultados obtidos podemos afirmar que o Neo4J é mais performant quando é utilizado a sua interface nativa CYPHER, e que não existe uma relação direta entre o tempo de execução da *query* e o tamanho do *dataset*, o que comprova a efetividade da propriedade *index free adjacency*, por outro lado, os sistemas multimodelo e relacionais apresentam uma performance superior a todos os outros sistemas, com uma ligeira degradação de performance para *datasets* maiores. Relativamente ao sistema relacional com interface orientada a grafos o seu desempenho foi o pior em todos os casos. Este facto poderá estar relacionado com o facto de não ser usada a estratégia descrita por Sun et. al [79] e Bornea et. al [8] referida anteriormente. Numa segunda fase foi simulado um *workflow* de processamento em tempo real, usando a arquitetura proposta. Ao nível de resultados podemos verificar que a *native graph database* tem melhor performance para *workflows* de leitura, suporta um maior número de leituras em simultâneo, e os sistemas relacionais para *workflows* de escrita, suportam um maior número de escritas em simultâneo. O Autor afirma que a nível *performance overall* a *native graph database* apresenta melhores resultados, seguido do sistema relacional e do sistema multimodelo, o que mostra que os últimos não devem ser ignorados para este tipo de problemas.

2.2.5 Casos de Uso

Como referido anteriormente, o uso de *graph systems* é otimizado quando o problema que estamos a resolver é um problema de grafos, ou seja, estamos perante domínio altamente acoplado e coeso. Exemplos disso são aplicações no campo de Energy Management Systems [47] (EMS), Power Grid Modeling [37], campo biomédico que é descrito extensivamente com a survey de Timón-Reina et al. [83], campo de redes de estradas [14] e claro o caso das redes sociais [55], [10]. Relativamente ao domínio de *Manufacturing e Indústria 4.0*, nos últimos 5 anos, podemos realçar os seguintes trabalhos:

Martinez-Gil et al. [52] propõe uma solução baseada em grafos para fazer *Tracking* de Produtos. Segundo o autor, um dos principais problemas na Indústria de *manufacturing* consiste no facto de ser muito difícil de fazer o *tracking* de um material ao longo da sua vida na fábrica, devido a sucessivas misturas, com outros materiais, que podem ser respetivamente compostos por outros materiais, em diferentes passos. A possibilidade de fazer *drill-down* de um material inicial até a um produto final é uma funcionalidade crucial para os *end-users*, esta funcionalidade é suportada nos dias de hoje por um sistema relacional, que segundo o autor tem dois problemas: (i) os tempos de resposta são muito demorados, (ii): a análise de dados não é direta. Para resolver este problema

o autor cria uma solução baseada em graph databases, que segundo o mesmo otimiza o tempo de resposta. Para resolver o problema o autor utiliza OrientDB, uma Graph Database (GDB) com um tipo de storage orientada a documentos. Ao nível de modelação representa como *nodes* os materiais e os *edges* informam sobre a hierarquia e precedência de montagem. Para responder ao requisito de drill-down utiliza o algoritmo Breadth-First Search (BFS). Para concluir o autor afirma que a sua abordagem é mais eficiente e que permite uma visualização de dados superior. Martinez-Gil et al. [53] propõe em 2020 uma extensão do seu trabalho, desta vez tendo em consideração diferentes dependências dos materiais, como, por exemplo, a sua evolução e a sua distribuição. Comparado este trabalho com o anterior podemos afirmar que o acréscimo de funcionalidade não levou a um grande esforço de desenvolvimento, o que comprova a praticabilidade e agilidade das *graph databases*. Numa perspetiva crítica podemos afirmar que o autor consegue com a sua solução, por um lado, melhorar a visualização dos dados ao usar uma graph database, no entanto, por outro lado, afirma que a sua abordagem é superior, mas não mostra dados que suportem a sua afirmação. Relativamente ao processo de escolha da *graph database* a abordagem também não foi explicada, bem como o porquê do SGBD escolhido. Por termos ignorado esta etapa perdemos a oportunidade de considerar as *native graph databases*, que podiam ter resultados mais animadores devido à sua propriedade de *index free adjecency*.

Kim et al. [43] propõe uma ferramenta para integrar dados heterogéneos num único *datasource* — um knowledge graph — para facilitar a pesquisa. Segundo o autor, a informação relativamente a *manufacturing* é heterogénea tanto ao nível de formato como ao nível de estratégia de persistência, e diferentes informações podem estar guardados em sistemas diferentes e estarem relacionadas. Atualmente a informação é guardada em sistemas de informação, como um MES ou um ERP, ou em formatados não estruturados, como, por exemplo, editores de texto, *spreadsheets* e e-mails. Esta informação pode estar relacionada e é gerada por diferentes utilizadores, com diferentes formatos e muitas vezes com vocabulário e linguagens diferentes. Segundo o autor, um operador fabril perde cerca de 19% do seu tempo à procura de informação, levando à necessidade de uniformizar o acesso à informação. Para responder a este requisito o autor propõe I-DATAQUEST, um sistema de query baseado numa GDB, o Neo4J, que permite a pesquisa de informação em fontes de dados heterogéneas. Um ponto a realçar do trabalho foi a estratégia de construção do grafo. Para converter a structured-data, modelada em modelo relacional, o autor converteu cada tabela em um *node*, e cada relação entre entidades como um *edge*. Para converter a informação não estruturada, cada documento, e-mail, item era convertido em *node* e as referências para *items* terceiros eram convertidas em *edges*.

2.3 Discussão

O corpo de texto do estado da arte é iniciado com um contexto da quarta revolução Industrial e da Indústria 4.0 onde percebemos que desde a sua conceção até agora tanto o mundo da academia como a indústria a tentam concretizar [2]. Historicamente verificamos que as revoluções anteriores foram positivamente disruptivas no que toca ao sector tecnológico e no que toca ao sector social

e económico. Surge assim uma grande oportunidade, mas também um grande obstáculo, para a sociedade, para a indústria, e em particular para as empresas que desenvolvem *software* de apoio ao processo de *manufacturing*. Como vimos anteriormente, com a visão de Indústria 4.0 vêm também novos requisitos relacionados com os dados, descritos pelos 5 *V's de Big Data*. Destes novos requisitos surge a necessidade dos sistemas existentes se adaptarem, o que leva à procura de novas estratégias de resolver os problemas.

O corpo de texto das *Graph Databases*, surge exactamente como estratégia inovadora de resolução de problemas. Esta secção é iniciada por uma contextualização histórica, o que faz todo sentido, porque vemos algumas conclusões adquiridas no passado que têm impacto nos sistemas e decisões de hoje. Logo no início, ao comparar o modelo hierárquico com o modelo *network*, percebemos que quando modelamos a informação com estruturas complexas, no caso um grafo, isso tem prós e contras, no caso um contra referido foi o tempo de upload ser maior, algo que volta a ser confirmado mais tarde como descrito na secção 2.2.4, onde verificamos que as *graph databases* têm melhor performance em *workflows* de leitura, quando comparado com *workflows* de escrita. Logo de seguida, no início da era relacional, é referido que uma das principais vantagens do modelo relacional é o conceito de abstração entre o modelo físico e o modelo conceptual, algo que hoje é um dado adquirido. A mesma lógica podemos estender para aquilo que são as *queries languages*. Quanto maior ao nível de abstração, melhor. O que significa que as linguagens declarativas são superiores na maior parte dos casos, esse facto volta a ser posto em causa na secção 2.2.2 onde são descritos vários tipos de linguagens de query a grafos, e o que vemos é uma proliferação das *graph query languages* declarativas como o CYPHER. Ainda sobre as *queries languages* podemos verificar a tendência para a padronização de acesso a *graph databases*, algo que já aconteceu no passado, como é referido no fim da descrição da era pré-relacional, e como aconteceu com o SQL na era relacional. Com a contextualização histórica percebemos também aquilo que foram as tendências ao longo dos anos, observamos um *paradigm-shift* de onde passamos de tentar otimizar o *storage* para otimizar propriedades de escalabilidade e *availability*, para, por fim, tentar extrair o máximo de valor dos dados, apesar deste *paradigm-shift* observamos que algumas das ideias apresentadas continuam a fazer sentido:

- **Importância da modelação:** A modelação tem impacto na performance dos sistemas, bem como na sua manutenção.
- **“Para problemas diferentes, diferentes soluções”:** Isto faz sentido a vários níveis de granularidade. A alto nível faz sentido na escolha de um SGBD, que deu origem às abordagens *Exploration based*, descrita na secção 2.2.3. A baixo nível faz sentido porque mesmo quando escolhemos a família SGBD, temos de proceder à comparação de tecnologias descrita na secção 2.2.4, porque como vimos, mesmo para problemas de grafos o uso de uma *graph database* pode não ser ótimo.

Um outro facto interessante a realçar é o crescimento no interesse por GDBs nos últimos anos. Isto está por um lado relacionado com o crescente interesse nos dados, mas também pelas propriedades descritas que fazem das *graph systems* uma ótima escolha, das quais:

- Facilidade de visualização de dados e mapeamento domínio-SGBD: Como verificamos nas secções 2.2.1, 2.2.5 o facto de os *graph systems* serem baseados em grafos, permite uma modelação mais natural dos dados, que se reflete também na possibilidade de os analisar mais facilmente, quando comparado com outros modelos de SGBD.
- Otimizadas para tratar dados altamente acoplados e coesos: Devido à sua arquitetura as relações entre as entidades são tratadas como *first-class citizens* [67]. No caso particular das *native graph databases* o facto de possuírem a propriedade de *index free adjacency* permite que trate as relações de forma mais eficiente que outras implementações.
- Graph Interface: Uma das propriedades dos *graph systems* é possuir uma interface otimizada para fazer *queries* a grafos, que como já vimos, independentemente da abordagem, permite otimizações e torna a manipulação de grafos mais *developer friendly*
- Schemaless: Uma das propriedades dos sistemas NoSQL, e, por isso, dos *graph systems* é o facto de não terem um *schema* fechado. Isto permite uma agilidade gigantesca quando precisamos de alterar o *schema* em versões diferentes do *software*, um excelente exemplo disso foi descrito na secção 2.2.5 no artigo [53]

Por outro lado, como não existem estratégias perfeitas, também podemos apontar:

- Standard Query Language: Não existe uma Standard Query Language proliferada nas diferentes implementações, o que é um grande obstáculo na complexidade do mundo das GDBs [62].
- Ecossistema pouco maduro: Quando comparado com o modelo relacional, podemos afirmar que os sistemas relacionais têm um ecossistema mais desenvolvido e são mais robustos, como podemos verificar na secção de 2.2.5.
- Escalabilidade: A maior parte das implementações de GDBs têm uma arquitetura *one-tier*, mas existem exceções como podemos verificar por [33]

Ao nível de processos de escolha de SGBD e de benchmarking, podemos debater que é dos pontos onde existe menos uniformização, o que é negativo, pois é impossível fazer uma escolha informada quando existem tantos pontos de variação. Um outro ponto a considerar é o aparecimento de novos *graph systems* que ainda não foram considerados pela bibliografia. Ao nível de bibliografia todos os processos possuem as seguintes etapas, mesmo que com nomenclatura e configurações diferentes:

- (i) Análise do problema: consiste em perceber o problema e descrevê-lo de maneira informal.
- (ii) Descrição dos Requisitos Funcionais e não Funcionais e respetivos indicadores: consiste em, com base no ponto anterior, enumerar os requisitos funcionais e não funcionais, bem como os indicadores que permitem a sua avaliação.

- (iii) Escolha dos Modelos e implementações: com base nos requisitos escolher os modelos que podem responder aos meus requisitos.
- (iv) Benchmarking: Consiste em, com base nas implementações escolhidas, calcular os indicadores de desempenho definidos no ponto (ii).
- (v) Escolha Final: Tomar a decisão. Este ponto pode variar, existem abordagens que usam métodos numéricos para tomar a decisão, existem abordagens que utilizam métodos de apoio a decisão e existem abordagens que utilizam métodos mais sensíveis.

Quando estamos perante um problema de grafos, um dos principais problemas está logo no terceiro ponto, a escolha do modelo. Como é que eu sei, sem avançar no processo, que o meu problema deve ser modelado por grafos? Porque devido à natureza desta estrutura de dados, é sempre possível. Algo que poderá ter impacto pode ser a tipologia do grafo do nosso problema, provavelmente, para grafos altamente densos, com *average degree* elevado, o uso de *graph databases* é um bom use-case, no entanto, isto nunca foi considerado na bibliografia referida. Relativamente ao quarto passo temos os seguintes problemas:

- Datasets: Para diferentes autores existem diferentes datasets considerados standard, alguns referem o Graph500 [33], outros o LDBC [3], outros o HPC [24].
- Need for standard Queries: Uma das queries que é sempre executada sobre graph databases é o *ShortestPath*, mas qual é a razão? Será por ser o primeiro algoritmo que pensamos quando falamos em grafos?

Numa perspetiva crítica podemos afirmar que é preciso perceber como é que tipologia do grafo, que pode ser descrita como o *average degree* e a *link density*, afeta a escolha de um determinado SGBD, precisamos de perceber como é que a modelação e o uso de estratégias como a desnormalização na pesquisa [22] podem afetar a escolha de um determinado SGBD e precisamos estratégias/datasets uniformes de benchmark bem como de um conjunto de queries uniformes para perceber a performance de um determinado sistema.

2.4 Conclusão

À medida que avançamos na construção da visão de Indústria 4.0, novos requisitos vão começar a aparecer que têm de ser suportados pelos sistemas de manufacturing. Estes requisitos estão relacionados, principalmente com os dados, com critérios de performance, mas acima de tudo com a extração de valor que conseguimos fazer dos dados. Para tal, sistemas como os Manufacturing Execution Systems (MES), que funcionam como a espinha dorsal da indústria, têm de inovar com novas estratégias e novas implementações que suportem tal disrupção. Como estratégia de disrupção surgem as *Graph Databases* que como vimos são soluções altamente otimizadas para domínios orientados a dados e altamente coesos. Por um lado podemos afirmar que, devido à natureza dos dados num sistema MES, onde uma entidade não tem sentido sozinha, mas é o contexto

da mesma que lhe traz valor, faz sentido perceber até que ponto o contexto de manufacturing pode ser otimizado pelas *graph databases*. Por outro lado, com este estudo bibliográfico percebemos que as *graph databases* não são a *silver bullet* para todos os problemas. Surge então a necessidade de empregar uma abordagem *Exploration Based* que terá como principal objetivo quantificar o grau de adequação das *graph databases* ao contexto de manufacturing, bem como responder às questões levantadas ao longo da secção de discussão (2.3).

Capítulo 3

Descrição do Problema

O presente capítulo tem a responsabilidade de descrever o problema que esta dissertação se propõe a resolver. A secção 3.1 tem o objetivo de descrever os conceitos de negócio do domínio *manufacturing* apresentado um modelo de domínio e descrevendo a interação entre as diferentes entidades do sistema. Em seguida a secção 3.2 descreve, tanto a nível físico como a nível lógico, a arquitetura atual do MES da Critical Manufacturing. Por fim a secção 3.3 integra os conceitos das secções anteriores e apresenta formalmente o problema que a presente dissertação se propõe a resolver.

3.1 Domínio

De forma a perceber o problema com melhor detalhe, temos de primeiro perceber o contexto do mesmo e o seu domínio. O processo de fabrico — *Manufacturing* — pode ser descrito em cinco conceitos:

- “Material”: representa os componentes de um processo de fabrico;
- “Product”: representa o *Material* no fim do processo de fabrico;
- “Production Process”: descreve os passos, ou *Steps*, do processo de fabrico. Os steps são aglomerados em *Flows*;
- “Equipement Paramters”: descreve o estado e configuração das máquinas — *Resources* — utilizadas nos *Steps* de fabrico;
- “Facility”: representa logicamente uma localização física de um chão de fábrica (*Shopping Floor*), ou seja, o local onde os *Materials* são transformados em *Products*.

O processo de manufacturing pode ainda considerar várias *Facilities*. Cada *Facility* é constituída por várias *Areas*, que instancia várias vezes os conceitos descritos anteriormente. Quando olhamos para este conjunto de conceitos em escala percebemos a necessidade de utilizar um sistema MES como suporte, tendo em conta que muitos dos processos existentes no *Shopping Floor*

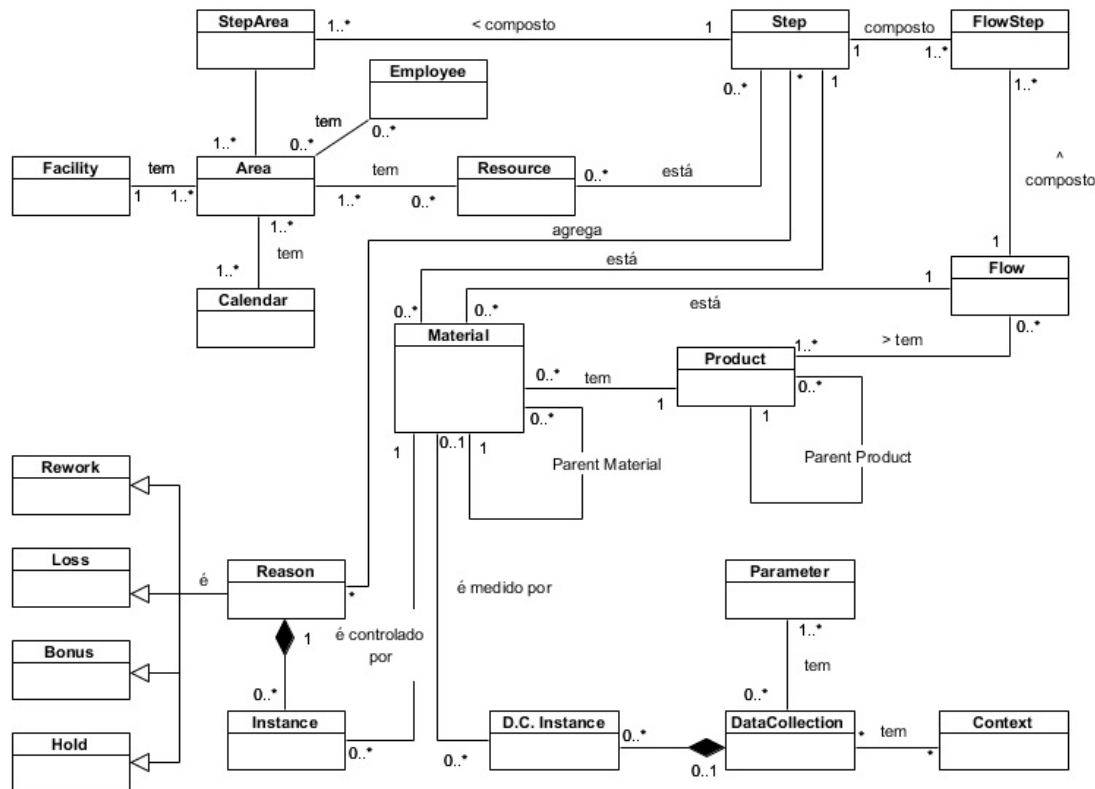


Figura 3.1: Modelo domínio — Manufacturing

podem ser executados em simultâneo. Os componentes, bem como as máquinas que os processam, podem ser caros e existe também a necessidade a monitorizar o processo. Podemos afirmar que os MES são sistemas de informação utilizados para gerir o processo de produção de uma fábrica em tempo real, visando tornar o contexto de manufacturing mais eficiente e *paperless*. Durante o processo de fabrico um determinado *Material* é alvo de um conjunto de operações. Estas operações são efetuadas num determinado *Step*, que é uma divisão lógica do chão de fábrica, onde podem existir *Resources* responsáveis pelo processamento de *Materials*. Fisicamente os *Resources* estão localizados em determinadas *Areas*. Deste processo podem ser originados outros *Materials* que podem, opcionalmente, ter uma relação genealógica com o original. Para efeitos de controlo de qualidade existem operações que permitem medir a quantidade de um material — via *Data Collections* — que quando instanciadas dão origem a uma *Data Collection Instance*. Quando no processo de fabrico existe alguma circunstância que altere o ciclo de vida normal do material existem operações que indicam isso mesmo, como é o caso da criação de “Loss/Bonus Reasons” no caso de perda/ganho de quantidade. Todas as operações efetuadas no sistema são ainda registadas no histórico de cada entidade. Todos estes conceitos podem ser mapeados para o modelo de domínio apresentado na Figura 3.1, de notar que o domínio de manufacturing está em constante alteração e que os conceitos apresentados são na quantidade mínima necessária para fornecer contexto sobre o problema.

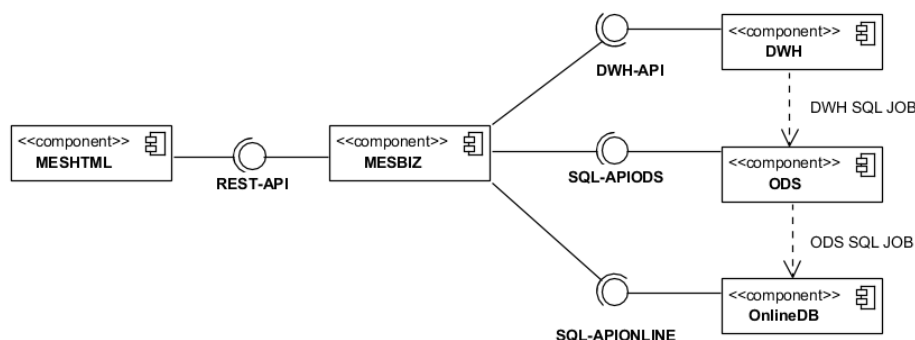


Figura 3.2: Granularidade de Sistema: Diagrama de Componentes

3.2 Arquitetura

De um ponto de vista arquitetural, ao nível de sistema, será apresentado o diagrama de componentes — que mostra a relação estrutural entre os diferentes componentes — e o diagrama de implantação — que explicita a arquitetura física implementada. A granularidade de sistema permite mostrar como é processada a informação do sistema com um escopo mais alargado.

