Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Computação e Sistemas

Implementação de um sistema para conversão de banco de dados no modelo relacional para modelo orientado a grafos

Pedro Lucas Evangelista de Paulo

TRABALHO DE CONCLUSÃO DE CURSO

ORIENTAÇÃO:
Rafael Frederico Alexandre

COORIENTAÇÃO:
Bruno Rabello Monteiro

Setembro, 2025
João Monlevade–MG

Pedro Lucas Evangelista de Paulo

Implementação de um sistema para conversão de banco de dados no modelo relacional para modelo orientado a grafos

Orientador: Rafael Frederico Alexandre

Coorientador: Bruno Rabello Monteiro

Monografia apresentada ao curso de Engenharia de Computação do Instituto de Ciências Exatas e Aplicadas, da Universidade Federal de Ouro Preto, como requisito parcial para aprovação na Disciplina "Trabalho de Conclusão de Curso II".

Universidade Federal de Ouro Preto

João Monlevade

Setembro de 2025

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E APLICADAS
DEPARTAMENTO DE COMPUTACAO E SISTEMAS

FOLHA DE APROVAÇÃO

Pedro Lucas Evangelista de Paulo

Implementação de um sistema para conversão de banco de dados no modelo relacional para modeloorientado a grafos

Monografi a apresentada ao Curso de Engenharia da Computação da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de bacharel.

Aprovada em 04 de setembro de 2025

Membros da banca

Doutor - Rafael Frederico Alexandre - Orientador (Universidade Federal de Ouro Preto)
Doutor - Bruno Rabello Monteiro - Coorientador (Universidade Federal de Ouro Preto)
Doutor - George Godim da Fonseca - (Universidade Federal de Ouro Preto)
Mestre - Zilton Cordeiro Jr.

Rafael Frederico Alexandre, orientador do trabalho, aprovou a versão fi nal e autorizou seu depósito na BibliotecaDigital de Trabalhos de Conclusão de Curso da UFOP em 17/12/2025

Documento assinado eletronicamente por **Rafael Frederico Alexandre**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 17/12/2025, às 17:40, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **1034359** e o código CRC **7168234C**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: (31)3808-0819 - www.ufop.br

Este trabalho é dedicado ao bondoso Deus, à minha família e aos meus amigos.

Agradecimentos

Agradeço à minha família, primeiramente ao meu pai Wilson, minha mãe Jaqueline e minha irmã Bárbara e também ao meu tio Thiago e minha vó Eunice por todo o suporte e apoio em todos os momentos, principalmente quando estive longe fisicamente.

Tenho que mencionar minha gratidão aos meus amigos, principalmente aqueles em que estavam todo o tempo comigo: minha grande dupla e parceiro de graduação Pedro Henrique Mendes (um grande gênio técnico) e meus amigos do dia a dia Daniel, Mateus, Vinícius, Mari e Kelly que traziam leveza no processo.

Ao meu orientador Prof. Dr. Rafael Alexandre por todo o acompanhamento, não somente nessa monografia, mas também durante toda a graduação, me dando oportunidades que tem valor imensurável, me mostrando o espaço da computação no âmbito da pesquisa e no mercado de trabalho.

Agradeço à Universidade Federal de Ouro Preto pelo curso e pela qualidade do ensino oferecido e aos meus professores que fazem isso ser possível por meio da universidade, dando mais do que eu mereço em ideias e conhecimento.

E por ultimo (e mais importante) agradeço à Deus, por sempre me dar tudo que precisei, principalmente paz e tranquilidade.

"Corrigir a rota do barco, às vezes, é simplesmente confiar.
Não sou eu a navegar, mas quem faz oceanos inteiro"

— Marcos Almeida, *in: Ventos.*

Resumo

Na era moderna, grande parte da tomada de decisões, análises e automações ocorre por meio da existência e da manipulação de dados. Por isso, os sistemas que os gerenciam de forma estruturada, desde o armazenamento até a recuperação, são de suma importância. O presente trabalho propõe e implementa um sistema automatizado para conversão de bancos de dados relacionais em estruturas orientadas a grafos (no Neo4j). Partindo da extração de metadados (como chaves primárias e estrangeiras) e dados das tabelas em SQL Server, o sistema realiza a transformação de registros para o formato JSON estruturado, define regras de aninhamento e identifica o que deve ser modelado como nós ou arestas. A abordagem foi validada por meio de testes com duas bases amplamente utilizadas: *Northwind* e *AdventureWorks2022*. Os resultados obtidos demonstram a viabilidade da ferramenta na geração de modelos compatíveis com grafos. Além disso, são apresentadas comparações entre consultas SQL e Cypher, destacando os benefícios do modelo orientado a grafos em cenários exploratórios e de múltiplos relacionamentos. O estudo também aponta os desafios enfrentados durante a implementação, como o tratamento de chaves compostas e a escolha de representações adequadas para diferentes tipos de tabelas.

Palavras-chave: banco de dados relacional. Banco de dados orientado a grafos. Conversão automática. Transformação de esquema. Neo4j.

Abstract

In the modern era, a significant portion of decision-making, analysis, and automation relies on the availability and handling of data. Therefore, systems that manage data in a structured way, from storage to manipulation and retrieval, are of utmost importance. This work proposes and implements an automated system for converting relational databases into graph-oriented structures (in Neo4j). Starting from the extraction of metadata (such as primary and foreign keys) and table data from SQL Server, the system transforms records into a structured JSON format, defines nesting rules, and identifies what should be modeled as nodes or edges. The approach was validated through tests with two widely used databases: *Northwind* and *AdventureWorks2022*. The results obtained demonstrate the feasibility of the tool in generating models compatible with graph databases. Additionally, comparisons between SQL and Cypher queries are presented, highlighting the benefits of the graph-oriented model in exploratory and multi-relational scenarios. The study also discusses the challenges encountered during implementation, such as handling composite keys and selecting appropriate representations for different types of tables.

Keywords: Relational database. Graph database. Automatic conversion. Schema transformation. Neo4j.

