UNIVERSIDADE FEDERAL DE ITAJUBÁ - UNIFEI
PROGRAMA DE PÓS-GRADUAÇÃO EM
CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

Mapeamento, Conversão e Migração automática de Bancos de Dados Relacionais para Orientados a Grafos

Anderson Tadeu de Oliveira Vicente

Itajubá - MG,
Agosto de 2020

UNIVERSIDADE FEDERAL DE ITAJUBÁ - UNIFEI
PROGRAMA DE PÓS-GRADUAÇÃO EM
CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

Anderson Tadeu de Oliveira Vicente

Mapeamento, Conversão e Migração automática de Bancos de Dados Relacionais para Orientados a Grafos

Dissertação submetida ao Programa de Pós-Graduação em Ciência e Tecnologia da Computação como parte dos requisitos para obtenção do Título de Mestre em Ciência e Tecnologia da Computação

Área de Concentração: Matemática da Computação

Orientador: Edmilson Marmo Moreira
Co-orientador: Enzo Seraphim

Agosto de 2020
Itajubá - MG

Agradecimentos

Agradeço à Deus.

Agradeço à minha família, em especial minha mãe Valéria e irmãs Aline, Dayane e Juliana, que me deixaram o ensinamento de que o estudo e a educação é a coisa mais valiosa que temos na vida.

Agradeço ao meu orientador Prof. Edmilson e Co-orientador Prof. Enzo por aceitarem me orientar na execução desta pesquisa e por toda a contribuição dada durante seu desenvolvimento. Ao Prof. Edmilson sou grato aos seus ensinamentos a respeito da Teoria de Grafos, que me fez gostar ainda mais da área e a sua forma de ensinar, que despertou em mim ainda mais paixão pelo ensino. Ao Prof. Enzo sou grato à sua rica contribuição sobre os conceitos de Banco de Dados, à atenção dedica por me co-orientar.

Agradeço à Prof.ª Vanessa Cristina por ter despertado desde a época da Graduação o meu interesse pela pesquisa, principalmente sobre a área de Banco de Dados e Estruturas de Dados mais complexas. Por ter me ajudado a melhorar muito a escrita e ter tido paciência com meus parágrafos desordenados. Por ter ministrado muito bem as aulas de COM112 e COM231, as quais recordo até hoje o aprendizado.

Agradeço ao Prof. Adler Diniz por toda a base que forneceu no início dessa pesquisa, com a elaboração de uma Revisão Sistemática que resultou em uma publicação em congresso internacional.

Agradeço às minhas colegas do CRAS ACPT, Anne, Fabiana, Isabela, Maiara e Maria Janayna que me apoiaram em todo o processo de formação neste Mestrado, seja nas mudanças de horários, seja por palavras de incentivo.

Aqueles que se enamoram da prática sem ciência são como um marinheiro que entra no navio sem timão ou bússola, que nunca tem certeza para onde vai; a prática deve sempre ser edificada sobre a boa teoria da qual a perspectiva é guia de entrada, e sem a qual nada se faz de bom[...]

(Leonardo da Vinci)

Resumo

Bancos de Dados Relacionais são os modelos mais utilizados em diversas aplicações em razão da facilidade existente em sua linguagem de consulta e utilização em ambientes multi-usuários. Com o grande volume de informação que se tem nos dias de hoje e, sendo que estes encontram-se cada vez mais relacionadas, surgem os bancos de dados orientados a grafos como forma de lidar com esta nova demanda, frente às dificuldades do modelo relacional a este novo cenário. Diante disto, esta pesquisa tratou dos processos de mapeamento, conversão e migração do modelo relacional para o orientado a grafos, tratando, sobretudo, a sobrecarga semântica de construtores entre os dois modelos. O objetivo deste estudo foi o desenvolvimento de uma aplicação, denominada *ThrusterDB*, que realiza esse processo de conversão do modelo relacional para o orientado a grafos de forma automática. A pesquisa traz contribuição ao integrar as fases de mapeamento, conversão e migração automática de um banco de dados relacional para um orientado a grafos. Esta dissertação apresenta resultados que mostram que o banco de dados gerado, após o processo, provê um desempenho melhor no tempo médio de consultas realizadas, além de preservar a semântica do banco de dados relacional de origem, sem qualquer perda ou redundância de dados.

Palavras-chave: Aninhamento de Relações. Grafos. ThrusterDB. MySQL. Neo4j.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ACID	Atomicidade, Consistência, Isolamento e Durabilidade	20
API	*Application Programming Interface*	12
BLOB	*Binary Large Object*	81
BSON	*Binary JSON*	32
CSV	*Comma-separated Values*	38
DDL	Linguagem de Definição de Dados	17
DER	Diagrama de Entidade-Relacionamento	41
DHT	*Distributed Hash Table*	31
DML	Linguagem de Manipulação de Dados	17
GDB	*Banco de Dados Orientado a Grafos*	32
GDBS	*Bancos de Dados Orientado a Grafos*	33
GEXF	*Graph Exchange XML Format*	43
JSON	*JavaScript Object Notation*	32
KCDC	*Korea Centers for Disease Control and Prevention*	66
NoSQL	*Not-Only SQL*	12
ORM	*Object-Relational Mapping*	46
RDBMS	*Relational Database Management Systems*	12
SGBD	Sistema Gerenciador de Banco de Dados	16
SGBDs	Sistemas Gerenciadores de Banco de Dados	16
SQL	*Structured Query Language*	12
UML	Linguagem de Modelagem Unificada	41
URL	*Uniform Resource Locator*	81
XML	*Extensible Markup Language*	32

Sumário

1 Introdução

Bancos de Dados Relacionais vêm sendo utilizados desde a década de 70. Eles são os modelos mais comumente utilizados em diversas aplicações, sejam comerciais ou não, em razão da sua facilidade de programação ao utilizar a linguagem SQL (*Structured Query Language*) e possibilitar a sua implementação em ambientes multiusuários. Tais características tornaram essa tecnologia de grande valor para o mercado (DATE, 2003).

A crescente demanda por armazenamento e processamento de conjunto de dados em larga escala vem definindo o desenvolvimento de sistemas de banco de dados na última década. Sendo assim, o armazenamento dos dados foi aprimorado de tal forma que de local passou a ser *clusterizado*, distribuído e na nuvem. Consequentemente, os novos tipos de aplicações frente a este cenário requerem uma nova forma de armazenamento dos dados em oposição aos tradicionais RDBMS (*Relational Database Management Systems*). Isso se justifica pelo fato de que os banco de dados relacionais possuem um esquema pré-definido e, consequentemente, tornando-os difíceis para se adaptarem aos novos modelos de dados (KUSZERA; PERES; FABRO, 2019; WU; SAKR; ZHU, 2017; MOHANTY, 2015).

Essa nova forma de armazenamento é chamada de NoSQL (*Not-Only SQL*) e caracteriza-se por ser altamente escalável, tolerante a falhas e é projetado para armazenar dados semi-estruturados e não estruturados. NoSQL é um conjunto de conceitos que possibilitam um rápido e eficiente processamento de um conjunto de dados, desempenho e confiabilidade. Existem diversos tipos de tecnologias de armazenamento NoSQL dentre os quais pode-se citar: Chave-Valor, Famílias de Colunas, Orientados a Grafos e Orientados a Documentos(ERL; KHATTAK; BUHLER, 2016; MCCREARY; KELLY, 2013).

Levando em consideração que o grande volume de dados atualmente se encontra, na maioria das vezes, fortemente conectados e, a representação desses dados em forma de grafo facilita a descoberta de informações que dificilmente seriam visualizadas ao utilizar o modelo relacional, nada mais adequado do que utilizar um banco de dados orientado a grafos para a persistência destes dados (MEGID; EL-TAZI; FAHMY, 2018; UNAL; OGUZTUZUN, 2018; OREL; ZAKOŠEK; BARANOVIČ, 2016).

Como a maioria das aplicações atualmente, ao trabalhar com um grande volume dados, demanda que o processamento destes tenha alto desempenho, realizar a migração de dados persistidos em banco de dados relacionais para um banco de dados NoSQL e, especificamente orientado a grafos, apresenta dificuldades em razão da não existência de uma API (*Application Programming Interface*) que realize tal tarefa de modo automático. O processo de mapeamento entre modelos e por fim, a migração, pode ser custoso para desenvolvedores, em razão de demandar grande tempo para o aprendizado (KUSZERA;

PERES; FABRO, 2019).

1.1 Visão Geral

As tecnologias de banco de dados tiveram, diversos modelos e abordagens, como: o modelo hierárquico e de redes, relacional, orientado a objetos, NoSQL e NewSQL, em ordem cronológica.

O processamento dos dados em forma de grafos se tornou, nos últimos tempos, algo de grande importância em diversas áreas da Ciência da Computação como o Aprendizado de Máquina, aplicações na Medicina, análise de redes sociais entre outros (BESTA et al., 2019).

De acordo com o site `DB-Engines`[1], que realiza uma classificação das tecnologias de bancos de dados por popularidade, fica evidente o crescimento do modelo orientado a grafos (Figura 1), justificando a crescente demanda por este modelo. Como o próprio *site* destaca, esse *rank* de popularidade é calculado seguindo os seguintes parâmetros: a quantidade de menções em sistemas de busca, interesse geral pelos sistemas (utiliza o *Google Trends* como parâmetro), frequência de discussões técnicas a respeito do sistema (em *sites* como o *Stack Overflow* e *DBA Stack Exchange*), quantidade de ofertas de trabalho dos referidos sistemas (em *sites* de oferta de empregos como o *Indeed* e *Simply Hired*), a quantidade de perfis em redes de profissionais (como o *LinkedIn* e *Upwork*) e a relevância em redes sociais (como o *Twitter*).

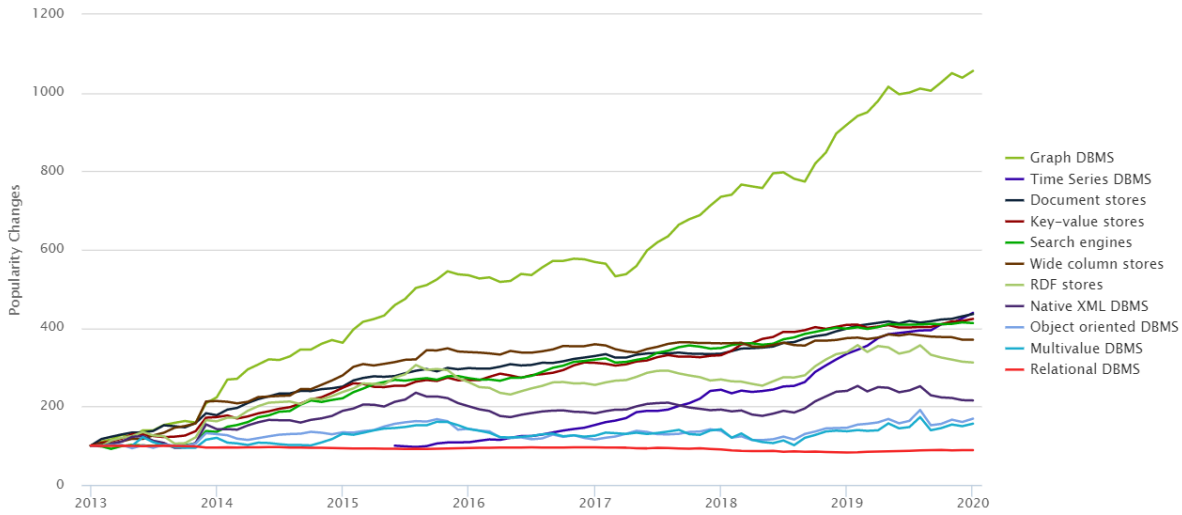


Figura 1 – Popularidade das tecnologias de Banco de Dados entre 2013 e 2020 (DB-ENGINES, 2020).

É possível observar o crescimento de tecnologias não só as que trabalham com o

[1] https://db-engines.com/en/

modelo de grafos, mas outras que integram o modelo NoSQL em geral, enquanto o modelo relacional se mantêm abaixo destas e com tendência a se estabilizar quanto à popularidade, embora ainda seja bastante utilizado em grandes organizações. O que se pode supor a partir deste fato, é que está ocorrendo uma mudança em que o modelo relacional vem sendo substituído por outros modelos que lidam melhor com dados pertinentes ao universo da Web 2.0.

1.2 Objetivos

1.2.1 Objetivo Geral

O principal objetivo desta pesquisa é desenvolver uma aplicação (*ThrusterDB*) que realize o mapeamento, a conversão e a migração automática de Bancos de Dados Relacionais para um Orientado a Grafos (Neo4j).

1.2.2 Objetivos Específicos

Os objetivos específicos da pesquisa são:

- Descrever as diferenças entre os modelos relacional e o orientado a grafos (**Capítulo 2**).
- Identificar as diversas técnicas que envolvem o mapeamento e a conversão do modelo relacional e o orientado a grafos (**Capítulo 3**).
- Comparar e combinar os pontos positivos das técnicas analisadas e melhorar lacunas na aplicação desenvolvida, *ThrusterDB* (**Capítulo 4**).
- Testar a aplicação desenvolvida quanto ao tempo de execução de cada um dos processos e o tempo total, além de comparar o modelo gerado com outros resultantes da aplicação de outras técnicas (**Capítulo 5**).

1.3 Questões de Pesquisa

No decorrer desta pesquisa, quatro questões foram definidas para nortear o seu desenvolvimento.

1. **Como realizar a migração dos dados persistidos em um banco de dados relacional para orientado a grafos?**
2. **Como elementos pertinentes ao projeto de banco de dados influenciam o desenvolvimento de algoritmos que realizam conversão para um modelo orientado a grafos?**

3. **Quais são os problemas, dificuldades, desafios, perspectivas futuras para algoritmos de conversão automática de um banco de dados relacional para modelos orientados a grafos?**

4. **Como os algoritmos e aplicações desenvolvidas podem testados?**

1.4 Organização do trabalho

Como forma de atingir os objetivos propostos e responder às questões de pesquisa, esta dissertação está estruturada da seguinte forma:

- **Capítulo 2:** contextualiza os sistemas de banco de dados em geral, apresentando um histórico e características do modelo relacional, situa o modelo NoSQL no cenário atual e aborda com mais detalhes o modelo orientado a grafos, foco principal desta pesquisa.
- **Capítulo 3:** apresenta os trabalhos relacionados ao tema desta pesquisa, incluindo os mais recentes e que estão no estado-da-arte na área científica.
- **Capítulo 4:** detalha a aplicação desenvolvida, apresentado a linguagem de programação, bibliotecas utilizadas e descrição de cada um dos algoritmos que integram o processo como um todo.
- **Capítulo 5:** apresenta os testes efetuados com a aplicação a fim de compará-la com outros modelos desenvolvidos e demonstrar a sua importância.
- **Capítulo 6:** realiza as considerações finais e responde as questões da pesquisa, além de apontar as perspectivas de trabalhos futuros.

2 Sistemas de Banco de Dados

Neste capítulo são discutidos elementos teóricos essenciais para um bom entendimento do desenvolvimento deste trabalho. Inicia-se com uma explicação sobre SGBDs (Sistemas Gerenciadores de Banco de Dados), abordando suas características e arquitetura. Em seguida, são mostrados os Banco de Dados Relacionais e NoSQL, evidenciando as particularidades e cenários de aplicação de cada um destes modelos. As definições e conceitos referentes ao modelo relacional partiu das ideias presentes em Connolly e Begg (2004), Ramakrishnan e Gehrke (2008), Garcia-Molina, Ullman e Widom (2008), Foster e Godbole (2014), Date (2003), Elmasri e Navathe (2011), Silberchatz, Korth e Sudarshan (2012), Haerder e Reuter (1983), Strawn e Strawn (2016), Codd (1970), Codd (1982), Codd (1990), Rob e Coronel (2011), Chen (1976), Teorey et al. (2011), Heuser (2009).

2.1 Sistemas Gerenciadores de Banco de Dados

Pode-se definir banco de dados como sendo:

Definição (Banco de Dados)**.** *Uma coleção compartilhada de dados logicamente relacionados e uma descrição destes dados, projetada para atender às necessidades de informações de uma organização.*

Desde o aparecimento dos primeiros computadores, o armazenamento e a manipulação dos dados vêm sendo o principal foco das aplicações de computador. Os primeiros sistemas de banco de dados comerciais surgiram na década de 1960 e originaram-se dos sistemas de arquivos, que armazenavam grande quantidade de dados por um longo período de tempo.

Atualmente, bancos de dados são essenciais para qualquer negócio. Ao visitar qualquer página na Web que fornece algum tipo de informação, há um banco de dados por trás recuperando e disponibilizando a informação requisitada. Além disso, grandes corporações e institutos de pesquisas científicas mantém suas informações armazenadas em banco de dados.

Um SGBD (Sistema Gerenciador de Banco de Dados) é:

Definição (Sistema Gerenciador de Banco de Dados)**.** *Um sistema de software que permite aos usuários definir, criar, manter e controlar o acesso ao banco de dados.*

Partindo da definição, SGBD permite que:

- Usuários possam criar novos bancos de dados e definir seus esquemas (estrutura lógica dos dados) utilizando uma linguagem específica, denominada DDL (Linguagem de Definição de Dados).
- Usuários possam consultar e modificar os dados fazendo uso de uma linguagem específica, denominada DML (Linguagem de Manipulação de Dados).
- Suporte o armazenamento de uma grande quantidade de dados por um longo período de tempo, permitindo acesso eficiente aos dados para consultas e modificações.
- O banco de dados seja recuperado em caso de falhas, erros de diversos tipos ou utilização indevida de forma intencional.
- Seja determinado um controle de acesso para diversos tipos de usuários ao mesmo tempo.

Ilustram-se a seguir algumas vantagens de um sistema de banco de dados:

- Redução de redundâncias.
- Prevenção de inconsistências.
- Compartilhamento de dados.
- Aplicação de restrições de segurança.
- Estabelecimento de padrões.
- Integridade pode ser preservada.
- Equilíbrio entre requisitos discordantes.
- Ganho de desempenho em razão da velocidade de processamento.
- Facilidade na manutenção e recuperação dos dados, sem necessidade de um programa de aplicação complexo.
- Independência de uma linguagem de programação de alto nível para utilização.
- Facilidade na criação de visões lógicas dos dados.

A Figura 2 esquematiza um sistema de banco de dados. Pode-se observar que, diversos usuários encontram-se utilizando o mesmo banco de dados, cada um com um objeto específico. A definição dos dados e suas relações estão separadas do programa de aplicação que é utilizado por estes usuários. A interação destes com o sistema é intermediado pelo SGBD que provê uma interface para com o banco de dados.

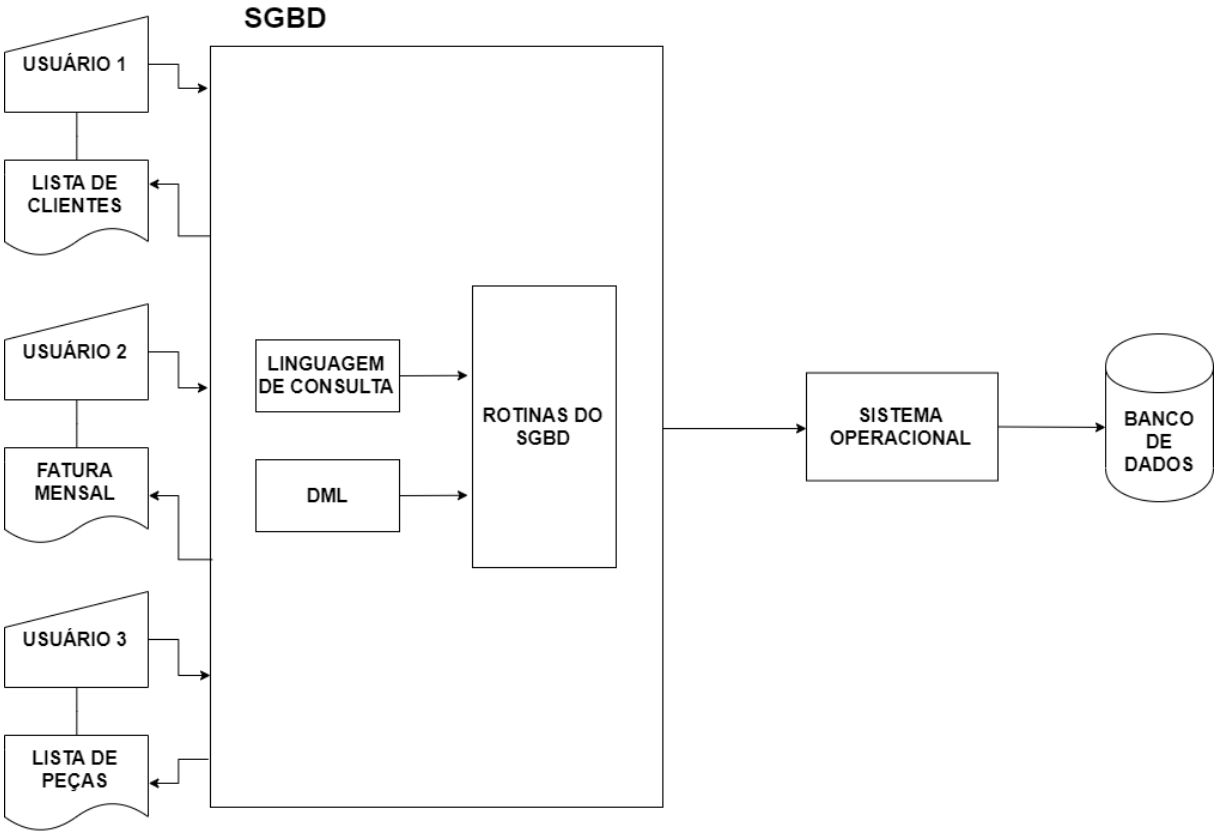


Figura 2 – Sistema de Banco de Dados, adaptado de (MITTRA, 1991)

.

2.1.1 Arquitetura do Sistema de Banco de Dados

Desde os primeiros sistemas monolíticos (onde o *software* do SGBD era um sistema fortemente integrado) até os mais atuais, caracterizados pela arquitetura cliente/servidor (estrutura modular), a arquitetura dos SGBDs vem evoluindo com base nas tendências da computação. A arquitetura de um sistema de banco de dados está intimamente ligada ao sistema de computador subjacente em que o banco de dados é executado.

Sendo um sistema de banco de dados uma coleção de dados que se encontram inter-relacionados e um conjunto de programas que dão aos usuários a possibilidade de acessar e modificar tais dados, faz-se necessário ocultar destes usuários os detalhes da forma em que os dados são armazenados e mantidos. Em outras palavras, fornece aos usuários uma visão *abstrata* dos dados. Essa abordagem permite que distintos usuários possam perceber os dados no nível de detalhe que preferirem.

Tal abstração é alcançada através do que se chama de **modelo de dados**. Este é um conjunto de elementos que são utilizados para representar a estrutura de um banco de dados.

Qualquer que seja o modelo de dados, é necessário diferenciar a descrição do banco de dados (**esquema do banco de dados**) e o banco de dados em si. O esquema é definido

no projeto do banco de dados, sendo representado por um **diagrama de esquema** e não é modificado frequentemente.

A arquitetura de um banco de dados, ilustrada na Figura 3, é similar à arquitetura proposta pelo ANSI/SPARC *Study Group on Data Base Management Systems* e o objetivo deste modelo de arquitetura é desassociar as aplicações do usuário do nível físico do banco de dados.

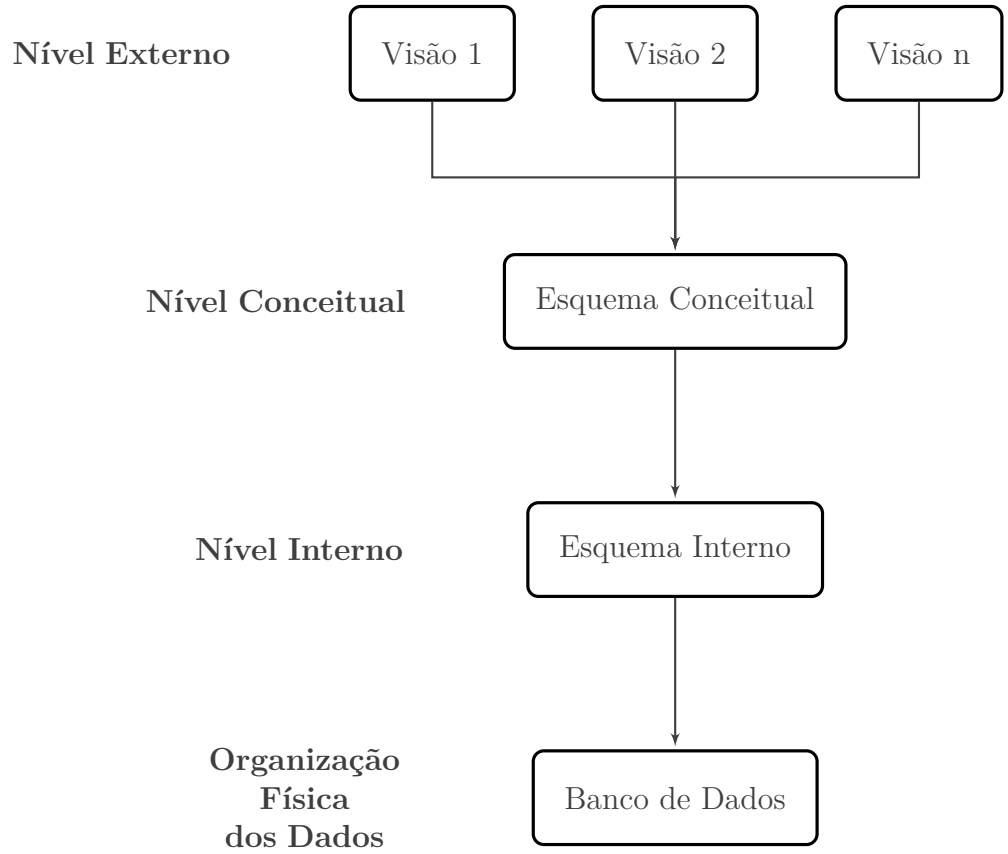


Figura 3 – Arquitetura em três camadas de um SGBD (CONNOLLY; BEGG, 2004).

2.1.1.1 Nível Externo

Corresponde ao nível de visão dos usuários do banco de dados. Este nível contém uma sucinta descrição do que apenas interessa ao usuário e apenas isso - um usuário neste contexto pode ser uma programação de aplicação ou mesmo um usuário final com qualquer grau de conhecimento que possui uma linguagem de programação disponível para acesso ao banco de dados. Caso o usuário seja um programador de aplicações, ele utilizará alguma linguagem de programação (Python, Java, C++, etc). Se este for um usuário final, será utilizada uma linguagem de consulta (como por exemplo, SQL, *Cypher*).

2.1.1.2 Nível Conceitual

Corresponde ao nível que descreve a estrutura do banco de dados como um todo para um grupo de usuários. Descreve quais dados são armazenados no banco de dados e os relacionamentos entre si, sendo uma visão completa dos requisitos dos dados da organização, independentemente de quaisquer considerações referentes ao armazenamento. Além disso, oculta informações detalhadas acerca do armazenamento físico e comumente é utilizado um modelo de dados representativo que descreve o esquema conceitual de um banco de dados.