3.2.1 Diagrama de Componentes

O diagrama de componentes explicita a estrutura de um sistema de software, que descreve os seus componentes, as suas interfaces e as suas dependências. Todos estes parâmetros estão apresentados na Figura 3.2.

O sistema é composto por uma camada de apresentação (Componente MESHTML), uma camada de processamento e aplicação de regras de negócio (Componente MESBIZ) e uma camada de persistência de dados que inclui os componentes OnlineDB, ODS e DWH que fazem uso da tecnologia relacional SQL Server. O Componente OnlineDB é responsável pela informação de runtime. O Componente ODS é uma réplica do componente OnlineDB e é a principal fonte de dados para efeitos de reporting, para evitar a execução de workflows de reporting sobre o componente transacional. O componente DWH (Data Warehouse) contém a informação de ODS agregada para efeitos de *business intelligence*.

Ao nível de interfaces, o componente MESBIZ expõe uma interface, REST-API, que é por sua vez consumida pelo componente MESHTML. A camada de persistência de dados expõe três interfaces, DWH-API, SQL-APIODS e SQL-APIONLINE, que são por sua vez consumidas pelo componente MESBIZ, sendo que apenas a API SQL-APIONLINE possibilita a escrita e leitura de dados, enquanto os outros só possibilitam a leitura. Internamente a camada de persistência possui uma série de SQL Server Jobs que servem para propagar e transformar o fluxo de dados entre as mesmas.

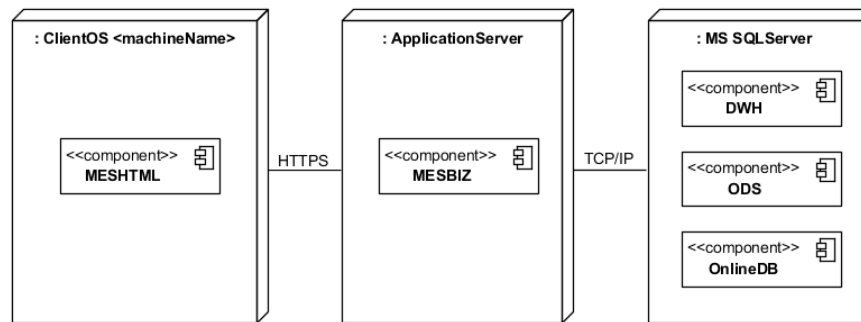


Figura 3.3: Granularidade de Sistema: Diagrama de Implantação

3.2.2 Diagrama de Implantação

Os diagramas de implantação fornecem uma vista da arquitetura física a um sistema, mostrando o relacionamento entre os componentes e a componente tecnológica implícita nesta interação.

Como podemos verificar na Figura 3.3, temos que o componente MESHTML comunica com o componente MESBIZ por HTTP, estando a ser executado numa máquina com o nome “machineName”. O Componente MESBIZ está implantado no “Application Server” e comunica com os componentes DWH, ODS e OnlineDB por TCP/IP, estes últimos implantados no servidor “MS SQLServer”.

3.3 Apresentação do problema

Como referido anteriormente, o principal uso dos sistemas de MES é, em tempo real, permitir a análise de dados e gestão do chão de fábrica. À medida que o sistema cresce, tanto ao nível de volume de dados, como ao nível de funcionalidades [50], aquilo que é o sistema vigente terá de se adaptar, ou deixará de ser uma mais-valia para o domínio de *manufacturing* [2]. Um sistema de *manufacturing* é, por definição, bastante denso ao nível de dados — como percebemos pela descrição do domínio realizada no ponto 3.1 — com entidades altamente interligadas, baseados numa entidade principal que é o Material. Existem duas operações, fulcrais no sistema e baseadas no material, que como consequência da sua interligação com entidades terceiras têm visto uma degradação de performance com o crescimento de dados — A *Material History* e a *Material Geneology*.

A *Material History* é uma operação fundamental num sistema de MES que permite para um determinado material saber todas as operações e sub-operações que foram efetuadas sobre o mesmo, bem como os seus detalhes e entidades intervenientes. Devido à sua natureza de aglomeração de dados, quanto mais tempo de vida tem o material, mais operações são executadas sobre o mesmo e por isso pior será a performance da operação.

A *Material Geneology* é uma operação que permite para um determinado material mostrar estruturadamente a relação entre *Parent Materials* e *Child Materials*, bem como a informação sobre as operações que derem origem a esta hierarquia, isto para N níveis de hierarquia, onde N é um número natural.

Atualmente o sistema MES da Critical Manufacturing implementa estas operações utilizando uma base de dados relacional, como descrito na secção 3.2.1. Como foi evidenciado pelo estado da arte e comprovado efetivamente o sistema relacional, apesar de robusto e confiável, não é um modelo ideal para quando a informação que queremos tratar é altamente densa ao nível de relações. Podemos também afirmar que o valor destas operações está relacionado com as relações entre entidades, e não com as entidades intrinsecamente. Com base nos argumentos apresentados ao longo do capítulo 2 percebemos que existem evidências que comprovam que Graph Databases (GDBs) podem ser sistemas otimizados para tratar este tipo de domínios altamente coesos, pelo que faz sentido explorar a sua aplicação como estratégia de inovação e suporte aos crescentes requisitos de dados. No entanto, da mesma maneira que existem evidências positivas, também existem evidências que indicam que as GDBs podem não ser ótimas para todos os problemas. Tendo em conta as conclusões obtidas no estado da arte, capítulo 2, e a problemática aqui apresentada, surgem as seguintes questões:

- **Q1:** O contexto de manufacturing pode ser otimizado com o uso de *Graph Databases*?
- **Q2:** As diferenças encontradas entre *Native Graph Databases* e *Non-Native Graph Databases* são evidenciadas no contexto de *manufacturing*?
- **Q3:** As propriedades do grafo têm impacto no sucesso da aplicação das *Graph Databases*?
- **Q4:** Que impacto tem a modelação no desempenho das *Graph Databases*?

Tendo em conta que o domínio de *manufacturing* é bastante denso ao nível de relações entre os dados e sintetizando o problema em questão nas questões anteriormente referidas faz sentido aplicar um processo *Exploration Based* para perceber se a utilização de *Graph Databases* faz sentido para o problema que nos propomos a resolver. Este processo será explorado no próximo capítulo.

Capítulo 4

Desenvolvimento

Este capítulo tem o objetivo de descrever os métodos utilizados para resolver o problema descrito ao longo da dissertação, com a finalidade de permitir avaliar a validade do trabalho desenvolvido do ponto de vista teórico. A secção 4.1 tem o objectivo de desenhar o processo para responder às questões levantadas no capítulo anterior. Por fim, a secção 4.2 tem a responsabilidade de, conforme a secção anterior, descrever e ilustrar o processo seguido, desde a justificação das diferentes decisões até à apresentação de resultados.

4.1 Definição do Processo

Para responder às questões propostas no capítulo anterior, vamos para cada questão propor uma hipótese nula (H_0) e uma hipótese alternativa (H_1). A hipótese nula mostra que não existe relação entre os conceitos/variáveis das questões, enquanto a hipótese alternativa mostra uma relação entre os conceitos/variáveis das questões. Relativamente aos conceitos, ao existir relação, podemos afirmar que existe um dependente e um independente. O conceito independente é aquele que fazemos variar para medir diferentes comportamentos do conceito dependente. Ao testar estes conceitos podemos afirmar, ou não, a existência de uma relação causa-efeito, permitindo responder às diferentes questões.

Para testar as hipóteses referidas na tabela 4.1, foi criado um ambiente que permita, por um lado, variar o conceito independente e, por outro lado, medir o conceito dependente.

Para a Questão 1, o conceito independente é relativo ao uso ou não uso das GDBs, e o conceito dependente é o quão otimizado é o contexto de manufacturing. Por otimização do contexto de manufacturing estamos a referir-nos à melhoria de performance nas operações *Material History* e *Material Geneology*. Podemos afirmar que H_1 é válida quando com a utilização das GDBs o contexto de manufacturing é otimizado. Para testar esta hipótese, temos de considerar um plural de GDBs, no caso 3, temos também de estabelecer um conjunto de indicadores que nos permitam medir esta otimização e, como o principal problema está relacionado com o volume de dados, temos de mesurar o comportamento das GDBs com o aumento do volume de dados. Para tal utilizaremos três *datasets* com volumes de dados diferentes.

Questão	H0	H1
Q1: O contexto de <i>manufacturing</i> é otimizado com o uso de GDBs?	O contexto de manufacturing não é otimizado com o uso de GDBs.	O uso de GDBs tem um impacto positivo no contexto de <i>manufacturing</i> .
Q2: As diferenças encontradas entre <i>Native</i> GDBs e <i>Non-Native</i> GDBs são evidenciadas no contexto de <i>manufacturing</i> ?	Para o contexto de <i>manufacturing</i> não existe diferença entre <i>native</i> e <i>non-native graph databases</i> .	Para o contexto de manufacturing existe diferença entre <i>native</i> e <i>non-native graph databases</i> .
Q3: As propriedades do grafo têm impacto no contexto de <i>manufacturing</i> ?	As propriedades do grafo não têm impacto na aplicação das <i>graph databases</i> .	As propriedades do grafo têm impacto na aplicação das <i>graph databases</i> .
Q4: Qual é a importância da modelação na performance das GDBs?	A modelação não tem impacto na performance.	A modelação tem impacto na performance.

Tabela 4.1: Tabela de Hipóteses

Para a Questão 2, o conceito independente é relativo ao uso ou não de uma *native* GDB, e o conceito dependente que queremos medir é o quão otimizado é o contexto de manufacturing. O processo de validação desta hipótese surge como consequência do processo de validação da primeira questão, pois temos requisitos semelhantes.

Para a Questão 3, o conceito independente é relativo às propriedades do grafo, e o conceito dependente ao sucesso da aplicação, ou seja, a otimização do contexto de manufacturing. Para testar esta hipótese temos de descrever os grafos existentes nos diferentes *datasets*. Para filtrar os indicadores escolhidos podemos dar continuidade às inferências descritas no capítulo de estado da arte relacionados com a densidade do grafo. Para tal vamos, para cada grafo do *dataset*, recolher informação sobre o número de nós, número de relações e propriedades derivadas da densidade do grafo para perceber se existe alguma correlação com o sucesso da aplicação das *graph databases*.

Para a Questão 4, o conceito independente é relativo à modelação, e o conceito dependente a performance das operações. Para testar o impacto que a modelação tem sobre a performance, terá de desenvolvido um processo de transformação de dados que permita modelar os mesmos de maneiras diferentes. Analisando o modelo de domínio e a implementação da Critical Manufacturing percebemos que as relações de muitos para muitos estão representadas por tabelas intermédias e quando queremos migrar este modelo para um grafo temos então duas opções: (i) a tabela intermédia é mapeada como *nó* ou (ii) a tabela intermédia é mapeada como uma relação.

De forma a testar as hipóteses anteriormente descritas e responder às questões apresentadas no capítulo anterior empregamos um processo *exploration based* inspirando no estado da arte, com as seguintes etapas:

(i) **Prólogo:** Consiste em escolher, descrever e justificar os três *datasets* a serem utilizados.

(ii) **Descrição dos Requisitos e respetivos indicadores:** consiste em, com base no capítulo de análise do problema, enumerar os requisitos funcionais e não funcionais, bem como os indicadores que permitem a sua avaliação.

(iii) **Escolha das implementações:** com base nos requisitos escolher, descrever e justificar os modelos que podem responder aos mesmos.

(iv) **Extração e Transformação dos dados:** consiste em extrair os dados do sistema origem e fazer um processo de transformação que permita validar a questão Q4. Este processo tem de ser efetuado de forma agnóstica, visto que existem, neste caso, três sistemas destinos.

(v) **Benchmarking:** Consiste em, com base nas implementações escolhidas, calcular os indicadores de desempenho definidos no ponto (ii).

(vi) **Epílogo:** Consiste em rematar todo o processo, respondendo às questões levantadas no capítulo anterior

(vii) **Escolha Final:** Tomar a decisão com base em todo o processo.

4.2 Instanciação do Processo

Com base na secção anterior, onde foram definidas as linhas gerais de desenho do processo, as seguintes secções irão ilustrar, para cada etapa, o procedimento empregado. Ao longo da descrição de cada etapa estão presentes as decisões tomadas e os respetivos resultados que permitem responder às hipóteses levantadas e presentes na tabela 4.1.

4.2.1 Prólogo

Como constatado anteriormente o problema das operações sobre o material está relacionado com o acréscimo volume de dados. Para validar o processo de escolha é importante perceber o comportamento do *System Under Test* (SUT) com o acréscimo do volume de dados, para tal foram escolhidos 3 datasets — que é o número mínimo aceitável para medir este comportamento. Como podemos apurar pelo estado da arte — “Para diferentes problemas diferentes soluções” — aquilo que é um SUT ideal irá depender do domínio do problema. Posto isto é imperativo que os *datasets* tenham origem num contexto de *manufacturing*, nomeadamente de origem no MES, e que o seu volume varie de forma crescente em ordem de grandeza.

O *Dataset 1* (QA), de menor volume, foi criado num ambiente de *staging* com apenas a informação mínima necessária para modelar o contexto de *manufacturing* e as operações sobre o material.

O *Dataset 2* (CD), de médio volume, foi criado num ambiente de *staging* onde sobre o mesmo é executada uma bateria de testes, com várias fábricas, vários materiais, portanto, mais dados. De

# Entidades	Dataset 1	Dataset 2	Dataset 3
Facilities	8	1 067	40
Materials	5	12 442	182 956
Areas	20	3 610	102
Resources	215	7 728	414
Data Collections	22	13 879	7 808
Checklists	56	5 375	8 343
Steps	68	13 392	9 224 639
Outros	5 528	1 406 280	4 155 860
Operações sobre Material	575	69 235	9 513 261
Relações Genealogia	10	4 590	151 501
Operações	29 621	1 377 233	52 761 610
Total	36 228	2 914 830	76 006 534

Tabela 4.2: Número de entidades nós diferentes datasets

notar que os dados simulam o contexto real, isto porque no processo de venda de sistemas *manufacturing* é importante os sistemas serem certificados para executar um conjunto de operações de forma estandardizada.

O *Dataset 3* (PROD), de grande volume, foi criado num ambiente de *staging* tendo por base dados reais de um sistema de *manufacturing* produtivo. De notar que o grande volume é contextualizado para o contexto de *manufacturing* e não de *big data*.

A informação relativa aos *datasets* pode ser analisada pela tabela 4.2.

De salientar que observamos no capítulo anterior, secção 3.2, que o sistema é composto por três componentes de persistência e análise de dados, sendo que o componente de ODS funciona como fonte de dados para efeitos de *reporting* o que implica que as operações *Material History* e *Material Geneology*, anteriormente descritas, tenham este sistema como fonte de informação, pelo que os dados presentes na tabela 4.2 têm origem no componente de ODS.

4.2.2 Descrição dos Requisitos e respetivos indicadores

O objetivo desta subsecção é explicitar os requisitos definidos, bem como os respetivos indicadores que permitem a sua avaliação de forma permitir a discussão do processo e validar os resultados do mesmo.

Como estratégia de padronização da descrição dos requisitos foi optado por seguir uma abordagem baseada no modelo FURPs+ [26] no sentido que consideramos os mesmos critérios do modelo. O modelo FURPs permite agrupar os requisitos em cinco principais categorias: Funcionalidade, Usabilidade, Confiabilidade, Desempenho e Suporte.

Do ponto de vista de funcionalidade e da ótica de persistência de dados a comunicação é efetuada atualmente com base em *SQL Server Jobs* o que não permite a integração com sistemas fora do ecossistema da Microsoft — como explicitado pela secção de arquitetura 3.2. Para mitigar esta limitação existe, portanto, o requisito interno da empresa que o novo sistema GDB suporte a integração com sistemas terceiros de forma desacoplada e agnóstica do ponto de vista tecnológico.

Portanto, foi definido internamente como requisito a integração ser efetuada com a utilização de um sistema distribuído de ‘streaming’ de eventos — o Kafka ¹

Do ponto de vista de usabilidade foi considerado a existência ou não de documentação. Este indicador é analisado de forma binária e, portanto, esmiuçado com uma grande granularidade porque apesar de ser um ponto a considerar não será determinante para a implementação e escolha de um novo sistema.

Do ponto de vista confiabilidade foi considerado a existência ou não de falhas na instanciação do processo descrito nesta secção. Este indicador é analisado de forma binária e, portanto, com uma grande granularidade porque, apesar de ser uma característica importante de um sistema, a análise desta propriedade está fora do escopo do presente trabalho.

Do ponto de vista de suporte e da ótica de licenciamento do sistema, um requisito importante é que o sistema seja um projeto *open-source* com uma comunidade saudável. Para especificar o conceito de comunidade saudável foi considerado diversas características do repositório como os descritos de seguida:

- O ano da última *release*: é um indicador que mostra se o projeto é mantido regularmente. Quanto maior for o indicador, mais próximo do estado da arte estará, em teoria, o sistema.
- O número de estrelas: é um indicador que mostra a popularidade de um determinado projeto. Quanto maior for o indicador, mais popular será o projeto e em teoria maior será a comunidade e a sua atividade.
- O número de *forks*: também é um indicador de atividade de um determinado projeto. O mecanismo de *forking* é utilizado pela comunidade *open-source* como forma de contribuir para um determinado projeto que serve muitas vezes para corrigir *issues* existentes no projeto. Quanto maior for o indicador, mais popular será o projeto e mais ativa a sua comunidade.
- O número de *issues*: é um indicador de atividade de um determinado projeto. Quanto maior for o indicador maior será o número de incoerências encontradas no projeto, isto porque um *issue* é aberto sempre que um determinado utilizador encontra algum comportamento inesperado no sistema. Apesar de o número de comportamentos inesperados ser à primeira vista um indicador negativo, a verdade é um *software* sem falhas é uma utopia, por isso este indicador é, na nossa opinião, considerado como um ponto a favor — quanto maior, maior será a atividade — no sentido em que a comunidade está preocupada em manter e melhorar o sistema. Com base neste indicador consideramos ainda um outro indicador composto: a média de *Issues* abertos por dia que permite analisar a atividade de forma mais precisa porque tem em conta o tempo de vida do projeto.
- O número de *pull-requests* (PR): é um indicador de atividade do projeto. É resultado do processo de *forking* no sentido que um contribuidor individual publica as suas alterações para usufruto da comunidade. Quanto maior for o número de PR completados mais ativa

¹<https://kafka.apache.org/>

Requisito	Indicador
<i>Integração com Kafka</i>	Binário
<i>Consumo de recursos</i>	Uso do disco (Gb)
<i>Rapidez carregamento de informação</i>	Tempo de upload (ms), tempo de criação de relações (ms)
<i>Query language declarativa</i>	Binário
<i>Rapidez de processamento</i>	Tempo de execução (ms) das operações <i>Material History</i> e <i>Material Geneology</i>
<i>Projeto Open-Source</i>	Binário
<i>Qualidade da comunidade Open-source</i>	Ano da última release, número de estrelas, <i>forks</i> , <i>issues</i> e <i>Pull-Requests</i> (PR) no repositório <i>github</i> , médias diárias de <i>Issues</i> e PR por dia
<i>Documentação</i>	Binário

Tabela 4.3: Indicadores de desempenho do processo de seleção

será a comunidade. De forma semelhante ao número de *issues*, outro indicador composto é o Número de PR por dia.

Do ponto de vista de desempenho do sistema, sendo efetivamente o requisito mais importante, a avaliação tem de ser efetuada comparativamente com o sistema atual. Os indicadores considerados foram o tempo de execução (t_1), em milissegundos (ms), das operações *Material History* e *Material Geneology*. Quanto maior for a diferença em relação ao sistema atual (t_0), melhor será o desempenho do sistema.

Do ponto de vista de ecossistema do sistema é importante que um determinado sistema possua um conjunto de ferramentas que potencie o seu uso, como, por exemplo, em nível de funcionalidades out-of-the-box, em nível de visualização de dados entre outros. Não existe propriamente um indicador absoluto que permite avaliar o ecossistema, portanto a avaliação neste caso tem de ser realizada de forma mais informal e comparativamente, entre os sistemas considerados.

Os requisitos acima descritos foram compactados na seguinte tabela 4.3.

4.2.3 Escolha das implementações

Para o processo de escolha dos sistemas a serem considerados para *benchmarking* foi seguido um esquema equivalente ao utilizado por outros artigos na bibliografia, utilizando a base de dados de *DB-Engines*² que classifica os sistemas conforme a sua popularidade, por diferentes modelos de base de dados, e que tem em conta as referências dos diferentes sistemas em fóruns de desenvolvimento. O critério de escolha consistiu em escolher os 3 primeiros sistemas de base de dados *open-source*. O que leva à escolha dos seguintes sistemas: Neo4J, ArangoDB e Virtuoso, por esta ordem. Devido a publicações recentes em fóruns comunitários e nos resultados apresentados por P. Cailliau et al. [12] foi decidido equacionar o uso do RedisGraph, que não está indexado na base de dados *DB-Engines*.