Lista de figuras

Lista de tabelas

Sumário

1 Introdução

A crescente digitalização de processos em diferentes setores da sociedade tem gerado uma explosão no volume, variedade e complexidade dos dados manipulados por sistemas computacionais. Em resposta a esse fenômeno, os bancos de dados relacionais, amplamente adotados desde os trabalhos pioneiros de Codd (1970), consolidaram-se como o paradigma dominante para o armazenamento estruturado de informações, oferecendo robustez, normalização de dados e operações baseadas em álgebra relacional.

Contudo, o modelo relacional não é o único utilizado, pois pode apresentar limitações significativas ao lidar com estruturas altamente conectadas. Operações de junção múltiplas, recorrentes em consultas complexas, podem comprometer o desempenho e a legibilidade dos esquemas, especialmente em contextos como redes sociais, sistemas de recomendação, motores de busca e análise de dependências. Nessas aplicações, os dados possuem uma natureza inerentemente relacional, onde o foco se desloca do conteúdo das entidades para a maneira como elas se conectam.

Diante desse cenário, os bancos de dados orientados a grafos emergem como uma alternativa mais expressiva e performática. Fundamentados em conceitos da teoria dos grafos (WEST, 2001), esses sistemas modelam informações como vértices (nós) e arestas (relacionamentos), permitindo uma representação mais natural e eficiente para domínios fortemente interconectados. O modelo de grafos elimina a necessidade de junções explícitas, o que reduz a complexidade de consultas e favorece tempos de resposta menores, especialmente em consultas recursivas ou com múltiplos saltos (ANGLES et al., 2020).

Entretanto, a conversão de um banco de dados relacional para um modelo orientado a grafos não é trivial. Essa migração envolve mais do que a simples transferência de dados — exige uma transformação conceitual, onde tabelas, colunas, chaves primárias e estrangeiras devem ser reinterpretadas no contexto de vértices e arestas. Tal processo demanda uma abordagem sistemática que preserve a semântica dos dados originais e otimize sua nova representação.

Dentre os esforços já existentes para solucionar esse problema, iremos destacar nesse projeto o ThrusterDB, proposto por (VICENTE et al., 2019), que estabelece uma arquitetura modular para a conversão automática de bancos relacionais em grafos. Essa abordagem inclui etapas de mapeamento conceitual, análise de dependências, classificação de tabelas e geração de saídas compatíveis com bancos de grafos como o Neo4j.

Inspirado por essas diretrizes, este trabalho propõe o desenvolvimento de um sistema automatizado para a transformação de bancos relacionais em estruturas orientadas a grafos. A solução é construída utilizando a linguagem **Python**, com a extração de

metadados de um banco **SQL Server** e a geração de representações compatíveis com o banco de grafos **Neo4j**.

1.1 O problema de pesquisa

Dado um banco de dados relacional com estrutura arbitrária, como automatizar a conversão de seu modelo e dados para uma representação orientada a grafos, minimizando perda de semântica e preservando os relacionamentos entre entidades?

1.2 Objetivos

O trabalho consiste em desenvolver um sistema capaz de converter automaticamente um banco relacional em uma estrutura grafo, respeitando suas entidades e relacionamentos e analisar suas utilizações.

Este trabalho possui os seguintes objetivos específicos:

- Discorrer sobre bancos de dados e as diferenças acerca de modelo relacional e orientado a grafos;
- Extrair metadados estruturais (tabelas, colunas, chaves primárias e estrangeiras) de um banco relacional;
- Classificar tabelas quanto ao seu papel no modelo (vértices, arestas ou aninhamentos);
- Comparar o modelo de conversão do algoritmo proposto com algoritmos existentes;
- Validar e entender os resultados obtidos de uma mapeamento de dados real;

1.3 Metodologia

A pesquisa segue uma abordagem aplicada, fundamentada no desenvolvimento de um protótipo funcional capaz de realizar, de forma automática, as etapas de Extração, Transformação e Carga (ETL) entre os modelos relacional e de grafos. O sistema realiza introspecção sobre o banco relacional para coletar seus metadados, analisa suas relações de chave e grau de conectividade, e gera estruturas em **JSON** (JavaScript Object Notation) e inserções **Cypher** para a persistência no banco de grafos.

A base teórica é apoiada na literatura de bancos de dados, teoria dos grafos e trabalhos existentes sobre transformação de modelos, especialmente o ThrusterDB (VICENTE et al., 2019). O sistema é validado com bases de dados públicas e privadas,

demonstrando sua capacidade de representar relacionamentos complexos em grafos e facilitar consultas conectadas.

Para toda a revisão da literatura, foi pensado em trazer toda a linha de pensamento e de conceitos de bancos de dados, desde de a sua concepão simples, passando pelos conceitos relacionais e orientado a grafo e finalizando nas análises de outras implementações de mapeamentos relacionados.

Para o desenvolvilmento do algoritmo foi utilizado o conhecimento de outros autores aplicados para obter um resultado diferente a ser analisado, visto que há muitas formas diferentes de se mapear um banco de dados relacional e a lidar com suas cardinalidades e propriedades.

1.4 Organização do trabalho

A fim de concluir os objetivos propostos e tentar solucionar a problemática, este trabalho está estruturado da seguinte forma:

Capítulo 2 Apresenta a revisão bibliográfica sobre bancos de dados, seus modelos de representação, bancos em grafos e abordagens de conversão;

Capítulo 3 Detalha o desenvolvimento do sistema proposto, os algoritmos utilizados, decisões de projeto e a arquitetura do processo de conversão;

Capítulo 4 Apresenta os resultados obtidos e análise das representações geradas;

Capítulo 5 Aponta as conclusões e possíveis direções para trabalhos futuros.

2 Revisão bibliográfica

Este capítulo apresenta o estado da arte, uma revisão da literatura e os trabalhos correlacionados ao tema apresentado, nele iremos aprofundar em conceitos de banco de dados e os modelos de aplicação, tais quais os modelos citados: Relacional e Orientado a grafos.