2.1.1.3 Nível Interno

O nível interno compreende a representação física do banco de dados no computador, descrevendo como os dados estão armazenados no banco de dados. Também é chamado de nível de armazenamento.

2.1.2 Transações e Propriedades ACID

Uma transação é uma ação ou conjunto de ações realizadas por um usuário ou aplicação que realiza a leitura ou atualização do banco de dados. Esta deve ser executada de forma atômica e isolada de outras transações. Além disso, o SGBD assegura que o trabalho concluído de uma transação não será perdido - característica chamada de durabilidade.

Pode ser que uma ou mais transações possam ser interrompidas durante a sua execução, por falha no sistema, por exemplo. Em razão disso, o SGBD precisa garantir que as mudanças feitas por essas transações incompletas sejam removidas do banco de dados. Para que isto seja feito, o SGBD possui um *log* que registra todas as escritas feitas no banco de dados. Caso haja falhas, o sistema retorna ao estado consistente antes da falha ter ocorrido.

Um SGBD (salvo alguns SGBDs NoSQL) possui quatro propriedades importantes de transação, denominadas ACID (Atomicidade, Consistência, Isolamento e Durabilidade), que garantem a indivisibilidade de uma transação (HAERDER; REUTER, 1983). Estas propriedades são:

- **Atomicidade:** a transação precisa ser do tipo *tudo ou nada*, ou seja, são atômicas. Isso quer dizer que uma transação é executada em sua totalidade ou não.
- **Consistência:** a transação deve transformar o banco de dados de um estado consistente para outro estado consistente. O SGBD garante a consistência aplicando todas as restrições especificadas no esquema do banco de dados.

- **Isolamento:** as transações encontram-se isoladas umas das outras. Cada transação parece ser executada como se nenhuma outra transação estivesse sendo executada ao mesmo tempo.
- **Durabilidade:** Após uma transação ter sido concluída (COMMIT), o sistema deve garantir que as atualizações sobrevivam a qualquer queda subsequente, ou seja, seus efeitos nunca devem desaparecer.

Essas quatro propriedades: atomicidade, consistência, isolamento e durabilidade (ACID), especificam os elementos fundamentais do paradigma de transações, que exerceram grande influência em diversos aspectos do desenvolvimento em sistemas de banco de dados.

2.2 O Modelo Relacional

2.2.1 Breve Histórico

O Modelo Relacional teve sua origem em 1970, sendo descrito formalmente em um artigo publicado por Edgar Frank Codd, um homem inglês que estudou matemática e química além de ter servido como piloto na Segunda Guerra Mundial.

Codd, com essa nova abordagem, mostrou que tal modelo era superior aos mais empregados no contexto vigente da época: o hierárquico e o de redes. Estes modelos surgiram na década de 60, sendo então implementados nos primeiros SGBDs ao final da década de 60 e início da década de 70. Tais modelos são conhecidos atualmente como *sistemas de banco de dados legados* (ELMASRI; NAVATHE, 2011). O artigo de Codd é visto como um marco nos sistemas de bancos de dados, embora anteriormente já havia sido proposta uma abordagem orientada a conjuntos, além de servir como referência para linguagens de alto-nível de banco de dados nos próximos 40 anos.

Com a Visão Relacional (*Relational View*), Codd visava fornecer uma base sólida para tratar aspectos como: a derivabilidade, a redundância e a consistência das relações. Sendo assim, seu modelo consistia na disposição dos dados em tabelas descritivas, o que foi um grande avanço na independência dos dados em relação aos programas de aplicação. Por independência de dados entende-se como a imunidade dos programas de aplicação às mudanças na estrutura e formas de acesso aos dados.

Em razão de sua simplicidade e base matemática, o referido modelo teve grande atenção por parte da comunidade e possui como fundamento a teoria dos conjuntos e lógica de predicados de primeira ordem, utilizando a ideia de relação matemática (semelhante a uma tabela de valores) como sua estrutura básica.

2.2.2 Definição e Conceitos Relacionados

A introdução do modelo relacional realizado por Codd é visto como o evento mais importante da história da área de bancos de dados, já que causou uma revolução nesta área e enfraqueceu os outros modelos até então vigentes. É um modelo bastante simples se comparado com seus antecessores, facilitando o trabalho de programadores de aplicações. Nos dias de hoje, este modelo é o modelo de dados dominante e, isso se comprova com a Figura 4, que mostra que dentre os 5 SGBDs dominantes, os 4 primeiros são relacionais.

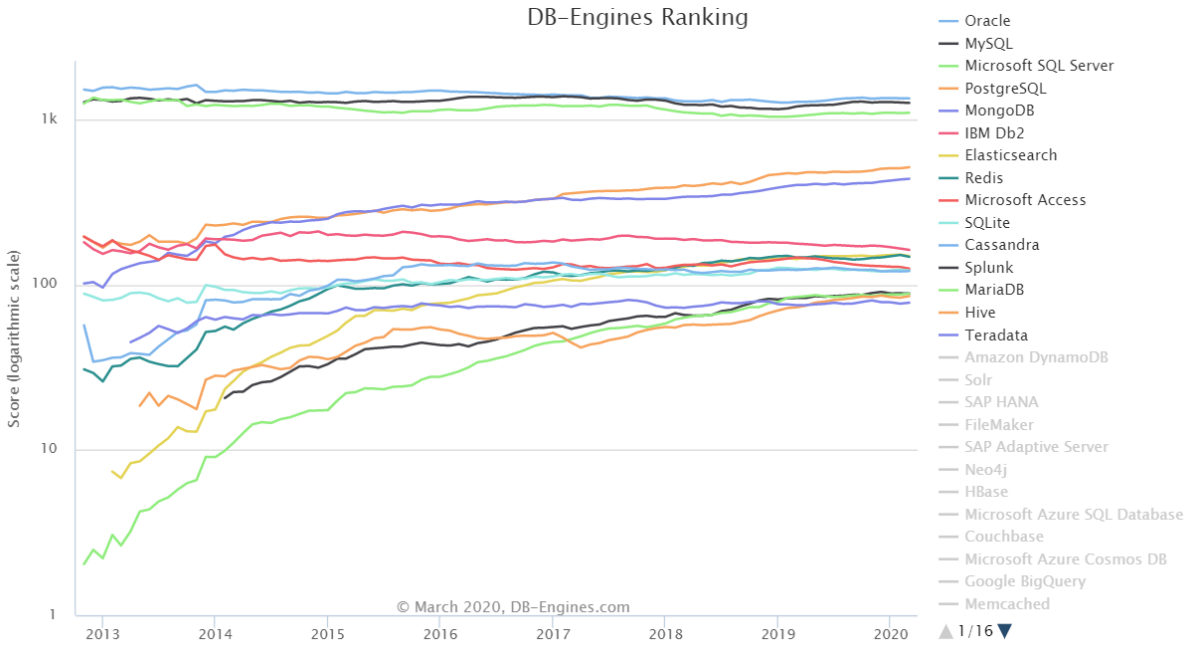


Figura 4 – *Ranking* de popularidade entre os SGBDs atuais.

O modelo relacional advém do conceito matemático de **Relação**, sendo representado fisicamente por uma tabela. Como era um matemático, Codd embasou sua ideia na Teoria dos Conjuntos e Lógica de Predicados. Neste caso, o conceito de relação pode ser definido como a seguir.

Nos primórdios das linguagens de programação, não havia uma separação entre o esquema lógico dos dados e sua representação física. O modelo relacional claramente realiza esta distinção ao esquematizar os dados como tabelas. A Tabela 2 sintetiza alguns termos presentes no modelo relacional e sua diferença no significado matemático. Estes termos serão discutidos com mais detalhes a seguir.

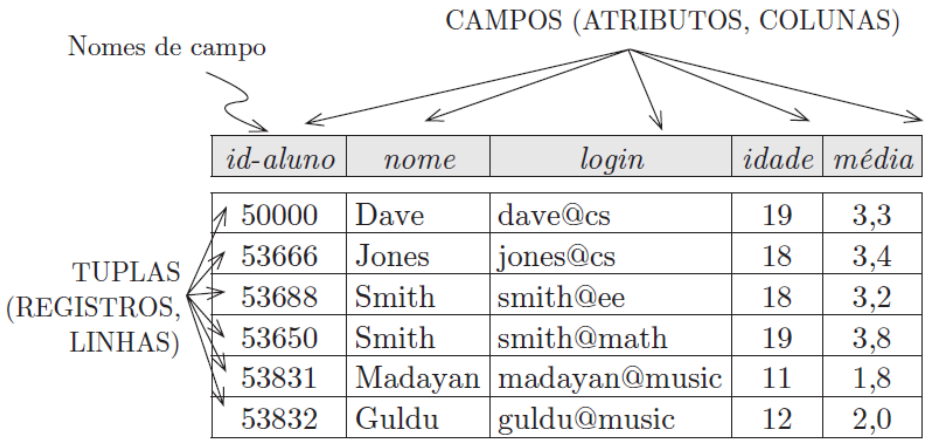
Embora sejam parecidos, os termos “Tabela” e “Relação” não são sinônimos. Uma relação (do modelo relacional) se coloca como um tipo especial de tabela. Sendo assim, em uma relação não existem conceitos posicionais, ou seja, linhas e colunas podem ser modificadas a sua ordem, sem que seja prejudicado o conteúdo das informações (é claro que o cabeçalho das colunas deve corresponder com às informações as quais se referem).

Termo Matemático	**Termo em Banco de Dados**
Relação de grau n	Tabela com n colunas
Atributo	Coluna da Tabela
Domínio	Tipo de dado estendido
Tupla	Linha da Tabela
Cardinalidade da Relação	Número de linhas na Tabela

Tabela 2 – Termos Matemáticos e em Banco de Dados (CODD, 1990)

Pontuadas as diferenças semânticas entre os dois conceitos, parte-se para a definição dos elementos envolvidos no modelo relacional.

Uma **Relação** é o principal elemento do modelo relacional. Esta consiste em uma **instância de relação** - tabela e um **esquema de relação** - cabeçalhos de coluna da tabela. Seguindo a denominação formal do modelo relacional, uma **relação** corresponde a uma tabela, uma **tupla** corresponde a uma linha da tabela, um **atributo** corresponde a um cabeçalho da coluna da tabela. O **domínio** dos valores representa o tipo de dado que descreve os valores que podem aparecer em cada coluna . A Tabela representada na Figura 5, representa uma instância de relação de uma tabela de Alunos de um determinado curso.



id-aluno	*nome*	*login*	*idade*	*média*
50000	Dave	dave@cs	19	3,3
53666	Jones	jones@cs	18	3,4
53688	Smith	smith@ee	18	3,2
53650	Smith	smith@math	19	3,8
53831	Madayan	madayan@music	11	1,8
53832	Guldu	guldu@music	12	2,0

Figura 5 – Uma instância de relação Alunos. (RAMAKRISHNAN; GEHRKE, 2002)

As terminologias citadas no parágrafo anterior podem ser definidas formalmente:

Definição (Atributo)**.** *Um atributo representa uma coluna de uma dada relação. Nomeia cada coluna da relação e pode aparecer em qualquer ordem, sem prejudicar o significado da relação.*

Definição (Domínio)**.** *Um domínio representa um conjunto de valores permitidos para um ou mais atributos. Domínios representam um poderoso elemento do modelo relacional, sendo que cada atributo de uma relação é definido em um domínio. Sua importância reside no fato de que permite que o usuário defina o significado e a fonte dos valores que um atributo pode ter.*

Definição (Tupla)**.** *Uma tupla representa cada linha de uma relação, com exceção do cabeçalho, que define o nome dos atributos.*

Definição (Esquema de Relação)**.** *Um nome de relação e um conjunto de atributos desta relação é chamado de esquema de relação.*

Definição (Esquema Relacional de Banco de Dados)**.** *Um conjunto de esquemas de relação, sendo cada uma com um nome diferente.*

Quando as relações são definidas, faz-se necessário distinguir cada uma de suas tuplas. Tal distinção é feita com base em valores de seus atributos. Isso implica dizer que, para cada par de tuplas de uma relação, seus valores em conjunto não devem ser iguais. Portanto, os valores dos atributos de cada uma destas tuplas as identificam *unicamente*.

O parágrafo anterior faz referência à utilização de **chaves** em relações. Os tipos de chaves utilizadas em banco de dados relacionais são descritos a seguir:

- **Superchave:** pode ser um atributo ou um conjunto de atributos que identifica unicamente uma tupla em uma relação. É certo que superchaves permitem atributos adicionais, que geralmente não são necessários para uma identificação única. Assim sendo, o que se quer é uma quantidade mínima de atributos para tal identificação.
- **Chave Candidata:** é, em suma, superchaves mínimas e garante que não haverá mais de uma tupla igual na relação. Esta pode conter diversas chaves candidatas.
- **Chave Primária (PK):** é a chave candidata escolhida para a identificação das tuplas de uma relação.
- **Chave Alternativa:** são chaves candidatas que não foram escolhidas como chave primária.
- **Chave Estrangeira (FK):** é um atributo ou uma combinação de atributos de uma relação (seja a relação B, por exemplo) dos quais os valores aparecem obrigatoriamente na chave primária de uma outra relação (seja a relação A, por exemplo). Este conceito é o que possibilita implementar relacionamentos em um banco de dados relacional. É possível uma relação A conter uma chave estrangeria para a própria relação, explicitando a ocorrência de um auto-relacionamento.

A definição de chave estrangeira estabelece o que se chama de **integridade referencial**. Esta integridade diz que se uma chave estrangeira existe em uma relação, o valor desta chave deve coincidir com o valor da chave primária da tabela referenciada, ou seu valor deve ser nulo.

A Figura 6 exemplifica o estabelecimento de um relacionamento entre duas relações (Filial e Funcionário) por meio das chaves primária e estrangeira.

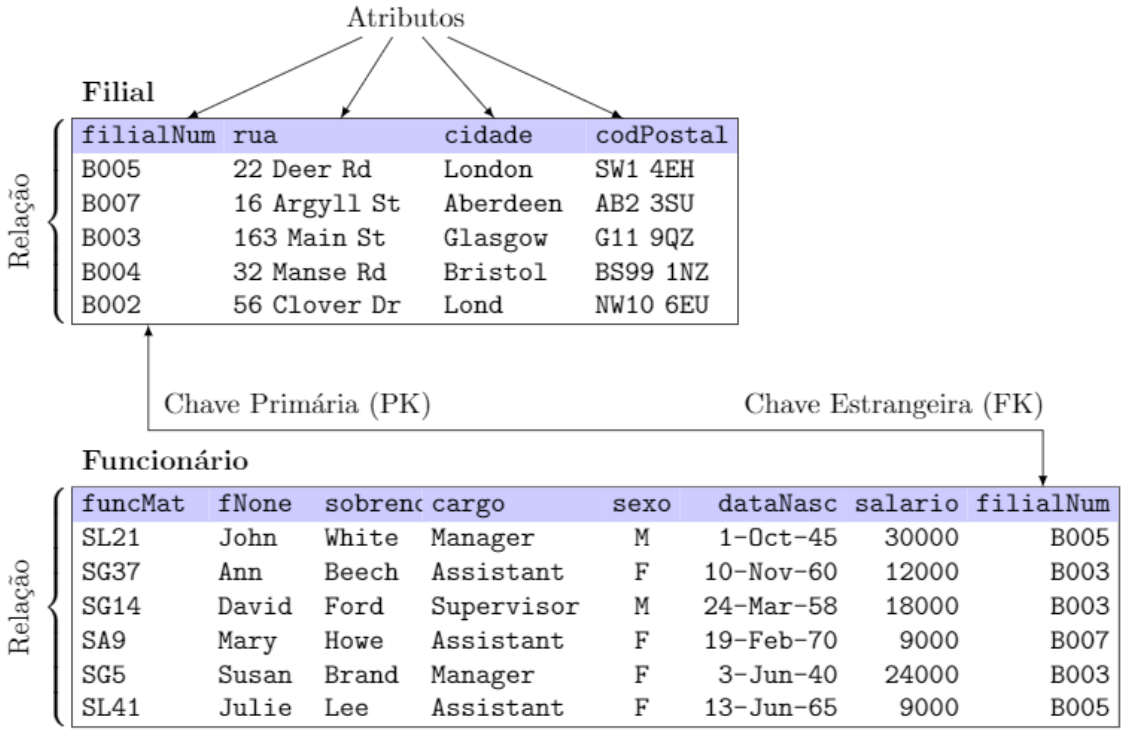


Figura 6 – Exemplo de Chaves Primária e Secundária (CONNOLLY; BEGG, 2004).

2.2.3 Projeto de Banco de Dados

O projeto de banco de dados compreende as ações realizadas com o objetivo de elaborar a estrutura que servirá para o armazenamento e gerenciamento dos dados do usuário final. Esta etapa é fundamental, já que um banco de dados mal projetado pode ocasionar erros difíceis de serem rastreados, além de gerar informações que norteiem tomadas de decisões equivocadas.

Dentre as atividades presentes no projeto de banco de dados até a sua implementação, podem-se citar: modelagem do banco em nível conceitual e lógico e as transformações entre os modelos de Entidade-Relacionamento para o Relacional.

O ponto inicial do projeto de uma aplicação de banco de dados compreende a caracterização das necessidades de dados para os usuários do banco que está sendo projetado. Para isto, o projetista do banco precisa interagir com os especialistas e usuários. Em seguida, é escolhido um modelo de dados que transcreve para um esquema conceitual do banco de dados (**projeto conceitual**) estas necessidades. Geralmente, o diagrama utilizado é o modelo de entidade-relacionamento, que representa as entidades, relacionamentos e restrições entre estas - detalhes que envolvem o armazenamento físico dos dados não são levados em consideração nessa etapa. A transição deste modelo abstrato para a implementação do banco de dados ocorre nas duas etapas finais do projeto.

Na etapa do **projeto lógico** é feito o mapeamento do modelo conceitual para o modelo de implementação do sistema de banco de dados. Na etapa do **projeto físico** são especificados os recursos físicos do banco de dados, como por exemplo a organização dos arquivos e estruturas de armazenamento internas.

Em suma, o processo de projetar um sistema de banco de dados pode ser resumido com o seguinte enunciado: "*dado algum conjunto de dados a ser representado em um banco de dados, de que modo decidimos sobre uma estrutura lógica adequada para esses dados?*".

2.2.4 Modelo de Entidade-Relacionamento

Chen (1976) publicou um artigo que trouxe a definição do modelo de entidade-relacionamento com a finalidade de ser utilizado como base para uma visão unificada dos dados. O modelo de dados de entidade-relacionamento possui três elementos fundamentais: entidades, relacionamentos e atributos.

Uma **Entidade** é uma "coisa" ou "objeto" do mundo real que pode ser distinguível de outros objetos. Uma ocorrência particular de uma entidade é chamada de **instância de entidade** (ou ocorrência de entidade).

Relacionamentos representam associações entre um ou mais entidades. Relacionamentos podem possuir *papéis* que são utilizados quando uma das extremidades do relacionamento precisa ter um nome distinto deste. Papéis são necessários quando é preciso eliminar ambiguidades e comumente são nomeados por substantivos. O **grau** de um relacionamento indica a quantidade de entidades a ele associadas, podendo ser unário, binário, ternário ou n-ário.

Um **relacionamento unário** ou **relacionamento recursivo** ou ainda **auto-relacionamento** acontece quando uma associação se dá entre ocorrências da mesma entidade - nesse caso, um *papel* pode ser utilizado para distinguir os papéis da entidade no relacionamento.

Um **relacionamento binário** é o que mais ocorre usualmente e se dá quando duas entidades estão associadas. O **relacionamento ternário** associa três entidades. Este é utilizado quando o relacionamento binário não é suficiente para descrever as semânticas da associação.

Um conceito importante que envolve relacionamentos é o chamado de **cardinalidade**, que indica quantas ocorrências de uma entidade podem se associar a uma dada ocorrência através do relacionamento. Considerando duas entidades, A e B, associadas no relacionamento, a cardinalidade deste pode ser uma das seguintes:

- **Um-para-um (*One-to-one*):** uma ocorrência de A está associada a **no máximo** com uma ocorrência de B, e uma ocorrência de B está associada **no máximo** com

uma ocorrência de A. O relacionamento um-para-um é representado como $1:1$.

- **Um-para-muitos (*One-to-Many*):** uma ocorrência de A está associada **a qualquer que seja** o número de ocorrências de B. Contudo, uma ocorrência de B pode estar associada a **no máximo** uma ocorrência de A. O relacionamento um-para-muitos é representado por $1:n$.
- **Muitos-para-muitos (*Many-to-many*):** uma ocorrência de A está associada a qualquer número de ocorrências de B (zero ou mais), e uma ocorrência de B pode estar associada a qualquer número de ocorrências (zero ou mais) de B.

Os **Atributos** são dados que se encontram associados a uma ocorrência de entidade ou relacionamento. Eles dão características às entidades e relacionamentos. Uma **instância de um atributo** dentro de uma entidade ou relacionamento é denominado *valor de atributo*. Cada atributo contém um conjunto de valores que lhes são permitidos, chamado de **domínio** ou **conjunto de valores** do atributo.

2.3 Banco de Dados NoSQL

Atualmente prevalece a necessidade de lidar com um grande volume de dados, o que requer uma classe de modelo de dados que permita tratar tais dados de forma eficiente e rápida. Embora o modelo relacional seja um paradigma consagrado e simples, não consegue lidar tão bem com estes dados, apresentando em diversos experimentos presentes na comunidade científica um desempenho inferior na recuperação e tratamento dos dados. Uma nova classe de banco de dados, denominada NoSQL, surge neste cenário como uma alternativa que vem prover estas deficiências do modelo relacional. Nesta seção será discutido este novo modelo de banco de dados e posteriormente com mais detalhes o modelo foco desta dissertação, o Orientado a Grafos e um de seus SGBDs, Neo4j.

2.3.1 Emergência e Características dos Bancos de Dados NoSQL

As aplicações comerciais que utilizam banco de dados relacionais possuem três componentes principais: interface de usuário, regras de negócio e código do banco de dados (comandos que realizam operações sobre o banco). Por diversos anos, os bancos de dados relacionais estiveram no topo das tecnologias utilizadas em aplicações, sendo que estes supriram as limitações dos modelos anteriores. Com o desenvolvimento da Internet, as limitações inerentes ao modelo relacional começaram a aparecer (SULLIVAN, 2015).

Esse novo contexto traz à tona um novo paradigma para desenvolvedores de aplicações para a Web, composto por um grande volume de dados e extensa quantidade de usuários, se deparando com a seguinte situação: grande volume de operações de leitura e

escrita dos dados, baixa latência no tempo de resposta e alta disponibilidade da informação.

A utilização de bancos de dados diante do cenário retratado anteriormente, se mostra problemático, em razão da necessidade de promover uma melhoria com aumento da quantidade de CPUs, memória adicional ou dispositivos de armazenamento mais rápidos. O problema é que existem limitações quanto à quantidade de CPUs e memórias a serem destinados a um servidor, além de técnicas que promovem a melhoria no desempenho do banco de dados (conhecidas como **desnormalização**) aumentam o risco de haver anomalias dos dados (SULLIVAN, 2015).

No início dos anos 2000, ocorreu uma súbita expansão de dados estruturados, semi-estruturados e não estruturados por aplicações de escopo global, resultantes dos avanços nas áreas das Redes Sociais, Tecnologias Móveis, Web, Internet das Coisas e o barateamento do *hardware* (DAVOUDIAN; CHEN; LIU, 2018).

Um modelo de dados descreve a maneira com a qual entidades do mundo real e seus relacionamentos são representados e operados (DAVOUDIAN; CHEN; LIU, 2018). Os modelos existentes de banco de dados NoSQL são categorizados em modelos orientados a:

- Chave-Valor;
- Colunas;
- Documentos e
- Grafos.

Cada um destes modelos possui características que se enquadram em um contexto que é requerido por um aplicativo específico (DAVOUDIAN; CHEN; LIU, 2018). Isso quer dizer que, cada modelo irá se enquadrar melhor em uma necessidade específica de uma aplicação em questão.

Definir NoSQL é um grande desafio, visto que o termo não descreve os conceitos principais do movimento NoSQL. Dessa forma, a definição vem sendo utilizada como meio de se distinguir do termo RDBMS. Para McCreary e Kelly (2013), NoSQL pode ser considerado como um conjunto de conceitos que possibilita o rápido e eficiente processamento de um conjunto de dados objetivando desempenho, confiabilidade e agilidade.

Como consequência da definição, NoSQL pode ser visto como:

- Mais do que tuplas em relações, já que os modelos NoSQL se organizam em forma de grafos, família de colunas, documentos e pares chave-valor.

- Livre de junções (*joins*), já que possível recuperar dados sem a necessidade de realizar operações *join* nas consultas.
- Livre de esquema, devido à flexibilidade dos modelos de dados que o NoSQL oferece.
- Funciona em diversos processadores, já que o modelo permite o armazenamento do banco em múltiplos processadores, mantendo um alto desempenho.
- Suporte à escalabilidade linear, com aumento consistente no desempenho ao acrescentar mais processadores.
- É inovador, com uma nova forma de armazenar, recuperar e manipular os dados.

2.3.2 Consistência e Disponibilidade em Banco de Dados NoSQL

A consistência em banco de dados significa que, após os dados serem persistidos, as consultas podem acessá-los e recuperá-los de forma consistente. No contexto das tecnologias de banco de dados NoSQL, a consistência recai em um dos dois campos (FOWLER, 2015):

- **ACID:** uma vez que o dado é persistido, tem-se consistência total nas leituras deste.
- **BASE (Consistência Eventual):** significa que, uma vez que o dado é persistido, ele eventualmente irá aparecer na leitura.

BASE é a premissa de operação para a maioria dos bancos de dados NoSQL, da mesma forma que RDBMS utilizam as regras ACID (LAKE; CROWTHER, 2013). Os conceitos por trás do BASE são (MCCREARY; KELLY, 2013):

- **Disponibilidade Básica (*Basic Availability*):** transações são gerenciáveis, com os sistemas tornando-se temporariamente inconsistentes.
- **Estado-Leve (*Soft-State*):** reconhece que alguma imprecisão é temporariamente permitida e os dados podem vir a ser modificados enquanto são utilizados para diminuir a quantidade de recursos consumidos.
- **Eventualmente Consistente (*Eventually Consistent*):** significa que eventualmente, quando toda a lógica de serviço é executada, o sistema é colocado em um estado consistente.

Ao analisar os conceitos do BASE, pode-se dizer que este apresenta um equilíbrio ajustável entre consistência e disponibilidade de dados. Sistemas BASE são bem simples e rápidos, já que não precisam lidar com o bloqueio e desbloqueio de recursos. O foco

é manter a execução do processo e posteriormente lidar com "partes quebradas" deste. Sendo os sistemas NoSQL altamente descentralizados e a falta de rigor ao garantir as propriedades ACID, eles utilizam uma abordagem mais "relaxada" ao adotar o BASE (MCCREARY; KELLY, 2013).

A consistência não é absoluta, é uma escala móvel. Diversos bancos de dados NoSQL permitem ajustar os níveis de consistência e disponibilidade, relacionados ao teorema CAP. Este teorema mostra um conjunto de opções sobre como o equilíbrio entre consistência, disponibilidade e particionamento pode ser mantido em um sistema de banco de dados BASE. A sigla CAP significa três propriedades: Consistência, Disponibilidade e Particionamento, elementos importantes no gerenciamento de sistemas de banco de dados (FOWLER, 2015).