²<https://db-engines.com/en/ranking/graph+dbms>

De notar que o ranking *DB-Engines*, bem como os dados descritos nas seguintes subsecções, foram consultados entre dia 25 de março a 8 de abril de 2022.

4.2.3.1 Neo4J

O Neo4J ³ (*Release 4.4.6*, abril 2022) é uma *native graph database* e é o sistema mais representativo das GDBs. É um sistema escalável, ACID-Compliant desenhado para workflows de alta desempenho e escalabilidade. Tem ofertas *self-hosted* e *cloud* disponíveis. Ao nível de API possui integrações com TinkerPop 3 e Cypher Query Language. É utilizado por mais de 200 000 utilizadores e mais de 800 clientes empresariais, dos quais 75% são *Fortune 100 companies*. Do ponto de vista de ecossistema possui ferramentas de integração de dados — Neo4j ETL —, possui ferramentas de processamento de dados — APOC Library —, possui ferramentas de análise de dados, possui integração com Kafka e ainda uma ferramenta visualização de dados — Neo4J Bloom. Ao nível de comunidade têm cerca de 10 011 estrelas no *github*, 2 083 *forks*, cerca de 3 090 *issues* abertos e 9 732 *pull requests* (PR) — uma média de 2.8 PR/dia e 0.89 *Issues*/dia

4.2.3.2 ArangoDB

A ArangoDB ⁴ (*Release 3.9.1*, abril 2022) é uma base de dados multimodelo *open-source*. O tipo de *storage* da ArangoDB são documentos JSON (mecanismo key-value) e um motor de pesquisa de texto permitindo combinar vários modelos de sistemas de bases de dados. É um sistema ACID-Compliant e possui a capacidade de fazer *sharding*. Ao nível de API possui integração com uma linguagem proprietária, a ArangoDB Query Language (AQL) e com Graph API (Gremlin). No relatório da *Gartner* “Peer Insight” foi selecionado como a principal GDB pela quarta vez consecutiva. Do ponto de vista de ecossistema possui integrações com sistema de *business intelligence*, ferramentas de análise de dados e integração com Kafka. Ao nível de comunidade têm 12 277 estrelas no *github*, 734 *forks*, cerca de 4 256 *issues* abertos e 11 993 *pull requests* (PR) — uma média de 3.11 PR/dia e 1.1 *Issues*/dia

4.2.3.3 Virtuoso

O Virtuoso ⁵ (*Release 08.03.3316*, março 2020) é uma base de dados multimodelo *open-source* que representa a informação como tabelas relacionais e/ou coleções de *property graphs*. É um sistema *ACID-compliant*. Ao nível de API possui integração com SPARQL e SQL. Do ponto de vista de ecossistemas possui um sistema de visualização de dados e outro de quering, não possuindo integração com Kafka. Ao nível de comunidade têm 742 estrelas no *github*, 193 *forks*, cerca de 969 *issues* abertos e 72 *pull requests* (PR) — uma média de 0.01 PR/dia e 0.26 *Issues*/dia

³<https://github.com/neo4j/neo4j>

⁴<https://github.com/arangodb/arangodb>

⁵<https://github.com/openlink/virtuoso-opensource>

4.2.3.4 RedisGraph

O RedisGraph ⁶ (Release 2.8.0, fevereiro 2022) é uma *in-memory native graph database* baseada no modelo *Property Graph Model* e representa o mesmo sobre *sparse adjacency matrices* que reduz o número de memória precisa para guardar os grafos. Ao nível de API possui integração com CYPHER. Do ponto de vista de ecossistema possui ferramentas de importação de dados, ferramentas de visualização e integração com Kafka. Ao nível de comunidade têm 1 614 estrelas no *github*, 184 *forks*, 848 *issues* abertos e 1 504 *pull requests* (PR) — uma média de 0.79 PR/dia e 0.44 *Issues*/dia

4.2.3.5 Sumário

De forma a condensar a consulta da informação anteriormente descrita foi construída a tabela 4.4. O sistema de pontuação utilizado para classificar o Ecossistema e a Comunidade serve apenas como comparação relativa, no sentido em que o sistema com melhor pontuação tem pontuação máxima e os restantes têm pontuação proporcional arredondada às unidades. De realçar que o sistema Virtuoso será descartado nesta fase devido à ausência de integração com o Kafka.

<i>Database</i>	<i>Native GDB</i>	<i>ACID Compliant</i>	<i>Query Language</i>	<i>Integração com Kafka</i>	<i>Ecossistema (1/5)</i>	<i>Comunidade (1/5)</i>
Neo4J	✓	✓	- TinkerPop - Cypher Query Language	✓	★★★★★	★★★★☆
ArangoDB	X	✓	- ArangoDB Query Language (AQL) - Graph API (Gremlin)	✓	★★★☆☆	★★★★★
Virtuoso	X	✓	- SPARQL - SQL	X	★★★☆☆	★★★☆☆
RedisGraph	✓	Sem informação	Cypher Query Language	✓	★★★★☆	★★★☆☆

Tabela 4.4: Catálogo de *Graph Databases*

4.2.4 Extração, Transformação e Carregamento dos dados

Para exemplificar o processo aqui tomado, será focada a migração da relação entre a entidade Flow e Step (Flow-Step) do sistema relacional (origem) para os diferentes SUT (alvo). A relação entre estas duas entidades é mapeada através de uma tabela intermédia (FlowStep) que permite a

⁶<https://github.com/RedisGraph/RedisGraph>

navegação entre as duas entidades via *Foreign keys*. A análise desta relação permite abordar todas as decisões tomadas e fazer o paralelo para a estratégia de implementação das outras entidades.

Como observamos no capítulo anterior, a solução anterior utilizava como sistema de base de dados um sistema relacional, e para avançarmos no processo temos de arranjar uma estratégia, tecnologicamente agnóstica, que permita a migração dos dados entre o sistema origem e o sistema alvo. Foi explorada a existência de ferramentas que permitissem fazer esta migração de dados, mas devido à falta de *drivers* — implementações *out-of-the-box* — para os diferentes sistemas alvo, este processo implicava o desenvolvimento dos mesmos, que consistia num esforço de desenvolvimento elevado e com baixo aproveitamento do ponto de vista de custo/benefício. De notar que o ecossistema do Neo4J tem uma ferramenta, Neo4j ETL ⁷, que permite efetuar este processo, mas até à data o suporte para Microsoft SQL Server não estava funcional. Para otimizar esta etapa e para garantir a igualdade de análise entre os diferentes sistemas o processo de extração consistiu em exportar o domínio considerado do sistema relacional para um formato CSV via SQL Server Management Studio. Para a relação Flow-Step foram mapeadas as três tabelas para um ficheiro CSV via SQL Server Management Studio, onde a primeira linha possuía o nome das diferentes colunas.

Relativamente ao processo de Transformação de dados foram utilizados os ficheiros CSV gerados como input desta etapa. O objetivo desta etapa consistia em interpretar o conteúdo do CSV e tendo em conta o seu *schema* possibilitar o carregamento para os diferentes sistemas, que no que lhes concerne possuem interfaces diferentes. De notar ainda que um dos requisitos consiste em avaliar o impacto da modelação no desempenho dos sistemas. Para automatizar este processo, que seria bastante moroso caso fosse feito manualmente, inicialmente foi previsto desenvolver um interpretador que tendo em conta a estrutura do ficheiro CSV gerava o *script* de carregamento de dados para os diferentes sistemas alvo, com os dois tipos diferentes de modelação. No processo de desenvolvimento das regras do interpretador foi utilizado o *vscode* como editor de código-fonte que tinha a extensão *GitHub Copilot* ⁸ instalada que foi perspicaz o suficiente para, tendo em conta um exemplo de interpretação, replicar para todas as entidades do sistema. O processo para a criação do *schema* da relação Flow-Step pode ser evidenciado pela figura 4.1. O primeiro passo consistiu em criar um modelo para ser interpretado pelo *GitHub Copilot*, representado pela secção amarela. O segundo passo consistiu em alterar a *keyword* “entidade” pelas entidades pretendidas, bem como as respetivas propriedades. Como resultado destes dois passos conseguimos assim criar um mecanismo de geração de *schema* para os diferentes modelos.

⁷<https://neo4j.com/labs/etl-tool/>

⁸<https://docs.github.com/en/copilot>

<pre> 1 //input 2 Entidade: EntidadeId, EntidadeName, prop1, ... , prop(n-1), prop n 3 4 //output neo4j: 5 LOAD CSV WITH HEADERS FROM 'file:///T_Entidade.csv' AS row FIELDTERMINATOR ';'; 6 WITH row WHERE row.EntidadeId IS NOT NULL 7 CREATE (d:Entidade { EntidadeId: row.EntidadeId, 8 EntidadeName: row.EntidadeName, 9 prop1: row.prop1, 10 ... , 11 prop(n-1): row.prop(n-1), 12 prop n: row.prop n}) 13 }); 14 //output arangodb: 15 arangoimport --file "C:\-path\T_Entidade.csv" --collection "Entidade" --create-collection true --type csv --server.database "" 16 -server.username "" -server.password "" 17 18 //output redisgraph: 19 redisgraph-bulk-insert *** -N Entidade C:\-path\T_Entidade.csv 20 21 //input 22 Flow: FlowId, ChangeSetId, CreatedBy, CreatedOn, DataGroupId, DataGroupName, DefinitionId, Description, DocumentationURL, Image, IsAlternat 23 24 //output neo4j: 25 LOAD CSV WITH HEADERS FROM 'file:///T_Flow.csv' AS row FIELDTERMINATOR ';'; 26 WITH row WHERE row.FlowId IS NOT NULL 27 CREATE (d:Flow { FlowId: row.FlowId, 28 ChangeSetId: row.ChangeSetId, CreatedBy: row.CreatedBy, CreatedOn: row.CreatedOn, DataGroupId: row.DataGroupId, 29 }); 30 //output arangodb: 31 arangoimport --file "C:\-path\T_Flow.csv" --collection "Flow" --create-collection true --type csv --server.database "" 32 -server.username "" -server.password "" 33 34 //output redisgraph: 35 redisgraph-bulk-insert *** -N Flow C:\-path\T_Flow.csv </pre>	Zona Gerador Zona Gerado
--	--

Figura 4.1: Geração Script via Copilot

Após efetuado a *load* do *schema* das propriedades temos de criar as relações entre as entidades. Para desenvolver este ponto temos de considerar que existem duas abordagens quando queremos mapear a informação de uma tabela intermédia para um sistema GDB:

- Tabela intermédia como nó: consiste em mapear a tabela intermédia como nó. Nesta situação as relações apenas têm a referência para o nó origem e destino, funcionando como “*dumb pipes*”. Em *workflows* de *leitura* quando queremos ter a informação dos três nós (Source, Relação e Target) implica um número de *hops* superior — que poderá ter impacto no desempenho. O exemplo prático para o FlowStep pode ser analisado pela *Schema Alvo 1* na figura 4.2
- Tabela intermédia como relação: consiste em mapear a tabela intermédia como relação. Nesta situação a relação contém a informação da tabela intermédia. Esta modelação é a mais *domain driven* no sentido que as relações são as mesmas do modelo de negócio. O exemplo prático para o FlowStep pode ser analisado pela *Schema Alvo 2* na figura 4.2

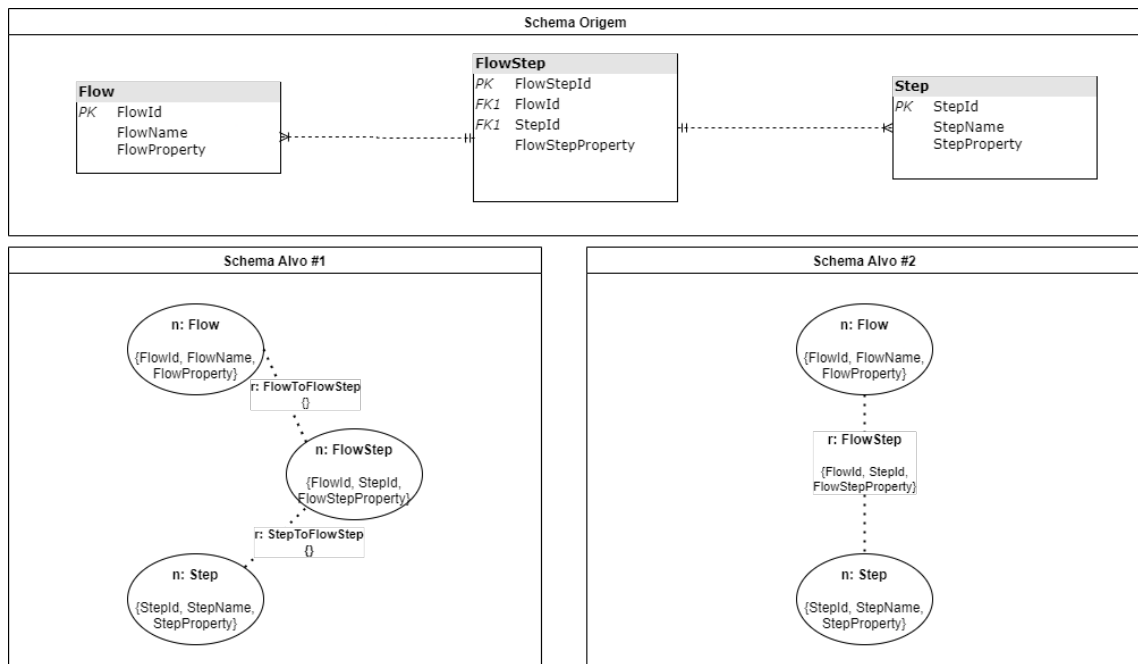


Figura 4.2: Comparação Schema Origem com Schemas Alvo — Exemplo FlowStep

Definidas as categorias de modelação desejadas, temos agora de criar os *scripts* para a criação das relações. Este processo foi semi-automatizado no sentido em que a lógica de relações entre os nós foi apenas desenvolvida uma vez. No caso começamos por desenvolver o *script* de criação das relações, com base no *schema* original (semelhante ao descrito no capítulo 3), pelo Neo4J. A criação das relações para as outras GDBs foi com o auxílio do *GitHub Copilot*, como podemos constatar pela figura 4.3.

Tabela Intermédia como Nó	Tabela Intermédia como Relação
<pre> 1 // input: FlowStep for Neo4J 2 MATCH (fs: FlowStep) 3 MATCH (f: Flow) 4 WHERE fs.SourceEntityId = f.FlowId 5 MERGE (f)-[r:FlowToFlowStep]->(fs); 6 7 MATCH (fs: FlowStep) 8 MATCH (s: Step) 9 WHERE fs.TargetEntityId = s.StepId 10 MERGE (s)-[r:StepToFlowStep]->(fs); 11 12 // output: FlowStep for ArangoDB 13 FOR fs IN FlowStep 14 FOR f IN Flow 15 FILTER fs["SourceEntityId"] == f["FlowId"] 16 INSERT {from: fs._id, to: f._id} IN FlowToFlowStep 17 RETURN NEW 18 19 FOR fs IN FlowStep 20 FOR s IN Step 21 FILTER fs["TargetEntityId"] == s["StepId"] 22 INSERT {from: fs._id, to: s._id} IN StepToFlowStep 23 RETURN NEW 24 25 // output: FlowStep for RedisGraph 26 GRAPH.QUERY *DBNAME* 27 *MATCH (fs: FlowStep) 28 *MATCH (f: Flow) 29 *WHERE fs.SourceEntityId = f.FlowId 30 *MERGE (f)-[r:FlowToFlowStep]->(fs)' 31 32 GRAPH.QUERY *DBNAME* 33 *MATCH (fs: FlowStep) 34 *MATCH (s: Step) 35 *WHERE fs.TargetEntityId = s.StepId 36 *MERGE (s)-[r:StepToFlowStep]->(fs)' </pre>	<pre> 38 39 // input: FlowStep Neo4J 2 40 41 MATCH (fs: FlowStep) 42 MATCH (s: Step) 43 MATCH (f: Flow) 44 WHERE fs.TargetEntityId = s.StepId AND fs.SourceEntityId = f.FlowId 45 MERGE (s)-[r:FlowStep_rel {FlowStepId: fs.FlowStepId, CorrelationID: fs.CorrelationID, CreatedBy: 46 }]->(fs); 47 48 49 // output: FlowStep for ArangoDB 2 50 51 FOR fs IN FlowStep 52 FOR s IN Step 53 FOR f IN Flow 54 FILTER fs["TargetEntityId"] == s["StepId"] AND fs["SourceEntityId"] == f["FlowId"] 55 INSERT {from: fs._id, to: s._id, FlowStepId: fs.FlowStepId, CorrelationID: fs.CorrelationID, 56 } IN FlowStep_rel 57 RETURN NEW 58 59 60 // output: FlowStep for RedisGraph 61 62 GRAPH.QUERY *DBNAME* 63 *MATCH (fs: FlowStep) 64 *MATCH (s: Step) 65 *MATCH (f: Flow) 66 *WHERE fs.TargetEntityId = s.StepId AND fs.SourceEntityId = f.FlowId 67 *MERGE (s)-[r:FlowStep_rel {FlowStepId: fs.FlowStepId, CorrelationID: fs.CorrelationID, Create 68 }]->(fs)' 69 70 71 72 73 </pre>

Figura 4.3: Criação de Relações — Exemplo FlowStep

<i>Ambiente</i>	<i>GDB</i>	<i>Modelação</i>	<i>TSchema (ms)</i>	<i>TRelation (ms)</i>	<i>Nós</i>	<i>Relações</i>	<i>Espaço (Gb)</i>
QA	Neo4J	#1	2 422	70 266	36 228	490 440	0.188
QA	Neo4J	#2	26 642	70 222	33 174	487 386	0.186
QA	RedisGraph	#1	1 667	3 013	36 228	490 440	0.168
QA	RedisGraph	#2	1 336	2 998	36 038	487 386	0.166
QA	ArangoDB	#1	1 000	1 826 970	36 228	490 440	0.402
QA	ArangoDB	#2	8 000	1 806 676	36 038	487 386	0.402
CD	Neo4J	#1	50 074	38 153	2 914 830	6 365 881	3.099
CD	Neo4J	#2	450 666	49 155	2 892 025	6 343 076	3.085
CD	RedisGraph	#1	133 503	52 804	2 914 830	6 365 881	5.099
CD	RedisGraph	#2	267 006	51 132	2 892 025	6 343 076	2.764
CD	ArangoDB	#1	14 000	5 536 961	2 914 830	6365881	3.556
CD	ArangoDB	#2	28 000	7 633 681	2 892 025	6 343 076	3.265
PROD	Neo4J	#1	4 431 000	90 772	75 997 636	215214199	98.695
PROD	Neo4J	#2	4 481 023	88 969	68 397 872	204 453 489	92.773
PROD	RedisGraph	#1	3 055 008	73 926	75 997 636	215 214 199	88.469
PROD	RedisGraph	#2	3 075 075	88 919	75 997 636	215 214 199	83.125
PROD	ArangoDB	#1	N/A	N/A	N/A	N/A	N/A
PROD	ArangoDB	#2	N/A	N/A	N/A	N/A	N/A

Tabela 4.5: Informações relacionados com o carregamento dos dados

Relativamente ao processo de carregamento de dados foram utilizados os *outputs* das etapas anteriores, os diferentes ficheiros de CSV e os *scripts* de carregamento. O processo de carregamento é composto por duas etapas: carregamento do *schema* e carregamento das relações. O carregamento do *schema* consiste em carregar a informação das diferentes entidades, tendo em conta as colunas do CSV. O carregamento das relações consiste em criar as relações entre as entidades, com base no *schema* original. Este processo foi o mais moroso de todos devido à dificuldade de trabalhar com *datasets* relativamente grandes. Antes de iniciar as diferentes etapas foram criadas seis bases de dados para cada sistema escolhida na secção 4.2.3.5, uma para cada combinação *dataset* e tipo de modelação. Os tempos de carregamento, a quantidade de nós e relações, bem como o espaço ocupado por cada base de dados podem ser analisados pela tabela 4.5.

Como referido anteriormente esta foi a fase mais morosa do processo, isto porque durante a execução deste processo e na geração dos dados para a tabela anterior ocorreram alguns problemas, que podemos ver refletidos nos artefactos apresentados, dos quais:

- ArangoDB — Criação Relações: No processo de criação de relações o ArangoDB possui uma limitação de implementação no que toca a criação de colunas compostas com o nome do nó mais o sufixo “Id”. No caso da figura 4.3 podemos reparar que quando estamos a efetuar o filtro do nó “Flow” o acesso à propriedade “FlowId” tem de ser efetuado de forma indireta com o uso de um indicador [“Nome da Propriedade”] senão o filtro não iria retornar resultados.
- ArangoDB — Criação *Schema*: No processo de criação de *Schema* não foi possível carregar os dados para o ambiente *PROD*, devido às dimensões do mesmo todas as operações de carregamento levavam ao *time-out* da operação e/ou *crash* da base de dados.