2.1 Banco de Dados

Banco de dados é um conjunto de informações estruturadas, e, para o contexto abordado, esse banco é armazenado de forma eletrônica em um computador. Esses dados, na maneira estruturada, ficam facilmente manipuláveis para gerenciamento, processamento, atualização e controle geral (ORACLE, 2020). São diversos os tipos e aplicações de bancos de dados, nesse projeto entraremos a fundo nos modelos relacionais e não relacionais, mais especificamente o modelo orientado a grafos.

É fato a importância de sua utilização no mundo técnologico que temos hoje, tudo que fazemos existem dados sendo processados. Esses dados, iniciamente de forma textual e numérica, se expandem até para processamentos geográficos (SIG). Coronel e Rob (2010) conta sobre os tipos de bancos de dados e sua história, passando de bancos de dados não computadorizados ate os que conhecemos hoje.

2.1.1 Dados e Metadados

Os dados presentes em um banco de dados representam informações reais e relevantes para uma aplicação, como nomes de usuários, valores de vendas, ou registros de transações. Já os metadados são descrições que fornecem informações estruturais sobre os dados armazenados, tais como tipos de atributos, definições de tabelas, domínios permitidos e restrições de integridade.

Como é dito por Elmasri e Navathe (2011), os metadados são essenciais para a organização do banco, pois permitem que o sistema compreenda a estrutura dos dados, possibilitando validação, consulta e manutenção de forma automatizada. Por exemplo, enquanto os dados armazenam o nome de um cliente, os metadados indicam que este campo é de um tipo específico (como uma sequência de caracteres - *string*) e que pode ou não ser nulo.

2.1.2 Sistema Gerenciador de Banco de Dados (SGBD)

Para o gerenciamento desses bancos de dados, especificamente os computadorizados, são utilizados sistemas chamados de SGDB. Esses são sistemas muitas vezes contam propriedades que tornam a experiência mais qualitativa (SILBERSCHATZ; KORTH; SUDARSHAN, 2006):

- Controle de redundância;
- Restrição de acesso;
- Armazenamento persistente;
- Consultas eficientes;
- Backup e restauração;
- Integridade dos dados;

O SGDB fornece funcionalidades para definição de estruturas de dados, inserção e modificação de registros, controle de acesso de usuários, execução de transações e recuperação de falhas.

A Figura 1 ilustra, de forma simplificada, a arquitetura de um sistema de banco de dados, destacando a interação entre os programas de aplicação, o software de gerenciamento (SGBD) e os dados armazenados.

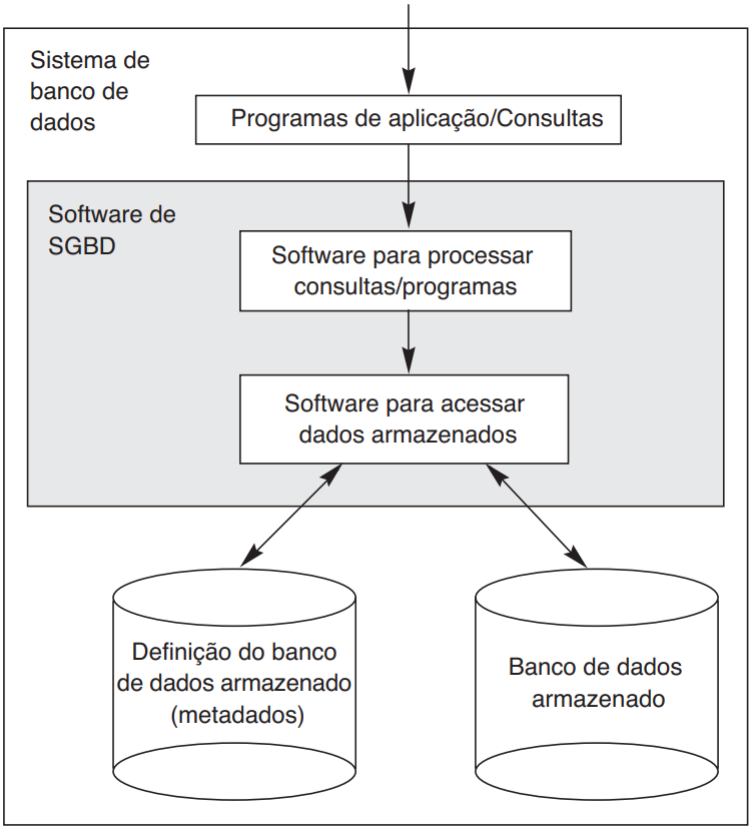
De acordo com McMillen (2024), os SGBDs modernos evoluíram significativamente para oferecer suporte a múltiplos modelos de dados (relacional, documentos, grafos), integração com ambientes distribuídos e computação em nuvem, além de fornecer ferramentas para análises em larga escala. Exemplos populares de SGBDs incluem *MySQL*, *PostgreSQL*, *Oracle*, *SQL Server*, *MongoDB* e *Neo4j*.

2.1.3 Linguagens de Consulta

A interação com o banco de dados é realizada por meio de linguagens específicas que permitem definir estruturas, manipular dados e controlar transações. No SQL Server (ferramenta que será utilizada nesse projeto), essas linguagens são divididas em categorias:

- Linguagem de Definição de Dados (DDL): utilizada para definir a estrutura do banco de dados, como criação de tabelas, índices e restrições *(CREATE, ALTER, DROP)*.
- Linguagem de Manipulação de Dados (DML): permite a inserção, atualização, consulta e exclusão de dados *(SELECT, INSERT, UPDATE, DELETE)*.