O teorema CAP indica que no máximo duas das três propriedades desejáveis podem ser obtidas simultaneamente. Entretanto, ao lidar explicitamente com as partições, pode-se otimizar a consistência e a disponibilidade, conseguindo assim uma compensação entre as três (BREWER, 2012). Estas propriedades são descritas a seguir (MCCREARY; KELLY, 2013):

- **Consistência (C):** não envolve a mesma ideia de consistência existente nas propriedades ACID. A consistência aqui diz respeito a vários clientes que leem os mesmos itens de partições replicadas e obtêm resultados consistentes.
- **Alta Disponibilidade (A):** se refere à capacidade do banco em servir dados aos clientes. Falhas na comunicação interna entre dados replicados não devem impedir atualizações.
- **Tolerância ao Particionamento (P):** se refere à capacidade do sistema de continuar respondendo às solicitações do cliente, mesmo que haja uma falha na comunicação entre as partições do banco de dados.

Ao perder uma das propriedades do teorema CAP, os bancos de dados NoSQL podem ser caracterizados como (DAVOUDIAN; CHEN; LIU, 2018; NEUMANN, 2017):

- **Sistemas CA:** os algoritmos utilizados por estes sistemas não têm qualquer suposição sobre as partições. Optam por priorizar a alta consistência e alta disponibilidade dos dados. Em sistemas distribuídos, é praticamente impossível atingir essa combinação, já que o particionamento das redes é inevitável.
- **Sistemas CP:** sistemas que dão prioridade à alta consistência e possibilidade de particionamento dos dados em vários servidores, mesmo que possa vir a ocorrer

atraso na disponibilidade de um dado. São ideais para sistemas que operam sobre uma rede confiável, como um único *datacenter*, em razão de existirem poucas partições da rede.

- **Sistemas AP:** sistemas que não sacrificam a disponibilidade (jamais podem ficar indisponíveis) e a possibilidade de particionamento dos dados em vários servidores, mesmo que isso implique em falta de consistência dos dados por um curto espaço de tempo. Isso quer dizer que, a execução de gravações conflitantes é permitida, mas acarreta em divergência de réplicas e necessita de um mecanismo de resolução de conflitos.

Nas seções a seguir, serão tratados com mais detalhes as formas de armazenamento de banco de dados NoSQL.

2.3.3 Banco de Dados Orientado a Chave-Valor

O armazenamento por chave-valor é uma DHT (*Distributed Hash Table*) simples que é utilizada quando é necessário que todo o acesso ao banco de dados seja feito pela chave primária (SADALAGE; FOWLER, 2012). Uma **chave** por ser simples ou composta, podendo ser também gerada pelo aplicativo que utiliza esta tecnologia. Um **valor** representa dados com um tipo, estrutura e tamanho arbitrários, sendo identificados por uma chave indexada. Esse modelo é adequado para aplicações que utilizam somente uma chave única para acesso aos dados, como por exemplo: carrinho de compras *online*, configuração de perfil de usuário e informação de sessão na Web (DAVOUDIAN; CHEN; LIU, 2018). Como é utilizado somente o acesso à chave primária na busca dos dados, modelos chave-valor possuem um ótimo desempenho e podem ser facilmente dimensionados (SADALAGE; FOWLER, 2012). Algumas tecnologias que utilizam o modelo orientado a chave-valor são: Redis, Riak KV e Oracle NoSQL.

2.3.4 Banco de Dados Orientado a Família de Colunas

Este modelo é inspirado pelo *Google Bigtable* onde os dados são representados em um formato tabular contendo linhas e uma quantidade fixa de colunas (DAVOUDIAN; CHEN; LIU, 2018). O armazenamento dos dados em famílias de colunas é feito de forma em que cada linha contém algumas colunas associadas com uma chave de linha (*row-key*). Geralmente os dados são acessados em conjunto e as colunas estão relacionadas logicamente com outra (SADALAGE; FOWLER, 2012; DAVOUDIAN; CHEN; LIU, 2018). O esquema é flexível e colunas podem ser adicionadas em tempo de execução. Uma coluna possui um nome e um valor com uma estrutura simples ou complexa. Este modelo estende o de chave-valor, onde o valor é representando por uma sequência aninhada de

pares chave-valor (DAVOUDIAN; CHEN; LIU, 2018). Exemplos de tecnologias que utilizam este modelo são: Apache Cassandra, Apache Hbase, Hypertable, e Google Cloud Bigtable.

2.3.5 Banco de Dados Orientado a Documentos

Este modelo estende os bancos de dados orientados a chave-valor, em que o valor é representado como um documento codificado em formatos semiestruturados, como XML (*Extensible Markup Language*), JSON (*JavaScript Object Notation*) ou BSON (*Binary JSON*) (DAVOUDIAN; CHEN; LIU, 2018). Estes documentos seguem uma estrutura de árvore hierárquica e são auto-descritivos. Os documentos armazenados são semelhantes entre si, mas não precisam ser exatamente iguais (SADALAGE; FOWLER, 2012).

Os documentos são agrupados em **coleções (*collections*)** ou **blocos (*buckets*)**, que se assemelham a estruturas de diretórios encontradas em sistemas operacionais como o Windows ou Linux e representam a mesma categoria de informações (DAVOUDIAN; CHEN; LIU, 2018; SADALAGE; FOWLER, 2012). Alguns exemplos de banco de dados orientados a documentos são: MongoDB, CouchDB, Terrastore, OrientDB, RavenDB.

2.3.6 Banco de Dados Orientado a Grafos

Os GDB (*Banco de Dados Orientado a Grafos*) surgiram com o intuito de possibilitar o armazenamento, processamento e análise de grandes conjuntos de dados em forma de grafos. Processar dados neste formato tornou-se atualmente uma parte essencial em diversas áreas de conhecimento. Muitos destes grafos possuem arestas na casa dos trilhões, sendo estruturalmente dinâmicos, com suas estruturas sendo modificadas a todo o tempo (BESTA et al., 2019). O armazenamento dos dados e relacionamentos é feito sem imposição de um esquema rígido, tornando rápida a sua modelagem e a identificação da associação entre os dados (LEE et al., 2015).

Um Grafo $G = (V(G), E(G), \psi_G)$ é um par ordenado de vértices (V) e arestas (E) disjuntos de V, com uma função de incidência (ψ) que associa cada aresta de G a um par não ordenado (não necessariamente distinto) de vértices de G (BONDY; MURTY, 2008). A Figura 7 ilustra um exemplo de Grafo G, com vértices $V(G) = \{A, B, C, D, E, F\}$ e o conjunto de arestas $E(G) = \{AB, AE, AD, BC, BE, CF, CD, DE, DF, EF\}$ associadas pela função de incidência ψ_G.

Bancos de Dados NoSQL suprem algumas deficiências de RDBMS, como, por exemplo, a dificuldade em lidar com modelos de dados flexíveis. Os Bancos de Dados Orientados a Grafos, em especial o Neo4j, são tomados como um tipo particular de banco de dados NoSQL, chamados de GDB Nativos. Este modelo é construído para persistir e processar dados estritamente em forma de grafos (BESTA et al., 2019).

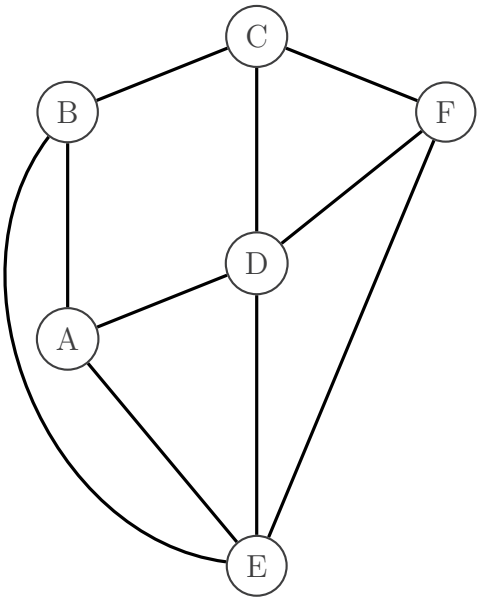


Figura 7 – Exemplo de um Grafo.

O ponto central de um GDB é o relacionamento, em que mesmo que exista valor nas informações discretas sobre as entidades representadas no grafo, o relacionamento entre estas possui um valor ainda maior (POKORNÝ; VALENTA; TROUP, 2019). Os dados armazenados são considerados em termos de um multigrafo (um grafo onde dois nós podem estar conectados por arestas múltiplas), comumente chamado de grafo de propriedades. Outra propriedade importante de GDBS (*Bancos de Dados Orientado a Grafos*) é chamada de *index-free adajcency* (VIRGILIO; MACCIONI; TORLONE, 2014).

Definição (Index-Free Adjacency)**.** *Um GDB é dito satisfazer a propriedade Index-Free Adjacency se a existência de uma aresta entre dois nós do grafo pode ser testada visitando estes nós sem a necessidade da existência de um índice externo e global.*

De acordo com a propriedade anterior, cada nó possui informação acerca de seus vizinhos e não existe qualquer índice de alcançabilidade entre nós. Consequentemente, o percurso de uma aresta de um nó independe do tamanho dos dados e pode ser testado em tempo constante. Além disso, a análise local pode ser realizada de forma mais eficiente tornando GDBS mais adequados para cenários onde dados crescem rapidamente (VIRGILIO; MACCIONI; TORLONE, 2014).

2.3.6.1 O Modelo de Grafo de Propriedades

A grande maioria dos sistemas de banco de dados orientados a grafos atualmente utilizam o modelo de grafo de propriedades. Ainda não há uma especificação formal padrão do modelo deste tipo de banco de dados, em razão do mercado ainda ser bastante fragmentado (ANGLES, 2018).

O **Modelo de Grafo de Propriedades** é a forma mais popular de representação do modelo de grafo para bancos de dados, possuindo as seguintes características (ROBINSON; WEBBER; EIFREM, 2015):

- Contém Nós (Vértices) e Relacionamentos (Arestas).
- Nós possuem propriedades (pares chave-valor).
- Nós podem ser rotulados com um ou mais rótulos (*labels*).
- Relacionamentos possuem um nome e são dirigidos, e sempre possuem um Nó de origem e destino.
- Relacionamentos também podem possuir propriedades.

A pesquisa feita por Angles (2018) traz uma definição formal do que seria o Modelo de Grafo de Propriedades, descrito anteriormente. Assumindo que **L** é o conjunto infinito de rótulos (*labels*), **P** é o conjunto infinito dos nomes de propriedades, **V** é o conjunto infinito de valores atômicos, **T** é o conjunto finito de tipos de dados (por exemplo, inteiros), segue a definição:

Definição (Grafo de Propriedades)**.** *Um Grafo de Propriedades é uma tupla* $G = (N, E, \rho, \lambda, \sigma)$*, onde:*

1. *N é o conjunto finito de nós (também chamado de vértices).*
2. *E é o conjunto finito de arestas em que E não possui qualquer elemento em comum com N.*
3. $\rho : E \longrightarrow (N \times N)$ *é uma função total que associa cada aresta E com um par de nós N (na Teoria dos Grafos corresponde à função de incidência).*
4. $\lambda : (N \cup E) \longrightarrow SUBSET(\boldsymbol{L})$ *é uma função parcial que associa um nó/aresta com um conjunto de rótulos de* $\boldsymbol{L}$*.*
5. $\sigma : (N \cup E) \times \boldsymbol{P} \longrightarrow SUBSET(\boldsymbol{V})$ *é uma função parcial que associa nós/arestas com propriedades e, para cada uma destas, um conjunto de valores de* $\boldsymbol{V}$*.*

Este é o modelo adotado nesta pesquisa para representação do modelo de grafo para um GDB.

2.4 Neo4j

2.4.1 Armazenamento e Organização de Arquivos

Os arquivos no Neo4j são armazenados para terem durabilidade a longo prazo, sendo que, estão localizados no diretório `data/databases/graph.db (v3.x+)` por padrão no diretório de dados do Neo4j (ROCHA, 2019).

Sendo o Neo4j um banco de dados sem a presença de um esquema rigidamente definido, o SGBD utiliza comprimentos de registros fixos para a persistência dos dados e as compensações nesses arquivos são acompanhadas para saber como buscar dados para responder às consultas. A tabela 3 demonstra os tamanhos fixos que o Neo4j utiliza para o tipo de objetos Java que estão sendo persistidos (ROCHA, 2019).

Nome do Arquivo	**Tamanho**	**Conteúdo**
`neostore.nodestore.db`	15 B	Nós
`neostore.relationshipstore.db`	34 B	Relacionamentos
`neostore.propertystore.db`	41 B	Propriedades de Nós e Relacionamentos
`neostore.propertystore.db.strings`	128 B	Valores de Propriedades do tipo *string*
`neostore.propertystore.db.arrays`	128 B	Valores de Propriedades do tipo *array*
Propriedade Indexada	1/3*AVG(X)	Cada entrada de índice tem aproximadamente 1/3 da média do tamanho do valor da propriedade

Tabela 3 – Tamanhos fixos de dados utilizados pelo Neo4j.

Algumas considerações importantes sobre as propriedades (ROCHA, 2019):

- O registro da propriedade possui uma carga útil de 32 *bytes*, sendo dividida em quatro blocos de 8B. Cada campo pode conter uma chave ou um valor, ou uma chave e um valor.
- A chave e o tipo ocupam 3,5 *bytes* (4 *bits* para a chave e 24 *bits* para o tipo).
- Valores do tipo *boolean*, *short*, *integer*, *char* e *float* são ajustados nos *bytes* restantes do mesmo bloco.

Como visto anteriormente, o Neo4j armazena os registros do grafo em diferentes arquivos. Cada arquivo possui informações de uma parte em específico do grafo. Essa separação da responsabilidade de armazenamento torna mais fácil o percurso no grafo, aumentando o seu desempenho, mesmo que isso signifique que a visualização do grafo pelo usuário e os registros reais no disco estejam estruturalmente diferentes.

Quando um nó é criado pelo usuário, seu armazenamento é feito no arquivo `neostore.nodestore.db` sendo de tamanho fixo, em que cada registro tem 9 *bytes* de comprimento. Manter os registros com tamanho fixo possibilita pesquisas mais rápidas dos nós. Como exemplo, suponha-se que o nó com `id` 100 seja buscado no banco. Sabe-se que seu registro tem início em 900 *bytes* no arquivo. Partindo do modelo considerado, o banco de dados efetua o cálculo da localização de um registro de forma direta, com custo $O(1)$, ao invés de realizar uma pesquisa, que custaria $O(logn)$. O primeiro *byte* do nó é uma *flag* que indica que o registro atual está sendo utilizado para guardar ou recuperar informações de um nó. Os 4 *bytes* seguintes guardam o ID do primeiro relacionamento conectado ao nó, e os próximos 4 *bytes* representam o ID da primeira propriedade do nó. Os 5 *bytes* reservados para os rótulos (*labels*) dos nós apontam para o armazenamento de rótulos desse nó. O *byte* final extra é reservado para outras *flags*. Uma dessas *flags* é utilizada na identificação de nós densamente conectados e o resto do espaço é reservado para uso futuro. O registro do nó é bastante simples e leve: são apenas alguns ponteiros para listas de relacionamentos, rótulos e propriedades (ROBINSON; WEBBER; EIFREM, 2015).

Da mesma forma que os nós, os relacionamentos são armazenados em um arquivo específico para relacionamentos, o `neostore.relationshipstore.db` e ocorre em registros de tamanho fixo. Cada registro de relacionamento contém os valores que identificam (`ids`) os nós de origem e destino do relacionamento, um ponteiro para o tipo de relacionamento (armazenado no arquivo que contém registros do tipo de relacionamento), ponteiros para os registros de relacionamento seguintes e anteriores para cada um dos pontos de origem e nós de destino e uma *flag* indicando se o registro atual é o primeiro do que é chamado de cadeia de relacionamento (ROBINSON; WEBBER; EIFREM, 2015).

O Neo4j permite a replicação e provê um certo nível de suporte à fragmentação. Isso quer dizer que o usuário pode particionar o grafo e armazenar cada partição em um banco de dados distinto, limitando a redundância de dados. Além disso, garante totalmente as propriedades ACID (BESTA et al., 2019).

2.4.2 A linguagem *Cypher*

O *Cypher* é a linguagem de consulta e atualização de bancos de dados orientados a grafos que teve início ao ser utilizado no Neo4j e atualmente está implementado comercialmente em outros produtos, como *SAP HANA Graph*, *Redis Graph*, *Agens Graph* (sobre o PostgreSQL) e *Memgraph* (FRANCIS et al., 2018). É uma linguagem relativamente simples, mas poderosa. Consultas complexas ao banco de dados podem ser facilmente expressas utilizando *Cypher*. Sendo uma linguagem declarativa, o foco do *Cypher* é expressar o que é recuperado em um grafo e não como recuperar. A estrutura de uma consulta em *Cypher* se assemelha ao SQL, ao utilizar diversas cláusulas que encontram-se interligadas.

Um ponto interessante da linguagem *Cypher* é que qualquer consulta que realiza alguma alteração no grafo ocorrerá em uma transação e sempre será bem sucedida ou não sucedida caso ocorra algum erro. Assim que a alteração dos dados se inicia, ela acontecerá iniciando uma nova transação ou será executada dentro de uma já existente (JORDAN, 2014). A linha de consulta da linguagem é linear, ou seja, permite que o usuário pense no processamento da consulta como partindo do texto inicial da consulta e progredindo linearmente até o fim. Cada cláusula em uma consulta é uma função que tem como entrada uma tabela, gerando então outra tabela que pode expandir a quantidade de campos e adicionar novas tuplas. Toda a consulta é, então, a junção de tais funções (FRANCIS et al., 2018).

Existe um projeto chamado *Open Cypher*[1] que procura estabelecer o uso do *Cypher* como uma linguagem padronizada para trabalhar com modelos de grafos de propriedades assim como o SQL é o padrão para bancos de dados relacionais (FRANCIS et al., 2018).

A tabela 4 traz uma breve equivalência entre cláusulas SQL e as existentes no *Cypher*.

Clásula SQL	**Cláusula em *Cypher***	**Operação**
INSERT	CREATE	Inserção de um registro.
ALTER TABLE	Apenas adiciona uma nova propriedade ao nó existente.	Adicionar uma coluna existente em em uma relação.
SELECT	START / MATCH	Recuperação de informações. O uso de START no Cypher inicia a recuperação em um nó contendo um valor em específico.
UPDATE	SET	Modifica registros existentes.
DELETE	DELETE	Remove um registro.
-	RETURN	Retorna partes do grafo.
-	WITH	Passa resultados de uma sub-consulta para a parte de uma consulta seguinte. Às vezes utiliza a cláusula DISTINCT para remover duplicatas.
-	SKIP	Define a partir de qual tupla o resultado será retornado.
LIMIT	LIMIT	Limita a quantidade de tuplas retornadas.
ORDER BY	ORDER BY	Especifica como o resultado será ordenado.

Tabela 4 – Equivalência entre cláusulas SQL e *Cypher*.

[1] https://www.opencypher.org/

3 Trabalhos Relacionados - Revisão Sistemática de Literatura

Este capítulo é resultante de uma Revisão Sistemática de Literatura realizada com o objetivo de explorar estudos voltados à conversão e migração de RDBMS para GDB. O resultado desta revisão resultou em publicação em congresso que pode ser vista em (OLIVEIRA et al., 2020).

Diversas técnicas utilizadas para o mapeamento e conversão entre modelos de banco de dados relacional para orientado a grafos além de outros modelos NoSQL de maneira geral foram encontradas nos artigos selecionados para o estudo. No que se refere ao mapeamento conceitual, foram identificadas técnicas que utilizam apenas o modelo conceitual (CLAUDINO; SOUZA; SALGADO, 2015) e outra que utiliza redes neurais artificiais para mapear o banco relacional para conversão (LIYANAARACHCHI et al., 2016). Quanto aos métodos que realizam a conexão com bancos de dados relacionais a fim de mapeá-los para outros bancos NoSQL ou em artigos que mapeiam e convertem para bancos orientados a grafos especificamente, são utilizados: conexão via *Java Database Connectivity* (JDBC) (MEGID; EL-TAZI; FAHMY, 2018; VIRGILIO; MACCIONI; TORLONE, 2013; MOUDEN; JAKIMI; HAJAR, 2019; UNAL; OGUZTUZUN, 2018; KUSZERA; PERES; FABRO, 2019; ALAMI; BAHAJ, 2016), no *core* do banco de dados relacional por meio de procedimentos armazenados (*stored procedures*) (OREL; ZAKOŠEK; BARANOVIČ, 2016). Quanto aos arquivos gerados após o mapeamento e que serão utilizados para a conversão, são utilizados os seguintes formatos: CSV (*Comma-separated Values*), XML e JSON. Estes tópicos são abordados de forma resumida nas seções seguintes, sendo apresentadas em duas categorias: a que trata do processo de mapeamento e a que trata do processo de conversão, quando serão abordadas, de forma geral, as técnicas que convertem um banco de dados relacional para outros NoSQL a fim de compreender o processo utilizado para a extração da estrutura do banco de origem e, de forma mais detalhada, as técnicas que convertem para banco de dados orientado a grafos.

3.1 Execução da Pesquisa

Para que a pesquisa fosse executada, foram definidos critérios com o objetivo de viabilizar a sua execução. A pesquisa ocorreu com a consulta nas bases de dados Scopus e IEEE Xplore, utilizando-se a seguinte expressão de busca:

```
("Relational Database Model" OR "Relational Database" OR mysql OR postgres OR
sqlserver OR Oracle) AND (Migration OR Conversion) AND (NOSQL OR "graph
databases" OR neo4j)
```

Foram incluídos os artigos que se enquadram dentro do período de 1º de Janeiro de 2013 até 31 de Maio de 2019. A Figura 8 ilustra as epatas do processo de pesquisa. Dos 55 artigos resultantes da aplicação da expressão de busca, foi aplicado na primeira etapa o critério que analisa o título. Os artigos que não possuíam em seu título menções à migração e/ou mapeamento entre banco de dados relacionais a bancos NoSQL, ou mais especificamente orientado a grafos, foram removidos. Na fase seguinte, foram aplicados os critérios de Inclusão e Exclusão, descritos nas Tabelas 5 e 6. Após a aplicação dos critérios na leitura dos *abstracts*, resultaram 27 artigos e a última etapa foi realizada com uma análise criteriosa por meio de leitura completa dos artigos, resultando ao final em 11 artigos que são apresentados na Tabela 7.

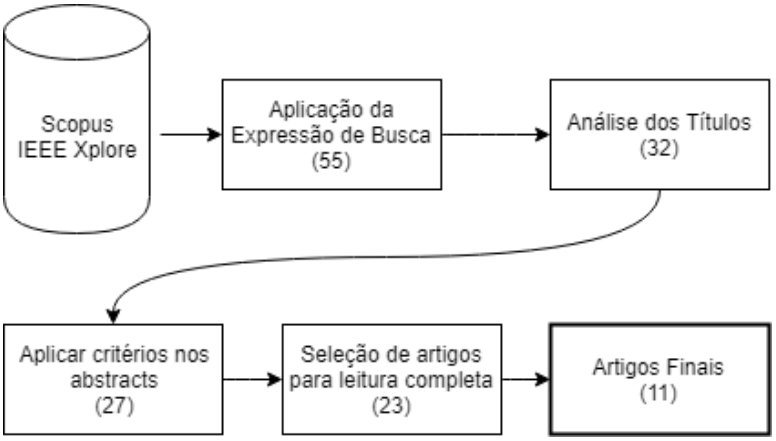


Figura 8 – Etapas do processo de pesquisa

Identificador	**Descrição**
CI-01	Podem ser selecionados artigos que tratem do processo de mapeamento e/ou conversão entre banco de dados relacionais para banco de dados orientados a grafos.
CI-02	Podem ser selecionados artigos que demonstrem algoritmos que mapeiem estruturas do banco relacional para bancos de dados NoSQL.
CI-03	Podem ser selecionados artigos que mencionam o mapeamento conceitual entre banco de dados relacionais para bancos de dados NoSQL.
CI-04	Podem ser selecionados artigos que realizem a representação de um banco de dados relacional para um modelo orientado a grafos (sem a persistência em um banco de dados).

Tabela 5 – Conjunto de Critérios de Inclusão

Identificador	Descrição
CE-01	Não serão selecionadas publicações que não contenham no título as palavras-chave contidas na expressão de busca.
CE-02	Não serão aceitas publicações que apenas façam menção das palavras presentes na expressão de busca e não apresentem qualquer metodologia de mapeamento ou conversão entre os modelos relacional para NoSQL e/ou orientado a grafos.

Tabela 6 – Conjunto de Critérios de Exclusão

nº	Autores (Ano)	Título	Critério
1	(UNAL; OGUZTUZUN, 2018)	Migration of Data from Relational Database to Graph Database	CI-01
2	(VIRGILIO; MACCIONI; TORLONE, 2013)	Converting Relational to Graph Databases	CI-01
3	(OREL; ZAKOŠEK; BARANOVIČ, 2016)	Property-Oriented Relational to Graph Database Conversion	CI-01
4	(MOUDEN; JAKIMI; HAJAR, 2019)	An Algorithm of Conversion Between Relational Data and Graph Schema	CI-04
5	(MEGID; EL-TAZI; FAHMY, 2018)	Using Functional Dependencies in Conversion of Relational to Graph Databases	CI-01
6	(LIYANAARACHCHI et al., 2016)	MigDB - Relational to NoSQL mapper	CI-02
7	(ROCHA, 2019)	A Framework for Migrating Relational Datasets to NoSQL	CI-02
8	(ALAMI; BAHAJ, 2016)	Migration of a Relational Databases to NoSQL: The Way Forward	CI-03
9	(HANINE; BOUTKHOUM, 2015)	Data Migration Methodology from Relational to NoSQL Databases	CI-03
10	(KUSZERA; PERES; FABRO, 2019)	Toward RBD to NoSQL: Transforming Data with Metamorfose Framework	CI-02
11	(CLAUDINO; SOUZA; SALGADO, 2015)	Mapeamentos conceituais entre os modelos relacional e NoSQL: uma abordagem comparativa	CI-03

Tabela 7 – Artigos incluídos na revisão

3.1.1 Extração dos Dados

Assim que foram definidos os artigos finais para análise, foi efetuado um processo de categorização destes com base na leitura completa de cada um. Os artigos foram categorizados seguindo os seguintes critérios:

- Tratam apenas do Mapeamento Conceitual entre os modelos relacional e NoSQL de forma geral.
- Definem algoritmos de Conversão e/ou Migração de banco de dados relacional para NoSQL.
- Realizam a Conversão e Migração de banco de dados relacional para orientado a grafos.

As seções seguintes discutem cada uma destas categorias apresentando resumidamente a técnica utilizada em cada artigo.

3.2 Mapeamento entre Modelos Relacionais para Orientados a Grafos

Uma das técnicas de mapeamento para bancos de dados NoSQL encontrada nos artigos utiliza o mapeamento conceitual para transformação em diversos modelos NoSQL (CLAUDINO; SOUZA; SALGADO, 2015). A técnica consiste em representar um cenário por meio de modelagem conceitual e sua representação semelhante ao modelo lógico. A estratégia de mapeamento para banco de dados orientados a grafos consiste em utilizar um diagrama de classes UML (Linguagem de Modelagem Unificada) em que, as instâncias de dados são representadas por nós (vértices) sendo distinguidos por meio de rótulos (que possuem os nomes das tabelas que referenciam) e os relacionamentos entre os nós são representados por arestas - no modelo relacional os relacionamentos representam as chaves estrangeiras.