- RedisGraph: No processo de criação de *Schema*, o RedisGraph possui uma limitação no que toca a carregar a informação via CSV. A primeira coluna do ficheiro CSV era importada com prefixo “0xE8”. Este problema foi bastante difícil de contornar porque a primeira propriedade a ser exportada era o *Primary Key* da tabela o que impossibilitava a operação de *JOIN* com outras entidades e o uso do prefixo ao nível da *query* não era interpretado. Para colmatar este problema foi adicionada uma nova coluna ao ficheiro CSV, o que complica com o acréscimo do volume do ficheiro.
- Neo4J: No processo de criação das relações, para os dois últimos *datasets*, a criação das mesmas levava ao *time-out* das operações. Após uma pesquisa nos fóruns do Neo4J foi recomendado o uso da *APOC Library*⁹ que permite fazer operações em *batch*. Como é evidenciado pela tabela 4.5 vimos que com o uso de *batch-processing* o tempo de carregamento diminui, mesmo para um volume de dados superior.

Para conhecer melhor os dados de cada *dataset*, bem como a sua estrutura, vamos atentar às seguintes tabelas, tabelas 4.6, 4.7 e 4.8, que apresentam dados estatísticos (Avg - média, Min - mínimo e Max - máximo) relativos ao número de propriedades de cada nó (PropertyCount) e ao número de relações de cada nó (Relationship Count). De notar que estes dados só foram calculados para a modelação “tabela intermédia como nó” porque as conclusões seriam semelhantes. Relativamente à tabela 4.8, que representa o *Dataset 3*, podemos afirmar que consiste apenas numa amostra do universo de dados, escolhida de forma aleatória e recorrendo a um gerador de números inteiros entre 0 e 1, visto que efetuar este cálculo para cerca de 70 milhões de entradas seria bastante exigente do ponto de vista computacional.

Nodes(n)	Avg_PropertyCount	Min_PropertyCount	Max_PropertyCount	Avg_RelationshipCount	Min_RelationshipCount	Max_RelationshipCount
Area	44	44	44	73.4	70	85
Calendar	50	50	50	9.5	5	19
Checklist	39	39	39	0	0	0
ChecklistInstancHistory	26	26	26	0	0	0
ChecklistItemId	36.90566	36	37	0	0	0
ChecklistItemInstance	30	30	30	2	2	2
DataCollection	37	37	37	1.136364	0	15
DataCollectionInstanceHistory	50	50	50	9.64	5	17
DataCollectionPoint	23	23	23	4.875	3	9
Facility	27	27	27	6.25	1	15
Flow	43	43	43	4.4	0	109
FlowStep	25	25	25	2	2	2
Material	150	150	150	7.2	5	14
MaterialAttributeHistory	14	14	14	0	0	0
MaterialGenealogy	39	39	39	0	0	0
MaterialHistory	147	147	147	234.7287	13	758
MaterialHoldReasonHistory	22	22	22	0	0	0
OperationHistory	11	11	11	5.463995	1	148
Parameter	32	32	32	1.5	0	30
Product	118	118	118	7.970874	0	328
ProductionOrder	56	56	56	3.565217	0	22
Reason	28	28	28	0	0	0
Resource	167	167	167	1511.033	1505	1712
Service	24	24	24	4.805556	0	212
ServiceHistory	11	11	11	6.290295	0	6392
Step	94	94	94	48.92647	25	622
StepArea	18	18	18	2	2	2

Tabela 4.6: Dados estatísticos relativos à estrutura de dados no dataset QA

⁹<https://neo4j.com/developer/neo4j-apoc/>

Nodes(n)	Avg_PropertyCount	Min_PropertyCount	Max_PropertyCount	Avg_RelationshipCount	Min_RelationshipCount	Max_RelationshipCount
Area	44	44	44	18.11	2	22716
Calendar	25	0	50	2.10	0	3040
Checklist	39	39	39	0.16	0	18
ChecklistInstacneHistory	13	0	26	2.75	0	13
ChecklistItemId	37	37	37	0.55	0	9
ChecklistItemInstance	30	30	30	2.45	2	4
DataCollection	8.09	0	37	0.11	0	200
DataCollectionInstanceHistory	0.095	0	50	0.03	0	421
DataCollectionPoint	11.44	0	23	1.83	0	8
Facility	27	27	27	5.47	1	1486
Flow	42.99	42	43	0.63	0	202
FlowStep	25	25	25	2	2	2
Material	42	42	42	17.13	3	448
MaterialAttributeHistory	13.99	13	14	1.62	0	10
MaterialGenealogy	19.5	0	39	0	0	0
MaterialHistory	147	147	147	64.06	16	1146
MaterialHoldReasonHistory	22	22	22	1.99	0	2
OperationHistory	11	11	11	4.20	1	97
Parameter	15.99	0	32	2.71	0	2700
Product	118	118	118	5.23	0	9114
ProductionOrder	56	56	56	3.93	0	60
Reason	28	28	28	0.13	0	176
Resource	166.98	166	167	14.70	2	690
Service	24	24	24	3.45	0	4517
ServiceHistory	11	11	11	5.29	0	6428
Step	93.99	93	94	19.85	5	11295
StepArea	18	18	18	2	2	2

Tabela 4.7: Dados estatísticos relativos à estrutura de dados no dataset CD

Nodes(n)	Avg_PropertyCount	Min_PropertyCount	Max_PropertyCount	Avg_RelationshipCount	Min_RelationshipCount	Max_RelationshipCount
Area	43	43	43	3	2	4
Checklist	29	29	29	27.40	0	1925
Calendar	25	0	50	2.19	0	4
ChecklistInstacneHistory	23	23	23	3.27	2	16
ChecklistItemId	34	34	34	34.25	0	10185
ChecklistItemInstance	28.99	28	29	3.78	2	4
DataCollection	29	29	29	89.65	0	8569
DataCollectionInstanceHistory	49	49	49	16.94	1	2177
DataCollectionPoint	22	22	22	5.93	3	56
Flow	35	35	35	0	0	0
FlowStep	24	24	24	0	0	0
Material	41.99	41	42	7.46	0	167
MaterialAttributeHistory	13.96	13	14	7.03	0	3023
MaterialGenealogy	35	35	35	0	0	0
MaterialHistory	141.99	141	142	22.18	14	808
MaterialHoldReasonHistory	19.99	19	20	2	2	2
OperationHistory	11	11	11	0.64	0	1098
Parameter	30.99	30	31	85.75	0	9381
Product	96	96	96	14.70	0	0
ProductionOrder	55	55	55	0.73	0	270
Reason	27	27	27	0	0	0
ServiceHistory	11	11	11	8.95	0	12024
Step	91	91	91	458598	458598	458598
StepArea	17	17	17	2	2	2

Tabela 4.8: Dados estatísticos relativos à estrutura de dados no dataset PROD

Analisando os dados das tabelas 4.6, 4.7 e 4.8, com fina granularidade, percebemos que ao nível de propriedades não existe diferenças visíveis relativamente ao número de propriedades. Ao nível de relações percebemos que a entidade “MaterialHistory” é a entidade com mais relações em todos os *datasets*, com a exceção do *Dataset 3* que pode ser justificado pela escolha aleatória — Note-se que, por exemplo, a *DataCollection* tem um número de *relations* superior aos outros nós. O facto de a tabela “MaterialHistory” ser a entidade com mais relações foi algo destacado no capítulo anterior onde foi afirmando ser a base da operação mais importante do sistema. Notamos também que as tabelas intermédias têm igual valor de média, mínimo e máximo, algo que pode ser explicado pelo facto de que cada instância ter sempre duas relações — uma com princípio no nó origem e outra em direção ao nó destino.

Analisando os dados da tabela 4.9, com grande granularidade, temos presente o número de

Dataset	Nós	Relações	R/N	ρ	$\bar{\rho}$	$\min(\bar{\rho})$	$\max(\bar{\rho})$	ρMax
QA	36228	490440	1.35E+01	7.47E-04	7.24E-06	0.00E+00	1.470E-04	6.56E+08
QA 2	33174	487386	1.47E+01	8.86E-04	8.46E-06	0.00E+00	1.470E-04	5.50E+08
CD	2914830	6 365 881	2.18E+00	1.50E-06	1.52E-01	0.00E+00	1.58E+00	4.25E+12
CD 2	2892025	6343076	2.19E+00	1.52E-06	1.64E-01	0.00E+00	1.58E+00	4.18E+12
PROD	75997636	215 214 199	2.83E+00	7.45E-08	3.01E+00	0.00E+00	1.83E+01	2.89E+15
PROD 2	68397872	204453489	2.99E+00	8.74E-08	3.12E+00	0.00E+00	1.83E+01	2.34E+15

Tabela 4.9: Propriedades dos diferentes Datasets

nós (*nós*), o número de relações (*relações*), o coeficiente entre o número de relações e número de nós (R/N), a densidade do grafo (ρ), o máximo teórico de densidade do grafo (ρMax), a densidade média de cada nó ($\bar{\rho}$), a densidade máxima ($\max(\bar{\rho})$) e a densidade mínima ($\min(\bar{\rho})$) de cada nó.

Estes indicadores permitem avaliar diferentes propriedades dos grafos. O R/N é um indicador que nos permite medir a proporção entre relações e nós num grafo. A densidade de um grafo é calculada por $\rho = \frac{2 \cdot R}{N \cdot (N-1)}$, onde R é o número de relações e N o número de nós, e quando avaliada com máximo teórico de densidade do grafo, calculado por $\rho\text{Max} = \frac{N \cdot (N-1)}{2}$, permite avaliar o grafo globalmente quanto à sua classificação como *dense* ou *sparse*. A densidade mínima, média e máxima de um nó permite avaliar o grafo localmente quanto à sua classificação, como *dense* ou *sparse*, no sentido em que faz a análise para cada “tipo” de nó, onde cada “tipo” de nó que representa as diferentes entidades do sistema.

Analisando a tabela 4.9 podemos visualizar que os indicadores globais R/N , ρ para os *datasets* considerados diminuem o seu valor com o acréscimo de relações e de nós — o que implica que a proximidade entre o indicador ρ e ρMax diminua, diminuindo a densidade relativa do grafo. Inversamente, os indicadores locais $\bar{\rho}$, e $\max(\bar{\rho})$ aumentaram. De notar também que o $\max(\bar{\rho})$ mantém-se dentro do *dataset* independentemente do tipo de modelação.

4.2.5 Benchmarking

Esta subsecção tem o objetivo de explicitar as decisões tomadas na etapa de *benchmarking*, bem como apresentar indicadores estatísticos e visuais que permitam analisar os indicadores previamente definidos.

4.2.5.1 Equipamento

No processo de *benchmarking* é muito importante ter informação sobre o hardware utilizado, considerando a validade e reprodutibilidade da operação. A experiência foi realizada num *Le-Novo ThinkPad T460p* equipado com um CPU *Intel Core i7-6700HQ* com 2.60GHz de *clocking speed* e 16.0 GB de RAM. Ainda a mencionar que o sistema operativo utilizado foi o *Windows 11 enterprise*.

4.2.5.2 Resultados

O processo de *benchmarking* consiste em calcular os diferentes indicadores a serem considerados, descritos na tabela 4.3 e relacionados com o requisito de rapidez de processamento. Para fazer o cálculo dos mesmos foi preciso migrar as diferentes operações do sistema *baseline* — o sistema original — para os diferentes SUT — os diferentes sistemas alvo — que como vimos anteriormente têm interfaces diferentes e como consequência da etapa de transformação de dados 2 modelações diferentes, o que dá um total de 12 implementações (2 operações * 3 SUT * 2 tipos de modelação) para serem executadas sobre 3 *datasets* diferentes. Na criação destas implementações foi seguido um processo semelhante à criação das relações com a diferença que em vez de usarmos operações de construção de *schema* usamos operações de leitura.

Os resultados do processo de *benchmarking* podem ser analisados pelo intervalo de figuras de 4.4 a 4.15. O intervalo de figuras 4.4 a 4.9 representa o tempo de execução em milissegundos (*ms*) ao longo dos 25 pedidos, o permite analisar o comportamento dos diferentes SUT com granularidade fina. Devido à discrepância de valores entre o sistema *baseline* e os diferentes SUT os gráficos mostram a informação utilizando a escala logarítmica de base 10. O número de pedidos escolhido permite fornecer validade estatística e perceber também como é que se comportam os diferentes SUT relativamente ao sistema *baseline*. O intervalo de figuras 4.10 a 4.15 permite analisar o comportamento dos diferentes SUT, com granularidade grossa, permitindo fazer uma comparação, utilizando a média, entre a performance dos diferentes pares sistemas-modelações permitindo responder aos indicadores necessários. A comparação é efetuada utilizando o indicador percentagem de sucesso (PS/%), calculado com base na seguinte expressão: $PS(SUT) = (1 - \frac{MEDIA(SUT)}{MEDIA(baseline)}) * 100$, onde $MEDIA(SUT)$ representa o tempo de execução médio, em *ms*, de um determinado SUT e $MEDIA(baseline)$ representa o tempo de execução médio, em *ms*, do sistema origem.

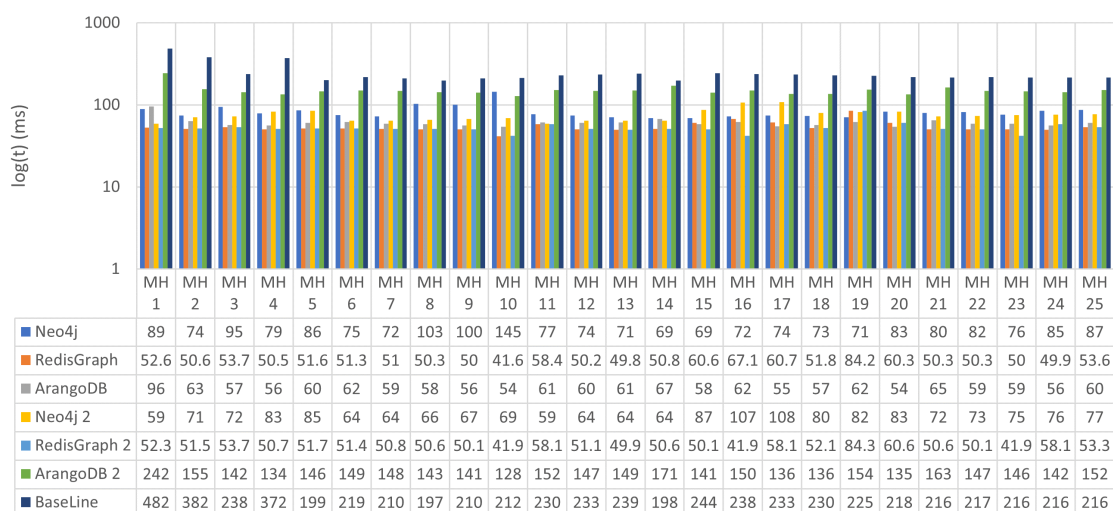


Figura 4.4: Tempo de execução *Material History* em milissegundos no Dataset 1 — escala logarítmica base 10

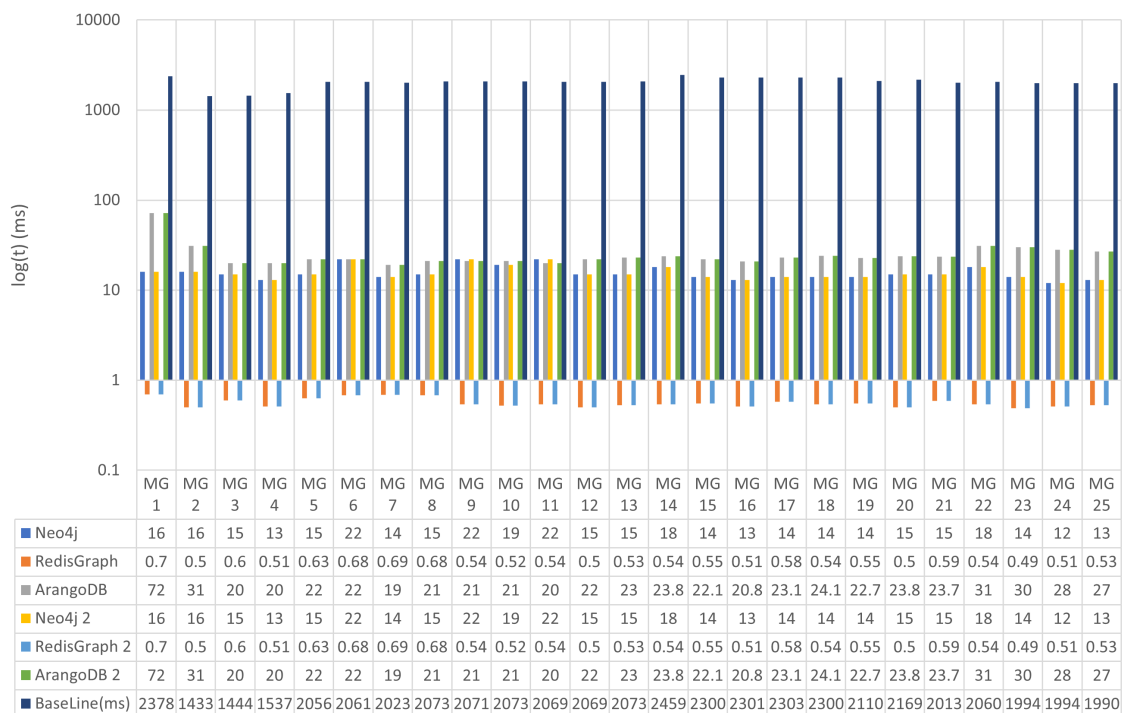


Figura 4.5: Tempo de execução *Material Geneology* em milissegundos no Dataset 1 — escala logarítmica base 10

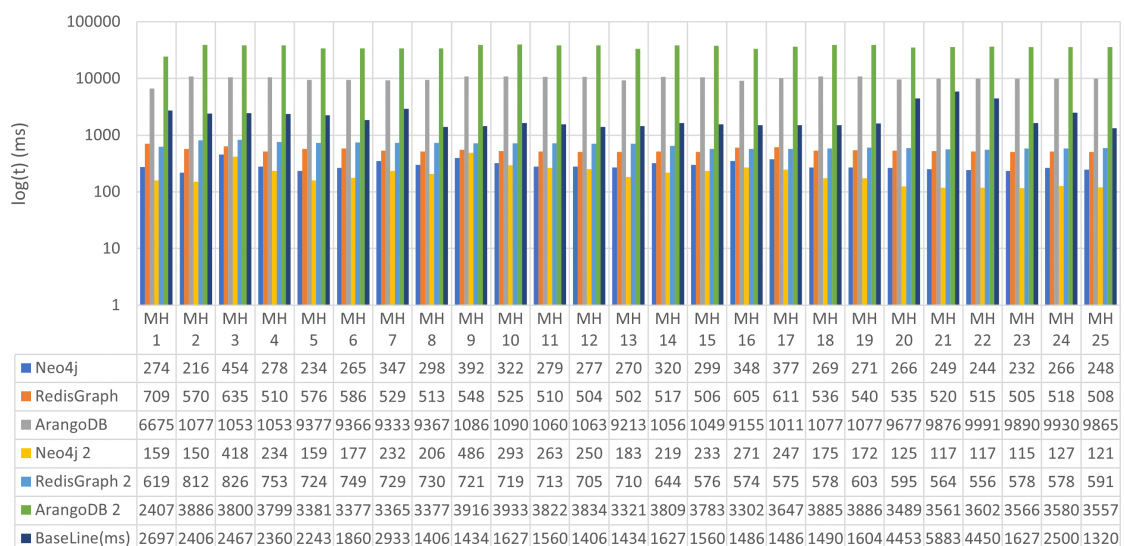


Figura 4.6: Tempo de execução *Material History* em milissegundos no Dataset 2 — escala logarítmica base 10

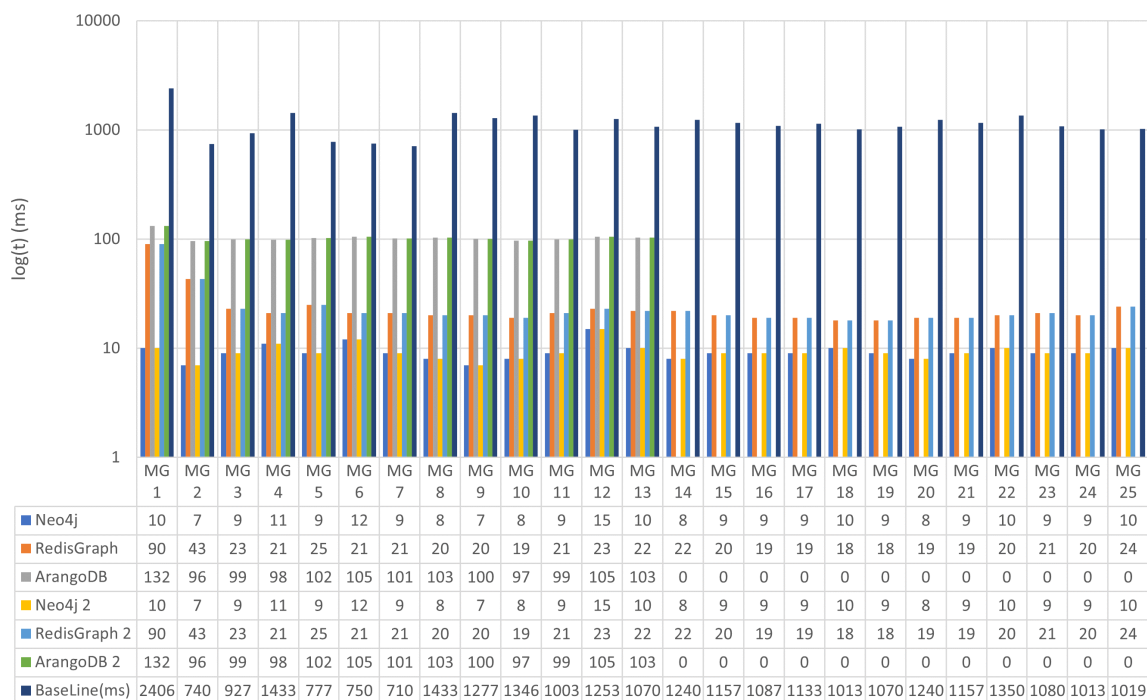


Figura 4.7: Tempo de execução *Material Geneology* em milissegundos no Dataset 2 — escala logarítmica base 10