Figura 1 – Representação de um ambiente de banco de dados



Fonte: Elmasri e Navathe (2011, p. 23)

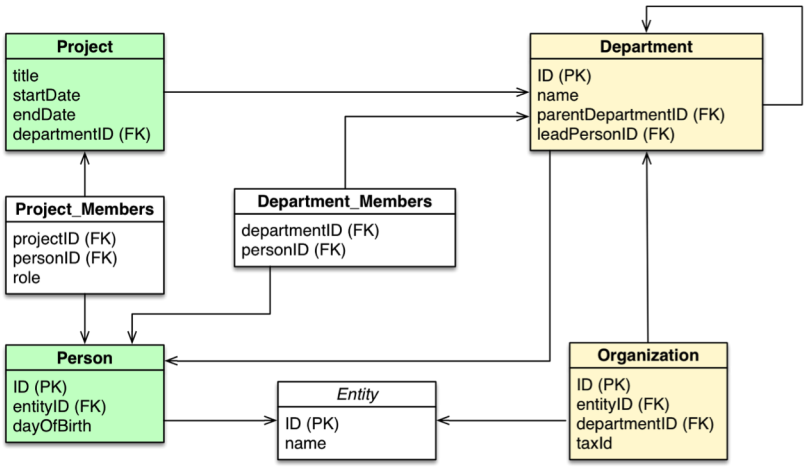
- Linguagem de Controle de Dados (DCL): responsável por gerenciar permissões e controle de acesso aos dados *(GRANT, REVOKE)*.
- Linguagem de Controle de Transações (TCL): utilizada para gerenciar transações e garantir a atomicidade das operações *(COMMIT, ROLLBACK)*.

Conforme Elmasri e Navathe (2011), considerando o modelo relacional, essas linguagens são fundamentais para garantir que os dados possam ser acessados e manipulados de forma segura, eficiente e consistente, mesmo em ambientes com múltiplos usuários concorrentes.

2.2 Modelo Relacional

O modelo relacional, abordado nesta seção, foi instânciado por Codd (1970) usando o conceito de relação matemática, como uma tabela de valores, visando independência dos dados no acesso e integridade estrutural. A Figura 2 é um exemplo de um banco no modelo relacional, suas tabelas e relacionamentos.

Figura 2 – Exemplo de um modelo relacional



Fonte: Hunger, Boyd e Lyon (2016)

2.2.1 Fundamentos do Modelo Relacional

O modelo relacional é uma das abordagens mais utilizadas para organização e gerenciamento de dados. Sua principal característica é a representação dos dados em estruturas chamadas *relações*, que são comumente implementadas como tabelas bidimensionais contendo linhas e colunas (CODD, 1970).

Cada relação (ou tabela) é composta por **tuplas** (linhas), que representam instâncias individuais de dados, e por **atributos** (colunas), que representam os diferentes campos ou propriedades de uma entidade. Por exemplo, uma tabela de "Alunos" pode conter atributos como *matricula*, *nome*, *email*, e *data de nascimento*, sendo que cada tupla armazenará os valores correspondentes a um aluno específico.

Um aspecto essencial do modelo relacional é a associação de cada atributo a um **domínio**, isto é, a um conjunto de valores válidos (ELMASRI; NAVATHE, 2011). Por exemplo, um atributo do tipo 'data' deve conter apenas valores pertencentes ao domínio de datas válidas. Essa definição de domínio contribui para garantir a integridade e a consistência dos dados armazenados.

Além disso, o modelo relacional adota princípios matemáticos da lógica de predicados e da álgebra relacional, conferindo-lhe um rigor formal que facilita a aplicação de operações sobre os dados e a verificação de integridade. As estruturas relacionais são independentes da ordem das tuplas e dos atributos, o que proporciona flexibilidade no armazenamento e na recuperação das informações (SILBERSCHATZ; KORTH; SUDARSHAN, 2006).

Outra característica central do modelo relacional é a noção de **esquema relacional**,

que define a estrutura da base de dados em termos de suas relações, atributos e restrições. Essa abstração separa a estrutura lógica dos dados da sua implementação física, o que contribui para a independência dos dados e facilita sua manutenção ao longo do tempo.

O sucesso do modelo relacional está atrelado à sua simplicidade conceitual, à robustez das operações definidas pela álgebra relacional e à sua adaptabilidade a diferentes domínios de aplicação. Posteriormente, linguagens como a SQL foram desenvolvidas para implementar e popularizar esses conceitos nos sistemas comerciais. No entanto, suas limitações em contextos que demandam relacionamentos altamente conectados ou dados semiestruturados motivaram o surgimento de modelos alternativos, como os bancos orientados a documentos e a grafos, os quais serão abordados nas próximas seções.

Além de sua estrutura conceitual, o modelo relacional é sustentado por formalismos matemáticos, como a álgebra relacional e o cálculo relacional, que servem de base para a construção e otimização de consultas (SILBERSCHATZ; KORTH; SUDARSHAN, 2006). No entanto, tais aspectos não serão explorados em profundidade neste trabalho, uma vez que o foco recai sobre a estrutura de dados e sua transformação para o modelo de grafos.

2.2.2 Chaves e Restrições

As chaves e restrições são elementos fundamentais no modelo relacional, pois definem a maneira como os dados se organizam e se relacionam dentro do banco. Elas são utilizadas para garantir a integridade dos dados, a consistência entre tabelas e a unicidade de registros. Sua correta definição é imprescindível tanto para o bom funcionamento de sistemas baseados em bancos relacionais quanto para transformações estruturadas entre modelos, como a migração para bancos orientados a grafos.

Chaves são atributos ou conjuntos de atributos que identificam unicamente uma tupla em uma relação. A mais comum delas é a **chave primária** (*primary key*), que assegura que não existam duas linhas idênticas na mesma tabela. Além da chave primária, é possível definir **chaves candidatas** (possíveis alternativas à chave primária), **chaves substitutas** (geradas artificialmente, como IDs) e **chaves estrangeiras** (*foreign keys*), que são atributos que criam um vínculo entre tabelas distintas.

As **chaves estrangeiras** são particularmente relevantes, pois expressam relacionamentos entre entidades. Por exemplo, uma tabela “Turmas” pode conter um campo ‘matricula_aluno‘ que referencia a chave primária da tabela “Alunos”. Esse tipo de vínculo é o que permite representar relacionamentos como um-para-um (1:1), um-para-muitos (1:N) e muitos-para-muitos (N:N), que serão explorados em seções posteriores do trabalho.