Em Liyanaarachchi et al. (2016), é feito o mapeamento de um banco de dados relacional para NoSQL utilizando redes neurais artificiais. A proposta apresentada pelos pesquisadores consiste na criação de um documento JSON com a estrutura do banco e um módulo que identifica a cardinalidade das relações entre tais tabelas. O mapeamento das cardinalidades 1:1, 1:N e N:N é feito utilizando redes neurais artificiais para posterior conversão em um banco de dados NoSQL.

Em Neumann (2017)[1] o mapeamento se dá conforme o conhecimento do projetista do banco de dados relacional do Diagrama de Entidade-Relacionamento. O diferencial do método empregado é o conceito de incorporação das entidades presentes no DER (Diagrama de Entidade-Relacionamento). Com isso, uma entidade do modelo pode ser unida a outra entidade ou não, transformando-se em uma entidade apenas caso seja incorporada. Mais detalhes deste processo é discutido no Capítulo seguinte.

3.3 Conversão entre Banco de Dados Relacionais para Orientados a Grafos

Em Virgilio, Maccioni e Torlone (2013) é proposta uma abordagem que automatiza a migração de um banco de dados relacional para outro orientado a grafos. No processo de conversão, o modelo relacional é representado em forma de grafo, que considera as

[1] A consulta desta referência ocorreu em ocasião posterior à conclusão da Revisão Sistemática de Literatura.

chaves primárias e estrangeiras. A técnica apresentada consiste na definição de caminhos baseados em junções entre as tabelas do banco. São agrupados em um mesmo nó os dados recuperados de diferentes tuplas mas que se encontram relacionadas - neste caso, na existência de junções entre as tabelas. Tal método pode ocasionar grande armazenamento dos dados em um nó - o que não é interessante para um modelo orientado a grafos. Como resultado da técnica empregada, elaboraram uma aplicação denominada R2G implementada em linguagem Java.

A pesquisa feita por Orel, Zakošek e Baranovič (2016) propõe um algoritmo universal que realiza a conversão entre os modelos relacional para orientado a grafos, e para isto, consideram alguns aspectos importantes no processo de migração: identificação de qual tabela irá ser convertida em um nó ou relacionamento do grafo; ordem de criação dos nós e relacionamentos; atributos que são chave primária que serão utilizados como id's, atributos que são chaves estrangeiras e se tornarão relacionamentos e atributos que não são partes de uma chave que se tornarão parte das propriedades de um nó ou relacionamento. O algoritmo foi implementado como uma *stored procedure* no SGBD *Informix* e os dados convertidos na estrutura de grafos foram persistidos no Neo4j. Em testes realizados comparando o algoritmo com a técnica desenvolvida por Virgilio, Maccioni e Torlone (2013) (foram utilizadas as mesmas bases de dados testadas por estes), os resultados mostraram uma redução na quantidade de nós e, consequentemente, nos relacionamentos gerados, mas com uma quantidade maior de propriedades para cada um destes.

Já Unal e Oguztuzun (2018) representaram a estrutura de um documento legal, neste caso uma Lei, de forma hierárquica para estudar o procedimento de migração entre as informações armazenadas em um banco de dados relacional (MySQL) para um orientado a grafos (Neo4j). A técnica de modelagem explorada pelos pesquisadores consistiu em estruturar os componentes que formam uma lei seguindo uma hierarquia em que, cada entidade representa uma parte da lei. As entidades representadas são (em ordem decrescente de hierarquia): Lei, Cláusula e Parágrafo, sendo importante destacar que estas entidades podem estar relacionadas com entidades de outros documentos legais. As informações dos documentos legais foram persistidas no banco de dados MySQL seguindo a modelagem citada. Na conversão entre o modelo relacional para grafo, as chaves primárias são transformadas em arestas do grafo e cada tabela é transformada em rótulo de um nó e cada tupla desta tabela é transformada em uma instância de nó no modelo de grafo. Ao realizar os experimentos de migração dos dados, foi estabelecida uma conexão *JDBC* para o acesso ao banco relacional e, para a extração dos metadados do banco, foi utilizada a aplicação *Schema Crawler*. As particularidades desta aplicação são: para atributos que aparecem frequentemente, foram criados índices; remoção de dados com valores *default*; dados que se encontram desnormalizados ou duplicados foram colocados em nós separados para obter um modelo mais limpo; tabelas de junções (*join tables*) são transformadas em relacionamentos (arestas do grafo) e as colunas em tais tabelas tornam-se propriedades

destes.

Em Megid, El-Tazi e Fahmy (2018) as autoras apresentam uma melhoria na técnica proposta por Virgilio, Maccioni e Torlone (2013) e implementam o algoritmo denominado FD2G. O algoritmo segue a ideia do método que serviu como base para a melhoria, com a diferença de que aquela busca superar falhas no suporte a múltiplos tipos de relacionamentos, como relações unárias e entidades associativas, sem a presença de chaves estrangeiras. Além disso, a técnica aproveita a existência de informações de dependências funcionais existentes no banco de dados relacional que será migrado a fim de executar automaticamente a conversão para um banco de dados orientados a grafos. Sendo assim, são apresentadas diversas falhas que o algoritmo da técnica R2G não apresenta no artigo escrito por Virgilio, Maccioni e Torlone (2013), como a grande quantidade de nós gerados e perda de dados no processo de migração. Além disso, o algoritmo FD2G normaliza o banco de dados relacional para a terceira forma normal antes da conversão ser inicializada e possui tempo de processamento superior, em comparação ao R2G, utilizando as mesmas bases de dados que foram testadas na aplicação desenvolvida por Virgilio, Maccioni e Torlone (2013).

A pesquisa de Mouden, Jakimi e Hajar (2019) apresenta uma abordagem para a conversão de dados em banco de dados relacionais para um modelo orientado a grafos utilizando a representação destes em formato XML (*eXtensible Markup Language*). Para tanto, deram continuidade a um trabalho anterior que consistiu em detectar comunidades em grafos utilizando algoritmos de clusterização espectrais. Sendo assim, no processo de classificação, a entrada é o dado persistido no modelo relacional e um algoritmo converte este em grafo utilizando um formato *open source* denominado GEXF (*Graph Exchange XML Format*). A análise do trabalho permitiu verificar que o propósito dos autores não era criar uma aplicação que executasse estritamente a conversão entre os modelos, mas sim processar e classificar os dados persistidos no banco relacional em razão de sua similaridade e, ao aplicar um algoritmo de conversão, criar um grafo explicitando seus relacionamentos de acordo com sua similaridade.

3.4 Discussão dos Métodos

Os artigos selecionados para a elaboração da revisão demonstraram diferenças quanto às técnicas empregadas, como nos métodos de conexão e inspeção do esquema do banco de dados de dados relacional, no mapeamento das informações tanto da estrutura do esquema inspecionado quanto dos dados a serem migrados em estruturas de dados distintas e no grafo resultante. Um ponto negativo observado quanto aos artigos coletados, é a ausência de mais estudos direcionados para o contexto de banco de dados orientados a grafos.

Foi observado que, em nenhum dos estudos que se propunham a realizar a conversão entre os modelos, não há testes com outros SGBDs relacionais e estudo do comportamento em ambientes distribuídos, a fim de que possam ser analisados os seguintes aspectos: tempo de execução, eventuais falhas durante a execução do processo, como: perda de dados, comportamento em caso de falha de comunicação durante a sua execução. Além disso, pôde-se verificar que o principal problema no processo de conversão entre os modelos é manter a mesma semântica do banco migrado para o de destino, considerando a abordagem de Virgilio, Maccioni e Torlone (2013) foi verificado por Megid, El-Tazi e Fahmy (2018) que, ao realizar a migração, ocorria perda de dados no processo, por exemplo.

A revisão sistemática realizada consistiu no estudo de referências que tratam do processo de mapeamento e conversão entre os modelos relacional e orientado a grafos, sendo incluídos também artigos que enfocam a migração para outros bancos de dados NoSQL, com o objetivo de identificar outras técnicas de mapeamento de banco de dados relacionais para tais modelos. Esta análise permitiu verificar que há distintas abordagens entre o processo de conversão de um modelo relacional para um orientado a grafos. Cada técnica explora uma forma de conexão com o SGBD Relacional e utiliza uma estrutura de dados que serve como base para a migração entre os modelos. O propósito do estudo foi servir de base para a construção de uma aplicação que, corrigindo as falhas observadas nas técnicas, pode realizar a migração de um banco de dados relacional para um orientado a grafos em qualquer cenário mantendo a mesma semântica do banco de dados de origem, sem que ocorram perda ou distorção dos dados.

4 ThrusterDB

Nesse capítulo são apresentados os detalhes da aplicação desenvolvida, chamada de ThrusterDB, que realiza a conversão automática de um RDBMS para um GDB. Inicia-se com uma apresentação da Linguagem de Programação utilizada além das bibliotecas incluídas. Ademais, são mostrados detalhes em alto nível da aplicação com diagramas UML. Por fim, detalhes em pseudocódigo dos algoritmos que efetuam a conversão são mostrados.

A aplicação foi desenvolvida pautando-se no estudo dos pontos positivos e falhas dos métodos descritos no Capítulo 3. Como resultado da revisão sistemática de literatura e contato com outras pesquisas em momento posterior à realização da revisão, observa-se que existem muitos pontos que não são levados em consideração pelos pesquisadores. Um deles está relacionado ao processo de mapeamento do modelo conceitual para o modelo orientado a grafos. Neste sentido, não há uma atenção à **sobrecarga semântica** que existe neste processo. O modelo relacional é formado por diversos construtores (conforme tratados no Capítulo 2), como entidades, relacionamentos entre as entidades, chaves primárias e estrangeiras, e cardinalidade de relacionamentos. Assim sendo, não existe uma tradução direta de cada um destes construtores para o modelo orientado a grafos, já que este possui apenas os seguintes construtores: nós/entidades (vértices), relacionamentos (arestas) e atributos.

A pesquisa realizada por Orel, Zakošek e Baranovič (2016) não analisa a conversão para quando existe no banco de dados relacionamentos n-ários, com $n > 2$ (n corresponde à quantidade de chaves estrangeiras em uma *joined table*). Já, no trabalho de Megid, El-Tazi e Fahmy (2018), quando foi executado o algoritmo desenvolvido pelos autores, verificou-se que, quando há uma tabela sem a presença de chave primária (mas que tem ao menos uma chave estrangeira), ocorre um erro na criação do relacionamento. Isso implica a não criação do relacionamento e o fim da execução do algoritmo, gerando um banco de dados inconsistente com o de origem. Tal erro, é decorrente do próprio método desenvolvido pela pesquisadora, que estabelece dependências funcionais no modelo a ser migrado.

Parte dos métodos implementados seguem algumas ideias lançadas por Neumann (2017), que trata do processo de aninhamento (chamada de incorporação pelo autor) de relações a outras relações ou a relacionamentos. A diferença do método desenvolvido nesta pesquisa com o elaborado por Neumann (2017) (denominado **Modelagem Participativa**), é que a pesquisa feita por este parte do modelo conceitual, utilizando o DER como ponto de partida para geração de um grafo. Já a presente pesquisa parte do modelo físico,

com o banco de dados já implementado e com dados persistidos em sua base. Detalhes pertinentes ao banco de dados utilizado para o teste da aplicação podem ser vistos no capítulo 5. Na pesquisa do referido autor, o que este chama de relacionamento, é entendido aqui como uma *joined table* e sua cardinalidade é dada considerando a quantidade de chaves estrangeiras que possui.

4.1 Linguagem de Programação

A linguagem de programação utilizada no desenvolvimento da aplicação descrita neste capítulo é a linguagem Python, em sua versão 3.8.2. A biblioteca SQLAlchemy (escrita totalmente em Python) foi utilizada para facilitar a extração dos metadados do banco de dados e facilitar o mapeamento e conversão do modelo relacional para o orientado a grafos.

4.1.1 Biblioteca SQLAlchemy

O SQLAlchemy é uma biblioteca que permite a interação com uma grande quantidade de SGBDs relacionais (como Postgres, MySQL, SQLite, Oracle entre outros). Foi criado em 2005 por Mike Bayer, sendo usado por diversas empresas, dentre elas WakaTime, Backend, e YOU-app (MYERS, 2015).

O objetivo desta biblioteca é fornecer ferramentas que expõe as camadas de interação do banco de dados como uma API rica, se dividindo em duas camadas (BROWN; WILSON, 2012):

- **Core:** possibilita a utilização de instruções na forma da linguagem SQL, que são entendidas pelo banco de dados e o gerenciamento do esquema do banco. Inclui o **Python Database API (DBAPI)**.
- **ORM (*Object-Relational Mapping*):** bastante semelhante a outros ORMs de outras linguagens, fornece uma abstração de alto nível sobre a linguagem SQL. Há a possibilidade de combinar essa abstração com o uso do SQL.

Para esta pesquisa, foi escolhida a camada do Core, pois esta apresenta uma visão voltada para o esquema do banco, que é focado em tabelas, chaves e índices, além de ser indicado quando se quer rigidamente controlar uma consulta (MYERS, 2015). A camada de ORM é indicada quando se tem uma abordagem de *design* controlado por domínio (*domain model driven*), em que grande parte do esquema e estrutura subjacentes nos metadados e objetos de negócios são encapsulados.

O ponto de partida para qualquer aplicação que utiliza o SQLAlchemy é a utilização de uma `Engine`, que permite a comunicação com qualquer combinação de banco de

dados/DBAPI. Por meio da `Engine` é possível utilizar o objeto `Inspector` que fornece métodos que retornam uma análise profunda do esquema do banco de dados.

Com isso, é viável a obtenção dos seguintes tipos de informações relacionadas ao banco (que foram utilizadas no desenvolvimento da aplicação desta pesquisa):

- **get_table_names(*schema_name*):** retorna o nome de todas as tabelas de um dado esquema.
- **get_foreign_keys(*table_name*):** retorna todas as informações referentes a uma chave estrangeira de uma dada tabela.
- **get_pk_constraint(*table_name*):** retorna informações referentes da chave primária de uma dada tabela.
- **get_columns(*table_name*:)** retorna informações detalhadas a respeito das colunas de uma dada tabela.

Dados os métodos descritos anteriormente, é evidente a facilidade de trabalhar com a biblioteca SQLAlchemy para o mapeamento do esquema do banco de dados e realizar sua conversão para um modelo orientado a grafos.

4.2 Arquitetura da Aplicação

Considerando o domínio do problema da ser resolvido, a aplicação foi desenvolvida seguindo o paradigma de programação orientado a objetos. Existem duas classes, uma denominada `GraphDatabase` e outra `RelationalDatabase` que contém as variáveis necessárias para a execução dos métodos. A execução é feita através do código a seguir, onde os valores são passados por argumentos via linha de comando.

```
python main.py DB_TYPE USER PASSWORD HOST DB_NAME
```

Os argumentos do código anterior correspondem aos valores necessários para a criação da conexão com o banco de dados relacional, iniciando a fase de mapeamento. Cada argumento corresponde a um determinado valor:

- `DB_TYPE`: tipo do SGBD relacional (MySQL, SQLServer, PostgreSQL, por exemplo).
- `USER`: nome do usuário do banco de dados.
- `PASSWORD`: senha do usuário do banco de dados.
- `HOST`: *host* de acesso ao banco de dados.

- `DB_NAME`: nome da base de dados.

Em razão da portabilidade da linguagem Python, a aplicação pode ser executada em qualquer sistema operacional. A Figura 9 contém parte do corpo do método principal (*main*) e mostra o momento da execução de cada fase e seus métodos.

```
db_name = sys.argv[1]
r = rdb.RelationalDatabase(sys.argv[2], sys.argv[3], db_name, sys.argv[4], sys.argv[5])
g = gdb.GraphDatabase()

total_reg = r.total_registros()
start_time = time.time()

# Fase de Mapeamento da Estrutura do Banco de Dados
print("**** Mapeamento da Estrutura ****")
start_map = time.time()
g_int = r.calcula_grau_integracao()

t.add_row(['I. Cálculo do Grau de Integração', (time.time() - start_map)])
start_rel = time.time()
fk = r.relacionamento_info(g_int)
db = r.db_info()
fk = r.integracao(g_int, fk, db)
t.add_row(['II. Realização da Integração', (time.time() - start_rel)])
t.add_row(['Mapeamento da Estrutura', (time.time() - start_map)])

# Fase de Conversão da Estrutura Mapeada para o Modelo de Grafo a ser migrado
print("**** Conversão do Modelo ****")
start_convert = time.time()
r.cria_nos_e_relacionamentos(g_int, fk, db)
t.add_row(['Conversão do Modelo', (time.time() - start_convert)])

# Fase de migração dos Dados para o Neo4j
print("**** Criação dos Nós ****")
start_node_create = time.time()
total_nos = g.cria_nos(r.nos)
t.add_row(['Criação dos Nós', (time.time() - start_node_create)])

print("**** Criação dos Relacionamentos ****")
start_relationship_create = time.time()
total_relacionamentos = g.cria_relacionamentos(r.relacionamento)
t.add_row(['Criação dos Relacionamentos', (time.time() - start_relationship_create)])

print("Mapeamento, Conversão e Migração do Banco de Dados finalizados!!!")
```

Figura 9 – Corpo do método principal da aplicação ThrusterDB

Durante a execução, o tempo de execução de cada etapa é armazenado em uma variável que será posteriormente disposta em uma tabela registrada em um arquivo de texto, que servirá para a contagem e análise no que se refere ao tempo e ao tamanho das estruturas de dados.

4.3 Processos de Mapeamento, Conversão e Migração

A aplicação ThrusterDB converte um esquema de banco de dados relacional para um esquema orientado a grafos em três fases de operação: Mapeamento, Conversão e Migração.

A Figura 10 esquematiza cada um dos processos, dando uma visão geral do que é realizado em cada etapa. Nas seções (4.3.1) a seguir, estes serão tratados com mais detalhes.

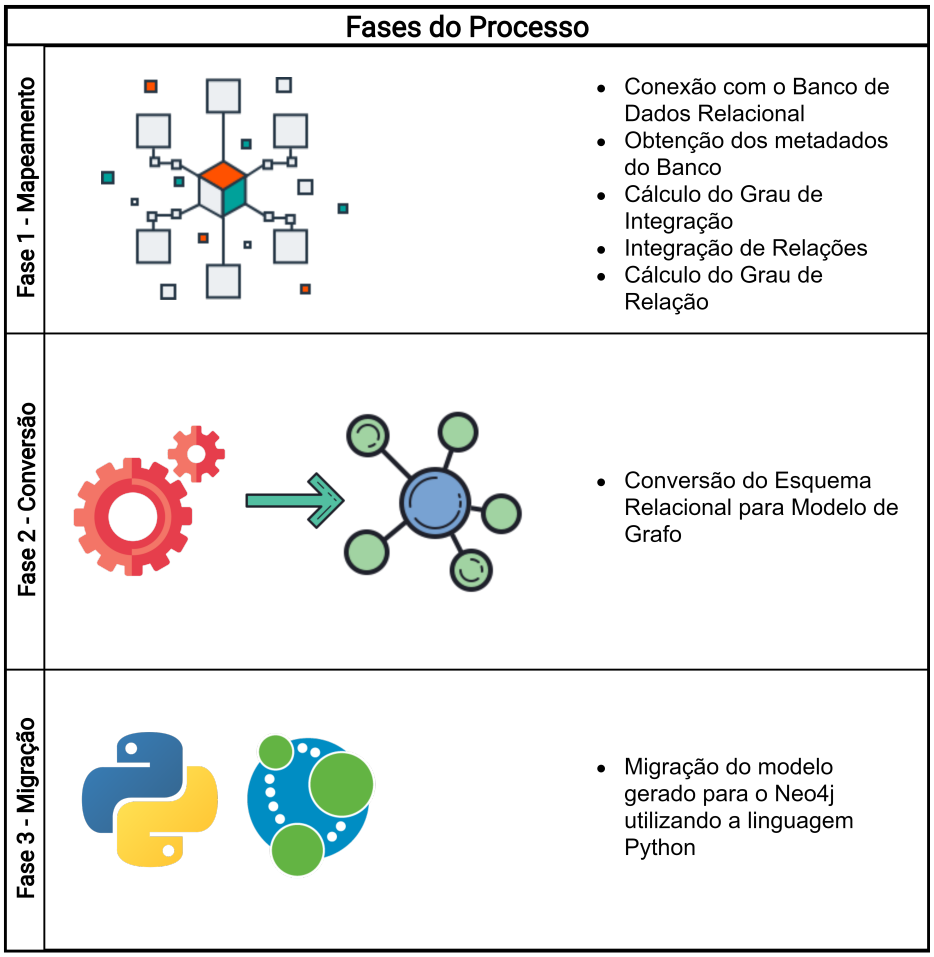


Figura 10 – Fases de operação da aplicação ThrusterDB

4.3.1 Fase 1 - Mapeamento do Esquema Relacional

A **Fase de Mapeamento** é a primeira parte da execução da aplicação e consiste de métodos que, tendo como entrada metadados do banco relacional exportados da biblioteca descrita na seção 4.1.1, são analisados e sobre estes são aplicadas as seguintes operações:

- Cálculo do Grau de Aninhamento das Relações.
- Aninhamento das Relações.

- Cálculo do Nível de Relações.

Cada uma destas operações é detalhada nas seções seguintes e apresentados os algoritmos em pseudocódigo para melhor entendimento do que o método realiza. O diagrama da Figura 11 representa um esquema de banco de dados que será utilizado como exemplo para ilustrar as operações executadas pelos métodos. Ele representa informações básicas sobre uma Universidade.

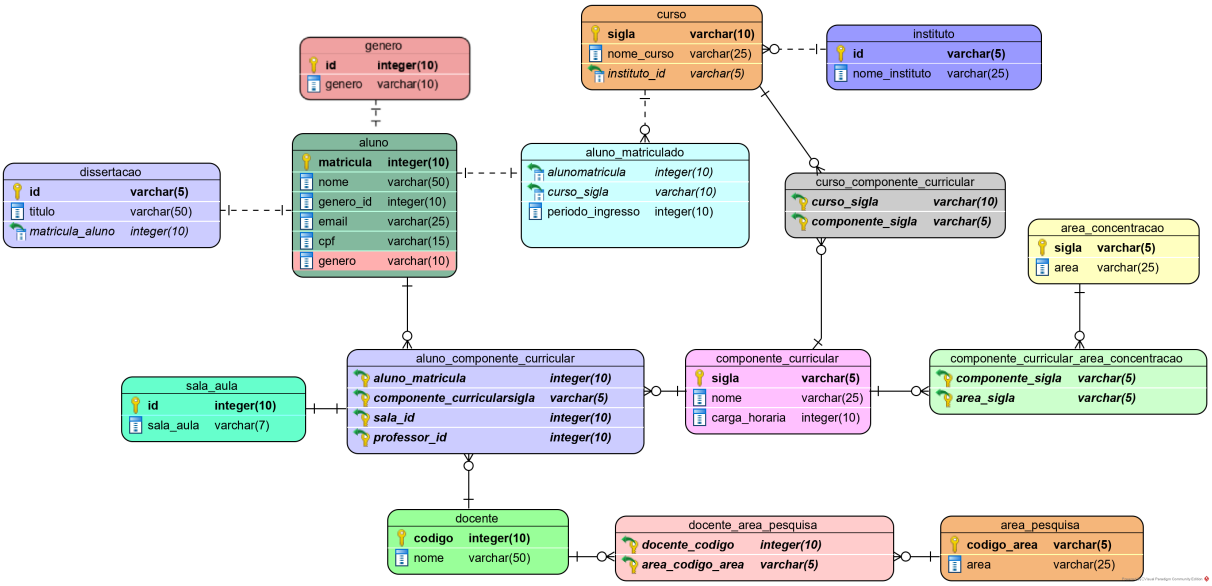


Figura 11 – Esquema Relacional de uma Universidade

4.3.1.1 Grau de Aninhamento de Relações

O algoritmo 1 recebe um conjunto Λ contendo todas as relações de um esquema relacional de banco de dados e analisa se, para cada relação λ pertencente a esse conjunto, esta pode vir a ser aninhada a outra relação ou relacionamento.

Uma relação será aninhada a outra relação ou relacionamento, se:

- fizer parte de apenas um relacionamento, mesmo que este referencie mais de uma relação (por exemplo, *joined table* com mais de duas chaves estrangeiras). A justificativa para ser apenas um relacionamento reside no fato de que, se for aninhada a duas ou mais relações, a manutenção de seus atributos pode ser onerosa, já que se encontram em mais de uma relação. Também não pode ser um auto-relacionamento, pois uma relação não pode ser aninhada a ela mesma;
- possuir atributos constantes (adaptado de Neumann (2017)), ou seja, haver uma pequena possibilidade de seus conteúdos sofrerem atualizações. São consideradas relações deste tipo aquelas que possuem apenas dois atributos, um sendo a chave primária (que não deve ser composta) e um dado qualquer.

Outro ponto a ser considerado é a questão das *joined tables* que estabelecem relacionamentos binários, mais especificamente relacionamentos do tipo $n : m$ do modelo conceitual. Considera-se que uma relação seja $n : m$ binária se ela conter duas chaves estrangeiras. Este conceito segue os exemplos contidos em Teorey et al. (2011) no processo de transformação do modelo conceitual para o lógico. É importante exemplificá-lo haja vista que na transformação entre os modelos, os conceitos do modelo conceitual se perdem. Dessa forma, para relações deste tipo, não é possível efetuar o aninhamento de uma relação à outra devido a impossibilidade de incluir os m atributos da relação para as n ocorrências da relação e vice-versa (NEUMANN, 2017).

Algoritmo 1: ANINHA RELAÇÃO

Entrada: $\Lambda = [\lambda_1, \lambda_2, ..., \lambda_n]$
Saída: Um conjunto Γ contendo informações das relações do esquema do banco de dados.

```
1  início
2      para cada λ ∈ Λ faça
3          ζ = len(fks(λ))
4          se (PK(λ) ≠ CompositeKey) & (len(attr(λ)) ≤ 2) &
            (len(refTbl(λ) = 1)) então
5              Φ(λ) ← 1
6          fim
7          senão
8              Φ(λ) ← 0
9          fim
10         Γ.add(λ, Φ(λ), cols(λ), ζ)
11     fim
12 fim
13 retorna Γ
```

Na execução do algoritmo são verificados os testes lógicos:

- **Quanto à relação λ:** Se esta não possui chave primária composta (primeira condição) e se possui no máximo dois atributos (segunda condição).
- **Quanto à(s) relação(ões) que referencia(m) a relação λ:** se existe **apenas uma** tabela que referencia a relação λ e se esta não é uma *joined table* com duas chaves estrangeiras (terceira condição).

O algoritmo retorna uma estrutura de dados Γ contendo os valores do Grau de Aninhamento Φ, o Nível de Relação ζ (tomada aqui como a quantidade de chaves estrangeiras da relação) e as colunas de cada relação λ de um esquema de banco de dados. A quantidade de chaves estrangeiras é considerada, pois posteriormente será importante para analisar relações com a quantidade de chaves estrangeiras superior a 2.