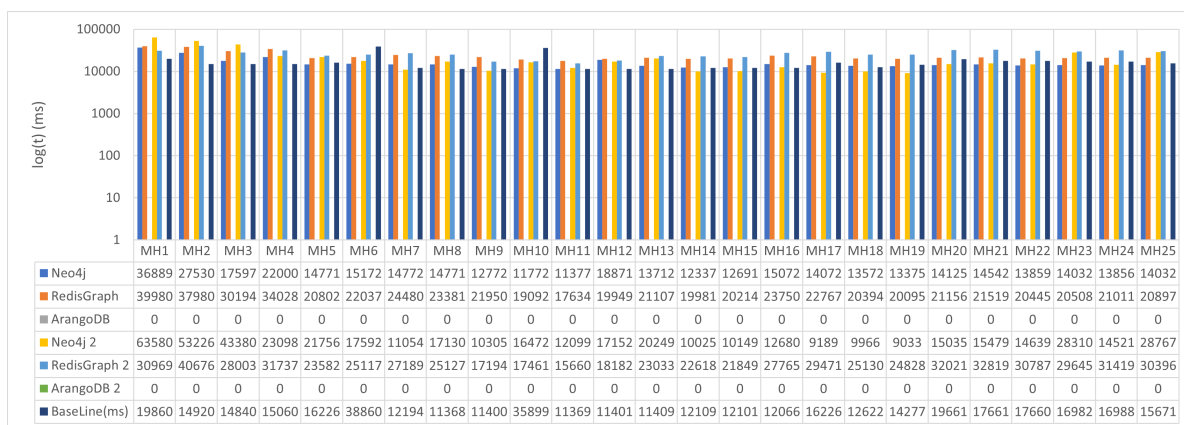


Figura 4.8: Tempo de execução *Material History* em milissegundos no Dataset 3 — escala logarítmica base 10

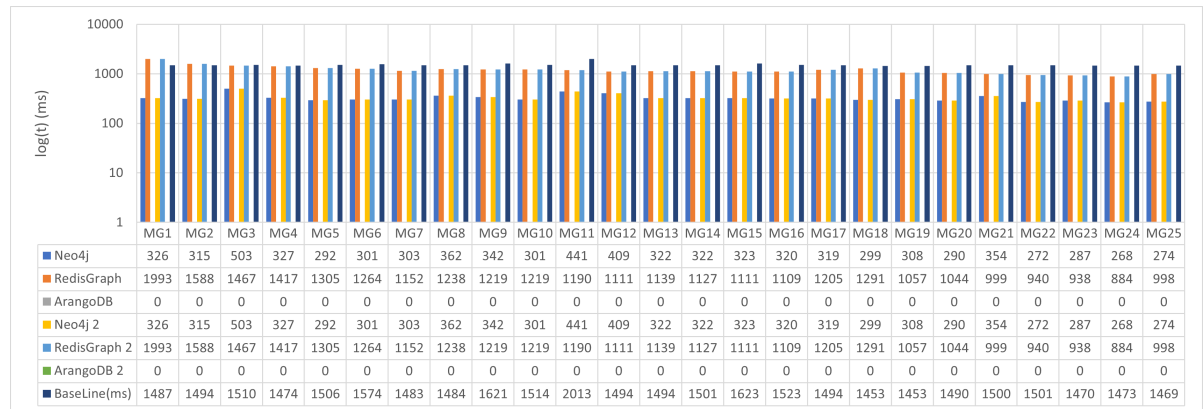


Figura 4.9: Tempo de execução *Material Geneology* em milissegundos no Dataset 3 — escala logarítmica base 10

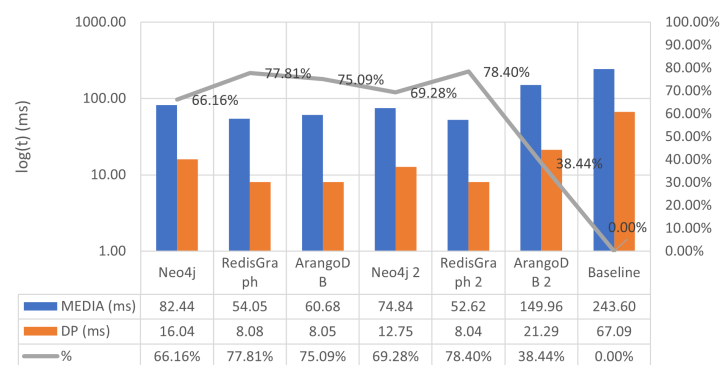


Figura 4.10: Tempo de execução médio *Material History* em milissegundos no Dataset 1 — escala logarítmica base 10

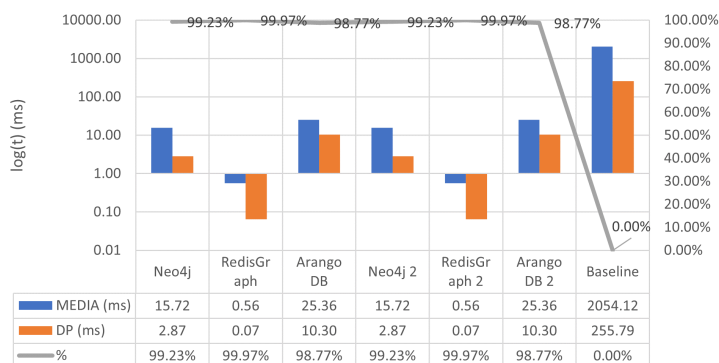


Figura 4.11: Tempo de execução médio *Material Geneology* em milissegundos no Dataset 1 — escala logarítmica base 10

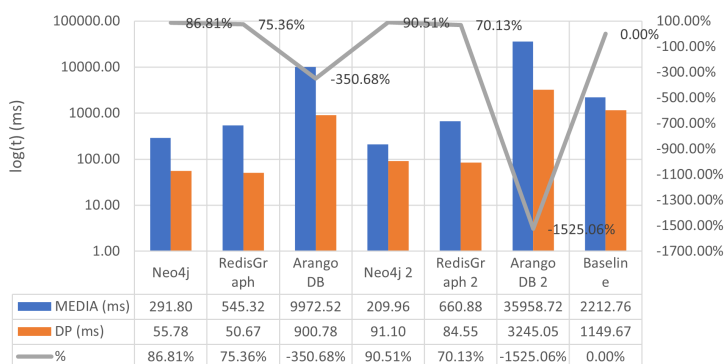


Figura 4.12: Tempo de execução médio *Material History* em milissegundos no Dataset 2 — escala logarítmica base 10

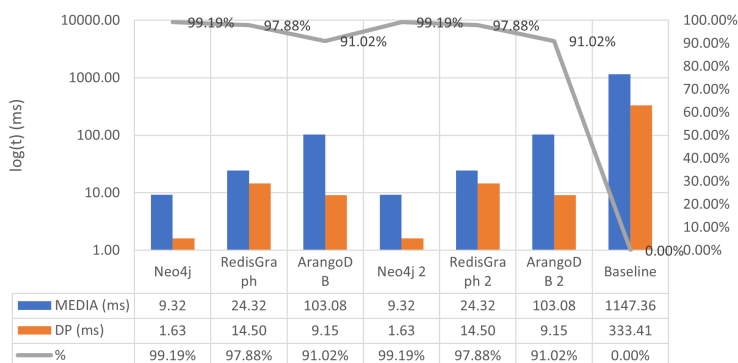


Figura 4.13: Tempo de execução médio *Material Geneology* em milissegundos no Dataset 2 — escala logarítmica base 10

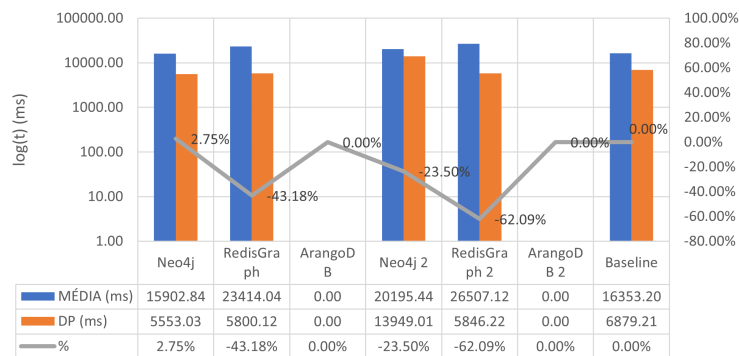


Figura 4.14: Tempo de execução médio *Material History* em milissegundos no Dataset 3 — escala logarítmica base 10

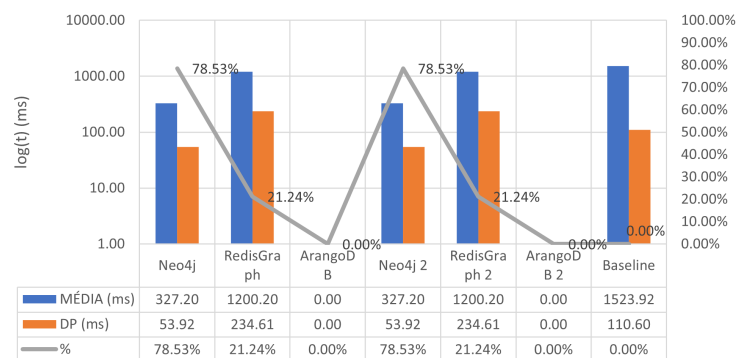


Figura 4.15: Tempo de execução médio *Material Geneology* em milissegundos no Dataset 3 — escala logarítmica base 10

Relativamente à operação *Material History*, para o *Dataset 1*, podemos aferir, pela figura 4.4, que todos os SUT têm um desempenho superior relativamente ao sistema *baseline* ao longo dos 25 pedidos. Percebemos também que todos os sistemas considerados diminuem o seu tempo de execução ao longo dos diferentes pedidos, com a exceção do RedisGraph e do Neo4J. Podemos também para este *dataset* fazer uma análise comparativa de resultados, com base na figura 4.10, onde observarmos que podemos ordenar a performance da seguinte maneira: RedisGraph 2 (com uma melhoria relativa ao *baseline* de 78.40%), RedisGraph, ArangoDB, Neo4J 2, Neo4 J, ArangoDB 2 (com uma melhoria relativa ao *baseline* de 38.44%). De realçar ainda, com base no desvio padrão (DP), a estabilidade dos pedidos é superior para todos os SUT, especialmente para o RedisGraph e ArangoDB.

Relativamente à operação *Material Geneology*, para o *Dataset 1*, podemos aferir, pela figura 4.5, que todos os SUT têm um desempenho superior relativamente ao sistema *baseline* ao longo dos 25 pedidos. Percebemos também que todos os sistemas considerados diminuem o seu tempo de execução ao longo dos diferentes pedidos. Podemos também para este *dataset* fazer uma análise comparativa de resultados, com base na figura 4.11, onde observarmos que podemos ordenar a performance da seguinte maneira: RedisGraph e RedisGraph 2 (com uma melhoria relativa ao *baseline* de 99.97%), Neo4J e Neo4J 2 (com uma melhoria relativa ao *baseline* de 99.23%), ArangoDB e ArangoDB 2 (com uma melhoria relativa ao *baseline* de 98.77%). De realçar ainda, com base no desvio padrão (DP), a estabilidade dos pedidos é superior para todos os SUT, especialmente para o RedisGraph (0.07 ms) e Neo4J (2.87 ms).

Relativamente à operação *Material History*, para o *Dataset 2*, podemos aferir, pela figura 4.6, que todos os SUT têm um desempenho superior relativamente ao sistema *baseline* ao longo dos 25 pedidos, com a exceção do ArangoDB e ArangoDB 2. Percebemos também que todos os sistemas considerados diminuem o seu tempo de execução ao longo dos diferentes pedidos, com a exceção do ArangoDB e ArangoDB 2. Podemos afirmar, com base no desvio padrão, que a flutuação de valores anda por volta de valores inferiores a 19%, em contraste com o sistema *baseline* com aproximadamente 30%, passando mesmo em valores absolutos de 2697 ms para 1320 ms, melhoria de aproximadamente 52%. Podemos também para este *dataset* fazer uma análise comparativa de resultados, com base na figura 4.12, onde observarmos que podemos ordenar a performance da seguinte maneira: Neo4J 2 (com uma melhoria relativa ao *baseline* de 90.51%), Neo4J, RedisGraph, RedisGraph 2, ArangoDB e ArangoDB 2 (com um declino de 350.68% e 1525.05%, respetivamente).

Relativamente à operação *Material Geneology*, para o *Dataset 2*, podemos aferir, pela figura 4.7, que todos os SUT têm um desempenho superior relativamente ao sistema *baseline* ao longo dos 25 pedidos. Percebemos também que o sistema Neo4J mantém o seu tempo de execução, o que poderá estar relacionado com a dimensão do resultado quando comparado com a avaliação para o *Dataset 1* — visto que tem um desvio padrão, em termos relativos, bastante semelhante (cerca de 17% para os dois *datasets*). O sistema RedisGraph e o *baseline* têm uma descida gradual. De notar que o sistema RedisGraph, após a 10 execução, deixou de retornar resultados. Podemos também para este *dataset* fazer uma análise comparativa de resultados, com base na figura 4.13, onde

observarmos que podemos ordenar a *performance* da seguinte maneira: Neo4J e Neo4J 2 (com uma melhoria relativa ao *baseline* de 99.19%), RedisGraph e RedisGraph 2 (com uma melhoria relativa ao *baseline* de 97.88%).

Relativamente à operação *Material History*, para o *Dataset 3*, podemos aferir, pela figura 4.8, que todos os SUT têm um desempenho ligeiramente inferior relativamente ao sistema *baseline* ao longo dos 25 pedidos, com a exceção do Neo4J. Percebemos também que todos os sistemas considerados diminuem o seu tempo de execução ao longo dos diferentes pedidos e que todos os SUT nos primeiros pedidos têm valores muito superiores aos restantes pedidos e ao sistema *baseline* — De realçar o Neo4J com um tempo de execução de 63 580 ms para o primeiro pedido, um aumento de cerca de 300% relativamente ao seu tempo de processamento médio. Ao longo dos 25 pedidos também podemos realçar execuções que fogem ao padrão de processamento, nomeadamente a execução MH6 e MH10 para a *baseline* e a MH1 e MH2 para o Neo4J 2. É importante realçar também que o Neo4J 2 é a implementação que possui o mínimo absoluto de execução, presente na execução MH19. Podemos também para este *dataset* fazer uma análise comparativa de resultados, com base na figura 4.14, onde observarmos que podemos ordenar a *performance* da seguinte maneira: Neo4J (com uma melhoria relativa ao *baseline* de 2.75%), *baseline*, Neo4J 2, RedisGraph, RedisGraph 2 (com um agravamento relativo ao *baseline* de -62.09%). De realçar ainda, com base no desvio padrão (DP), a instabilidade dos pedidos é bastante elevada para todos os sistemas, com a seguinte ordem de estabilidade: RedisGraph 2 (22%), RedisGraph (24%), Neo4j (35%), *baseline* (42%), Neo4j 2 (69%),

Relativamente à operação *Material Geneology*, para o *Dataset 3*, podemos aferir, pela figura 4.9, que todos os SUT têm um desempenho superior relativamente ao sistema *baseline* ao longo dos 25 pedidos, onde a implementação Neo4J tem um desempenho bastante superior ao longo dos 25 pedidos. Percebemos também que todos os sistemas considerados mantêm, ou diminuem ligeiramente, o seu tempo de execução ao longo dos diferentes pedidos, com a exceção do RedisGraph que diminui o seu tempo de execução. Podemos também para este *dataset* fazer uma análise comparativa de resultados, com base na figura 4.15, onde observarmos que podemos ordenar a *performance* da seguinte maneira: Neo4J e Neo4J 2 (com uma melhoria relativa ao *baseline* de 78.53%) e RedisGraph e RedisGraph 2 (com uma melhoria relativa ao *baseline* de 21.24%). De realçar ainda, com base no desvio padrão (DP), a estabilidade dos pedidos é superior para todos os SUT (valores entre 16% a 19%) relativamente ao *baseline* (aproximadamente 7%).

Para avaliar o impacto da modelação na resolução deste problema foram desenvolvidos o intervalo de gráficos 4.16 a 4.18. Estes gráficos permitem avaliar para as duas modelações e para cada SUT a *performance* relativa ao sistema *baseline* nos *diferentes datasets*. Para este efeito foi considerada a percentagem de sucesso, *PS*. A *PS* pode ser consultada no intervalo de figuras 4.10 a 4.15 e é calculada com base na percentagem de melhoria relativamente ao sistema *baseline*. Com base no *PS* de cada SUT foi feita a diferença entre a *PS* do sistema modelado como tabela intermédia como nó — abordagem #1 — e o sistema tabela intermédia como relação — abordagem #2. Esta análise permite perceber a tendência e o sinal da *performance* das diferentes abordagens em função do tamanho de dados. O sinal informa-nos sobre qual é o melhor tipo de

modelação (se for positivo significa que a 1.^a abordagem é superior). A tendência informa-nos sobre o comportamento do sinal em função do tamanho dos dados — se for crescente significa que o grau de adequação da modelação #1 cresce com o aumento dos *dataset*, se for decrescente significa que o grau de adequação da modelação #2 cresce com o aumento do *dataset*. De notar que esta abordagem apenas tem o objetivo de ilustrar uma possível tendência da *performance* em função do volume de dados, não de formalizar uma regressão linear entre as variáveis.

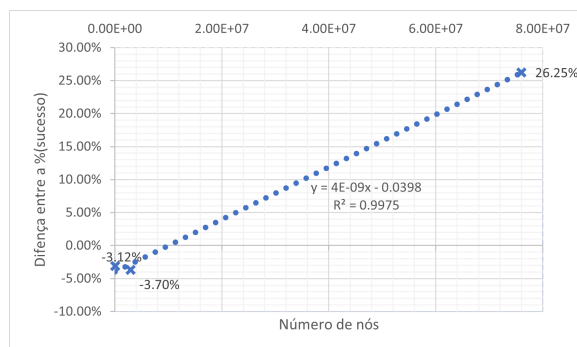


Figura 4.16: Diferença de desempenho dos diferentes tipos de modelação em função do tamanho de dados — Neo4J

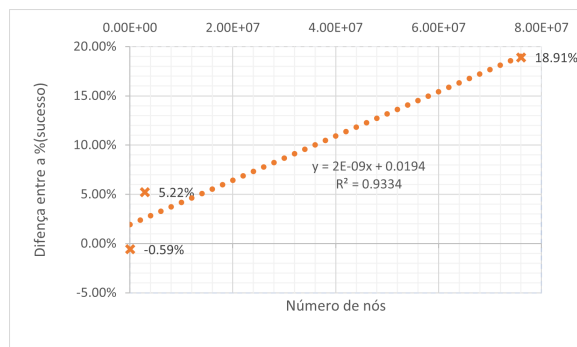


Figura 4.17: Diferença de desempenho dos diferentes tipos de modelação em função do tamanho de dados — RedisGraph

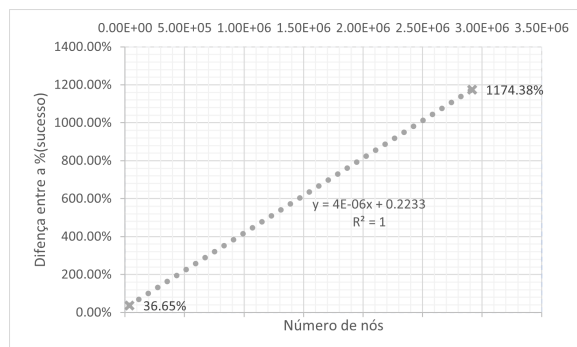


Figura 4.18: Diferença de desempenho dos diferentes tipos de modelação em função do tamanho de dados — ArangoDB

Analisando a figura 4.16 — para o Neo4J — podemos afirmar que a modelação #2 é superior para os dois primeiros *datasets* — na ordem dos 3%, e devido ao declive da reta e como o sinal do dataset 3 é positivo podemos afirmar que tendencialmente o ganho de usar a modelação #1 aumenta com o volume de dados. De realçar ainda que o desvio padrão para o Neo4J 2 foi de cerca de 69% do seu valor médio, o que é uma anomalia quando comparado com os restantes resultados, podendo ser possível argumentar que, excluindo esta anomalia, o declive da reta seria menos íngreme.

Analisando a figura 4.17 — para o RedisGraph — podemos afirmar que a modelação #2 é superior apenas para o primeiro *dataset* — na ordem dos 0.5%. Devido ao declive da reta podemos afirmar que tendencialmente o ganho de usar este tipo de modelação diminui com o aumento do volume de dados.

Analisando a figura 4.18 — para o ArangoDB — podemos afirmar que a modelação #1 é superior para todos os *datasets*, e devido ao declive da reta podemos afirmar que tendencialmente o ganho de usar este tipo de modelação aumenta com o volume de dados — na ordem dos 1100%.