Além das chaves, o modelo relacional adota diversas **restrições de integridade**, que asseguram a validade dos dados inseridos ou atualizados. (ELMASRI; NAVATHE, 2011) - As principais restrições são:

- **Integridade de entidade**: assegura que a chave primária de uma relação nunca seja nula.
- **Integridade referencial**: garante que valores de uma chave estrangeira correspondam a uma tupla existente na tabela referenciada.
- **Restrição de domínio**: impõe que os valores de um atributo pertençam a um conjunto pré-definido (tipo de dado, faixa de valores, etc.).
- **Restrições adicionais**: incluem unicidade, nulabilidade, valores padrão, e expressões condicionais mais complexas.

Essas restrições são automaticamente aplicadas e verificadas pelos Sistemas Gerenciadores de Banco de Dados (SGBDs), proporcionando robustez ao sistema e evitando inconsistências que poderiam comprometer a confiabilidade da aplicação.

Para efeitos de transformação para modelos orientados a grafos, as chaves primárias serão representadas como identificadores únicos de nós, enquanto as chaves estrangeiras frequentemente se transformam em arestas (relacionamentos) entre esses nós ou em nós, a depender da estratégia utilizada. Assim, o entendimento das chaves e restrições no modelo relacional é um passo indispensável na construção de estruturas equivalentes no modelo de grafos.

2.2.3 Cardinalidade dos Relacionamentos

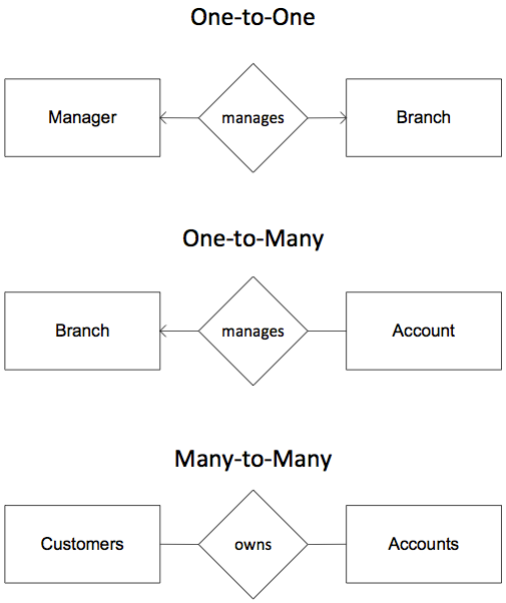
A cardinalidade é um conceito central na modelagem de dados relacionais e refere-se à quantidade de ocorrências de uma entidade que pode estar associada a ocorrências de outra entidade em um relacionamento. De acordo com (ELMASRI; NAVATHE, 2011), essa definição é essencial para compreender como as entidades se conectam e se influenciam, tanto em bancos relacionais quanto em outros modelos, como os orientados a grafos.

No modelo relacional, os tipos mais comuns de cardinalidade são:

- **Um para um (1:1)**: Cada registro de uma tabela está associado a, no máximo, um registro da outra tabela, e vice-versa. Exemplo: uma pessoa pode ter apenas um passaporte, e cada passaporte pertence a uma única pessoa.
- **Um para muitos (1:N)**: Um registro de uma tabela pode estar associado a vários registros da outra, mas cada registro da segunda tabela está vinculado a apenas um da primeira. Exemplo: um cliente pode ter vários pedidos, mas cada pedido pertence a um único cliente.
- **Muitos para muitos (N:N)**: Vários registros de uma tabela podem estar associados a vários registros da outra. Exemplo: estudantes e disciplinas — um estudante pode

cursar várias disciplinas e uma disciplina pode ter vários estudantes. Esse tipo de relacionamento geralmente requer uma tabela intermediária (tabela de junção).

Figura 3 – Tipos de cardinalidade



Fonte: USI Informatics

Na Figura 3 temos os tipos de cardinalidade e nela é exemplificado o que foi explicado anteriormente. No relacionamento 1:1 o gerente só gerencia uma filial, no 1:N a filial gerencia várias contas, no N:N cada cliente pode estar associado a muitas contas, e cada conta pode estar associada a muitos clientes.

A definição correta da cardinalidade é fundamental para evitar problemas como duplicação indevida de dados ou perda de integridade nas relações entre tabelas. A escolha equivocada da cardinalidade pode comprometer a lógica de negócios representada no banco de dados e dificultar operações como consultas e atualizações (CONNOLLY; BEGG, 2015).

Além disso, a cardinalidade tem implicações diretas na conversão de dados relacionais para o modelo de grafos. Em sistemas orientados a grafos, os relacionamentos são representados como arestas entre nós, e a multiplicidade (ou cardinalidade) determina a natureza e a direção dessas conexões. Por exemplo, uma relação 1:N pode ser representada como múltiplas arestas saindo de um nó origem para vários nós destino, enquanto uma relação N:N exige modelagem específica (como a criação de nós intermediários ou uso de propriedades nas arestas).

O entendimento preciso da cardinalidade é essencial tanto para o projeto de bancos de dados relacionais quanto para transformações estruturais entre diferentes modelos (CORONEL; ROB, 2010).

2.2.4 Vantagens e Limitações do Modelo Relacional

O modelo relacional consolidou-se como a principal abordagem para gerenciamento de dados nas últimas décadas, sendo adotado em uma ampla gama de aplicações corporativas, científicas e administrativas. Seu sucesso está relacionado a diversos fatores técnicos e conceituais que o tornaram robusto, confiável e amplamente padronizado.