A Figura 12 ilustra a estrutura de dados gerada ao final da execução do algoritmo 1. Como resultado, tem-se uma lista Γ desta estrutura, onde cada índice corresponde à informação de uma relação do banco.

```
Γ(λ) = [{tabela: nome_tabela
         colunas: [col_1, ..., col_n]
         grau_relacao: quantidade_fks
         grau_integracao: valor_grau
        },
        {...}
       ]
```

Figura 12 – Estrutura de dados resultante da execução do algoritmo de aninhamento de relações.

A Figura 13 mostra uma parte da estrutura de dados gerada após a execução do método para o banco de dados Universidade (as informações de outras relações do esquema foram suprimidas).

```
Γ(universidade) = [{tabela: genero
                    colunas: [id, genero]
                    grau_relacao: 0
                    grau_integracao: 1
                   },
                   {tabela: aluno
                    colunas: [matricula, nome, genero_id, email, cpf]
                    grau_relacao: 1
                    grau_integracao: 0
                   }, {...}
                  ]
```

Figura 13 – Parte do resultado da execução do algoritmo de aninhamento de relações para o banco de dados Universidade.

4.3.1.2 Informação de Relacionamentos

O algoritmo 2 recebe uma lista Λ contendo as relações do esquema do banco de dados. Para cada relação λ é obtido o conjunto *fks* de suas chaves estrangeiras. Para cada chave estrangeira, é calculado o grau total de aninhamento do relacionamento, sendo de no mínimo 0 e no máximo 1.

O grau total de cada relacionamento (*degree_rel*) é utilizado para analisar posteriormente qual relacionamento contém relação λ que pode ser aninhada a outra. Além disso, o algoritmo retorna um mapeamento completo do relacionamento, contendo também o nome deste, o(s) atributo(s) que forma(m) a chave estrangeira, a relação λ que contém

Algoritmo 2: Informação de Relacionamentos

Entrada: $\Lambda = [\lambda_1, \lambda_2, ..., \lambda_n]$
Saída: Um conjunto Θ contendo as informações de cada relacionamento do esquema do banco de dados.

início
  para *cada* $\lambda \in \Lambda$ **faça**
    $fks = get\_fks(\lambda)$
    se *fks* **então**
      para *cada* $fk \in fks$ **faça**
        degree_rel = $\Phi(\lambda) + \Phi(refTbl(\lambda))$
      fim
      $\Theta.add(fk\_name, col\_fk, \lambda, degree\_rel, refTbl(\lambda), col(refTbl(\lambda))$
    fim
  fim
fim
retorna Θ

a chave estrangeira, o grau do relacionamento, a relação referenciada e seu(s) atributo(s) referenciado(s).

A Figura 14 ilustra a estrutura de dados gerada ao final da execução do algoritmo 2. Como resultado, tem-se uma lista Θ desta estrutura, onde cada índice corresponde à informação de um relacionamento do banco.

```
Θ(λ) = [{fk: nome_fk
         coluna_fk: nome_coluna_fk
         tabela: nome_tabela
         grau_relacionamento: valor_grau
         tabela_referenciada: nome_tabela
         coluna_referenciada: nome_coluna
        },
        {...}
       ]
```

Figura 14 – Estrutura de dados resultante da execução do algoritmo de informação de relacionamentos.

Na Figura 15 é possível observar parte da estrutura de dados resultante após a execução do método para o banco de dados Universidade (as informações de outras chaves estrangeiras do esquema foram suprimidas).

4.3.1.3 Nível de Relações

O nível ζ de uma relação não é calculado por um método em específico, mas ele merece atenção. O grau, está relacionado ao DER. Quando é feita a transformação do modelo conceitual para o lógico, este conceito se perde. Para facilitar o entendimento, considera-se o nível da relação como a quantidade de chaves-estrangeiras que uma relação

```
Θ(universidade) = [{fk: fk_aluno_componente_curricular
                    coluna_fk: aluno_matricula
                    tabela: aluno_componente_curricular
                    grau_relacionamento: 0
                    tabela_referenciada: aluno
                    coluna_referenciada: matricula
                   },
                   {...}
                  ]
```

Figura 15 – Parte da estrutura gerada pelo algoritmo de informação de relacionamentos para o banco de dados Universidade.

possui. Este valor é importante, pois, quando for feito a incorporação de uma relação à outra e após este processo ainda tal relação tiver seu nível ζ superior a 2, é necessário efetuar algumas análises que irão impactar o processo de conversão para o modelo orientado a grafos.

A importância dessa informação, no processo de mapeamento e conversão, reside no fato de que pesquisas que não efetuam o procedimento de aninhamento, como em Megid, El-Tazi e Fahmy (2018), criam no grafo resultante um nó representando a relação e tantas quantas arestas forem a quantidade de chaves estrangeiras existentes na relação original.

Por exemplo: considere que **D** seja uma *joined table* que referencia as relações **A**, **B** e **C**. No cenário descrito, **D** representa um nó no grafo e nele existirão 3 arestas incidentes, uma para cada nó que está relacionado a ele. Uma busca em que é preciso retornar uma consulta que parte de **B** e que precisa ter um valor de chave primária de **A** com código igual a *a_pk = 2591*, por exemplo, é necessário percorrer dois caminhos: um que parte de **B** e chega em **D**, e outro que parte de **A** e chega em **D**. Agora, considerando o método da presente pesquisa e no final da conversão a relação **A** tenha sido aninhada à relação **D**, teríamos, ao final do processo, um percurso no grafo ao executar a consulta. Para grande quantidade de registros armazenados no banco, isso reduz o tempo de percurso do grafo.

4.3.1.4 Aninhamento de Relações

É a fase do mapeamento que aninha ou não uma relação à outra. Todo o processo realizado para a escolha e aninhamento de uma relação à outra é mostrado no algoritmo 3. O método recebe como entrada a estrutura de dados Θ contendo informações sobre os relacionamentos (esquema da Figura 14) e outra estrutura η, representada na Figura 16, contendo os atributos e tuplas de cada relação λ, ambos existentes no banco de dados.

A Figura 17 exemplifica o resultado da execução do algoritmo sobre as relações **gênero** e **aluno**, do esquema representado anteriormente pela Figura 11.

Pode-se observar no algoritmo 3 que o ponto de partida é iterar sobre cada relaci-

```
η = [{tabela: nome_tabela
      info: [[col_1, col_2, ..., col_n],
             [tupla_1_col_1, tupla_1_col_2, ..., tupla_1_col_n],
             ...,
             [tupla_m_col_1, tupla_m_col_2, ..., tupla_m_col_n],
            ]
     },
     {...}
    ]
```

Figura 16 – Estrutura de dados contendo informações dos atributos e tuplas de cada relação do banco de dados.

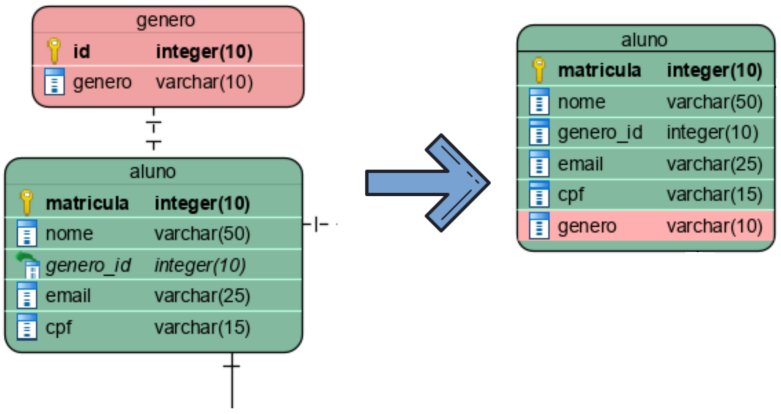


Figura 17 – Resultado do Aninhamento de duas Relações.

onamento *fk* existente no conjunto Θ e verificar se o grau do relacionamento é igual a 1 (linhas 2 e 3). Se a condição for verdadeira, as seguintes variáveis são criadas:

- Dois índices *idx_a* (linha 5) e *idx_b* (linha 6) que armazenam, respectivamente, os índices dos atributos que são FK e PK da tabela que receberá os dados(a) e da tabela que será aninhada (b).
- Dois índices *idx_tb* (linha 8) e *idx_ref_tb* (linha 9) que recebem, respectivamente, os índices do conjunto η onde se encontram os dados da tabela que receberá os dados e a que será aninhada.
- Duas estruturas de dados *dict_1* (linha 11) e *dict_2* (linha 12) contendo, respectivamente, os dados da tabela que receberá os dados da que referencia.

O aninhamento tem início com o laço da linha 14. O conjunto contendo o nome de cada atributo da relação λ está presente no índice 0 (consulte Figura 16) das estruturas criadas nas linhas 11 e 12. Sendo assim, são aninhados primeiro os nomes destes (linha 15). Após esse passo, é verificado se o valor da chave estrangeira não é *nulo* (linha 18). Caso não seja, a variável *search* (linha 19) armazena o valor atual do atributo que é chave estrangeira e este é buscado na estrutura (*dict_2*) que corresponde à tabela que

Algoritmo 3: Aninhamento de Relações

Entrada: O conjunto de Θ e o conjunto η de atributos e tuplas de cada λ.
Saída: O conjunto η com as devidas relações aninhadas.

```
1  início
2      para cada fk ∈ Θ faça
3          se dg_rel = 1 então
4
5              idx_a = col_fk(λ)
6              idx_b = col_pk(refTbl(λ))
7
8              idx_tb = η(λ)
9              idx_ref_tb = η(refTbl(λ))
10
11             dict_1 = η[idx_tb]
12             dict_2 = η[idx_ref_tb]
13
14             para cada (i, j) ∈ (idx_a, idx_b) faça
15                 dict_1[info][0].add(dict_2[info][0][j + 1])
16                 k = 1
17
18                 se dict_1[k][i] então
19                     search = dict_1[info][k][i]
20
21                     para cada f ∈ dict_2[info] faça
22                         se search ∈ f então
23                             dict_1[info][k].add(f[j + 1])
24                         fim
25                     fim
26                 fim
27                 senão
28                     dict_1[info][k].add(null)
29                 fim
30                 k = k + 1
31             fim
32             ζ(λ) = ζ(λ) − 1
33             η.update(λ(dict_1[info]))
34             η.remove(λ(idx_ref_tb))
35             FKS.remove(fk)
36         fim
37     fim
38 fim
39 retorna η
```

será aninhada. Para cada correspondência de *search* com o valor da chave primária desta estrutura, o dado que não é a chave primária é aninhado. Caso a verificação da linha 18 seja falsa, não existe um valor de chave estrangeira na relação e, portanto, não é inserido nenhum dado da relação de origem nessa tupla.

Terminado o aninhamento, o nível ζ é decrementado (linha 32), já que diminuiu em 1 a quantidade de relações referenciadas a ela. A estrutura η é atualizada com os novos valores da relação λ que recebeu os dados (linha 33) e removida a relação que foi aninhada (linha 34). Por fim, é removido do conjunto de chaves estrangeiras (linha 35) o relacionamento que era existente entre as tabelas envolvidas neste processo, já que uma delas deixou de existir.

4.3.2 Fase 2 - Conversão do Modelo Relacional

A **Fase de Conversão** é a segunda parte da execução da aplicação e se encarrega de converter a estrutura do modelo relacional que foi mapeada e aninhada na fase anterior para dois conjuntos: os de nós e os de relacionamentos entre estes. Estes conjuntos serão entradas dos métodos que realizam a migração para o GDB - em particular nesta pesquisa para o *Neo4j*.

4.3.2.1 Criação de Nós e Relacionamentos

Ao ser encerrada a execução do método de aninhamento de relações (algoritmo 3), é terminada a fase de mapeamento do banco. Com o algoritmo 4, a fase de conversão da estrutura do banco relacional para o modelo orientado a grafos é iniciada.

O algoritmo tem como entrada o conjunto Γ de informações de cada relação λ, o conjunto Θ de informações de chaves estrangeiras e o conjunto η de todas as informações de atributos e tuplas do banco de dados. A sua função é analisar para cada relação (linha 2) e seus relacionamentos com outra(s) relação(ões), como estes serão convertidos como nós e relacionamentos a serem migrados.

Um ponto importante, tratado neste método, é a criação de um atributo adicional chamado `node_id` que identifica unicamente o nó criado (além da chave estrangeira). A importância de sua criação reside principalmente no fato da criação de relacionamentos de entidades com nível maior que 2, pois, suponha que uma entidade tenha uma combinação de chaves estrangeiras **AX**, **BX** e **CX**. A relação de origem, que possui chave primária **AX**, ao ser estabelecido o relacionamento entre o nó que tem origem esta chave e um nó de destino com essa chave referenciando a origem, seria preciso criar uma estrutura adicional que tratasse esta condição em especial (além do valor da chave estrangeira em questão, a combinação de todas as outras chaves). É um procedimento bastante trabalhoso, aumentando ainda mais a complexidade do problema. A criação de um valor identificador para cada nó se mostrou mais simples e, ao final da migração, este é removido de cada nó persistido do banco. Outro ponto importante da utilização desse atributo está no estabelecimento de relacionamentos com relações que possuem apenas chaves estrangeiras e não possuem chave primária. Na execução do método desenvolvido por Megid, El-Tazi e Fahmy (2018), ao se deparar com este caso, o algoritmo simplesmente interrompe a sua execução deixando o banco de dados inconsistente. Este `node_id` soluciona este problema.

Para relações com nível ζ valendo 1 e 2, o processo de conversão segue parte do modelo desenvolvido por Orel, Zakošek e Baranovič (2016). Para relações com nível ζ superior a 2, é utilizado o conceito empregado por Neumann (2017). Por fim, relações com nível ζ igual a zero (que não possuem qualquer relacionamento com outra), são também consideradas.

A linha 2 inicia a execução do algoritmo. É feita uma análise de cada relação λ do conjunto Γ (linha 2). A primeira estrutura condicional analisa se a relação λ possui nível ζ igual a 1. Se a comparação for verdadeira, indica que a relação atual referência apenas uma tabela. A relação λ atual é chamada de **B** no algoritmo e a tabela referenciada é chamada de **A** para facilitar o entendimento. A Figura 18 exemplifica a ocorrência dessa condição. É verificado no conjunto μ de nós se não existem ocorrências de A e B (linhas 6-10). Caso não, estes são adicionados ao conjunto μ. É criado um relacionamento entre A e B e adicionado ao conjunto π (linha 12).

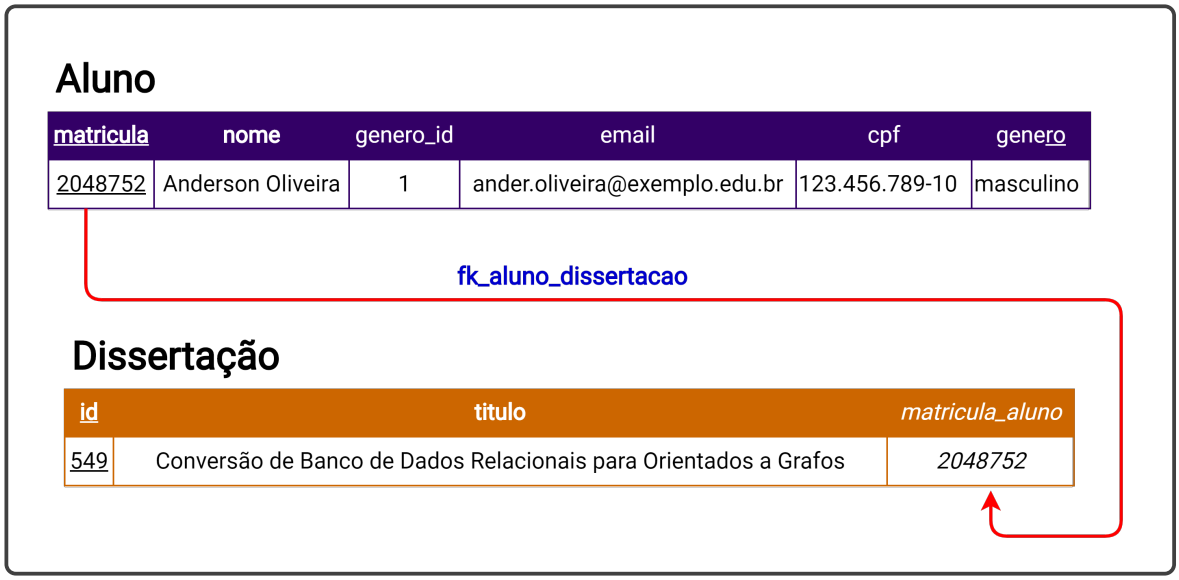


Figura 18 – Relação de nível 1.

O modelo de grafo de propriedades resultante com a execução deste passo pode ser visto na Figura 19.

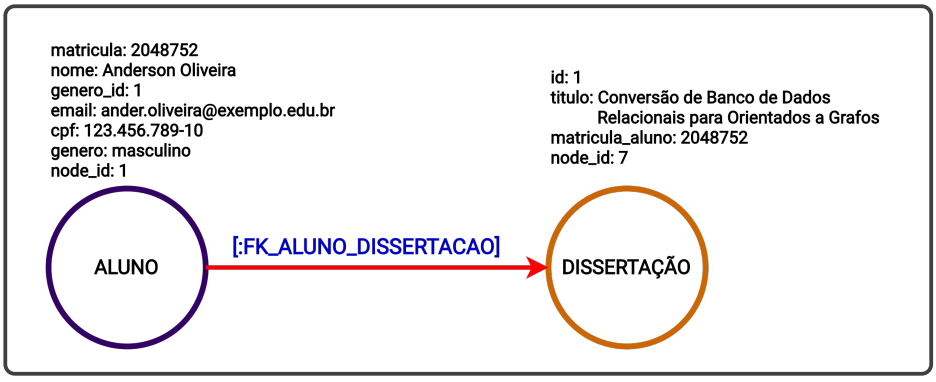


Figura 19 – Grafo gerado para relação de nível 1.

Para relações de nível 2 e que possuem **apenas** dois atributos (colunas), linhas 14-28, o processo de conversão é bastante simples, sendo criado apenas um relacionamento entre os nós. Ao final, a tabela λ se transforma em relacionamento entre as duas relações que referencia. O relacionamento gerado possui o mesmo nome da relação que o originou. A Figura 20 exemplifica a ocorrência de uma relação de nível 2.

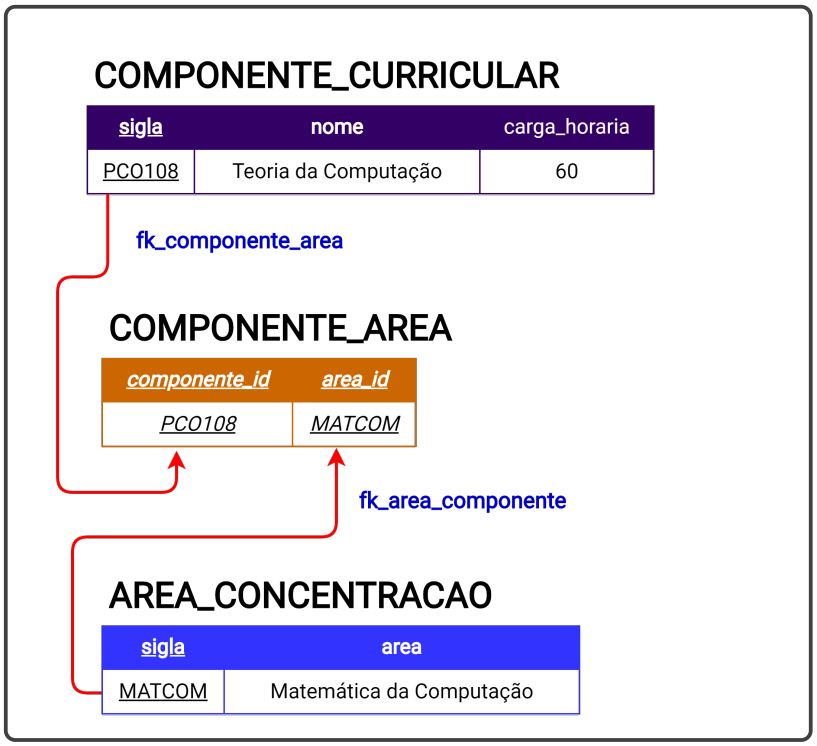


Figura 20 – Relação de nível 2 e suas relações referenciadas.

Tomando como exemplo, após a execução do método sobre a relação ilustrada na Figura 20, o modelo de grafo resultante pode ser observado na Figura 21. Percebe-se que a relação de nível 2 de origem nomeou o relacionamento existente entre os dois nós. Além disso, o relacionamento não possui qualquer propriedade já que a relação correspondente não possuía outros atributos a não ser a duas chaves estrangeiras.

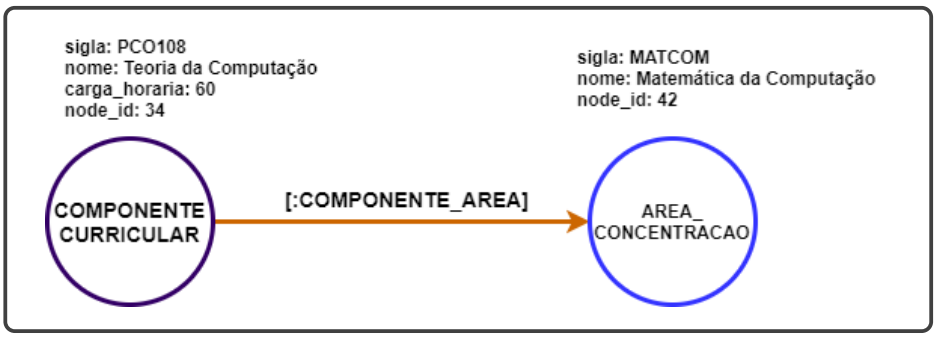


Figura 21 – Grafo gerado para relação de nível 2.

Relações com nível igual a 2 e com quantidade de atributos superior a 2 **ou** com nível superior a 2 são tratadas nas linhas 30-37. Este processo segue parte da ideia proposta por Neumann (2017), que é criar um nó para esta relação e, para cada relação a ela relacionada (por uma chave estrangeira), criar um nó para cada uma desta e estabelecer um relacionamento entre tais nós. A Figura 22 exemplifica uma relação de nível 4.

O grafo resultante da execução deste algoritmo sobre uma relação de nível superior a 2, pode ser visto na Figura 23. Esta relação serve como nó central, estabelecendo uma

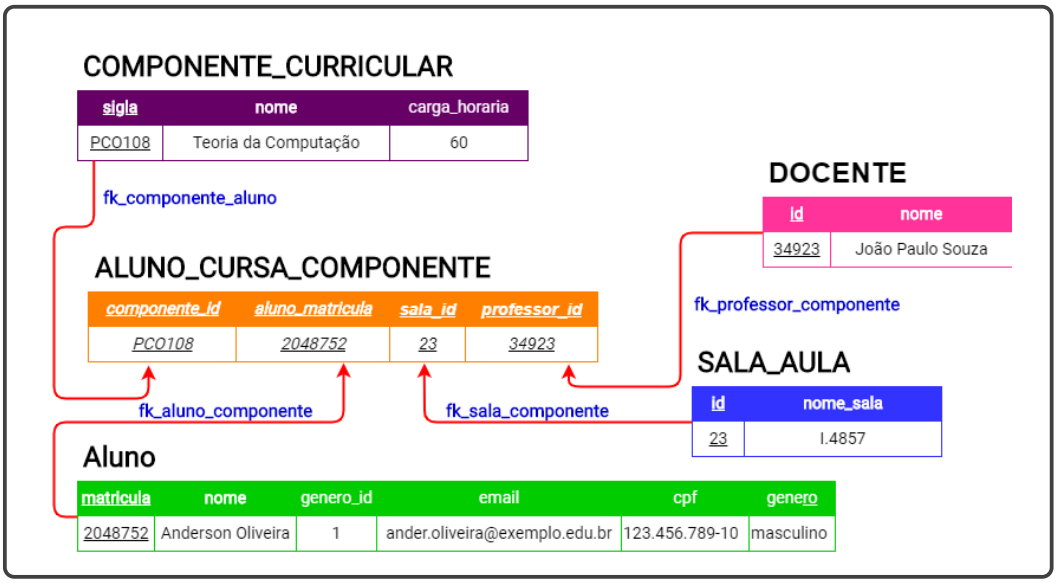


Figura 22 – Relação de nível 4 e suas relações referenciadas.

ligação entre todos os outros nós a ele relacionados.

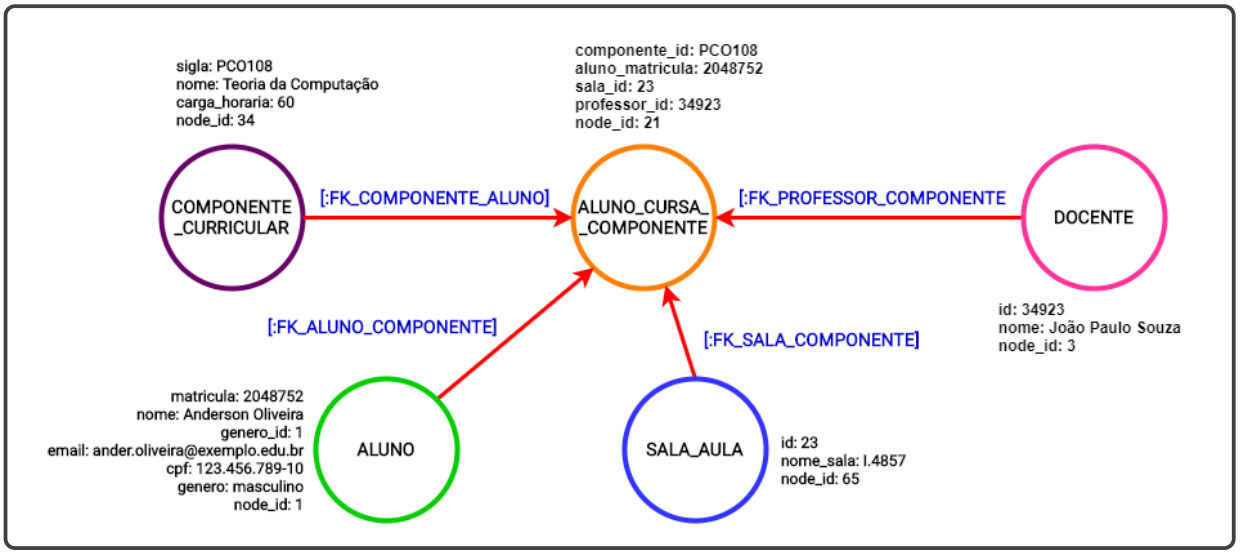


Figura 23 – Grafo gerado para relação de nível 2.

Por fim, são tratadas relações que não possuem relacionamento ou não são referenciadas por qualquer outra relação (linhas 39-41). Simplesmente é criado um nó isolado no modelo do grafo para esta.

Neste algoritmo, percebe-se que cada relação e cada relacionamento são processa-

dos em num menor de iterações, tornando processo de conversão mais rápido.