4.2.6 Epílogo

O processo empregado e o resultado do mesmo permitem salientar algumas propriedades do sistema em geral e em particular que precisam de ser justificadas. Em geral, podemos realçar as seguintes observações:

- Relativamente aos diferentes *Datasets* podemos verificar pela tabela 4.2 um número crescente no volume de dados entre o *Dataset 1* ao *Dataset 3*. Apesar de se tratar de *Datasets* diferentes, reparamos que a maior frequência absoluta de entradas nos diferentes sistemas está relacionada com o registo de operações e com operações sobre o material validando a ideia da importância e centralidade desta entidade — algo estabelecido em capítulos anteriores.
- Relativamente ao processo de Escolha das Implementações podemos afirmar que, como descrito nas conclusões do estado da arte, o aparecimento de sistemas GDBs emergentes devem ser um fator a ter em conta, algo validado através do *RedisGraph* que teve resultados bastante satisfatórios demonstrados na secção 4.2.5 e não estava representado nem no catálogo *DB-Engines* nem, significativamente, ao nível de literatura.
- Relativamente à robustez no processo de carregamento de dados é de notar que existiram alguns comportamentos não esperados descritos no fim da subsecção 4.2.4 que poderão estar relacionadas com a novidade do ecossistema. De salientar que devido às ferramentas disponíveis e à sua comunidade o Neo4j foi a tecnologia que mais se demonstrou eficaz no processo de resolução de imprevistos. Relativamente ao resultado desta migração de dados conseguimos reparar as propriedades dos *datasets*. Com o auxílio da informação presente na tabela 4.9 percebemos que, como seria de esperar, o número de nós e relações aumenta com o volume dos *datasets*, mas propriedades como a densidade do grafo diminuem. Por

outro lado, propriedades como densidade média e o máximo da densidade média aumentam. O que implica que, no domínio de *manufacturing*, o acréscimo do volume de dados torna o grafo globalmente menos denso, mas como o modelo de dados está orientado para uma entidade principal (o Material) e as operações sobre a mesma, aquilo que será a sua densidade local aumenta com o aumento do volume de dados.

- Relativamente ao processo de *benchmarking* também percebemos que todos os sistemas, na sua generalidade, diminuem o seu tempo de execução ao longo dos diferentes pedidos, isto poderá ser justificado pelo facto quando um *workflow* de leitura é executado pela primeira vez a informação não se encontra em *cache* — *cold cache* — o que significa que os SGBD têm de ler a informação do disco — o que é mais dispendioso do ponto de vista de *performance*. Quando este *workflow* é executado múltiplas vezes, os dados já se encontram em *cache*, o que implica que o tempo de execução seja inferior. [56]

Em particular e como forma de rematar o processo respondendo às questões presentes na tabela 4.1 foram agrupadas possíveis justificações nas seguintes subsecções.

4.2.6.1 Q1: O contexto de manufacturing é otimizado com o uso de GDBs?

Como referido anteriormente, por otimização do contexto de manufacturing estamos a referir-nos à melhoria de *performance* nas operações *Material History* e *Material Geneology*. Podemos afirmar que H1 é válida quando com a utilização das GDBs o contexto de manufacturing é otimizado. Para testar esta hipótese, foi considerado um plural de GDBs, o Neo4J, o RedisGraph e o ArangoDB bem como um conjunto de três *datasets* com um volume de dados crescente.

Relativamente à *Material History* reparamos que, para os primeiros 2 *datasets*, independentemente do tipo de modelação, os resultados de todos os SUT têm uma *performance* superior ao sistema original, com a exceção do ArangoDB. De notar que a percentagem de sucesso relativo relativamente ao sistema origem diminui à medida que o volume de dados aumenta. Para o *Dataset* de maior volume, o *Dataset* 3, apenas uma implementação teve um tempo de execução médio inferior, o Neo4J 2. Esta evidência mostra a robustez dos sistemas relacionais quando tratamos de grandes volumes de dados.

Relativamente à *Material Geneology* reparamos que, independentemente do tipo de modelação e para os três *datasets*, os resultados de todos os SUT têm uma *performance* superior ao sistema original, tendo uma percentagem de sucesso relativo relativamente ao sistema origem na ordem dos 90% para dos dois primeiros *datasets*.

Tendo em conta estes argumentos e a definição de otimização do contexto de manufacturing, podemos afirmar que o uso de GDBs otimiza o contexto de manufacturing, validando assim a hipótese positiva.

4.2.6.2 Q2: As diferenças encontradas entre *Native GDBs* e *Non-Native GDBs* são evidenciadas no contexto de *manufacturing*?

Como vimos anteriormente, o Neo4J e o RedisGraph são *Native GDBs* e o ArangoDB é uma base de dados multimodelo orientada a documentos e, portanto, uma *Non-Native GDB*. Analisando os resultados do processo de *benchmarking*, presentes na subsecção 4.2.5, percebemos que o ArangoDB para Datasets relativamente pequenos tem uma *performance* muito satisfatória e próxima dos resultados do RedisGraph que é um indicador muito positivo porque o RedisGraph é conhecido exatamente por ter excelentes resultados para *datasets* relativamente pequenos [12], [82] devido à sua arquitetura orientada a operações *in-memory*. Com o acréscimo do volume de dados temos uma degradação do sistema ArangoDB tanto ao nível de *performance* como ao nível de confiabilidade, como podemos analisar pelos artefactos do processo de *benchmarking*, presentes na subsecção 4.2.5.

Para perceber este comportamento temos de perceber o funcionamento interno dos diferentes sistemas. O ArangoDB guarda os seus documentos num formato chamado *VelocityPack* que é uma versão compactada de documentos JSON. Os documentos são guardados em *collection* com um atributo chave — chamado “*_key*”. Para permitir o suporte de grafos o ArangoDB usa o conceito de *edges*, modelados em *VelocityPack*, que permitem fazer a referência entre as coleções via as “*_keys*”, com o uso das *keywords* “*_from*” e “*_to*”, reduzindo a tempo de travessia em *workflows* de grafos. Isto implica que, tanto os nós, como os *edges* sejam guardados em documentos diferentes, em espaços de memória diferentes, otimizados via *indexes* baseados no par “*from*”-“*to*”. Por outro lado, as *Native GDBs*, como vimos no estado da arte, possuem a propriedade de *index-free adjacency*. Tal facto diminui a *performance* de travessias do grafo para múltiplos nós [7], podendo mesmo ocorrer problemas que ponham mesmo em causa o plano de execução como sucedido para o ArangoDB.

Tendo em conta os argumentos apresentados podemos afirmar, que para o problema e conjunto de dados em questão, o uso de *Native GDBs* tem um grau de adequação superior e, por isso, as diferenças entre estas duas estratégias são evidenciadas no contexto de *manufacturing*, validando assim a hipótese positiva.

4.2.6.3 Q3: As propriedades do grafo têm impacto no contexto de *manufacturing*?

Como forma de descrever os grafos foi, para cada *Dataset*, calculado um conjunto de informação relacionada com os diferentes grafos que pode ser consultada na tabela 4.9. No decorrer do processo também foi feito o *benchmarking* dos diferentes sistemas e das diferentes modelações. Para perceber se, de facto, existe alguma relação entre as propriedades e os resultados do processo foi empregue um processo de análise de correlação. A correlação é um termo estatístico utilizado para descrever a relação entre duas variáveis, por exemplo, ao afirmarmos que as pessoas com mais Altura têm mais Peso, estamos a dizer que existe um elevado índice de correlação entre Altura e Peso. Quando temos variáveis com elevados índices de correlação, significa que temos uma variável dependente e uma independente. Aplicando a correlação de Pearson a todos os pares

de variáveis podemos gerar um *Heatmap* que representa graficamente a correlação entre variáveis, representado pela figura 4.19. A mesma informação também pode ser analisada paralelamente e com maior detalhe pela matriz de correlação presente na tabela 4.10.

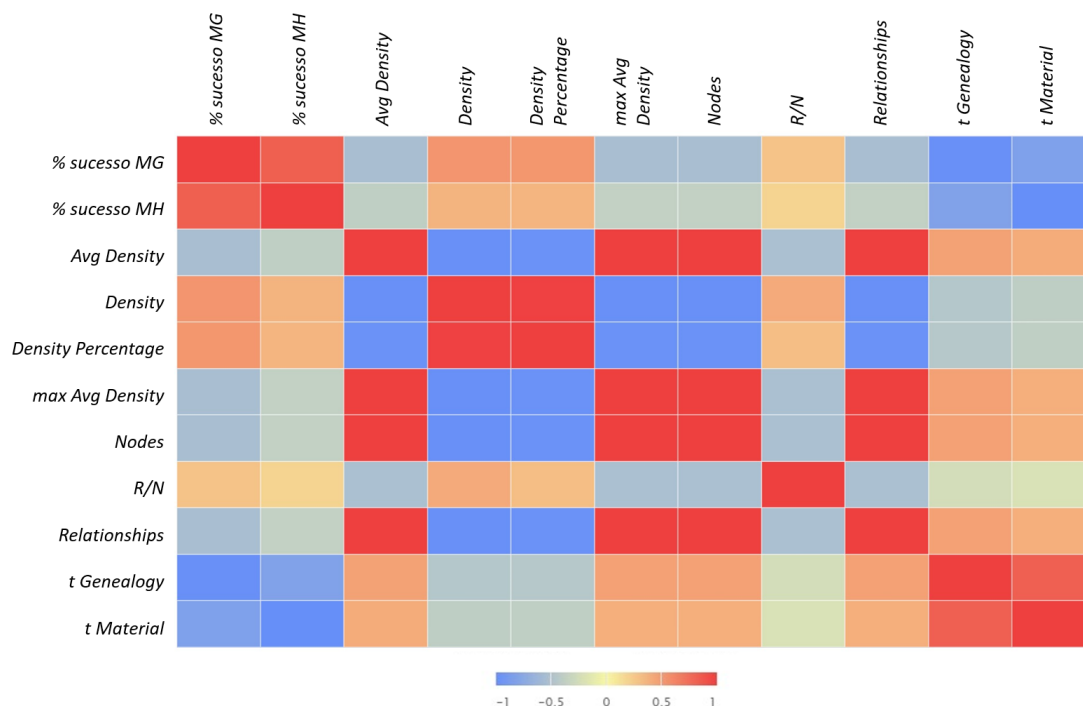


Figura 4.19: Heatmap de correlação

Attributes	%sucesso MG	%sucesso MH	Avg Density	Density	Density Percentage	MaxAvg Density	Nodes	R/N	Relationships	t Genealogy	t Material
% sucesso MG	1.00	0.83	-0.54	0.54	0.53	-0.54	-0.54	0.29	-0.54	-0.99	-0.83
% sucesso MH	0.83	1.00	-0.39	0.37	0.36	-0.37	-0.37	0.19	-0.37	-0.82	-1.00
Avg Density	-0.54	-0.39	1.00	-0.99	-0.97	1.00	1.00	-0.53	1.00	0.47	0.41
Density	0.54	0.37	-0.99	1.00	0.99	-0.99	-0.99	0.42	-0.99	-0.46	-0.39
Density Percentage	0.53	0.36	-0.97	0.99	1.00	-0.97	-0.97	0.31	-0.97	-0.45	-0.38
max Avg Density	-0.54	-0.37	1.00	-0.99	-0.97	1.00	1.00	-0.53	1.00	0.47	0.39
Nodes	-0.54	-0.37	1.00	-0.99	-0.97	1.00	1.00	-0.53	1.00	0.47	0.39
R/N	0.29	0.19	-0.53	0.42	0.31	-0.53	-0.53	1.00	-0.53	-0.25	-0.21
Relationships	-0.54	-0.37	1.00	-0.99	-0.97	1.00	1.00	-0.53	1.00	0.47	0.39
t Genealogy	-0.99	-0.82	0.47	-0.46	-0.45	0.47	0.47	-0.25	0.47	1.00	0.82
t Material	-0.83	-1.00	0.41	-0.39	-0.38	0.39	0.39	-0.21	0.39	0.82	1.00

Tabela 4.10: Matriz de correlação

Para analisar os valores apresentados na tabela 4.10 foi utilizado o *Cohen's Standard*, presente na tabela 4.11, para interpretar os coeficientes de correlação relativamente ao grau de associação entre as variáveis.

Para perceber se as propriedades do grafo têm impacto na performance dos diferentes GDBs faz sentido avaliar a correlação entre as variáveis de performance (Tempo de execução da *Material Genealogy* — t Genealogy —, Tempo de execução da *Material History* — t Material —, Percentagem de sucesso da *Material Genealogy* — % sucesso MG — e Percentagem de sucesso da *Material History* - % sucesso MH) e as variáveis de características do grafo (*Avg Density*, *Density*, *Density Percentage*, *R/N*, *Nodes* e *Relationships*).

Grau de correlação	Coefficientes de Correlação
<i>Pequeno</i>	[0.10 - 0.30[
<i>Médio</i>	[0.30 - 0.50[
<i>Forte</i>	[0.50 - 0.90[
<i>Muito Forte</i>	[0.90 - 1.00]

Tabela 4.11: *Cohen's Standard* para a interpretação de coeficientes de correlação

Relativamente à “% sucesso MG” reparamos que as variáveis *Avg Density*, *Density*, *Density Percentage* e *Max Avg Density* têm uma correlação forte. Relativamente à “% sucesso MH”, à “*t Geneology*” e à “*t Material*” reparamos que as variáveis *Avg Density*, *Density*, *Density Percentage* e *Max Avg Density* tem uma correlação média.

Estes resultados permitem concluir que, de facto, as propriedades do grafo têm impacto na *performance* das GDBs. Isto permite afirmar que quanto mais denso for o grafo, melhor serão os indicadores “% sucesso”, validando assim a hipótese positiva. Isto é especialmente verdade para operações orientadas a grafos, como a *Material Geneology* que representa a hierarquia de materiais. De notar que as variáveis de “% sucesso” têm uma correlação negativa com o número de nós, o que poderá ser um indicador que para grandes volumes de dados a “% sucesso”, que apesar de ser maioritariamente positiva para os casos estudados, tem tendência a diminuir.

4.2.6.4 Q4: Qual é a importância da modelação na performance das GDBs?

Para testar o impacto que a modelação tem sobre a *performance* foram desenvolvidas duas abordagens relativamente à modelação de tabelas intermédias: (#1) Tabela intermédia como nó e (#2) Tabela intermédia como relação — ilustradas pela figura 4.2. Devido à natureza das operações esta diferença só é evidenciada na operação *Material History* que é a única que efetua travessia por tabelas intermédias. Devido ao tamanho relativo do desvio padrão para o *Dataset 3* - repare-se que o desvio padrão passa em média de aproximadamente 18% nos *Datasets 1* e *2* para 38% no *Dataset 3* - podemos dividir a análise em duas partes: uma primeira que considera os dois primeiros *datasets*, com maior validade estatística, e uma segunda que considera o terceiro, com menor validade estatística.

Para a primeira parte da análise e com o auxílio das figuras 4.16 e 4.17 percebemos que para as *Native GDBs* a abordagem #2 é equivalente à abordagem #1 para os dois primeiros *datasets*, com uma diferença de *performance* residual inferior a 6%. Para as *Non-Native GDBs* e com o auxílio da figura 4.18 podemos afirmar que a abordagem #1 é superior para os dois primeiros *datasets*. Estes dois factos poderão ser justificados pelas diferenças de arquitetura dos dois tipos de sistemas. Para as *Native GDBs* o facto de o tipo de modelação ser negligenciável pode ser explicado, mais uma vez, pela propriedade de *index-free adjacency*, que torna o custo de navegar entre nós relativamente constante, $O(1)$ do ponto de vista de complexidade temporal [67], com o aumento do volume de dados. O que, na prática, implica que o facto de existir mais um nó para a abordagem #1 é negligenciável do ponto de vista de complexidade computacional. No caso das *Non-Native GDBs*, e em particular para o ArangoDB, a evidência que a modelação #1 é superior

pode ser explicada com a implementação que o sistema faz dos *Edges*. Como forma de competir com a propriedade de *index-free adjacency* os *Edges* no ArangoDB funcionam como *indexes*, o que melhora o tempo de execução na travessia de nós. Devido à informação aqui descrita podemos alegar que quanto mais informação tiverem estes *indexes*, pior será o particionamento dos dados, maior será o tempo de carregamento do *index* e, portanto, pior será a sua *performance*.

Para a segunda parte da análise e com o auxílio das figuras 4.16 a 4.17 percebemos que a abordagem #1 é superior para os sistemas Neo4J e RedisGraph. Devido ao declive da reta podemos, numa primeira análise, argumentar que a abordagem #1 é tendencialmente melhor quando o volume de dados aumenta, mas na realidade esta ilação é questionável devido à perturbação dos resultados demonstrada pelo valor do desvio padrão e ao intervalo de volume de dados entre o *Dataset 3* e *Dataset 2*, especialmente para o Neo4J. Esta evidência pode ser explicada em duas partes. A primeira está relacionada com a propriedade de *index-free adjacency*, que torna o custo de navegar entre nós relativamente constante com o aumento do volume de dados que, na prática, implica que o facto de existir mais um nó para a abordagem #1 é negligenciável do ponto de vista de complexidade computacional. A segunda parte está relacionada com o suporte a filtros para as propriedades dos *nodes* ser mais otimizado do que para o filtro de *relations*, o que implica que quando informação relevante é guardada na relação o seu filtro, a grande escala, é mais demorado que filtro por propriedades dos nós [67]. Como forma de colmatar este comportamento, foi recentemente introduzido o suporte de *indexes* para *relations*, no *RedisGraph 2.8.0*¹⁰ e no *Neo4J 4.3.0*¹¹.

Devido aos argumentos apresentados anteriormente podemos afirmar globalmente que, de facto, a modelação tem impacto na *performance*, validando assim a hipótese positiva. No entanto, não podemos deduzir nenhuma heurística generalizável de modelação, salientando a ideia frisada no estado da arte de que “Para diferentes problemas diferentes soluções”, no caso diferentes modelações. Como observamos anteriormente, conforme o tipo de sistema em uso e dependendo do tipo de *queries* executadas sobre a informação, a melhor modelação dependerá, por exemplo, do volume de dados. Por outro lado, e não diretamente relacionado com *performance* de execução, uma vantagem de usar uma *Graph Database* é a proximidade do modelo de dados com o mundo real pelo que faz todo o sentido, para melhorar questões relacionadas com manutenção e coerência da informação, tomar uma abordagem baseada no conceito de *Domain Driven Design* [29].

4.2.7 Escolha Final

Tendo em conta o processo até aqui descrito, bem como os argumentos apresentados, foi tomada a decisão de utilizar o *Neo4J* seguindo a estratégia de modelação #1, representada pela figura 4.2.

¹⁰<https://redis.com/blog/redisgraph-2-8-is-generally-available/>

¹¹<https://neo4j.com/product/neo4j-graph-database/whats-new-in-neo4j-4-3/>

Capítulo 5

Integração da Solução

Este capítulo descreve a abordagem de integração do sistema escolhido no capítulo anterior no MES da Critical Manufacturing. A secção 5.1 apresenta o contexto bem como as decisões tomadas no processo de integração do Neo4J com o MES da Critical Manufacturing. De seguida a secção 5.1 apresenta uma vista arquitetural da solução empregada, com uma descrição de cada componente interveniente. Por fim, na secção 5.3 são apresentadas as limitações da solução de integração apresentada.

5.1 Processo de Integração

Como foi descrito no capítulo Descrição de Problema, secção 3.2, os dados utilizados nas operações de *reporting Material History* e *Material Geneology* têm origem através de operações no sistema MES da Critical Manufacturing que são posteriormente replicados para a base de dados de ODS, que serve a jusante como fonte de dados para estes reports. Este fluxo de dados funciona por *SQL Server Jobs*, como tal, foi definido como requisito no capítulo de Desenvolvimento, secção 4.2.2, que a integração com um novo sistema de GDB seguisse uma abordagem desacoplada e agnóstica do ponto de vista tecnológico com o uso do Kafka — que já é utilizado atualmente para fazer *streaming* de eventos do MES para *workflows* de *analytics*. Como observado pelo capítulo 4 reparamos que o Neo4J tem integração, *open-source* e *off-the-shelf*, com Kafka através do *Kafka Connect* ¹ e do *Neo4j-Streams Plugin* ², cujas diferenças podem ser avaliadas na tabela 5.1.

¹<https://www.confluent.io/hub/neo4j/kafka-connect-neo4j>

²<https://neo4j.com/labs/kafka/>

	Kafka Connect	Neo4j-Streams
Prós	✓ É instalado e executado numa instância diferente do Neo4J. O que implica que não tem impacto na <i>performance</i> do Neo4J; ✓ Maior configurabilidade e possibilidade de estender o comportamento do consumidor; ✓ Pertence ao ecossistema da Confluent.	✓ É instalado e executado no Neo4J; ✓ Funciona como <i>Consumer</i> e <i>Producer</i> de tópicos de <i>Kafka</i>; ✓ Implementa o padrão <i>Change Data Capture</i> que permite espoletar eventos quando uma mensagem é recebida via <i>Kafka</i>; ✓ Melhor performance na comunicação com Neo4J.
Contras	✗ Não implementa o padrão <i>Change Data Capture</i> e funciona como um cliente normal para o Neo4J; ✗ Apenas permite escrever para o Neo4J e não publicar para o <i>Kafka</i>; ✗ Pior performance na comunicação com Neo4J.	✗ Uso de memória e CPU do servidor Neo4J.

Tabela 5.1: Prós e Contras: Integrações Kafka [46]

Analisando a tabela 5.1 percebemos que as principais diferenças estão relacionadas com a independência, entre o *Kafka* e o *Neo4J*, e a performance — tanto no processo de streaming de eventos como nas operações sobre a base de dados.