Entre as principais **vantagens do modelo relacional**, especificados por Connolly e Begg (2015), Silberschatz, Korth e Sudarshan (2006):

- **Simplicidade conceitual**: a estrutura tabular de dados (linhas e colunas) é intuitiva e facilmente compreendida por analistas, desenvolvedores e usuários finais.
- **Independência física e lógica**: permite que alterações na estrutura física do armazenamento não afetem a lógica do modelo nem as aplicações que o utilizam.
- **Linguagem padronizada (SQL)**: a linguagem SQL tornou-se padrão de mercado, facilitando interoperabilidade entre sistemas e reduzindo a curva de aprendizado.
- **Integridade e consistência**: graças a chaves, restrições e mecanismos de transação, o modelo relacional oferece forte suporte à integridade referencial e à consistência dos dados.
- **Maturidade e suporte**: os SGBDs relacionais possuem décadas de desenvolvimento, com suporte técnico consolidado, comunidades ativas e documentação extensa .

Apesar de sua ampla adoção, o modelo relacional apresenta algumas **limitações**, especialmente em contextos modernos como big data, redes sociais e aplicações com dados altamente conectados (HECHT; JABLONSKI, 2011) e (MONIRUZZAMAN; HOSSAIN, 2013):

- **Dificuldade em modelar relacionamentos complexos**: relacionamentos muitos-para-muitos ou redes altamente interligadas exigem junções múltiplas, o que afeta o desempenho e a clareza da modelagem.
- **Baixa performance em joins extensivos**: consultas que envolvem junções de várias tabelas, especialmente em grandes volumes de dados, podem resultar em operações custosas do ponto de vista computacional .
- **Rigidez no esquema**: alterações no esquema relacional (como adição de colunas ou tabelas) podem demandar migrações complexas, o que dificulta a evolução de sistemas dinâmicos ou orientados a prototipação rápida.

- **Escalabilidade limitada horizontalmente**: embora seja possível escalar bancos relacionais, isso geralmente requer técnicas mais complexas como particionamento (sharding) ou replicação, que não são nativas à arquitetura relacional tradicional.

Essas limitações foram fatores determinantes para o surgimento e a adoção crescente de modelos de banco de dados não relacionais, como os orientados a documentos, chave-valor e grafos. Estes modelos oferecem maior flexibilidade, escalabilidade e desempenho em contextos específicos, como veremos nas próximas seções.

Vale destacar que, apesar das limitações pontuadas, o modelo relacional continua sendo uma tecnologia extremamente consolidada e robusta, como discutido ao longo do Capítulo 2.2. Sua ampla adoção ao longo das décadas comprova sua eficiência e estabilidade em diversos contextos. Portanto, este trabalho não tem a intenção de desvalorizar ou substituir o modelo relacional, mas sim propor uma alternativa viável para cenários específicos. O modelo relacional permanece essencial e não deve, de forma alguma, ser descartado ou subestimado.

2.3 Modelos Não Relacionais

Com o crescimento exponencial de dados e a diversidade de aplicações modernas, surgiram novas demandas que desafiaram os limites do modelo relacional tradicional. Em resposta a esses desafios, emergiu uma nova classe de bancos de dados conhecida como *NoSQL* (Not Only SQL), agrupando sistemas que não seguem rigidamente o modelo tabular proposto (CODD, 1970), mas que oferecem alternativas mais flexíveis para lidar com grandes volumes de dados, dados semi-estruturados e estruturas altamente conectadas.

Os bancos NoSQL caracterizam-se por sua escalabilidade horizontal, flexibilidade de esquemas e desempenho otimizado em cenários específicos, como redes sociais, sistemas de recomendação, processamento de eventos e aplicações em tempo real. (MONIRUZZAMAN; HOSSAIN, 2013) O termo NoSQL não significa ausência de SQL, mas sim que esses bancos de dados vão além do paradigma relacional, explorando novos modelos de organização e consulta dos dados.

Como especifica Borad, Metah e Chauhan (2017), as principais categorias de bancos de dados NoSQL incluem:

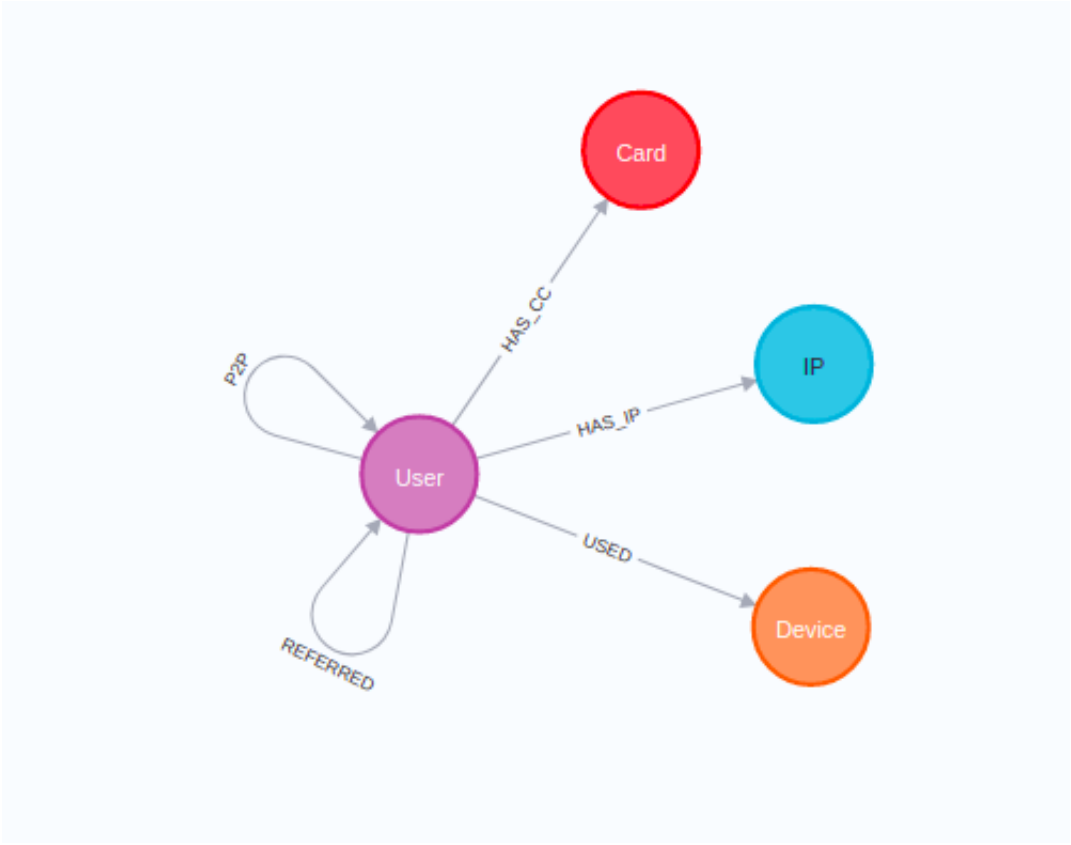
- **Chave-valor**: armazenam pares simples de chave e valor, sendo altamente performáticos para buscas diretas (ex: Redis, Riak).
- **Orientados a documentos**: armazenam estruturas semi-estruturadas como JSON ou BSON, permitindo alta flexibilidade de esquemas (ex: MongoDB, CouchDB).