Algoritmo 4: Cria Nós e Relacionamentos

Entrada: Γ, Θ, η

Saída: Os conjuntos de Nós (μ) e Relacionamentos π a serem migrados.

```
1  início
2      para cada λ ∈ Γ faça
3          se ζ(λ) = 1 então
4              tb_b = λ
5              tb_a = Θ.find(refTbl(λ))
6              se tb_a ∉ μ então
7                  μ.add(η(tb_a, node_id))
8              fim
9              se tb_b ∉ μ então
10                 μ.add(η(tb_b, node_id))
11             fim
12             π.add(rel(tb_a ⟶ tb_b))
13         fim
14         senão
15             se ζ(λ) = 2 & len(attr(λ)) = 2 então
16                 tb_a = f(refTbl(λ_A))
17                 tb_b = f(refTbl(λ_B))
18
19                 /* Include A and B like nodes, as in lines 6-11 */
20
21                 π.add(rel(tb_a ⟶ tb_b))
22             fim
23         fim
24         senão
25             se ζ(λ) > 2 ‖ (ζ(λ) = 2 & len(attr(λ) > 2) então
26                 μ.add(η(λ), node_id)
27                 para cada tb ∈ refTbl(λ) faça
28                     μ.add(η(tb, node_id))
29                     π.add(rel(tb ⟶ λ))
30                 fim
31             fim
32         fim
33         senão
34             se ζ(λ) = 0 então
35                 μ.add(η(λ))
36             fim
37         fim
38     fim
39 fim
40 retorna μ, π
```

4.3.3 Fase 3 - Migração dos Dados

A última etapa e a mais simples é a de migração dos dados para o GDB, Neo4j. Como os dados já se encontram no modelo próprio de um grafo de propriedades, os únicos esforços são os de criar um nó no banco e depois, ao criar o relacionamento, buscar os nós e estabelecer a relação entre eles.

O algoritmo 5 mostra como os nós são criados no banco. A entrada do método é o conjunto de nós η criado na fase anterior e, seguindo o modelo de grafo de propriedades, cada um conterá um rótulo (*label*), representado por $\mathscr{L}$(linha 3) e um conjunto de propriedades *props* que representam pares chave-valor que serão propriedades do nó. Estes valores são adicionados na criação do nó (linha 5) à instância $\mathscr{G}$ do grafo no banco de dados.

Algoritmo 5: Migração de Nós

Entrada: $\eta = [n_1, n_2, ..., n_n]$
Saída: Todos os n nós do conjunto migrados para o banco de dados orientado a grafos.

```
1 início
2     para cada n ∈ η faça
3         L = n(name)
4         para cada props ∈ n(info) faça
5             G.add(L, props)
6         fim
7     fim
8 fim
```

Após todos os nós terem sido criados no banco, o algoritmo 6 estabelece os relacionamentos existentes entre eles.

Algoritmo 6: Migração de Relacionamentos

Entrada: $\Theta = [r_1, r_2, ..., r_n]$
Saída: Todos os r relacionamentos do conjunto migrados para o banco de dados orientado a grafos.

```
1 início
2     para cada r ∈ Θ faça
3         ϱ = r(name)
4         n_1 = G.match(r(origin), r(node_id))
5         n_2 = G.match(r(destiny), r(node_id))
6
7         se props ⊂ r então
8             G.add(rel(ϱ, n_1 ⟶ n_2), props)
9         fim
10        senão
11            G.add(rel(ϱ, n_1 ⟶ n_2))
12        fim
13    fim
14    para cada n ∈ nodes( G ) faça
15        n.remove(node_id)
16    fim
17 fim
```

A entrada do método é o conjunto Θ de relacionamentos entre os nós, criado na fase anterior. Para cada relacionamento r existente no conjunto (linha 2), é obtido o seu rótulo ϱ e feita a busca dos nós de origem (linha 4) e o de destino (linha 5) pelo `node_id`. Ao ser estabelecido o relacionamento entre os nós, é verificado se este possui propriedades (linha 7), sendo adicionadas ou não, caso existam. Ao final, o `node_id` é removido de cada nó existente no banco, já que esta propriedade foi utilizada somente para criar os relacionamentos. Terminada a execução do algoritmo 6, o processo por completo é concluído.

Este Capítulo mostrou com detalhes cada passo executado pela aplicação e com isso, percebe-se a sua completude ao tratar cada fase da execução para gerar um modelo de grafo que melhor correspondesse ao modelo relacional de origem, sem alterar a sua semântica e ocorrer perdas de dados durante os processos. O capítulo seguinte tratará dos experimentos que exploram os algoritmos tanto no tempo de execução de cada fase, quanto na análise dos modelos gerados e tempo de consultas efetuadas nos bancos de dados de origem (relacional) e o gerado (orientado a grafos).

5 Testes e Resultados

Nesse capítulo são apresentados o banco de dados utilizado na pesquisa, o planejamento e execução dos testes com a aplicação desenvolvida e os resultados destes. A escolha do conjunto de dados é pertinente à conjuntura atual, que passa por uma pandemia de coronavírus. Tal cenário traz relevância para o método desenvolvido, já que grande parte destes conjuntos são fornecidos em forma de tabelas que são armazenadas em banco de dados relacionais.

A pesquisa feita por Lysenko et al. (2016) mostra como a utilização de banco de dados orientados a grafos é adequada para a representação de informações biológicas (como por exemplo, no caso referente a doenças), que se encontram fortemente conectadas, muitas vezes semiestruturadas e imprevisíveis. Assim, o estudo demonstra que este tipo de banco de dados fornece uma solução flexível para o aninhamento de diversos tipos de dados biológicos e facilitam a mineração exploratória de dados para apoio à geração de hipóteses.

Existem diversas pesquisas recentes em periódicos da área de saúde como *BioData Mining* (BI et al., 2019; DENAXAS et al., 2017; PERDIGÃO; ROSA; O'DONOGHUE, 2017; TSUR, 2017; MOORE et al., 2016; MUÑOZ-TORRES et al., 2016; MURUGAN et al., 2015) e *Nature* (SCHöPF et al., 2020; KATAKA et al., 2020; AGUADO et al., 2020; ZENG et al., 2020; CHITNIS et al., 2019; LADUMOR et al., 2019; SINGH et al., 2019) que utilizam dados persistidos em banco de dados relacionais para visualização e análise de informações referentes à área de saúde e biologia. Para a maioria das organizações, atualmente, seus dados ainda se encontram armazenados em banco de dados relacionais e, efetuar a migração para um modelo orientado a grafos, embora este seja bem mais simples de se entender, demanda tempo e conhecimento do administrador do banco de dados (KUSZERA; PERES; FABRO, 2019). Este necessita ter um sólido conhecimento do esquema do banco de dados da organização para gerar um modelo de grafo que não traga perdas dos dados e da semântica das informações persistidas.

Dessa forma, o algoritmo desenvolvido nesta pesquisa, e que foi detalhado no capítulo anterior, busca minimizar o trabalho necessário da organização no processo de migração do banco de dados, efetuando de maneira automática, o processo de mapeamento e conversão do modelo relacional para o orientado a grafos.

Para efetuar a análise do comportamento da aplicação com base na estrutura do banco e na variação da quantidade de registros, o processo experimental envolveu as seguintes etapas:

1. Criação de duas bases de dados relacionais.
2. Execução da aplicação e registro dos tempos de cada fase e o tempo execução completa.
3. Realização das consultas nas bases relacionais e orientadas a grafos para comparação dos tempos de execução. Cada consulta foi realizada 11 vezes, excluindo o registro do tempo da primeira execução.
4. Análise dos resultados dos tempos (todos em milissegundos) registrados em cada etapa anterior.

Durante a realização dos experimentos, a aplicação desenvolvida por Megid, El-Tazi e Fahmy (2018) foi executada, mas ocorreram erros durante sua execução referentes, ora à criação dos nós ora com a criação dos relacionamentos. Sendo assim, a aplicação encerrava a execução gerando um modelo de grafo incompleto.

Com relação à etapa 3, relatada anteriormente, a realização das consultas vem para estudar o desempenho de cada um dos modelos envolvidos, já que a aplicação desenvolvida altera a estrutura do banco de dados de origem. Dessa forma, esta parte do experimento seguiu as seguintes etapas (semelhante aos passos executados por Neumann (2017) em seus experimentos):

1. Reinicialização do servidor do MySQL e Neo4j a cada execução de uma consulta.
2. Execução de um *script* que percorre todos os vértices e arestas de uma base no Neo4j retornando o total de vértices e arestas percorridos, a fim de melhorar o desempenho, conforme descrito na pesquisa referenciada.
3. Todas as consultas foram executadas descartando o primeiro resultado.

As consultas procuraram responder os seguintes questionamentos:

- O desempenho muda conforme é alterada a estrutura (relacional e de grafos) em que os dados estão armazenados?
- A resposta da consulta trazida pelo modelo de grafo gerado coincide com a retornada pela consulta no banco de dados relacional de origem?

Todos os procedimentos descritos neste Capítulo foram executados em uma máquina com a seguinte configuração:

- **Dispositivo:** Acer Predator Helios 300.

- **Sistema Operacional:** Windows 10 Education.
- **Processador:** Core i7-9750H, 2.60 GHz, Hexa-core (6 núcleos).
- **Memória:** 16GB de RAM.
- **Armazenamento:** SSD com leitura/escrita sequencial até 545 MB/s.

5.1 Coronavirus Dataset

O banco de dados utilizado nesta pesquisa consiste de um conjunto de dados disponibilizado na plataforma Kaggle[1], que trata de informações referentes à pandemia do COVID-19 (Coronavírus) na Coréia do Sul. Os dados encontram-se disponibilizados no formato CSV e organizados da seguinte maneira (cada arquivo corresponde a uma tabela):

- **Case:** dados dos casos de infecção por COVID-19 na Coréia do Sul. Possui os seguintes campos[2]:
  1. **case_id:** identificador do caso.
  2. **province:** Capital / Cidade Metropolitana / Província.
  3. **city:** Cidade / Distrito. Contém um valor "*from other city*"que especifica que a infecção teve origem em outra cidade.
  4. **group:** grupo de infecção. Valores como *contact with patient*, *overseas inflow* e *etc* não são considerados como grupo de infecção.
  5. **infecction_case:** nome do grupo de infecção ou de outros casos. O valor *overseas inflow* indica que a infecção ocorreu em outro país. O valor *etc* inclui casos individuais e casos sob investigação.
  6. **confirmed:** total acumulado de casos confirmados.
  7. **latitude:** latitude do grupo.
  8. **longitude:** longitude do grupo.
- **PatientInfo:** dados epidemiológicos dos pacientes infectados pelo COVID-19. Contém os seguintes campos:
  1. **patient_id:** identificador do paciente.
  2. **global_num:** número identificador global fornecido pela KCDC (*Korea Centers for Disease Control and Prevention*).

[1] https://www.kaggle.com/kimjihoo/coronavirusdataset
[2] https://www.kaggle.com/kimjihoo/ds4c-what-is-this-dataset-detailed-description

3. **sex:** gênero do paciente.
4. **birth_year:** ano de nascimento do paciente.
5. **age:** faixa de idade do paciente. Os dados estão divididos na seguinte escala:
   - 0s: 0 a 9 anos.
   - 10s: 10 a 19 anos.
   - ...
   - 90s: 90 a 99 anos.
   - 100s: 100 a 109 anos.
6. **country:** país de residência do paciente.
7. **province:** província do paciente.
8. **city:** cidade do paciente.
9. **disease:** indicador se o paciente possui (1) ou não (0) doença subjacente.
10. **infection_case:** caso de infecção do paciente.
11. **infection_order:** ordem de infecção do paciente.
12. **infected_by:** identificador do paciente que infectou.
13. **contact_number:** total de contatos do paciente.
14. **symptom_onset_date:** data de início dos sintomas.
15. **confirmed_date:** data de confirmação da doença.
16. **released_date:** data de alta.
17. **deceased_date:** data do falecimento.
18. **state:** situação do paciente. Dividido em três tipos:
    - *isolated:* o paciente está internado isolado no hospital.
    - *released:* o paciente foi liberado do isolamento.
    - *deceased:* o paciente veio a óbito.

- **PatientRoute:** informações de rotas de paciente. Contém os seguintes campos:

1. **patient_id:** identificador do paciente.
2. **global_num:** número identificador global fornecido pela KCDC.
3. **date:** data da rota (YYYY-MM-DD).
4. **province:** província em que a rota está localizada.
5. **city:** cidade em que o paciente passou.
6. **latitude:** latitude do local.
7. **longitude:** longitude do local.

- **Region:** localização e informações estatísticas de regiões da Coréia do Sul. Contém os seguintes campos:

  1. **code:** código da região.
  2. **province:** Capital / Cidade Metropolitana / Província.
  3. **city:** Cidade / Distrito.
  4. **latitude:** latitude do local.
  5. **longitude:** longitude do local.
  6. **elementary_school_count:** quantidade de escolas de ensino básico.
  7. **kindergarten_count:** quantidade de jardins de infância.
  8. **university_count:** total de universidades.
  9. **academy_ratio:** total de academias.
  10. **elderly_population_ratio:** proporção de população idosa.
  11. **elderly_alone_ratio:** proporção de idosos que vivem sozinhos.
  12. **nursing_home_count:** total de casas de repouso para idosos.

- **SeoulFloating:** dados da população flutuante em Seoul. Contém os seguintes campos:

  1. **date:** data do registro das informações.
  2. **hour:** hora do registro das informações.
  3. **birth_year:** faixa da idade da população registrada.
  4. **sex:** gênero da população.
  5. **province:** Capital / Cidade Metropolitana / Província.
  6. **city:** Cidade / Distrito.
  7. **fp_num:** total da população flutuante.

- **Weather:** informações climáticas nas regiões da Coréia do Sul. Contém os seguintes campos:

  1. **code:** código da região.
  2. **province:** Capital / Cidade Metropolitana / Província.
  3. **date:** data do registro das informações.
  4. **avg_temp:** temperatura média.
  5. **min_temp:** menor temperatura registrada.
  6. **max_temp:** maior temperatura registrada.

7. **precipitation:** precipitação diária.
8. **max_wind_speed:** velocidade máxima dos ventos.
9. **most_wind_direction:** direção mais frequente dos ventos.
10. **avg_relative_humidity:** umidade relativa média.

Os dados foram coletados em Maio de 2020 e persistidos no SGBD MySQL *Community Edition* na versão 8.0.19. A Figura 24 representa o esquema do banco de dados normalizado gerado a partir dos dados coletados na base referida.

A Tabela 8 mostra a quantidade de registros que cada relação do banco de dados possui após a persistência dos dados.

Relação	**Quantidade de Registros**
caso	112
caso_infeccao	43
cidade	158
coordenadas	2.891
estado	3
genero	2
paciente	3.388
paciente_infectado	817
paciente_info	3.388
pais	12
populacao_flutuante	432.000
provincia	18
regiao	244
rota_paciente	6.714
tempo	25.295
tipo_rota	24
Total de Registros	475.109

Tabela 8 – Quantidade de registros em cada relação do banco de dados.

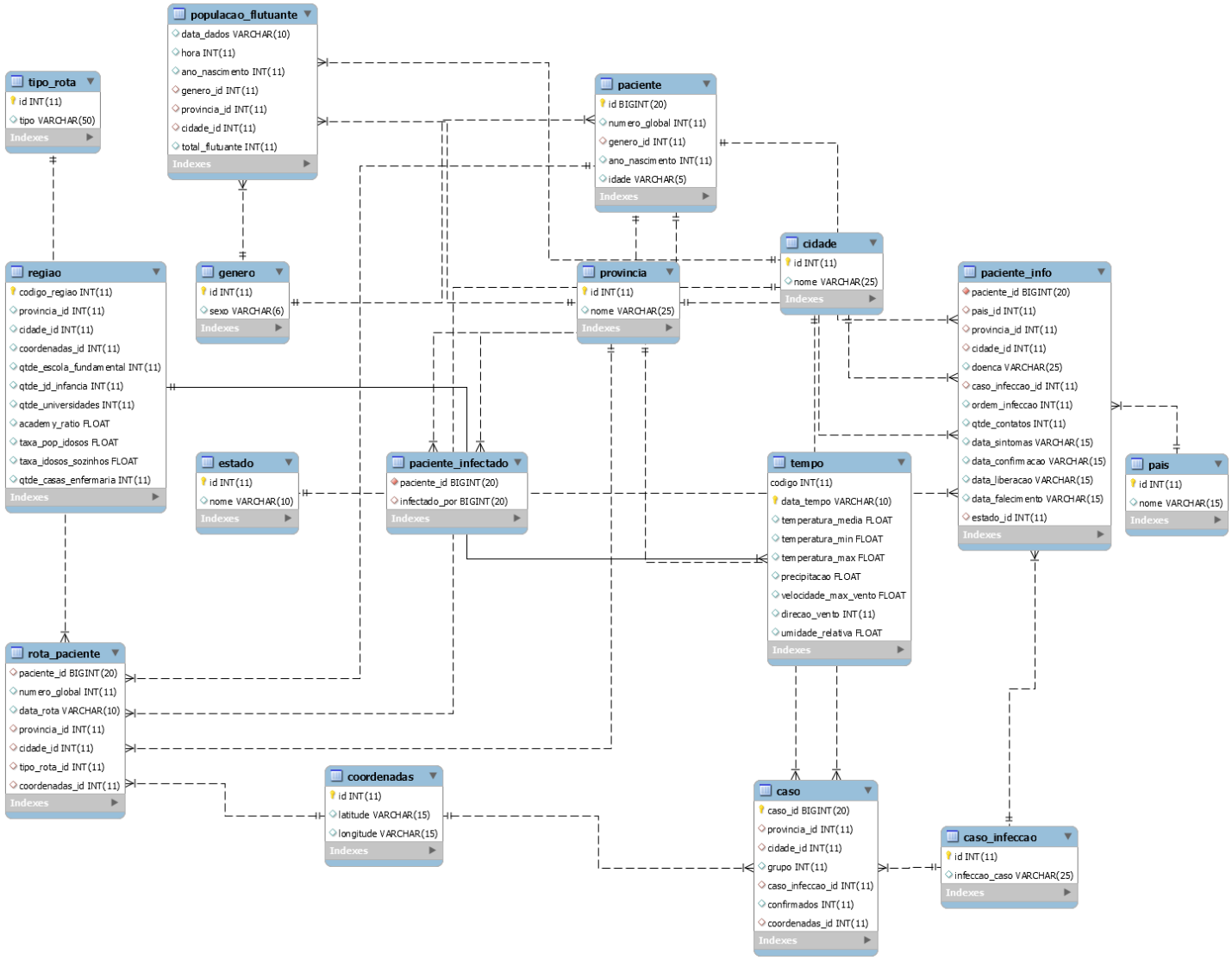


Figura 24 – Modelo físico do Banco de Dados *Coronavirus*.

5.1.1 Mapeamento, Conversão e Migração

Os tempos de execução de cada uma das etapas foram registrados, mas a sua comparação com os de outra aplicação foi impossibilitada em razão do erro descrito na introdução deste capítulo. Sendo assim, a análise presente nesta parte da pesquisa concentra-se no modelo de grafo gerado pela aplicação.

A primeira etapa do processo consiste no mapeamento da estrutura, com o cálculo do grau de aninhamento de cada relação presente no banco de dados e o aninhamento das relações, caso seja possível. A Tabela 9 contém o grau de aninhamento e a cardinalidade de cada relação.

Relação	**Grau de Aninhamento**	**Nível**
caso	0	4
caso_infeccao	0	0
cidade	0	0
coordenadas	0	0
estado	1	0
genero	0	0
paciente	0	1
paciente_infectado	0	2
paciente_info	0	6
pais	1	0
populacao_flutuante	0	3
provincia	0	0
regiao	0	3
rota_paciente	0	5
tempo	0	1
tipo_rota	1	0

Tabela 9 – Grau de Aninhamento e Cardinalidade de cada relação.

A fase seguinte, da etapa de mapeamento, é a de aninhamento das relações de grau 1. Pela Tabela 9 e análise do modelo lógico do banco de dados na figura 24 verificam-se os seguintes aninhamentos:

- A relação **estado** aninhada à relação **paciente_info**.
- A relação **pais** aninhada à relação **paciente_info**.
- A relação **tipo_rota** aninhada à relação **rota_paciente**.

O aninhamento altera a estrutura de dados das relações do banco de dados. Após o aninhamento das relações, é iniciada a etapa de conversão do modelo. Nesta fase, a análise da cardinalidade é efetuada para determinar como o modelo do grafo será estruturado. A Tabela 10 contém a cardinalidade de cada relação após a etapa de mapeamento.

A figura 25 ilustra o modelo de grafo gerado após a execução da etapa de conversão (este modelo foi retirado do esquema do banco de dados já migrado, no Neo4j). Observe

Relação	Cardinalidade
caso	4
caso_infeccao	0
cidade	0
coordenadas	0
genero	0
paciente	0
paciente_infectado	2
paciente_info	4
populacao_flutuante	3
provincia	0
regiao	3
rota_paciente	4
tempo	1

Tabela 10 – Cardinalidade de cada relação após o aninhamento.

que houve a redução de uma relação, onde a **paciente_infectado** se transformou um auto-relacionamento entre o nó **paciente**.

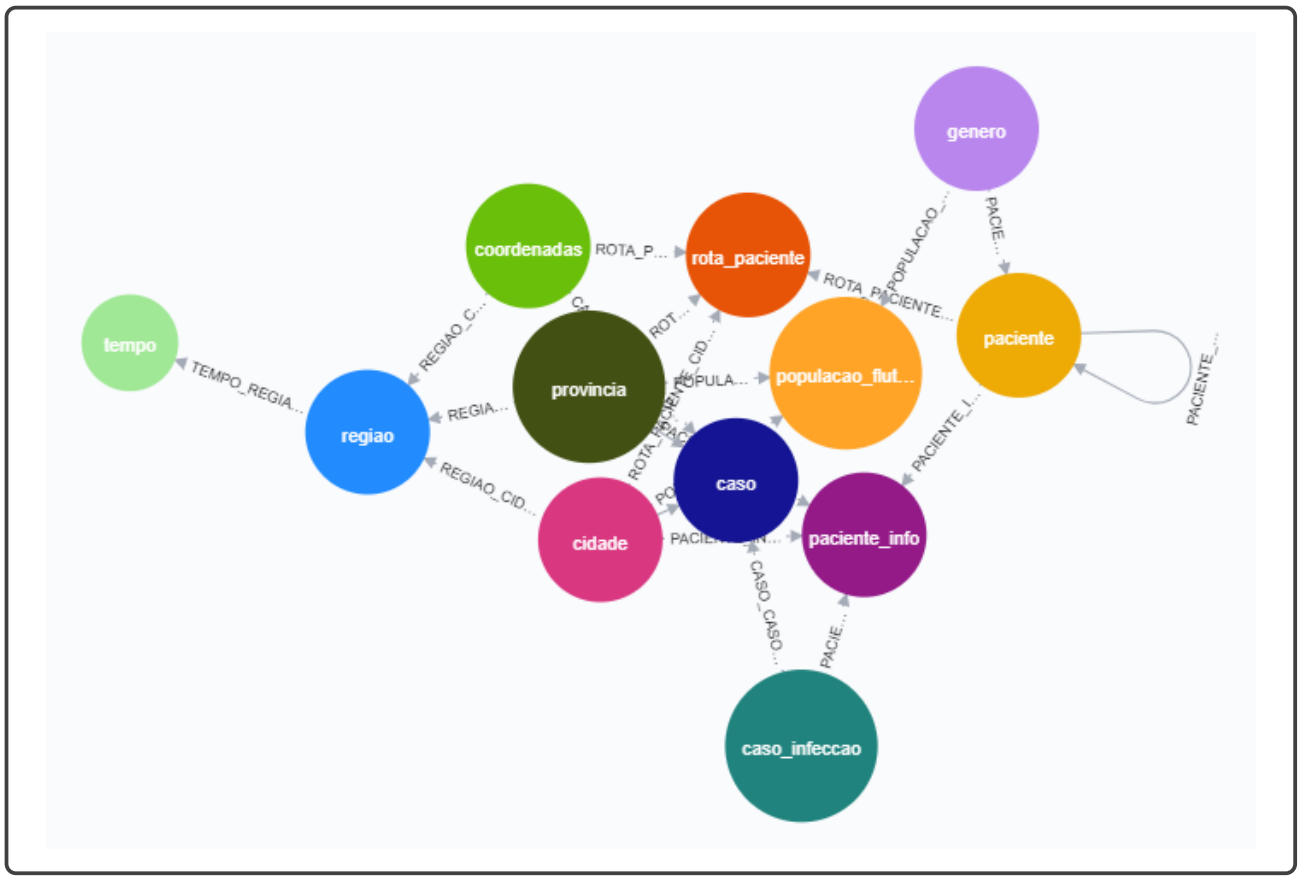


Figura 25 – Modelo de grafo gerado após a fase de conversão.

5.1.2 Realização de Consultas

Para esta base de dados, as consultas foram definidas baseando-se na lista de discussão da plataforma onde o conjunto de dados foi extraído. As questões definidas para

cada consulta (Apêndice A, seção A.1) pautaram-se nas tarefas existentes no repositório, voltado para a Ciência de Dados. Sendo assim, procurou-se adequar a análise com o que é requerido do tipo de base de dados em questão, podendo ser estendido para outras bases de mesma característica, voltada para a área de saúde.

Código	**Descrição**
Q1	Total de casos por forma de infecção
Q2	Distribuição do total de casos de infecção por mês
Q3	Total de casos confirmados por Gênero
Q4	Total de casos confirmados por Faixa de Idade
Q5	Cidades e Rotas de Pacientes Infectados (Total)
Q6	Distribuição do Total de Casos confirmados por Província e Cidade
Q7	Dados Demográficos da População Flutuante antes e 14 dias depois da primeira confirmação

Tabela 11 – Consultas realizadas no experimento

A Tabela 12 contém o tempo médio (ms), o desvio padrão (DP) e o intervalo de confiança (IC) da distribuição *t-student* para as execuções com 95% de confiança, da execução de cada uma das consultas realizadas no banco de dados de origem (relacional) e o gerado após o processo de migração para o Neo4j.

	Média		**DP**		**IC**	
	MySQL	**Neo4j**	**MySQL**	**Neo4j**	**MySQL**	**Neo4j**
Q1	31,7	3,9	6,05	2,33	4,33	1,66
Q2	32,4	28,6	4,92	14,09	3,52	10,08
Q3	28,10	14,40	1,91	3,34	1,37	2,39
Q4	26,70	9,00	3,68	0,94	2,63	0,67
Q5	50,20	19,50	4,42	2,68	3,16	1,92
Q6	27,10	17,20	2,18	15,56	1,56	11,13
Q7	278,00	33,40	19,79	2,22	14,16	1,59

Tabela 12 – Tempo médio, desvio padrão (DP) e intervalo de confiança (IC) das consultas do experimento

A análise dos tempos médios contidos na tabela anterior permite verificar que, o modelo orientado a grafos possui desempenho superior na maioria das consultas. Além disso é possível verificar que:

- O desvio padrão (DP) foi utilizado para entender se os tempos coletados estão próximos do tempo médio (se o desvio é baixo) ou se o tempo das consultas dispersa muito da média (caso seja alto).
- De modo geral, o modelo orientado a grafos ganhou em desempenho.
- Os resultados retornados pela consulta foram os mesmos em ambos os modelos, evidenciando que o processo de aninhamento das relações não interfere no resultado da consulta.