A independência dos sistemas está relacionada com a instância onde o *consumer*³ é instalado, que tem impacto nas características de disponibilidade e confiabilidade do sistema — pois pode funcionar como um *single point of failure* — e na independência no que toca a autonomia relativamente a operações como *update* e *restart*. A independência também tem impacto no desempenho, no sentido em que se o *consumer* estiver instalado na instância do Neo4J, como no caso do Neo4J-Streams, isto vai ter impacto ao nível de uso de memória e uso de CPU, mas como a integração com o Neo4J é nativa isto implica uma performance superior na interação com o Neo4J. Se o *consumer* estiver instalado numa instância externa ao Neo4J, como no caso do Kafka Connect, a interação com o Neo4J é feita através da interface *bolt*⁴ que é menos eficiente.

Devido aos argumentos apresentados, percebemos que não existe uma solução melhor que a outra, apenas soluções com contrapartidas diferentes, pelo que a decisão foi desempatada pela existência mecanismo de *Change Data Capture*⁵ implementado no *Neo4J-Streams* que permite espoletar eventos sobre o *Neo4J* quando uma mensagem é emitida pelo *Kafka*, simplificando assim o processo de integração.

³<https://docs.confluent.io/platform/current/clients/consumer.html>

⁴<https://neo4j.com/docs/java-reference/current/java-embedded/bolt/>

⁵<https://www.redhat.com/en/topics/integration/what-is-change-data-capture>

5.2 Arquitetura de Integração

Com base na descrição apresentada anteriormente, bem como a arquitetura apresenta no capítulo Descrição de Problema, secção 3.2, podemos concluir que existem então três componentes, representados pelo diagrama de componentes presente na figura 5.1. A título de exemplo e de forma semelhante à abordagem tomada pelo capítulo de Desenvolvimento, secção 4.2.4, demonstraremos o processo de integração usando como exemplo as entidades *Flow* e *Step* bem como a relação entre estas.

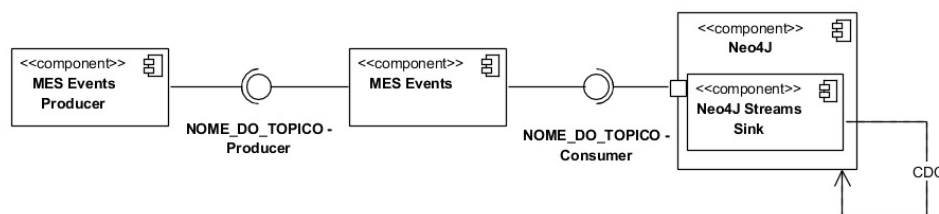


Figura 5.1: Arquitetura de Integração — Diagrama de Componentes

O Componente *MES Events* foi desenvolvido internamente pela Critial Manufacturing e é responsável por emitir todas as transações que ocorrem num sistema MES. Sempre que uma transação é efetuada, este componente tem a responsabilidade de emitir as alterações ao modelo de dados para um tópico de Kafka em formato de documento JSON, onde cada *key* corresponde à tabela alterada e o *value* à nova informação de uma determinada instância dessa tabela. Um exemplo de um tópico pode ser observado pela figura 5.2 que corresponde a uma transação sobre a relação Flow-Step.

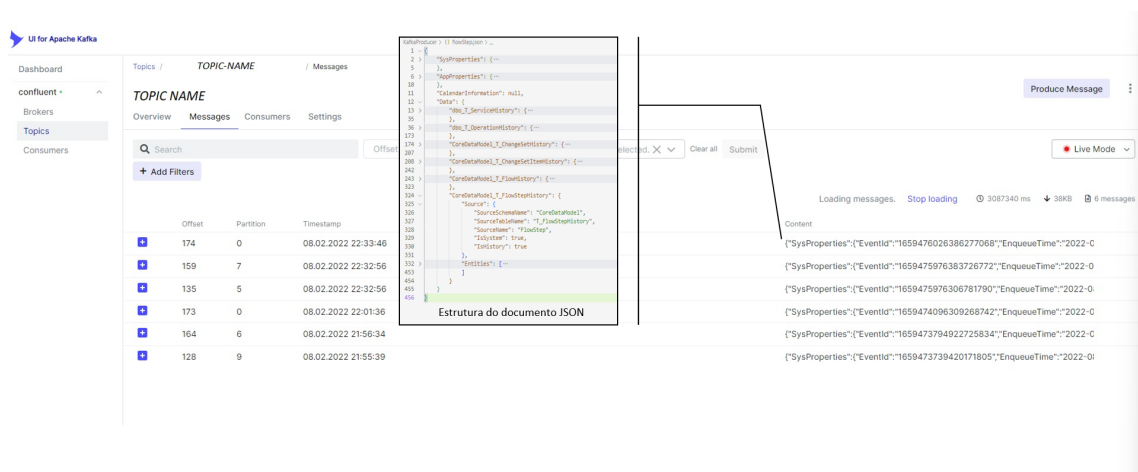


Figura 5.2: Mensagens do MES Events — Operação sobre relação Flow-Step

O Componente *Neo4j Streams Sink* é um *plugin* desenvolvido pelo *Neo4J* que permite estar à escuta de determinados tópicos *Kafka*, e conforme esse tópico despoletar determinadas ações numa determinada base de dados *Neo4J* — Implementação do padrão *Change Data Capture*. Este comportamento é efetuado através do ficheiro de configurações *stream.conf* (disponível em *PastaDeInstalaçãoBaseDeDados > conf > streams.conf*) e a sua implementação pode ser observada pela figura 5.3.

```

streams.conf
1  kafka.bootstrap.servers=enderecobootstrapserver1:porta,enderecobootstrapserver2:porta
2  streams.sink.enabled=true
3  streams.sink.topic.cypher.NOME_DO_TOPICO=
4      UNWIND event.data.Data as tables
5      WITH apoc.map.values(tables, keys(tables)) as items
6      UNWIND items as item
7      RETURN
8  CASE item.Source.SourceTableName
9      WHEN "T_FlowHistory" THEN "MERGE (n:Flow {id: item.Entities[0].FlowId}) ON CREATE SET n += item.Entities[0]"
10     WHEN "T_FlowStepHistory" THEN "MERGE (n:FlowStep {id: item.Entities[0].FlowStepId}) ON CREATE SET n += item.Entities[0]"
11     WHEN "T_StepHistory" THEN "MERGE (n:Step {id: item.Entities[0].StepId}) ON CREATE SET n += item.Entities[0]"
12     ... // Outras Tabelas
13     ELSE "MERGE (n:KafkaConnectError {id: event.key}) ON CREATE SET n += event.properties"
14     END as RESULT
15

```

Figura 5.3: Configuração do *Neo4j Streams Sink* — Operação sobre relação Flow-Step

Analisando a figura 5.3 podemos afirmar que apenas temos de definir os *kafka.bootstrap.servers*⁶, que correspondem aos endereços dos *Brokers* de *Kafka*, definir *streams.sink.enabled=true*, que configura o estado do *plugin Neo4j Streams Sink* e definir o código a ser executado na leitura de uma mensagem, com a seguinte estrutura: *streams.sink.topic.cypher.<NOME-DO-TOPICO>=<QUERY-EM-CYPHER>* — onde o *<NOME-DO-TOPICO>* corresponde ao tópico *kafka* que queremos que o *Neo4J* consuma e a *<QUERY-EM-CYPHER>* o código em *CYPHER* que queremos que seja executado.

O Componente *Neo4J* é o alvo deste processo de replicação. O resultado deste processo de replicação pode ser analisado pelas figuras 5.4 e 5.5, onde é possível observar o grafo *Step-Area* antes e depois do processo de replicação.

⁶<https://kafka.apache.org/documentation/>

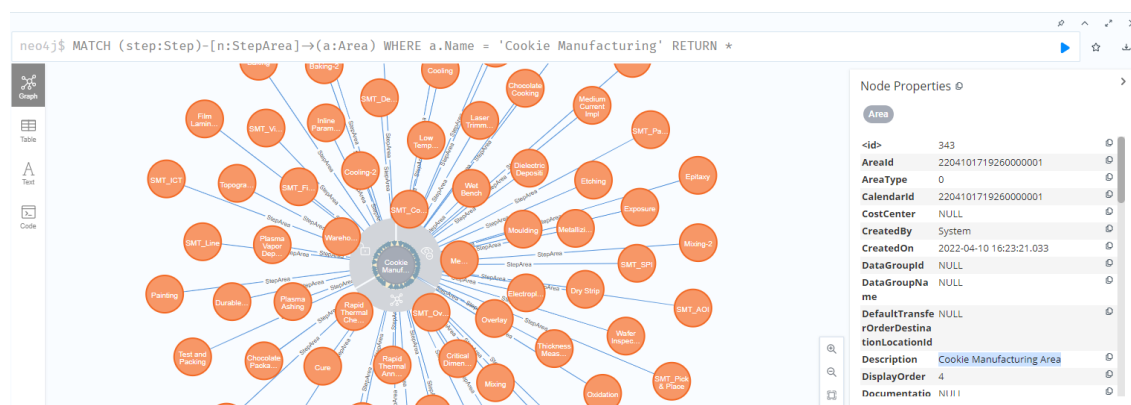


Figura 5.4: Grafo Step-Area — Descrição da Area antes da replicação

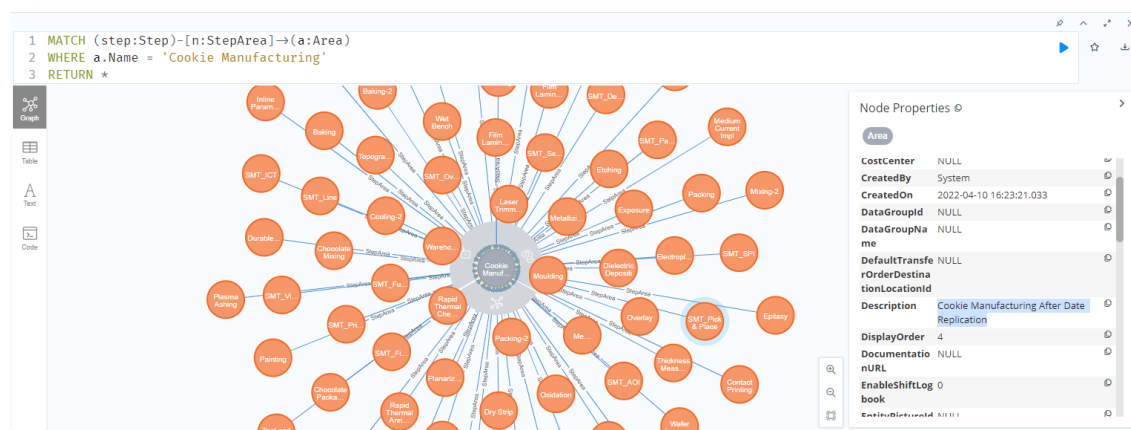


Figura 5.5: Grafo Step-Area — Descrição da Area depois da replicação

5.3 Limitações

Atualmente as operações de *Material History* e *Material Geneology* descritas nas seções anteriores são exclusivamente utilizadas para efeitos de *reporting*, como referido no capítulo 3.2. Os reports no sistema *MES* da *Critical Manufacturing* têm como fonte de dados a base de dados de ODS, mas são materializados através do *Microsoft SQL Server Reporting Services (SSRS)* ⁷ que permitem mapear determinadas queries, como a *Material History* e *Material Geneology*, em forma de *reports*. O SSRS é uma tecnologia limitada ao ecossistema Microsoft, e em particular ao *SQL Server*, pelo que a integração com sistema externos, como sistemas NoSQL, não é possível [77].

⁷<https://docs.microsoft.com/en-us/sql/reporting-services/create-deploy-and-manage-mobile-and-paginated-reports?view=sql-server-ver16>

Capítulo 6

Conclusões e Trabalho Futuro

Este capítulo tem a responsabilidade de fazer um sumário da dissertação. A secção 6.1 apresenta uma visão global do trabalho desenvolvido, onde podemos encontrar a descrição do processo, as essenciais conclusões que tiveram origem ao longo do mesmo, bem como a interpretação do autor sobre estas. Para terminar é feita uma retrospectiva ao nível de implicações das conclusões e do processo empregado ao longo da dissertação. A secção 6.2 tem a responsabilidade de compactar as principais conclusões da dissertação. De seguida a secção 6.3 apresenta aquilo que são as principais contribuições do presente documento. A secção 6.4 apresenta as limitações do processo demonstrado ao longo da dissertação. Por fim a secção 6.5 apresenta aquilo que poderão ser possíveis respostas às limitações levantadas na secção anterior e enumera potenciais diretrizes para trabalho futuro.

6.1 Discussão

À medida que a humanidade avança na construção da visão de Indústria 4.0, novos requisitos, relacionados com os 5 V's da *Big Data*, vão começar a aparecer que têm de ser suportados pelos sistemas de *manufacturing*, como o MES da *Critical Manufacturing*. Como estratégia de disrupção surgem as *Graph Databases*, pelo que surge a questão: “poderão as *Graph Databases* otimizar o contexto de *Manufacturing*?”. Para responder a esta questão foi consultado o estado da arte que nos dá uma resposta inconclusiva no sentido em que afirma que para diferentes problemas devemos tomar diferentes abordagens e para escolher um determinado sistema de base de dados temos de elaborar um processo *Exploration Based* — o que é bastante moroso. Posto isto, e devido à arquitetura das *Graph Databases*, foi levantada a necessidade de tentar arranjar heurísticas como forma de otimizar os processos de seleção, de onde surgiu a questão “Que impacto têm as propriedades do grafo no sucesso de implementação de uma *Graph Database*?”. No processo de investigação destas questões reparamos que existem dois tipos de *Graph Databases* — as *Native* e as *Non-Native* — o que torna a resposta à primeira pergunta ainda mais exigente, pelo que surgiu a necessidade de perceber se existe diferença relativamente à adequação das duas abordagens.

Adicionalmente podemos alegar que no corpo de conhecimento do estado da arte nunca era abordado o tipo de modelação nem as escolhas de modelação tomadas sempre que era desenvolvido um trabalho comparativo entre diferentes SGBD, apesar do facto de existirem trabalhos que nos indicam que a modelação de um SGBD tem impacto significativo no desempenho do sistema — de realçar que estes trabalhos, até à data, não consideram o caso das *Graph Databases*.

Como forma de responder a estas questões foi empregue um processo *Exploration Based*, cujos resultados são apresentados no capítulo 4, e que nos indicam que o seguinte:

- (i) O uso de GDBs otimiza o contexto de *manufacturing*;
- (ii) Devido à sua arquitetura, o uso de Native GDBs, para este problema, tem um grau de adequação superior;
- (iii) As propriedades do grafo, têm impacto na performance das GDBs, tanto mais quanto mais orientada for a operação a grafos;
- (iv) Para diferentes problemas devemos utilizar diferentes estratégias de modelação.

Na opinião do autor isto deve-se a algo que foi observado no decorrer do capítulo 3 e comprovado no capítulo 4 onde vimos que o domínio de *manufacturing* é baseado numa entidade principal, o Material, e que a maior parte das operações incidem sobre este tornando o modelo de dados bastante coeso e conectado, o que implica que valor da informação está muitas vezes relacionado com as relações que o material tem com entidades terceiras do sistema e não com o conteúdo das suas propriedades propriamente ditas. A juntar a esta característica as operações analisadas ao longo deste trabalho também são operações orientadas a grafos, no sentido em que a estratégia de *query* pode seguir uma abordagem em grafo — note que, simplificando, a *Material Geneology* é a representação hierárquica de um material e o *Material History* é o histórico do material no que toca às relações com entidades terceiras do MES. Em linha com as conclusões da bibliografia, com algumas exceções, é que para este tipo de domínios o uso de *Graph Databases* pode ser benéfico, algo que foi corroborado por este trabalho, como podemos concluir pelo capítulo 4.

Este processo *Exploration Based* promove um novo ponto de vista naquilo que é a adoção de um novo sistema de base de dados. Com o processo aqui demonstrado conseguimos comprovar que as propriedades dos grafos têm um impacto direto naquilo que é o sucesso da aplicação das *Graph Databases*, pelo que nos permite utilizar este facto como heurística em processos futuros.

Numa perspetiva crítica e em retrospectiva, podemos afirmar que existem pontos do processo descrito que podem ser melhorados. Em primeiro lugar é importante perceber que o desenho deste processo está orientado para a indústria e visa principalmente responder à questão “poderão as *Graph Databases* otimizar o contexto de *Manufacturing*?”, otimizando assim operações-chave num contexto real, pelo que faz sentido utilizar como fonte de dados o contexto de mundo real. O facto de usarmos dados de um contexto real implica que não podemos medir as diferentes variáveis de forma desacoplada isto porque quando afirmarmos que propriedades como a *Graph Density*

têm impacto na aplicação de Graph Databases a verdade é que para validar este conceito na sua totalidade teríamos de ter uma distribuição normal de diferentes valores para a *Graph Density*, pelo que fazia sentido, por exemplo, utilizar um gerador de grafos agnóstico a qualquer domínio. Tal facto poderá implicar que, apesar de a análise deste problema nos permitir deduzir um padrão, esta solução pode estar sobre-ajustada (*overfitted*) para o contexto de *manufacturing*. Numa perspetiva mais técnica reparamos que as *Graph Databases*, em particular as *Non-Native Graph Databases*, tiveram problemas de funcionamento no processo de *benchmarking*, apesar de isto ser um indicador de maturidade e robustez do sistema, este facto levanta a questão: que impacto teve o dispositivo onde foram executados os testes? Teríamos os mesmos resultados, e por consequência as mesmas conclusões, se tivesse sido utilizado um servidor aplicacional desenvolvido para o efeito?

Em retrospectiva e como forma de rematar o processo considero importante realçar a importância de utilizar dados reais para resolver problemas concretos, mas considero que seria pertinente estudar as propriedades do grafo, ou outras, com base em *datasets* gerados de forma agnóstica, com origem em geradores, por exemplo.

6.2 Conclusão

Como evidenciado ao longo do documento, na construção da visão de Indústria 4.0, novos requisitos vão começar a aparecer que têm de ser suportados pelos sistemas de manufacturing. Estes requisitos estão relacionados, principalmente com os dados, com critérios de performance, mas acima de tudo com a extração de valor que conseguimos fazer dos dados. Para tal, sistemas como os Manufacturing Execution Systems (MES), que funcionam como a espinha dorsal da indústria, têm de inovar com novas estratégias e novas implementações que suportem tal disrupção. Esta dissertação surge assim como estratégia de disrupção, propondo as *Graph Databases* como potencial solução. Como descrito pelo estado da arte, as *Graph Databases* são soluções que podem ser altamente otimizadas para domínios orientados a dados e altamente coesos, pelo que surge a hipótese de ser uma boa solução para o contexto do MES. Para fazer esta ponte foi preciso analisar o domínio de MES de onde vimos e comprovamos que o mesmo é, efetivamente, altamente coeso, isto devido à natureza dos dados num sistema MES — onde uma entidade não tem sentido sozinha, mas é o contexto da mesma que lhe traz valor.

Com o conteúdo desta dissertação comprovamos, efetivamente, que as *Graph Databases* são uma solução que permitem otimizar o contexto de *manufacturing*, algo evidenciado pela performance demonstrada pelas *Graph Databases* em operações fulcrais como a *Material Geneology* e a *Material History*.

Para suportar a alegação anterior foi empregue um processo *Exploration Based*, suportado pelo estado da arte, com pequenas modificações. As modificações ao processo permitiram analisar o domínio de *manufacturing*, comprovando as suas características, bem como analisar o grafo que o compõe permitindo descrever o mesmo e criar possíveis relações entre as suas propriedades e o sucesso da aplicação das *Graph Databases*. Esta abordagem permitiu evidenciar a correlação

existente entre propriedades relacionados com a densidade do grafo com o sucesso na aplicação das *Graph Databases* permitindo afirmar que quando maior for a densidade do grafo, maior será a adaptação das *Graph Databases* ao problema.

Com base na correlação existente entre a densidade do grafo e a adequação das *Graph Databases* podemos otimizar os processos *Exploration Based* analisando, a montante, as características do grafo criando assim uma heurística de otimização. O que permite reforçar a recomendação do uso de *Graph Databases* a contextos parecidos com o de manufacturing, onde os dados são altamente coesos e o valor das operações sobre os dados está relacionado com as relações existentes entre os mesmos e não, diretamente, com as características de uma determinada entidade.

6.3 Contribuições

No decorrer desta dissertação, várias contribuições foram feitas. Das quais é pertinente realçar:

- **State-of-the-art:** O estado da arte compreende uma contextualização histórica das *Graph Databases*, comparando decisões feitas anteriormente, com problemas que estão em discussão hoje. Está presente uma análise tanto do estado da arte, como do estado da prática no que toca ao universo das *Graph Databases* aglomerado definições, desenvolvimentos recentes e os conceitos principais no que toca ao uso de *Graph Databases*.
- **Otimização do contexto de Manufacturing:** O processo descrito ao longo do documento permitiu otimizar operações fulcrais para o domínio de manufacturing ao nível de performance.
- **Otimização de processos Exploration Based:** Como resultado do processo foi demonstrada a correlação existente entre a densidade de um grafo e o sucesso de aplicação de *Graph Databases*, permitindo assim definir uma heurística de otimização de processo baseada nesta propriedade.
- **Integração com o MES da Critical Manufacturing:** O processo descrito ao longo do documento permitiu transformar o *workflow* de *reporting* da Critical Manufacturing poliglota, funcionando assim com um sistema relacional e com o Neo4J, ao nível de *back-end*.