- **Orientados a colunas**: otimizados para leitura em larga escala de grandes volumes de dados distribuídos por colunas (ex: Apache Cassandra, HBase).
- **Orientados a grafos**: modelam dados como nós (entidades) e arestas (relacionamentos), sendo ideais para representar estruturas altamente conectadas (ex: Neo4j, ArangoDB).

Cada modelo atende a diferentes necessidades de aplicação. Este trabalho foca especialmente no modelo orientado a grafos, dada sua adequação para representar relacionamentos complexos e sua crescente aplicação em domínios como redes sociais, sistemas de recomendação, segurança cibernética, entre outros.

Na próxima subseção, exploraremos com mais profundidade o funcionamento e as características dos bancos de dados orientados a grafos, destacando como suas estruturas contrastam com o modelo relacional e os benefícios que oferecem em contextos altamente relacionais. Além disso, na Figura 4 está exemplificado um esquema de grafos (não relacional) que demonstra um sistema de conexões entre usuário, aparelho e IP.

Figura 4 – Figura exemplo de um modelo não relacional



Fonte: Site oficial neo4j

2.3.1 Contexto e Motivação para NoSQL

Com o crescimento exponencial da quantidade de dados gerados por aplicações modernas — especialmente com o advento de redes sociais, dispositivos móveis e Internet das Coisas (IoT) — surgiu a necessidade de sistemas de banco de dados capazes de lidar com grandes volumes de dados, alta variabilidade e mudanças frequentes de estrutura. Esse contexto motivou o surgimento dos bancos de dados NoSQL, que visam atender às limitações enfrentadas pelo modelo relacional tradicional em cenários de alta escalabilidade e flexibilidade (MONIRUZZAMAN; HOSSAIN, 2013).

O termo *NoSQL* surgiu inicialmente como uma negação ao modelo relacional, mas, com o tempo, passou a ser interpretado como *Not Only SQL*, enfatizando que esses sistemas não substituem necessariamente os relacionais, mas oferecem uma alternativa viável para casos específicos. Dentre os principais motivadores para a adoção de bancos NoSQL, destacam-se:

- **Escalabilidade Horizontal**: Sistemas NoSQL foram projetados para escalar horizontalmente de forma mais eficiente do que os relacionais, utilizando clusters de servidores distribuídos (POKORNY, 2013).
- **Alta Disponibilidade e Tolerância a Falhas**: Muitos bancos NoSQL seguem princípios da computação distribuída, como o teorema CAP, e garantem alta disponibilidade mesmo em situações de falha de nós (PADHY; PATRA; SATAPATHY, 2011).
- **Flexibilidade de Esquema**: Ao contrário dos bancos relacionais, os bancos NoSQL permitem a persistência de dados sem a necessidade de um esquema fixo, o que facilita a modelagem de dados heterogêneos e em constante evolução (BORAD; METAH; CHAUHAN, 2017).
- **Desempenho em Altas Cargas**: Para aplicações que exigem leitura e escrita em alta velocidade, como logs em tempo real, mensagens ou analytics, bancos NoSQL frequentemente oferecem desempenho superior (CATTELL, 2011).

Dessa forma, o surgimento dos bancos NoSQL reflete uma resposta prática às novas demandas de processamento e armazenamento impostas pelos sistemas modernos. Ainda assim, o modelo relacional continua sendo amplamente utilizado, sendo que a escolha entre modelos depende do tipo de aplicação e das necessidades específicas do negócio.

2.4 Banco de Dados Orientado a Grafos

Bancos de dados orientados a grafos (ou simplesmente Banco de Dados de Grafos) constituem uma categoria de sistemas de armazenamento de banco de dados NoSQL projetados para armazenar, mapear e consultar dados por meio de representações baseadas em grafos. Nesses bancos, os dados são modelados por meio de nós (vértices) e arestas (relacionamentos), sendo ambos capazes de conter propriedades, o que facilita a modelagem de dados altamente conectados e complexos, como redes sociais, sistemas de recomendação, cadeias logísticas, entre outros (ROBINSON; WEBBER; EIFREM, 2015).

A principal motivação por trás do uso de bancos de grafos está na eficiência na execução de consultas com múltiplos relacionamentos, uma vez que sistemas relacionais tradicionais tendem a perder desempenho quando as junções (*joins*) se tornam profundas e numerosas. Em contrapartida, bancos de grafos exploram a navegação direta por relacionamentos, permitindo acessos em tempo constante para saltos de relacionamento (Angles et al., 2020).

De acordo com Borad, Metah e Chauhan (2017), os bancos orientados a grafos diferenciam-se por serem capazes de representar a semântica do domínio de forma mais intuitiva, aproximando-se da estrutura real dos dados e facilitando a interpretação e exploração por parte dos desenvolvedores. Esses sistemas tornam explícitas as conexões entre entidades, o que favorece análises como detecção de comunidades, centralidade e caminhos mínimos.