5.2 *Academic Colaboration and Citation Network Dataset*

O segundo conjunto de dados utilizados nesta pesquisa foi o *Citation Network Dataset*[3], que contém informações de 4.107.340 publicações e 36.624.464 citações relacionadas na versão 11 do conjunto. As informações que podem ser extraídas deste conjunto permitem buscar informações de autores, conferências e artigos e, além disso, informações baseadas em semântica (como perfis estruturados de pesquisadores) (TANG et al., 2008). Outra aplicação deste tipo de informação é nos chamados *Collaboration Graphs*, utilizados para modelar redes sociais onde duas ou mais pessoas estão relacionadas em uma pesquisa realizada em conjunto (ROSEN, 2019).

A fim de estudar o comportamento do algoritmo quanto ao tempo de migração à medida que a quantidade de registros aumenta, foram criadas 4 bases de dados sobre o conjunto. Dessa forma, utilizando amostragem aleatória simples, foram criadas amostras contendo um total de 1.000 (**Base_A**), 10.000 (**Base_B**), 50.000 (**Base_C**) e 100.000 (**Base_D**) registros. A escolha aleatória destes ocorreu por uma aplicação que efetuava a leitura do arquivo de registros linha por linha e, em cada uma desta, tinha 50% de chance de ser incluída na amostra. A leitura encerrava quando cada amostra continha a quantidade de registros especificados anteriormente.

Após a geração das amostras, estas foram tratadas e os dados persistidos no SGBD *MySQLCommunity Edition* na versão 8.0.19. O esquema normalizado resultante deste processo pode ser visto na Figura 26.

A Tabela 13 contém a quantidade de registros por Relação (Tabela) e a quantidade total presente em cada base de dados.

Tabela 13 – Quantidade de Registros por Base de Dados.

	Base de Dados			
Relação	*Base_A*	*Base_B*	*Base_C*	*Base_D*
author	1.989	19.844	89.987	167.776
author_organization	1.336	13.273	71.541	169.655
author_paper	2.010	20.952	108.816	229.902
document_type	2	2	3	3
field_study	2.593	10.926	23.416	31.341
organization	702	6.180	29.271	68.621
paper	749	7.660	39.219	82.784
paper_field_study	5.881	60.243	318.858	730.620
paper_info	749	7.660	39.219	82.784
paper_publication_raw	749	7.660	39.219	82.784
paper_references	2	84	2.373	14.889
publisher	88	344	854	1.467
raw_publisher	453	1.509	2.711	3.363
Total de Registros	**17.303**	**156.337**	**765.487**	**1.665.989**

[3] https://www.aminer.org/citation

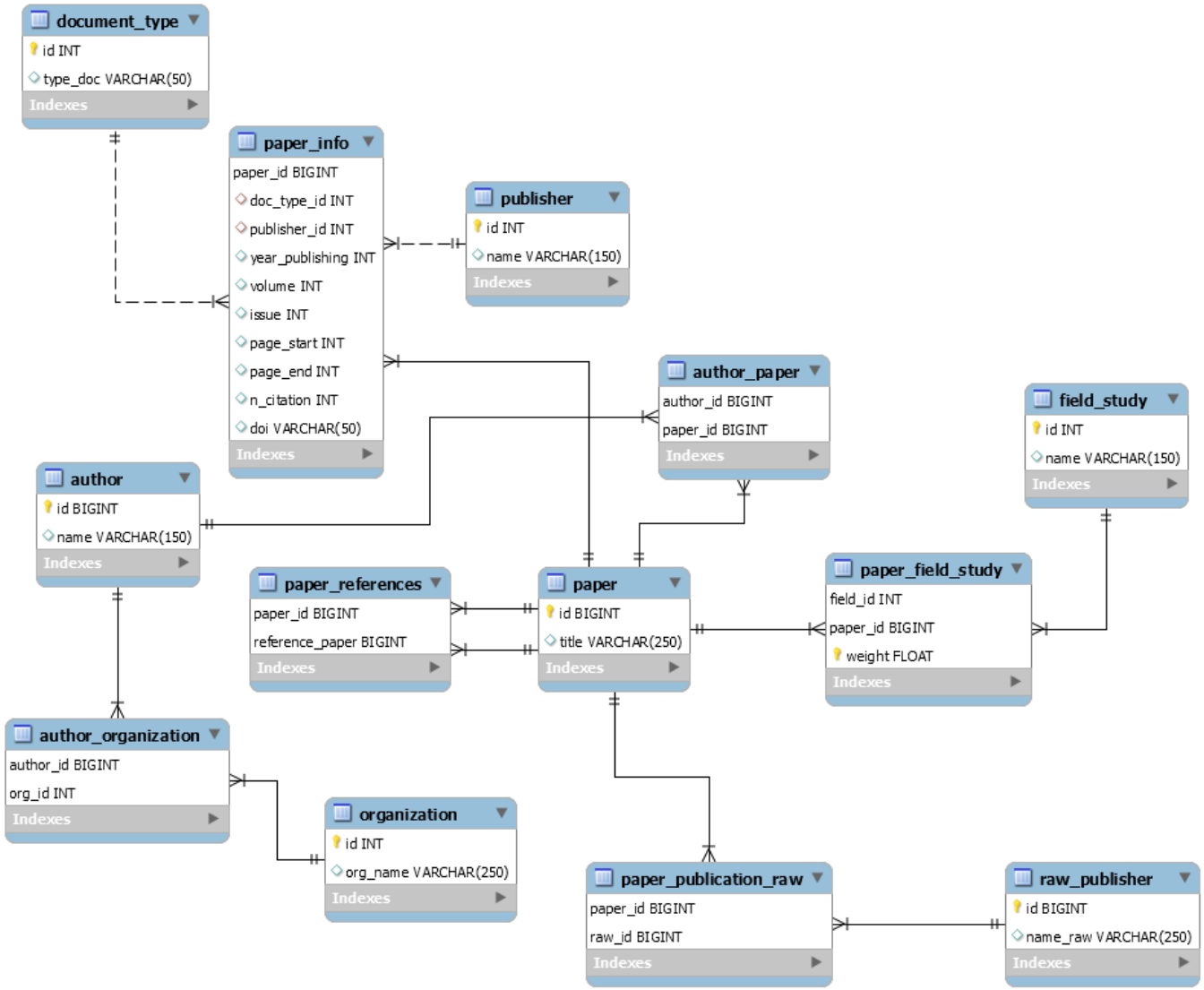


Figura 26 – Modelo Físico do Banco de Dados *Academic Colaboration and Citation Network*.

5.2.1 Mapeamento, Migração e Conversão

Para esta base de dados, foi possível observar melhor o tempo médio de cada etapa de execução da aplicação, já que foram feitos testes com 4 bases do mesmo conjunto de dados, mas com quantidade de registros variável. Dessa forma, a análise desta seção contemplará a comparação do tempo médio *versus* quantidade de registros no banco de dados e avaliação do modelo de grafo gerado.

A execução da etapa de mapeamento da estrutura, realizando o cálculo do grau de aninhamento de cada relação presente no banco de dados e o aninhamento das relações é executado da mesma forma como feito com a base de dados *coronavirus*. A tabela 10 contém o grau de aninhamento e a cardinalidade de cada relação.

Relação	**Grau de Aninhamento**	**Cardinalidade**
author	0	0
author_organization	0	2
author_paper	0	2
document_type	1	0
field_study	0	0
organization	0	0
paper	0	0
paper_field_study	0	2
paper_info	0	2
paper_publication_raw	0	2
paper_references	0	2
publisher	0	0
raw_publisher	0	0

Tabela 14 – Grau de Aninhamento e Cardinalidade de cada relação.

A fase seguinte da etapa de mapeamento é a de aninhamento das relações de grau 1. Pela Tabela 9 e análise do modelo físico do banco de dados na Figura 24 verifica-se o seguinte aninhamento:

- A relação **document_type** aninhada à relação **paper_info**.

A Figura 27 traz o modelo de grafo gerado após a execução da etapa de conversão (este modelo foi retirado do esquema do banco de dados já migrado, no Neo4j). Pode ser visto que o modelo final gerado se tornou bem mais simples que o de origem (Figura 26). A razão disto é a maior predominância de relações com nível igual a 2, em que uma relação intermediária entre duas relações, torna-se um relacionamento (aresta) no modelo de grafo gerado.

A Tabela 15 registra os tempos médios, em milissegundos (ms), de cada uma das etapas que são executas.

Com a análise dos resultados mostrados pela tabela anterior pode-se notar que:

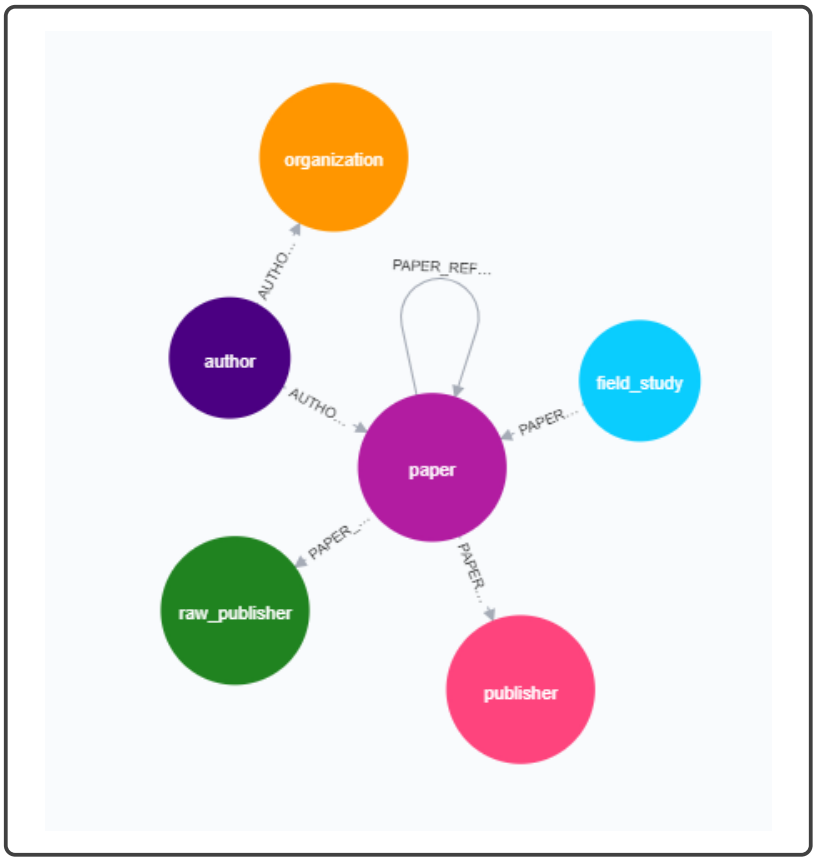


Figura 27 – Relação de grau 4 e suas relações referenciadas.

	Tempo (em segundos) para cada Base de Dados			
Etapas do Processo	*Base_A*	*Base_B*	*Base_C*	*Base_D*
I. Cálculo do Grau de Aninhamento	0,06	0,09	0,10	0,11
II. Realização do Aninhamento	0,16	1,34	6,66	14,54
Fase 1 - Mapeamento da Estrutura	**0,22**	**1,43**	**6,76**	**14,65**
Fase 2 - Conversão do Modelo	**0,06**	**0,75**	**3,84**	**9,43**
I. Criação dos Nós	25,55	162,11	633,61	1.338,08
II. Criação dos Relacionamentos	59,31	1.257,12	16.242,89	53.593,57
Fase 3 - Migração	**84,86**	**1.419,23**	**1.687,50**	**54.931,65**
Tempo total de execução	**85,14**	**1.421,41**	**1.698,10**	**54.955,73**
Total de registros da origem	**17.303**	**156.332**	**765.487**	**1.665.989**

Tabela 15 – Tempo Médio de Execução da Aplicação.

- A etapa do **Cálculo Grau de Aninhamento** possui pouca variação, já que a análise é feita apenas sobre a estrutura das relações, não tendo influência direta da quantidade de registros presentes no banco.
- A etapa de **Realização do Aninhamento** apresenta crescimento expressivo do tempo médio, por ser influenciada pela quantidade de registros presente na estrutura de dados das relações. A estrutura é percorrida por completa, sendo analisado em cada registro se precisa ser aninhado a outro ou receber o aninhamento de outra relação.

- Na fase de **Mapeamento da Estrutura** pode-se observar que o tempo total gasto (com a conjunção dos tempos das etapas anteriores) é influenciado em grande parte pela complexidade das operações residentes na etapa anterior, já que a primeira etapa desta fase possui pouca influência.
- A fase de **Conversão do Modelo** possui crescimento de tempo semelhante à etapa de aninhamento, já que o seu comportamento se assemelha ao método anterior citado no que diz respeito à sua complexidade.
- A etapa de **Criação dos Nós** possui crescimento de tempo linear, como pode ser visto no gráfico da Figura 26 (primeiro gráfico). O método tem complexidade linear, já que percorre a estrutura completamente e, para cada registro, é criado um nó no Neo4j.
- A etapa de **Criação dos Relacionamentos** comportamento exponencial de n^2, como pode ser visto no gráfico da Figura 26 (segundo gráfico). O método possui essa complexidade pois, além de percorrer a estrutura completa de relacionamentos, é preciso ser feita uma busca dos nós no Neo4j para ser criado o relacionamento.

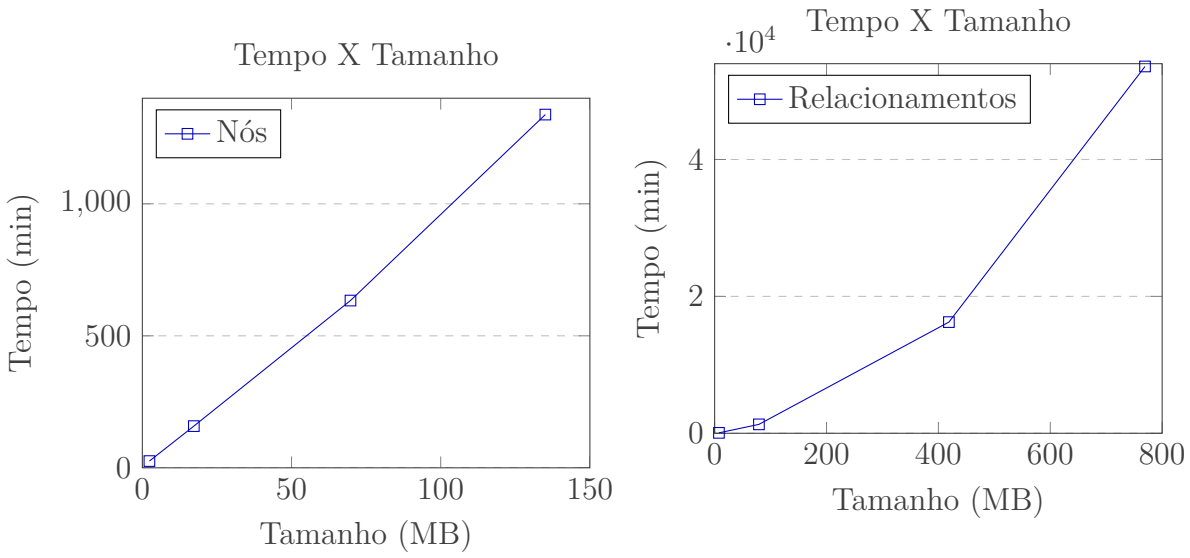


Figura 28 – Crescimento do tempo de migração dos nós e relacionamentos para o Neo4j em razão do tamanho da estrutura de dados.

5.2.2 Realização de Consultas

As consultas definidas para execução dos testes (Apêndice A, seção A.2) dessa seção foram criadas imaginando um cenário onde existe uma rede social que guarda informações de autores, editoras e suas publicações científicas - bastante semelhante com

as bases de publicações científicas atuais. Cada uma das 9 consultas executadas nas bases de dados pode ser verificada na Tabela 16.

Código	**Descrição**
Q1	Editoras que mais publicam
Q2	Tipos de documentos mais publicados por uma editora
Q3	Autores com mais publicações
Q4	Áreas de estudo com mais publicações
Q5	Áreas de estudo mais pesquisadas pelos 10 autores com mais publicações
Q6	Áreas de estudos mais frequentes das 10 editoras com mais publicações
Q7	Organizações com mais artigos publicados
Q8	Áreas de pesquisa das 10 organizações com mais artigos publicados
Q9	Artigos que mais foram citados

Tabela 16 – Consultas realizadas no experimento.

As Tabelas 17, 18, 19 e 20 registram os tempos das execuções das consultas nas bases de dados *Base_A*, *Base_B*, *Base_C* e *Base_D*, respectivamente. Elas contêm os mesmos dados coletados para a base de dados da seção anterior e o processo também foi executado da mesma forma.

	Base_A					
	Média		**DP**		**IC**	
	MySQL	**Neo4j**	**MySQL**	**Neo4j**	**MySQL**	**Neo4j**
Q1	33,10	2,10	6,45	0,74	4,62	0,53
Q2	25,90	1,30	2,38	0,48	1,70	0,35
Q3	26,50	4,30	1,84	1,16	1,32	0,83
Q4	29,70	6,80	2,00	1,03	1,43	0,74
Q5	73,40	27,20	7,81	6,73	5,58	4,81
Q6	32,10	11,30	3,67	2,98	2,62	2,13
Q7	24,90	4,20	2,51	0,79	1,80	0,56
Q8	48,40	26,50	4,52	2,84	3,23	2,03
Q9	21,30	1,60	2,75	0,70	1,97	0,50

Tabela 17 – Tempo médio, desvio padrão (DP) e intervalo de confiança (IC) das consultas do experimento para a base *ref_1mil*.

Ao realizar a análise das tabelas, pode-se perceber que o modelo orientado a grafos apresentou um desempenho superior ao relacional no que se refere ao tempo médio das consultas. Este fato evidencia a aplicabilidade melhor deste modelo em cenários que representam uma rede social. A quantidade de junções presentes nas consultas dessa base foi maior do que a da primeira, demonstrando que, para este tipo de consulta, um banco de dados relacional perde em desempenho. E isto fica evidente quando o banco de dados aumenta em quantidade de registros.

	Base_B					
	Média		DP		IC	
	MySQL	Neo4j	MySQL	Neo4j	MySQL	Neo4j
Q1	34,50	6,60	7,62	3,06	5,45	2,19
Q2	32,20	2,40	3,58	0,84	2,56	0,60
Q3	74,80	21,40	5,37	2,27	3,84	1,62
Q4	80,10	35,10	8,65	3,07	6,19	2,20
Q5	459,10	296,00	12,02	15,46	8,60	11,06
Q6	111,10	75,90	7,29	7,00	5,22	5,01
Q7	78,20	23,30	5,92	4,72	4,24	3,37
Q8	512,60	283,20	9,29	30,31	6,64	21,68
Q9	22,40	10,80	2,72	2,44	1,94	1,75

Tabela 18 – Tempo médio, desvio padrão (DP) e intervalo de confiança (IC) das consultas do experimento para a base *ref_10mil*.

	Base_C					
	Média		DP		IC	
	MySQL	Neo4j	MySQL	Neo4j	MySQL	Neo4j
Q1	46,30	14,90	8,38	2,56	6,00	1,83
Q2	48,90	7,30	4,23	1,42	3,02	1,01
Q3	563,90	123,70	18,80	6,60	13,45	4,72
Q4	734,80	213,90	56,65	26,56	40,52	19,00
Q5	4708,50	1845,00	79,23	66,01	56,68	47,22
Q6	3255,00	427,80	39,99	13,75	28,61	9,84
Q7	10242,40	139,30	138,85	5,38	99,33	3,85
Q8	72853,30	2392,00	3313,79	130,24	2370,54	93,17
Q9	26,00	39,00	3,09	3,33	2,21	2,38

Tabela 19 – Tempo médio, desvio padrão (DP) e intervalo de confiança (IC) das consultas do experimento para a base *ref_50mil*.

	Base_D					
	Média		DP		IC	
	MySQL	Neo4j	MySQL	Neo4j	MySQL	Neo4j
Q1	68,50	22,00	8,22	7,48	5,88	5,35
Q2	414,30	27,50	83,76	6,20	59,92	4,44
Q3	1379,50	361,70	632,12	58,36	452,19	41,75
Q4	1385,10	274,90	207,08	29,96	148,14	21,43
Q5	36441,60	2528,10	1372,93	118,21	982,14	84,56
Q6	12863,90	752,40	106,75	58,78	76,37	42,05
Q7	83800,60	382,10	2754,88	58,55	1970,72	42,10
Q8	212633,20	3009,50	2048,24	123,95	1465,22	88,67
Q9	677,00	152,20	477,18	22,21	341,35	15,88

Tabela 20 – Tempo médio, desvio padrão (DP) e intervalo de confiança (IC) das consultas do experimento para a base *ref_100mil*.

6 Conclusões

O desenvolvimento de uma aplicação que trata de diversos processos antes de migrar um banco de dados relacional para um orientado a grafos permitiu mostrar que este processo vai além de transformar relações e relacionamentos em vértices e arestas de um grafo. Isso ocorre em razão da diferença semântica de como os dados de um banco de dados relacional e de um orientado a grafos estão armazenados em disco. Com o intuito de analisar se os algoritmos desenvolvidos no decorrer desta pesquisa realizavam o trabalho designado, foram elaboradas consultas que permitiram analisar a variação do tempo médio em diversos casos. Além disso, o resultado das consultas permitiu ver que a aplicação não causou perda de dados no processo e nem modificação semântica do banco de dados de origem, ou seja, existência de redundâncias, dados presentes em nós que não estão relacionados, por exemplo.

Os testes aplicados nas bases permitiram observar que o banco de dados originado pelo processo apresenta um desempenho de consulta melhor do que o de origem. Aninhar uma relação à outra diminui o tempo de percurso do grafo, trazendo melhor desempenho para uma consulta.

Houve também dificuldades e problemas durante o desenvolvimento da aplicação. O tratamento do tipo de dado de cada atributo de uma relação do banco de dados relacional precisou ser feito minuciosamente, em razão da semântica da linguagem *Cypher*. Além disso, o trabalho com dados do tipo BLOB (*Binary Large Object*) se mostrou impraticável, já que o trabalho com este tipo de dado em bancos de dados orientados a grafos representa um anti-padrão do próprio modelo. No Neo4j, existe o tipo *byte* que lida com tipos BLOB, mas, caso uma propriedade tenha um tamanho de dado enorme, esta fica desordenada no arquivo de armazenamento e, consequentemente, prejudica o tempo de leitura dos dados. Um possível tratamento seria o armazenamento dos atributos deste tipo em forma de URL (*Uniform Resource Locator*) que aponta para os dados binários. Outro problema encontrado, foi lidar com as estruturas de dados geradas durante a execução da aplicação em memória primária, a fim de ganhar em desempenho. Em máquinas com memória limitada, a execução da aplicação pode ser prejudicada, afetando a qualidade do modelo final migrado.

6.1 Respostas às Questões da Pesquisa

1. **Como realizar a migração dos dados persistidos em um banco de dados relacional para orientado a grafos?** Existem diversas abordagens no processo

de mapeamento e migração dos dados entre os modelos citados. Embora existam diferenças entre as técnicas descritas, grande parte destas seguem três passos denominados ETL (*Extract*, *Transformation* e *Load*) que consiste em: extrair os dados de um banco de dados de origem, transformar os dados e migrar os dados para o banco de dados de destino. A pesquisa buscou integrar os dois processos encontrados na literatura: o mapeamento e a conversão entre os modelos.

2. **Como elementos pertinentes ao projeto de banco de dados influenciam o desenvolvimento de algoritmos que realizam conversão para um modelo orientado a grafos?** Foi visto que a modelagem conceitual ajudou no processo de conversão entre os modelos pois facilita a identificação das cardinalidades e de como os nós e relacionamentos serão definidos. Essa ideia foi adaptada para o modelo físico do banco de dados a fim de estrutura a fim de estruturar a etapa de conversão dos modelos.

3. **Quais são os problemas, dificuldades, desafios, perspectivas futuras para algoritmos de conversão automática de um banco de dados relacional para modelos orientados a grafos?** Pôde-se verificar que o maior problema no processo de conversão entre os modelos é manter a mesma semântica do banco migrado para o de destino, em Virgilio, Maccioni e Torlone (2013) foi verificado por (MEGID; EL-TAZI; FAHMY, 2018) que ao realizar a migração, ocorria perda de dados no processo. A aplicação *ThusterDB* conseguiu manter a semântica do modelo resultante além de criar modelo mais simples de grafo em algumas ocasiões. Problemas existentes estão relacionados à incompatibilidade de tipos de dados utilizados por SGBDs relacionais e o Neo4j.

4. **Como os algoritmos e aplicações desenvolvidas foram testados?** Os testes realizados em pesquisa desenvolvidas nas literaturas consultadas consistem na utilização de bases de dados com grande volume, sendo os mesmos utilizados em Virgilio, Maccioni e Torlone (2013), Megid, El-Tazi e Fahmy (2018), **??**) apresentando resultados diferentes para cada pesquisa.

6.2 Contribuições da Pesquisa

Esta pesquisa traz contribuições importantes para a atualização do estado-da-arte na comunidade científica. A principal contribuição reside no processo de mapeamento e conversão entre os modelos. Como foi tratado no decorrer desta dissertação, o método desenvolvido possui como base alguns pontos positivos de pesquisas que foram realizadas na área e procura preencher as suas lacunas. Um exemplo é o aninhamento de relações, que, ao fim da execução da aplicação, traz como resultado um modelo de grafo condizente semanticamente com o modelo relacional de origem e mais simples de se entender em

comparação com outro grafo gerado por um método mais recente (MEGID; EL-TAZI; FAHMY, 2018). Além disso, todos os processos envolvidos, desde o mapeamento até a migração, não dependem da forma que o modelo de origem se encontra estruturado, trabalhando da mesma maneira para qualquer cenário. Com relação à migração, o grafo gerado não apresenta perda de dados, estando consistente com o modelo de origem (foram realizadas comparações entre as tabelas resultantes das consultas no MySQL e no Neo4j).

6.3 Trabalhos Futuros

A aplicação desenvolvida apresenta inovações importantes para a área de banco de dados. Sugere-se, como possíveis trabalhos futuros, testes em ambientes corporativos, com banco de dados distribuídos a fim de analisar se a complexidade do tempo se mantém neste cenário, também com a seleção de relações que se quer migrar e não todas existentes no banco de dados relacional. Além disso, sugere-se verificar se a utilização de *threads* e processos concorrentes reduzem o tempo de criação dos relacionamentos no Neo4j. Com o intuito de lidar com o problema da limitação do uso da memória primária das variáveis que armazenam as estruturas de dados tanto do banco de dados relacional, quanto a que será migrada, sugere-se o teste da leitura e escrita de arquivos armazenados em disco, com o teste de desempenho neste caso.

Apêndices

APÊNDICE A – Consultas

Este apêndice contém as consultas dos testes realizados no Capítulo 5. Estas são mostras em SQL e sua equivalência em *Cypher*, que foi executada no Neo4j com o banco de dados resultante ao final da migração.