6.4 Limitações

Tanto do ponto de vista de implementação, como no ponto de vista de construção de conhecimento foram enumeradas algumas limitações do presente trabalho, descrito no decorrer da dissertação.

Do ponto de vista de implementação, a principal limitação foi identificada no capítulo de integração, secção 5.3, que se prende ao facto de apesar a performance das operações terem sido melhoradas com o uso de *Graph Databases* a verdade é que a integração com o componente atual

de visualização de reports não foi possível, pois este estava acoplado, tecnologicamente, com o ecossistema da Microsoft não permitindo usar outro tipo de sistemas como fonte de dados.

Do ponto de vista de construção de conhecimento, a principal limitação foi identificada na secção 6.1, que está relacionada com o possível *overfitting* da correlação entre a *Graph Density* e o sucesso na aplicação das *Graph Databases*. De notar que deste tipo de generalização não é recomendado pelo estado da arte, onde é reforçada a ideia de para diferentes problemas diferentes soluções.

6.5 Trabalho Futuro

O processo descrito nesta dissertação respondeu à principal questão a que se propôs — a questão de otimizar o contexto de manufacturing. Contudo, no decorrer do processo, foram encontrados algumas limitações e novos desafios que podem ser endereçados como trabalho futuro.

De um ponto de vista de implementação e relativamente à impossibilidade de integração de *Graph Databases* com o componente de visualização de *reports* da *Critical Manufacturing* podemos realçar a necessidade de construir um componente que não esteja acoplado tecnologicamente a uma tecnologia ou a um ecossistema de tecnologias. Numa prescritiva de longo termo isto é uma decisão que tem de ser tomada como forma de contornar o problema de *vendor lock-in* [61]. Da mesma forma que, hoje, observamos a vantagem de integrar as *Graph Databases* no *work-flow* de *reporting*, no futuro poderá surgir uma outra tecnologia que seja vantajosa. Exemplos de componentes visuais de Reporting, *off-the-shelf* e *open-source*, é, por exemplo, o *Grafana* ¹, mas como vimos anteriormente, para tomar este tipo de decisões é recomendável seguir um processo *Exploration Based* como forma de validar e definir a melhor opção.

De um ponto de vista de construção de conhecimento, como forma de combater o possível *overfitting* da correlação entre a *Graph Density* e o sucesso na aplicação das *Graph Databases*, seria interessante combinar resultados de outros domínios e resultados de grafos gerados aleatoriamente com origem em geradores.

Numa perspetiva mais abrangente e, considerando os avanços recentes do estado da arte e o crescente interesse nesta tecnologia, podemos afirmar que a integração das *Graph Databases* com o MES da *Critical Manufacturing* abre a porta ao universo das *Graph Analytics* que como vimos é uma estratégia, baseada em grafos, de analisar a informação e extrair padrões da mesma, criando assim valor acrescentado. Exemplos podem ser a utilização de algoritmos de *Traversal*, *Pathfinding*, *Centrality* e *Community Detection* [58] para resolver problemas de negócio.

¹<https://grafana.com/grafana/>

Referências

- [1] Soad Almagbady. Comparative Analysis of Relational and Graph Databases for Social Networks. Em *2018 1st International Conference on Computer Applications Information Security (ICCAIS)*, páginas 1–4, 2018.
- [2] Francisco Almada-Lobo. The Industry 4.0 revolution and the future of Manufacturing Execution Systems (MES). *Journal of Innovation Management*, 3(4):16–21, janeiro 2016.
- [3] Renzo Angles, Peter Boncz, Josep Larriba-Pey, Irini Fundulaki, Thomas Neumann, Orri Erling, Peter Neubauer, Norbert Martinez-Bazan, Venelin Kotsev, e Ioan Toma. The Linked Data Benchmark Council: A Graph and RDF Industry Benchmarking Effort. *SIGMOD Rec.*, 43(1):27–31, maio 2014.
- [4] Renzo Angles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Keith W. Hare, Jan Hidders, Victor E. Lee, Bei Li, Leonid Libkin, Wim Martens, Filip Murlak, Josh Perryman, Ognjen Savković, Michael Schmidt, Juan Sequeda, Slawek Staworko, e Dominik Tomaszuk. PG-Keys: Keys for Property Graphs. Em *Proceedings of the 2021 International Conference on Management of Data, SIGMOD/PODS '21*, páginas 2423–2436, New York, NY, USA, 2021. Association for Computing Machinery.
- [5] Renzo Angles e Claudio Gutierrez. Survey of Graph Database Models. *ACM Comput. Surv.*, 40(1), fevereiro 2008.
- [6] Timothy G. Armstrong, Vamsi Ponnkanti, Dhruba Borthakur, e Mark Callaghan. Link-Bench: A Database Benchmark Based on the Facebook Social Graph. Em *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, SIGMOD '13*, páginas 1185–1196, New York, NY, USA, 2013. Association for Computing Machinery.
- [7] Maciej Besta, Emanuel Peter, Robert Gerstenberger, Marc Fischer, Michal Podstawski, Claude Barthels, Gustavo Alonso, e Torsten Hoefer. Demystifying Graph Databases: Analysis and Taxonomy of Data Organization, System Designs, and Graph Queries. *CoRR*, abs/1910.09017, 2019.
- [8] Mihaela A. Bornea, Julian Dolby, Anastasios Kementsietsidis, Kavitha Srinivas, Patrick Dantressangle, Octavian Udrea, e Bishwaranjan Bhattacharjee. Building an Efficient RDF Store over a Relational Database. Em *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, SIGMOD '13*, páginas 121–132, New York, NY, USA, 2013. Association for Computing Machinery.
- [9] Ulrik Brandes. A faster algorithm for betweenness centrality. *The Journal of Mathematical Sociology*, 25(2):163–177, 2001.

- [10] Nathan Bronson, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov, Hui Ding, Jack Ferris, Anthony Giardullo, Sachin Kulkarni, Harry Li, Mark Marchukov, Dmitri Petrov, Lovro Puzar, Yee Jiun Song, e Venkat Venkataramani. TAO: Facebook’s Distributed Data Store for the Social Graph. Em *Proceedings of the 2013 USENIX Conference on Annual Technical Conference*, USENIX ATC’13, páginas 49–60, USA, 2013. USENIX Association.
- [11] Clair Brown e Greg Linden. *Chips and Change: How Crisis Reshapes the Semiconductor Industry*. The MIT Press, agosto 2009.
- [12] Pieter Cailliau, Tim Davis, Vijay Gadepally, Jeremy Kepner, Roi Lipman, Jeffrey Lovitz, e Keren Ouaknine. RedisGraph GraphBLAS Enabled Graph Database. Em *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, páginas 285–286, 2019.
- [13] Natalia Chaudhry e Muhammad Murtaza Yousaf. Architectural assessment of NoSQL and NewSQL systems. *Distributed and Parallel Databases*, 38(4):881–926, dezembro 2020.
- [14] Jinhua Chen, Qingyu Song, Can Zhao, e Zhiheng Li. Graph Database and Relational Database Performance Comparison on a Transportation Network. Em Mayank Singh, P. K. Gupta, Vipin Tyagi, Jan Flusser, Tuncer Ören, e Gianluca Valentini, organizadores, *Advances in Computing and Data Sciences*, páginas 407–418, Singapore, 2020. Springer Singapore.
- [15] Mikhail Chernoskutov. Data Structure for Faster Graph Processing. Em *2021 Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology (USBREIT)*, páginas 0297–0300, 2021.
- [16] E. F. Codd. *The Relational Model for Database Management: Version 2*. Addison-Wesley Longman Publishing Co., Inc., USA, 1990.
- [17] M.E. Coimbra, A.P. Francisco, e L. Veiga. An analysis of the graph processing landscape. *Journal of Big Data*, 8(1), 2021.
- [18] Richard Cyganiak, David Hyland-Wood, e Markus Lanthaler. RDF 1.1 Concepts and Abstract Syntax. *W3C Proposed Recommendation*, janeiro 2014.
- [19] Nicholas Davis. What is the fourth industrial revolution. volume 19, 2016.
- [20] Ali Davoudian, Liu Chen, e Mengchi Liu. A Survey on NoSQL Stores. *ACM Comput. Surv.*, 51(2), abril 2018.
- [21] Miyuru Dayarathna e Toyotaro Suzumura. Benchmarking Graph Data Management and Processing Systems: A Survey. *CoRR*, abs/2005.12873, 2021.
- [22] Denise Gosnell Denise e Broecheler Matthias. *The Practitioner’s Guide to Graph Data [Book]*. O’Reilly Media, Inc., março 2020.
- [23] Alin Deutsch, Nadime Francis, Alastair Green, Keith Hare, Bei Li, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Wim Martens, Jan Michels, Filip Murlak, Stefan Plantikow, Petra Selmer, Hannes Voigt, Oskar van Rest, Domagoj Vrgoc, Mingxi Wu, e Fred Zemke. Graph Pattern Matching in GQL and SQL/PGQ. *CoRR*, abs/2112.06217, 2021.
- [24] D. Dominguez-Sal, P. Urbón-Bayes, A. Giménez-Vañó, S. Gómez-Villamor, N. Martínez-Bazán, e J. L. Larriba-Pey. Survey of Graph Database Performance on the HPC Scalable

- Graph Analysis Benchmark. Em Heng Tao Shen, Jian Pei, M. Tamer Özsu, Lei Zou, Jiaheng Lu, Tok-Wang Ling, Ge Yu, Yi Zhuang, e Jie Shao, organizadores, *Web-Age Information Management*, páginas 37–48, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
- [25] The Economist. The world’s most valuable resource is no longer oil, but data. *The Economist*, maio 2017.
- [26] Peter Eeles. Capturing Architectural Requirements. novembro 2001.
- [27] Jing Fan, Adalbert Gerald Soosai Raj, e Jignesh M. Patel. The Case Against Specialized Graph Analytics Engines. Em *Seventh Biennial Conference on Innovative Data Systems Research, CIDR 2015, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings*. www.cidrdb.org, 2015.
- [28] Diogo Fernandes e Jorge Bernardino. Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB. Em *Proceedings of the 7th International Conference on Data Science, Technology and Applications, DATA 2018*, páginas 373–380, Setubal, PRT, 2018. SCITEPRESS - Science and Technology Publications, Lda.
- [29] Martin Fowler. Domain Driven Design, abril 2020.
- [30] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, e Andrés Taylor. Cypher: An Evolving Query Language for Property Graphs. Em *Proceedings of the 2018 International Conference on Management of Data, SIGMOD ’18*, páginas 1433–1445, New York, NY, USA, 2018. Association for Computing Machinery.
- [31] Edward Griffor, Christopher Greer, David Wollman, e Martin Burns. Framework for Cyber-Physical Systems: Volume 1, Overview, junho 2017.
- [32] Harshit Gujral, Abhinav Sharma, e Parmeet Kaur. Empirical Investigation of Trends in NoSQL-Based Big-Data Solutions in the Last Decade. Em *2018 Eleventh International Conference on Contemporary Computing (IC3)*, páginas 1–3, 2018.
- [33] Yong Guo, Marcin Biczak, Ana Lucia Varbanescu, Alexandru Iosup, Claudio Martella, e Theodore L. Willke. How Well Do Graph-Processing Platforms Perform? An Empirical Performance Evaluation and Analysis. Em *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, páginas 395–404, 2014.
- [34] Víctor H. Guzmán, Leobardo Ruiz, Luis Gutierrez, e Francisco Cervantes. A Selection Process of Graph Databases Based on Business Requirements. páginas 80–90. janeiro 2020.
- [35] Omar Hajoui, Rachid Dehbi, Mohamed Talea, e Zouhair Ibn Batouta. An advanced comparative study of the most promising NoSQL and NewSQL databases with a multi-criteria analysis method. 81:579–588, novembro 2015.
- [36] Kagermann Henning. Recommendations for implementing the strategic initiative INDUSTRIE 4.0. 2013.
- [37] Hua Huang, Zonghe Gao, Jiangpeng Dai, Yi Wang, Xin Shan, Mingyu Zhai, Guangyi Liu, e Wei Wang. Power Grid Modeling and Topology Analysis Based on Graph Database Conforming with CIM/E. páginas 575–591. janeiro 2020.
- [38] IBM. Program with SVG, 2005.

- [39] Salim Jouili e Valentin Vansteenberghe. An Empirical Comparison of Graph Databases. Em *2013 International Conference on Social Computing*, páginas 708–715, 2013.
- [40] Henning Kagermann. Change through digitization—Value creation in the age of Industry 4.0. Em *Management of permanent change*, páginas 23–45. Springer, 2015.
- [41] William Kent. A Simple Guide to Five Normal Forms in Relational Database Theory. *Commun. ACM*, 26(2):120–125, fevereiro 1983.
- [42] Hamzeh Khazaei, Marios Fokaefs, Saeed Zareian, Nasim Beigi, Brian Ramprasad, Mark Shtern, Purwa Gaikwad, e Marin Litoiu. How do I choose the right NoSQL solution? A comprehensive theoretical and experimental survey. *Journal of Big Data and Information Analytics (BDIA)*, 2, outubro 2015.
- [43] L. Kim, E. Yahia, F. Segonds, P. Véron, e A. Mallet. i-DATAQUEST: A Proposal for a Manufacturing Data Query System Based on a Graph. *IFIP Advances in Information and Communication Technology*, 594:227–238, 2020.
- [44] G. AND CARROL KLYNE. RDF 1.1 Concepts and Abstract Syntax, 2014.
- [45] Kazuma Kusu e Kenji Hatano. A Hub-based Graph Management for Efficient Repetition Path Traversing. Em *2021 IEEE International Conference on Big Data and Smart Computing (BigComp)*, páginas 188–191, 2021.
- [46] Neo4J Labs. When to use Kafka Connect vs. Neo4j Streams as a Plugin - Neo4j Streams Docs, julho 2022.
- [47] Guangyi Liu, Xi Chen, Zhiwei Wang, Renchang Dai, Jingjin Wu, Chen Yuan, e Jun Tan. Evolving Graph Based Power System EMS Real Time Analysis Framework. Em *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, páginas 1–5, 2018.
- [48] João Lourenço, Bruno Cabral, Paulo Carreiro, Marco Vieira, e Jorge Bernardino. Choosing the right NoSQL database for the job: a quality attribute evaluation. *Journal of Big Data*, 2:18, agosto 2015.
- [49] Peter Macko, Daniel Margo, e Margo Seltzer. Performance Introspection of Graph Databases. Em *Proceedings of the 6th International Systems and Storage Conference, SYSTOR '13*, New York, NY, USA, 2013. Association for Computing Machinery.
- [50] Soujanya Mantravadi e Charles Møller. An Overview of Next-generation Manufacturing Execution Systems: How important is MES for Industry 4.0? *Procedia Manufacturing*, 30:588–595, 2019.
- [51] Critical Manufacturing. What is MES?, 2022.
- [52] J. Martinez-Gil, R. Stumpner, C. Lettner, M. Pichler, e W. Fragner. Design and Implementation of a Graph-Based Solution for Tracking Manufacturing Products. *Communications in Computer and Information Science*, 1064:417–423, 2019.
- [53] Jorge Martinez-Gil, Reinhard Stumpner, Christian Lettner, Mario Pichler, Salma Mahmoud, Patrick Praher, e Bernhard Freudenthaler. General Model for Tracking Manufacturing Products Using Graph Databases. páginas 86–100. abril 2020.

- [54] Michael McClellan. *Applying manufacturing execution systems*. The St. Lucie Press/APICS series on resource management. St. Lucie Press ; APICS, Boca Raton, Fla. : Falls Church, Va, 1997.
- [55] Scott Meyer. Liquid: The soul of a new graph database, 2020.
- [56] Microsoft. DBCC DROPCLEANBUFFERS (Transact-SQL) - SQL Server, maio 2022.
- [57] Mohamed Morsey, Jens Lehmann, Sören Auer, e Axel-Cyrille Ngonga Ngomo. *DBpedia SPARQL Benchmark – Performance Assessment with Real Queries on Real Data*. janeiro 2011.
- [58] Mark Needham e Amy E. Hodler. *Graph algorithms: practical examples in Apache Spark and Neo4j*. O'Reilly Media, Sebastopol, California, first edition edição, 2019. OCLC: on1066191517.
- [59] Thuan L Nguyen. A framework for five big v's of big data and organizational culture in firms. páginas 5411–5413. IEEE, 2018.
- [60] T. William Olle. *The Codasyl Approach to Data Base Management*. John Wiley & Sons, Inc., USA, 1978.
- [61] Justice Opara-Martins, Reza Sahandi, e Feng Tian. Critical review of vendor lock-in and its impact on adoption of cloud computing. Em *International Conference on Information Society (i-Society 2014)*, páginas 92–97, 2014.
- [62] Anil Pacaci, Alice Zhou, Jimmy Lin, e M. Tamer Özsu. Do We Need Specialized Graph Databases? Benchmarking Real-Time Social Networking Applications. Em *Proceedings of the Fifth International Workshop on Graph Data-Management Experiences & Systems, GRADES'17*, New York, NY, USA, 2017. Association for Computing Machinery.
- [63] Andrew Pavlo e Matthew Aslett. What's Really New with NewSQL? *SIGMOD Rec.*, 45(2):45–55, setembro 2016.
- [64] Vanda Pinheiro. Indústria 4.0 a Quarta Revolução industrial, 2016.
- [65] Jaroslav Pokorný. Graph Databases: Their Power and Limitations. Em Khalid Saeed e Wladyslaw Homenda, organizadores, *Computer Information Systems and Industrial Management*, páginas 58–69, Cham, 2015. Springer International Publishing.
- [66] Pawan Rawal e Manish Rai Jain. Neo4j vs Dgraph - The numbers speak for themselves - Dgraph Blog, 2017.
- [67] Ian Robinson, Jim Webber, e Emil Eifrem. *Graph Databases: New Opportunities for Connected Data*. O'Reilly Media, Inc., 2nd edição, 2015.
- [68] Marko A. Rodriguez e Peter Neubauer. The Graph Traversal Pattern. *CoRR*, abs/1004.1001, 2010.
- [69] Everett M. Rogers. *Diffusion of innovations*. Free Press, New York, 5th ed edição, 2003.
- [70] Noa Roy-Hubara, Peretz Shoval, e Arnon Sturm. A Method for Database Model Selection. páginas 261–275. janeiro 2019.

- [71] Noa Roy-Hubara, Peretz Shoval, e Arnon Sturm. Selecting databases for Polyglot Persistence applications. *Data & Knowledge Engineering*, 137:101950, 2022.
- [72] Pramod J. Sadalage e Martin Fowler. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley Professional, 1st edição, 2012.
- [73] B. Saenz de Ugarte, A. Artiba, e R. Pellerin. Manufacturing execution system – a literature review. *Production Planning & Control*, 20(6):525–539, setembro 2009.
- [74] G Schuh, T Potente, C Wesch-Potente, e A Hauptvogel. 10.6 Sustainable increase of overhead productivity due to cyber-physical-systems. 2013.
- [75] Monika Shah, Amit Kothari, e Samir Patel. Influence of Schema Design in NoSQL Document Stores. Em Subarna Shakya, Robert Bestak, Ram Palanisamy, e Khaled A. Kamel, organizadores, *Mobile Computing and Sustainable Informatics*, páginas 435–452, Singapore, 2022. Springer Singapore.
- [76] Jeffrey Soplop, Jenifer Wright, Khris Kammer, e Rafael Rivera. Manufacturing execution systems for sustainability: Extending the scope of MES to achieve energy efficiency and sustainability goals. páginas 3555–3559. IEEE, 2009.
- [77] Maggie Sparkman. Data Sources Supported by Reporting Services - SQL Server Reporting Services (SSRS), outubro 2021.
- [78] Michael Stonebraker e Joey Hellerstein. What Goes Around Comes Around. *Readings in database systems*, página 43, 2005. Edition: 4.
- [79] Wen Sun, Achille Fokoue, Kavitha Srinivas, Anastasios Kementsietsidis, Gang Hu, e Guotong Xie. SQLGraph: An Efficient Relational-Based Property Graph Store. Em *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, SIGMOD '15*, páginas 1887–1901, New York, NY, USA, 2015. Association for Computing Machinery.
- [80] Enqing Tang e Yushun Fan. Performance Comparison between Five NoSQL Databases. Em *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*, páginas 105–109, 2016.
- [81] RedisGraph Team. Benchmarking RedisGraph 1.0, 2019. Publication Title: Redis.
- [82] TigerGraph. Graph Database Benchmarks and Performance Comparison, 2020. Publication Title: TigerGraph.
- [83] Santiago Timón-Reina, Mariano Rincón, e Rafael Martínez-Tomás. An overview of graph databases and their applications in the biomedical domain. *Database*, 2021:baab026, outubro 2021.
- [84] Harley Vera-Olivera, Ruizhe Guo, Ruben Cruz Huacarpuma, Ana Paula Bernardi Da Silva, Ari Melo Mariano, e Maristela Holanda. Data Modeling and NoSQL Databases - A Systematic Mapping Review. *ACM Comput. Surv.*, 54(6), julho 2021.
- [85] Aleksa Vukotic. How much faster is a graph database, really?, 2014. Publication Title: Neo4j Graph Database Platform.
- [86] Claudius Weinberger. Benchmark: MongoDB, PostgreSQL, OrientDB, Neo4j and ArangoDB, 2018.