A.1 Coronavirus Database

A.1.1 Q1: Total de casos por forma de infeccção

```
SELECT infeccao_caso,
    SUM(confirmados) AS Total
FROM caso
JOIN caso_infeccao ci
    ON caso.caso_infeccao_id = ci.id
GROUP BY infeccao_caso
ORDER BY Total DESC
```

Listagem A.1 – Q1 em SQL

```
MATCH(a:caso_infeccao)-[r]->(b:caso)
RETURN a.infeccao_caso, SUM(b.confirmados) AS total
ORDER BY total DESC
```

Listagem A.2 – Q1 em *Cypher*

A.1.2 Q2: Distribuição do total de casos de infecção por mês

```
SELECT ci.infeccao_caso,
   MONTH(str_to_date(data_confirmacao, '%Y-%m-%d')) AS Mes,
   COUNT(data_confirmacao) as Total
FROM paciente_info pi
JOIN caso_infeccao ci
    ON pi.caso_infeccao_id = ci.id
WHERE pi.data_confirmacao IS NOT NULL
GROUP BY mes, infeccao_caso
ORDER BY mes, infeccao_caso
```

Listagem A.3 – Q2 em SQL

```
MATCH (b:caso_infeccao)-[r]->(a:paciente_info)
WHERE a.data_confirmacao IS NOT NULL
```

```
WITH b,r, apoc.date.format(apoc.date.parse(a.data_confirmacao, 'ms', 'yyyy-
MM-dd'), 'ms', 'MM') AS mes
RETURN mes,b.infeccao_caso, COUNT(mes) AS total
ORDER BY mes, total
```

Listagem A.4 – Q2 em *Cypher*

A.1.3 Q3: Total de casos confirmados por Gênero

```
SELECT g.sexo AS Genero,
   COUNT(pi.data_confirmacao) As Total
FROM paciente_info pi
JOIN paciente p
    ON pi.paciente_id = p.id
JOIN genero g
    ON p.genero_id = g.id
GROUP BY Genero
```

Listagem A.5 – Q3 em SQL

```
MATCH(a:genero)-[r]->(b:paciente)-[s]->(c:paciente_info)
WHERE c.data_confirmacao IS NOT NULL
RETURN a.sexo AS genero, COUNT(c.data_confirmacao) AS total
```

Listagem A.6 – Q3 em *Cypher*

A.1.4 Q4: Total de casos confirmados por Faixa de Idade

```
SELECT p.idade AS Faixa_Idade,
   COUNT(pi.data_confirmacao) AS Total
FROM paciente_info pi
JOIN paciente p
    ON pi.paciente_id = p.id
WHERE pi.data_confirmacao IS NOT NULL
GROUP BY Faixa_Idade
ORDER BY Total DESC
```

Listagem A.7 – Q4 em SQL

```
MATCH(a:paciente)-[r]->(b:paciente_info)
RETURN a.idade AS Faixa_Idade, COUNT(b.data_confirmacao)
```

Listagem A.8 – Q4 em *Cypher*

A.1.5 Q5: Cidades e Rotas de Pacientes Infectados (Total)

```
SELECT tr.tipo AS Tipo_Rota,
   c.nome AS Cidade,
   COUNT(pi.data_confirmacao) AS Total
FROM paciente_info pi
JOIN rota_paciente rp
    ON pi.paciente_id = rp.paciente_id
JOIN cidade c
    ON pi.cidade_id = c.id
JOIN tipo_rota tr
    ON rp.tipo_rota_id = tr.id
WHERE pi.data_confirmacao IS NOT NULL
GROUP BY Tipo_Rota, Cidade
```

Listagem A.9 – Q5 em SQL

```
MATCH(a:paciente_info)-[r]-(b:paciente)-[s]-(c:rota_paciente)-[t]-(d:cidade)
RETURN c.tipo, d.nome, COUNT(a.data_confirmacao)
ORDER BY c.tipo
```

Listagem A.10 – Q5 em *Cypher*

A.1.6 Q6: Distribuição do Total de Casos confirmados por Província e Cidade

```
SELECT p.nome AS Provincia,
   c.nome AS Cidade,
   COUNT(pi.data_confirmacao) as Total
FROM paciente_info pi
JOIN provincia p
    ON pi.provincia_id = p.id
JOIN cidade c
    ON pi.cidade_id = c.id
WHERE pi.data_confirmacao IS NOT NULL
GROUP BY Provincia, Cidade
```

Listagem A.11 – Q6 em SQL

```
MATCH(c:cidade)-[r]-(a:paciente_info)-[s]-(b:provincia)
WHERE a.data_confirmacao IS NOT NULL
RETURN b.nome AS Provincia, c.nome AS Cidade, COUNT(a.data_confirmacao) AS
total
```

Listagem A.12 – Q6 em *Cypher*

A.1.7 Q7: Dados Demográficos da População Flutuante antes e 14 dias depois da primeira confirmação

```sql
SELECT str_to_date(pf.data_dados, '%Y-%m-%d') AS data,
   pf.ano_nascimento, pf.hora,
   c.nome AS Cidade,
   g.sexo AS Genero,
   total_flutuante AS total
FROM populacao_flutuante pf
JOIN cidade c
    ON pf.cidade_id = c.id
JOIN genero g
    ON pf.genero_id = g.id
WHERE str_to_date(pf.data_dados, '%Y-%m-%d')
    BETWEEN '2020-01-17'
    AND '2020-01-31'
    AND pf.hora = 12
GROUP BY data, pf.ano_nascimento, Cidade, Genero
```

Listagem A.13 – Q7 em SQL

```
MATCH(c:genero)-[r]-(a:populacao_flutuante)-[s]-(b:cidade)
with c,r,a,s,b,
    apoc.date.format(apoc.date.parse(a.data_dados, 'ms', 'yyyy-MM-dd'),
        'ms', 'yyyy-MM-dd') AS data
WHERE data = '2020-01-17' OR data = '2020-01-31'
RETURN data, a.ano_nascimento, a.hora AS Hora,
    b.nome AS Cidade, c.sexo AS Genero, a.total_flutuante AS total
```

Listagem A.14 – Q7 em *Cypher*

A.2 Academic Colaboration and Citation Network Dataset

A.2.1 Q1: Editoras que mais publicam

```
SELECT p.name,
    COUNT(*) AS Total
FROM paper_info
JOIN publisher p
    ON paper_info.publisher_id = p.id
GROUP BY p.name
ORDER BY Total DESC LIMIT 10
```

Listagem A.15 – Q1 em SQL

```
MATCH (a:publisher)<-[r:PAPER_INFO_PUBLISHER_ID_FK]-(b:paper)
RETURN a.name, COUNT(r) AS Total
ORDER BY Total DESC LIMIT 10
```

Listagem A.16 – Q1 em *Cypher*

A.2.2 Q2: Tipos de publicações mais publicados por uma editora

```
SELECT dt.type_doc,
    COUNT(pi.paper_id)
FROM document_type dt
JOIN paper_info pi
    ON dt.id = pi.doc_type_id
JOIN publisher p
    ON pi.publisher_id = p.id
WHERE p.name = 'Springer, Berlin, Heidelberg'
GROUP BY dt.type_doc
```

Listagem A.17 – Q2 em SQL

```
MATCH (a:paper)-[r:PAPER_INFO_PUBLISHER_ID_FK]->(b:publisher)
WHERE b.name = "Springer, Berlin, Heidelberg"
RETURN r.type_doc, COUNT(r)
```

Listagem A.18 – Q2 em *Cypher*

A.2.3 Q3: Autores com mais publicações

```
SELECT name,
   COUNT(ap.paper_id) AS total
FROM author
JOIN author_paper ap
    ON author.id = ap.author_id
GROUP BY name
ORDER BY total DESC
LIMIT 10
```

Listagem A.19 – Q3 em SQL

```
MATCH (a:author)-[r]->(b:paper)
RETURN a.name, COUNT(r) AS total
ORDER BY total DESC LIMIT 10
```

Listagem A.20 – Q3 em *Cypher*

A.2.4 Q4: Áreas de estudo com mais publicações

```
SELECT name,
   COUNT(pfs.paper_id) AS total
FROM field_study
JOIN paper_field_study pfs
    ON field_study.id = pfs.field_id
GROUP BY name
ORDER BY total DESC
LIMIT 10
```

Listagem A.21 – Q4 em SQL

```
MATCH (a:field_study)-[r]->(b:paper)
RETURN a.name, COUNT(r) AS total
ORDER BY total DESC LIMIT 10
```

Listagem A.22 – Q4 em *Cypher*

A.2.5 Q5: Áreas de estudo pesquisadas pelos 10 autores com mais publicações

```
SELECT a.name,
   COUNT(ap.paper_id) AS total,
   GROUP_CONCAT(DISTINCT fs.name)
FROM author a
JOIN author_paper ap
    ON a.id = ap.author_id
JOIN paper p
    ON ap.paper_id = p.id
JOIN paper_field_study pfs
    ON p.id = pfs.paper_id
JOIN field_study fs
    ON pfs.field_id = fs.id
GROUP BY name
ORDER BY total DESC
LIMIT 10
```

Listagem A.23 – Q5 em SQL

```
MATCH (a:author)-[r]->(b:paper)<-[s]-(c:field_study)
RETURN a.name, COUNT(r) AS total, COLLECT(DISTINCT c.name)
ORDER BY total DESC LIMIT 10
```

Listagem A.24 – Q4 em *Cypher*

A.2.6 Q6: Áreas de estudos das 10 editoras com mais publicações

```
SELECT p.name,
    COUNT(*) AS Total,
    GROUP_CONCAT(DISTINCT fs.name)
FROM paper_info
JOIN publisher p
    ON paper_info.publisher_id = p.id
JOIN paper p2
    ON paper_info.paper_id = p2.id
JOIN paper_field_study pfs
    ON p2.id = pfs.paper_id
JOIN field_study fs
    ON pfs.field_id = fs.id
GROUP BY p.name
ORDER BY Total DESC LIMIT 10
```

Listagem A.25 – Q6 em SQL

```
MATCH (a:publisher)<-[r:PAPER_INFO_PUBLISHER_ID_FK]-(b:paper)<-[s]-(c:
field_study)
RETURN DISTINCT a.name, count(r) AS total, COLLECT(DISTINCT c.name)
ORDER BY total DESC LIMIT 10
```

Listagem A.26 – Q6 em *Cypher*

A.2.7 Q7: Organizações com mais artigos publicados

```
SELECT org_name AS organization,
   COUNT(ap.paper_id) AS total
FROM organization
JOIN author_organization ao
    ON organization.id = ao.org_id
JOIN author a
    ON ao.author_id = a.id
JOIN author_paper ap
    ON a.id = ap.author_id
GROUP BY organization
ORDER BY total DESC
LIMIT 10
```

Listagem A.27 – Q7 em SQL

```
MATCH(a:organization)<-[r]-(b:author)-[s]->(c:paper)
RETURN a.org_name, COUNT(s) AS total
ORDER BY total DESC LIMIT 10
```

Listagem A.28 – Q7 em *Cypher*

A.2.8 Q8: Áreas de pesquisa das 10 organizações com mais artigos publicados

```
SELECT org_name AS organization,
   COUNT(ap.paper_id) AS total,
   GROUP_CONCAT(DISTINCT fs.name)
FROM organization
JOIN author_organization ao
    ON organization.id = ao.org_id
JOIN author a
    ON ao.author_id = a.id
JOIN author_paper ap
    ON a.id = ap.author_id
JOIN paper p
    ON ap.paper_id = p.id
JOIN paper_field_study pfs
    ON p.id = pfs.paper_id
JOIN field_study fs
    ON pfs.field_id = fs.id
GROUP BY organization
ORDER BY total DESC
LIMIT 10
```

Listagem A.29 – Q8 em SQL

```
MATCH(a:organization)<-[r]-(b:author)-[s]->(c:paper)<-[t]-(d:field_study)
RETURN a.org_name, COUNT(s) AS total, COLLECT(DISTINCT d.name)
ORDER BY total DESC LIMIT 10
```

Listagem A.30 – Q8 em *Cypher*

A.2.9 Q9: Artigos que mais foram citados

```
SELECT title, COUNT(pr.reference_paper) AS total
FROM paper
JOIN paper_references pr
    ON paper.id = pr.paper_id
GROUP BY title
ORDER BY total DESC
LIMIT 10
```

Listagem A.31 – Q9 em SQL

```
MATCH(a:paper)-[r]->(b:paper)
RETURN a.title, COUNT(r) AS total
ORDER BY total DESC LIMIT 10
```

Listagem A.32 – Q9 em *Cypher*

Referências

AGUADO, A. et al. MorbiNet: multimorbidity networks in adult general population. analysis of type 2 diabetes mellitus comorbidity. *Scientific Reports*, Springer Science and Business Media LLC, v. 10, n. 1, fev. 2020. Disponível em: <https://doi.org/10.1038/s41598-020-59336-1>.

ALAMI, A. el; BAHAJ, M. Migration of a relational databases to nosql: The way forward. In: *Proceedings of the 5th International Conference on Multimedia Computing and Systems (ICMCS)*. [S.l.: s.n.], 2016. p. 18–23.

ANGLES, R. The property graph database model. In: OLTEANU, D.; POBLETE, B. (Ed.). *Proceedings of the 12th Alberto Mendelzon International Workshop on Foundations of Data Management, Cali, Colombia, May 21-25, 2018*. [S.l.]: CEUR, 2018. (CEUR Workshop Proceedings, v. 2100).

BESTA, M. et al. Demystifying graph databases: Analysis and taxonomy of data organization, system designs, and graph queries. *arXiv:1910.09017v4*, 2019.

BI, J.-H. et al. ClickGene: an open cloud-based platform for big pan-cancer data genome-wide association study, visualization and exploration. *BioData Mining*, Springer Science and Business Media LLC, v. 12, n. 1, jun. 2019. Disponível em: <https://doi.org/10.1186/s13040-019-0202-3>.

BONDY, A.; MURTY, M. R. *Graph Theory*. first. USA: Springer-Verlag London, 2008. v. 1. (Graduate Texts in Mathematics, v. 1).

BREWER, E. CAP twelve years later: How the "rules"have changed. *Computer*, Institute of Electrical and Electronics Engineers (IEEE), v. 45, n. 2, p. 23–29, fev. 2012.

BROWN, A.; WILSON, G. *The Architecture Of Open Source Applications*. 1st. ed. Morrisvile, North Carolina, USA: lulu.com, 2012. v. 2. ISBN 978-1105571817.

CHEN, P. P.-S. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems*, Association for Computing Machinery (ACM), v. 1, n. 1, p. 9–36, mar. 1976.

CHITNIS, T. et al. Quantifying neurologic disease using biosensor measurements in-clinic and in free-living settings in multiple sclerosis. *npj Digital Medicine*, Springer Science and Business Media LLC, v. 2, n. 1, dez. 2019. Disponível em: <https://doi.org/10.1038/s41746-019-0197-7>.

CLAUDINO, M.; SOUZA, D.; SALGADO, A. C. Mapeamentos conceituais entre os modelos relacional e nosql: Uma abordagem comparativa. *Revista Principia - Divulgação Científica e Tecnológica do IFPB*, v. 1, p. 37, 12 2015.

CODD, E. F. A relational model of data for large shared data banks. *Commun. ACM*, ACM, New York, NY, USA, v. 13, n. 6, p. 377–387, jun. 1970. ISSN 0001-0782.

CODD, E. F. Relational database: A practical foundation for productivity. *Commun. ACM*, ACM, v. 25, n. 2, p. 109–117, fev. 1982. ISSN 0001-0782.

CODD, E. F. *The Relational Model for Database Management: Version 2*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1990. ISBN 0-201-14192-2.

CONNOLLY, T. M.; BEGG, C. E. *Database Systems: A Practical Approach to Design, Implementation and Management (4th Edition)*. Harlow, England: Pearson Addison Wesley, 2004. ISBN 0321210255.

DATE, C. J. *Introdução a Sistemas de Banco de Dados*. 8th. ed. Rio de Janeiro: Elsevier, 2003. ISBN 9788535212730.

DAVOUDIAN, A.; CHEN, L.; LIU, M. A survey on NoSQL stores. *ACM Computing Surveys*, Association for Computing Machinery (ACM), v. 51, n. 2, p. 1–43, jun. 2018.

DB-ENGINES. 2020. Disponível em: <https://db-engines.com/>.

DENAXAS, S. et al. Methods for enhancing the reproducibility of biomedical research findings using electronic health records. *BioData Mining*, Springer Science and Business Media LLC, v. 10, n. 1, set. 2017. Disponível em: <https://doi.org/10.1186/s13040-017-0151-7>.

ELMASRI, R.; NAVATHE, S. *Sistemas de Banco de Dados*. 6. ed. SP, Brasil: Pearson Education do Brasil Ltda, 2011. ISBN 978-85-7936-08-5.

ERL, T.; KHATTAK, W.; BUHLER, P. *Big Data Fundamentals: Concepts, Drivers & Techniques*. 1st. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2016. ISBN 0134291077, 9780134291079.

FOSTER, E. C.; GODBOLE, S. *Database Systems: A Pragmatic Approach*. 1st. ed. Berkely, CA, USA: Apress, 2014. ISBN 1484208781, 9781484208786.

FOWLER, A. *NoSQL For Dummies*. 1st. ed. Hoboken, New Jersey: John Wiley and Sons, Inc., 2015. ISBN 1118905741.

FRANCIS, N. et al. Cypher. In: *Proceedings of the 2018 International Conference on Management of Data - SIGMOD '18*. [S.l.]: ACM Press, 2018.

GARCIA-MOLINA, H.; ULLMAN, J. D.; WIDOM, J. *Database Systems: The Complete Book*. 2. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2008. ISBN 9780131873254.

HAERDER, T.; REUTER, A. Principles of transaction-oriented database recovery. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 15, n. 4, p. 287–317, dez. 1983. ISSN 0360-0300.

HANINE, M.; BOUTKHOUM, O. Data migration methodology from relational to nosql databases. *International Journal of Information, Control and Computer Sciences*, v. 12, 2015.

HEUSER, C. A. *Projeto de Banco de Dados*. 6. ed. Porto Alegre, Brasil: Bookman, 2009. (Série Livros Didáticos Informática - UFRGS). ISBN 978-85-7780-382-8.

JORDAN, G. *Practical Neo4j*. 1st. ed. USA: Apress, 2014. ISBN 1484200233.

KATAKA, E. et al. Edgetic perturbation signatures represent known and novel cancer biomarkers. *Scientific Reports*, Springer Science and Business Media LLC, v. 10, n. 1, mar. 2020. Disponível em: <https://doi.org/10.1038/s41598-020-61422-3>.

KUSZERA, E. M.; PERES, L. M.; FABRO, M. D. D. Toward rdb to nosql: Transforming data with metamorfose framework. In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. New York, NY, USA: ACM, 2019. (SAC '19), p. 456–463. ISBN 978-1-4503-5933-7. Disponível em: <http://doi.acm.org/10.1145/3297280.3299734>.

LADUMOR, M. K. et al. A repository of protein abundance data of drug metabolizing enzymes and transporters for applications in physiologically based pharmacokinetic (PBPK) modelling and simulation. *Scientific Reports*, Springer Science and Business Media LLC, v. 9, n. 1, jul. 2019. Disponível em: <https://doi.org/10.1038/s41598-019-45778-9>.

LAKE, P.; CROWTHER, P. *Concise Guide to Databases: A Practical Introduction*. [S.l.]: Springer London, 2013. (Undergraduate Topics in Computer Science). ISBN 9781447156017.

LEE, S. et al. Table2graph: A scalable graph construction from relational tables using map-reduce. In: *2015 IEEE First International Conference on Big Data Computing Service and Applications*. Redwood City, CA: IEEE, 2015.

LIYANAARACHCHI, G. et al. Migdb - relational to nosql mapper. In: *Proceedings of the IEEE International Conference on Information and Automation for Sustainability (ICIAfS)*. [S.l.: s.n.], 2016.

LYSENKO, A. et al. Representing and querying disease networks using graph databases. *BioData Mining*, Springer Science and Business Media LLC, v. 9, n. 1, jul. 2016.

MCCREARY, D.; KELLY, A. *Making Sense of NoSQL - A Guide for Managers and the Rest of Us*. Shelter Island, NY: Manning Publications Co., 2013. 314 p. ISBN 9781617291074.

MEGID, Y. A.; EL-TAZI, N.; FAHMY, A. Using functional dependencies in conversion of relational databases to graph databases. Springer International Publishing, p. 350–357, 2018.

MITTRA, S. S. *Principles of Relational Database Design*. 1st. ed. Upper Saddle River, NJ, USA: Prentice Hal, 1991. ISBN 0137167962.

MOHANTY, H. Big data: An introduction. In: *Big Data: A Primer*. [S.l.]: Springer International Publishing, 2015. p. 1–28.

MOORE, C. C. B. et al. A biologically informed method for detecting rare variant associations. *BioData Mining*, Springer Science and Business Media LLC, v. 9, n. 1, ago. 2016. Disponível em: <https://doi.org/10.1186/s13040-016-0107-3>.

MOUDEN, Z. A. E.; JAKIMI, A.; HAJAR, M. An algorithm of conversion between relational data and graph schema. In: _____. *Smart Innovation, Systems and Technologies*. [S.l.]: Springer International Publishing, 2019. p. 594–602. ISBN 978-3-030-03576-1.

MUÑOZ-TORRES, P. M. et al. msBiodat analysis tool, big data analysis for high-throughput experiments. *BioData Mining*, Springer Science and Business Media LLC, v. 9, n. 1, ago. 2016. Disponível em: <https://doi.org/10.1186/s13040-016-0104-6>.

MURUGAN, K. et al. TaxKB: a knowledge base for new taxane-related drug discovery. *BioData Mining*, Springer Science and Business Media LLC, v. 8, n. 1, jun. 2015. Disponível em: <https://doi.org/10.1186/s13040-015-0053-5>.

MYERS, J. *Essential SQLAlchemy: Mapping Python to Databases*. Gravenstein Highway North, Sebastopol: O'Reilly Media, 2015. ISBN 149191646X.

NEUMANN, L. A. *Modelagem Participativa – Uma Nova Abordagem para Modelar Bancos de Dados Voltados a Grafos*. Dissertação (Mestrado em Ciência e Tecnologia da Computação) — Universidade Federal de Itajubá, 2017.

OLIVEIRA, A. T. de et al. Mapping and conversion between relational and graph databases models: A systematic literature review. In: *Advances in Intelligent Systems and Computing*. [S.l.]: Springer International Publishing, 2020. p. 539–543.

OREL, O.; ZAKOŠEK, S.; BARANOVIČ, M. Property oriented relational-to-graph database conversion. *Automatika*, Informa UK Limited, v. 57, n. 3, p. 836–845, jan. 2016.

PERDIGÃO, N.; ROSA, A. C.; O'DONOGHUE, S. I. The dark proteome database. *BioData Mining*, Springer Science and Business Media LLC, v. 10, n. 1, jul. 2017. Disponível em: <https://doi.org/10.1186/s13040-017-0144-6>.

POKORNÝ, J.; VALENTA, M.; TROUP, M. Graph pattern index for neo4j graph databases. In: *Communications in Computer and Information Science*. [S.l.]: Springer International Publishing, 2019. p. 69–90.

RAMAKRISHNAN, R.; GEHRKE, J. *Database Management Systems*. 3. ed. USA: McGraw-Hill, Inc., 2002. ISBN 0072465638.

RAMAKRISHNAN, R.; GEHRKE, J. *Sistemas de gerenciamento de banco de dados*. 3. ed. Porto Alegre: McGraw Hill Brasil, 2008. ISBN 9788563308771.

ROB, P.; CORONEL, C. *Sistemas De Banco De Dados: Projeto, Implementação e Gerenciamento*. 8. ed. São Paulo: Cengage Learning, 2011. ISBN 9788522107865.

ROBINSON, I.; WEBBER, J.; EIFREM, E. *Graph Databases: New Opportunities for Connected Data*. 2nd. ed. Sebastopol, CA: O'Reilly Media, Inc, 2015.

ROCHA, J. *Understanding Neo4j's data on disk*. 2019. Disponível em: <https://neo4j.com/developer/kb/understanding-data-on-disk/>.

ROSEN, K. H. *Discrete Mathematics and Its Applications*. 8th. ed. [S.l.]: McGraw-Hill Higher Education, 2019. ISBN 978-1-259-67651-2.

SADALAGE, P. J.; FOWLER, M. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. 1st. ed. [S.l.]: Addison-Wesley Professional, 2012. ISBN 0321826620.

SCHöPF, B. et al. OXPHOS remodeling in high-grade prostate cancer involves mtDNA mutations and increased succinate oxidation. *Nature Communications*, Springer Science and Business Media LLC, v. 11, n. 1, mar. 2020. Disponível em: <https://doi.org/10.1038/s41467-020-15237-5>.

SILBERCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. *Sistemas de Banco de Dados*. 6th. ed. Rio de Janeiro: Elsevier, 2012. ISBN 9788535245356.

SINGH, N. et al. MorCVD: A unified database for host-pathogen protein-protein interactions of cardiovascular diseases related to microbes. *Scientific Reports*, Springer Science and Business Media LLC, v. 9, n. 1, mar. 2019. Disponível em: <https://doi.org/10.1038/s41598-019-40704-5>.

STRAWN, G.; STRAWN, C. Relational databases: Codd, stonebraker, and ellison. *IT Professional*, v. 18, p. 63–65, 03 2016.

SULLIVAN, D. *NoSQL for Mere Mortals*. 1st. ed. Ann Arbor, Michigan: Addison-Wesley Professional, 2015. ISBN 0134023218.

TANG, J. et al. ArnetMiner. In: *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 08*. [S.l.]: ACM Press, 2008.

TEOREY, T. et al. *Database Modeling and Design*. 5. ed. Burlington, USA: Morgan Kaufmann, 2011. ISBN 9780123820204.

TSUR, E. E. Rapid development of entity-based data models for bioinformatics with persistence object-oriented design and structured interfaces. *BioData Mining*, Springer Science and Business Media LLC, v. 10, n. 1, mar. 2017. Disponível em: <https://doi.org/10.1186/s13040-017-0130-z>.

UNAL, Y.; OGUZTUZUN, H. Migration of data from relational database to graph database. In: *Proceedings of the 8th International Conference on Information Systems and Technologies*. New York, NY, USA: ACM, 2018. (ICIST '18), p. 6:1–6:5. ISBN 978-1-4503-6404-1. Disponível em: <http://doi.acm.org/10.1145/3200842.3200852>.

VIRGILIO, R. D.; MACCIONI, A.; TORLONE, R. Converting relational to graph databases. In: *First International Workshop on Graph Data Management Experiences and Systems*. New York, NY, USA: ACM, 2013. (GRADES '13), p. 1:1–1:6. ISBN 978-1-4503-2188-4. Disponível em: <http://doi.acm.org/10.1145/2484425.2484426>.

VIRGILIO, R. D.; MACCIONI, A.; TORLONE, R. Model-driven design of graph databases. In: *Conceptual Modeling*. Switzerland: Springer International Publishing, 2014. p. 172–185.

WU, D.; SAKR, S.; ZHU, L. Big data storage and data models. In: *Handbook of Big Data Technologies*. [S.l.]: Springer International Publishing, 2017. p. 3–29.

ZENG, X. et al. PIC, a paediatric-specific intensive care database. *Scientific Data*, Springer Science and Business Media LLC, v. 7, n. 1, jan. 2020. Disponível em: <https://doi.org/10.1038/s41597-020-0355-4>.