Entre os principais exemplos de bancos orientados a grafos disponíveis no mercado destacam-se:

Neo4j: sistema amplamente utilizado, com suporte à linguagem declarativa Cypher;

ArangoDB: banco multimodelo com suporte a grafos;

Amazon Neptune: serviço gerenciado de banco de grafos da AWS;

OrientDB: combina grafos com recursos de bancos orientados a documentos;

Além disso, linguagens específicas como Cypher, Gremlin e SPARQL foram desenvolvidas ou adaptadas para esse paradigma, permitindo a formulação de consultas complexas e semânticas por meio da navegação por padrões de relacionamentos (Angles et al., 2020).

2.4.1 Fundamentos de Teoria de Grafos

O modelo orientado a grafos se fundamenta na Teoria de Grafos, uma área da matemática discreta que estuda estruturas compostas por entidades chamadas de *vértices*

(ou nós) e conexões entre elas, denominadas *arestas* (WEST, 2001). Essa representação é especialmente adequada para modelar relacionamentos complexos, como redes sociais, sistemas de recomendação, mapas de rotas, entre outros.

Formalmente, um grafo G é definido como um par ordenado $G = (V, E)$, onde V é o conjunto de vértices (nós) e E é o conjunto de arestas, que representam pares de elementos de V. Em bancos de dados orientados a grafos, tanto os nós quanto as arestas podem conter um conjunto arbitrário de *propriedades* na forma de pares chave-valor, o que permite enriquecer o modelo com informações adicionais sem perda de desempenho (ANGLES; GUTIERREZ, 2008).

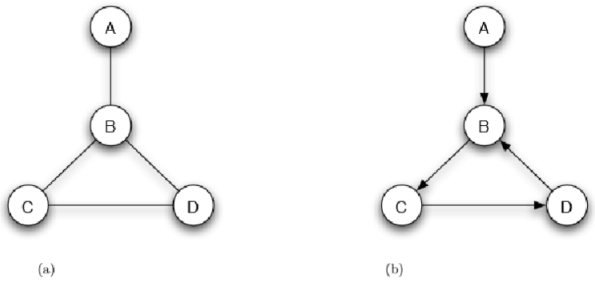
Exemplo: Em uma rede social, um nó pode representar uma pessoa (com propriedades como nome, idade), e uma aresta pode representar uma relação de amizade (com propriedade como data de início da amizade).

Os grafos podem ser classificados quanto à direção das suas arestas:

- **Grafos Não Direcionados**: As conexões entre os nós não possuem direção, ou seja, se um nó A está conectado a um nó B, então B também está conectado a A. São úteis para representar relações simétricas, como amizade mútua ou interconexões físicas.
- **Grafos Direcionados (Dígrafos)**: As arestas possuem direção, representando relações assimétricas. Se uma aresta vai de A para B, isso não implica uma conexão inversa. Essa estrutura é adequada para representar fluxos, como relações hierárquicas, ligações entre páginas da web ou transações financeiras (NEWMAN, 2010).

O uso de grafos direcionados é predominante em bancos de dados orientados a grafos, pois permite modelar com precisão os sentidos dos relacionamentos. Além disso, conceitos como grafos ponderados (com pesos associados às arestas) e grafos cíclicos ou acíclicos também são relevantes em aplicações específicas, aplicações essas que não serão aprofundadas nesse projeto. Na Figura 5 encontramos os exemplos das duas classificações citadas aqui.

Figura 5 – Figura demonstrando a) grafo não direcional e b) grafo direcional



Fonte: Networks, Crowds, and Markets: Easley e Kleinberg

2.4.2 Estrutura de um Banco de Dados de Grafos

Os bancos de dados orientados a grafos adotam uma estrutura nativa baseada em vértices (nós), arestas (relacionamentos) e propriedades, refletindo diretamente os conceitos da Teoria de Grafos. Cada nó representa uma entidade (como uma pessoa, produto ou local), enquanto as arestas representam relações entre essas entidades (como "amigo de", "comprou" ou "situado em") (ROBINSON; WEBBER; EIFREM, 2015).

- **Nós (Vertices)**: São instâncias de entidades e contêm propriedades em pares *chave-valor*. Podem ter rótulos (*labels*) que os classificam em categorias.
- **Arestas (Edges)**: Conectam dois nós e também podem armazenar propriedades. Arestas são direcionadas (ou seja, vão de um nó de origem a um nó de destino) e possuem um tipo ou nome de relacionamento, como "AMIGO_DE" ou "TRABALHA_EM".
- **Propriedades**: São metadados associados a nós ou arestas, permitindo um modelo rico e flexível. Por exemplo, um nó "Pessoa" pode conter as propriedades "nome", "idade" e "cidade".

Diferentemente de bancos relacionais, onde relações são indiretamente modeladas por chaves estrangeiras e junções, os bancos de grafos mantêm os relacionamentos como elementos de primeira classe. Isso permite que consultas complexas de múltiplos saltos (por exemplo, "amigos de amigos que trabalham na mesma empresa") sejam executadas com desempenho superior e menor complexidade computacional (ANGLES et al., 2020).

Além disso, bancos de grafos modernos como Neo4j, JanusGraph, TigerGraph e ArangoDB implementam armazenamento e indexação otimizados para operações de travessia de grafos, frequentemente empregando estruturas como índices por label ou tipo de relacionamento.

2.4.3 Linguagens de Consulta (Cypher, Gremlin)

As linguagens de consulta para bancos de dados orientados a grafos foram desenvolvidas com o objetivo de facilitar a navegação e manipulação de estruturas conectadas, permitindo expressar de forma natural e eficiente as travessias de grafos. Duas das linguagens mais amplamente utilizadas são **Cypher** e **Gremlin**.

O Cypher é uma linguagem declarativa inspirada em SQL, criada para uso com o banco de dados Neo4j. Ela permite descrever padrões de grafos por meio de uma sintaxe expressiva e legível. Um dos seus principais diferenciais é o uso de símbolos gráficos para representar nós e arestas, por exemplo:

```
MATCH (p:Pessoa)-[:AMIGO_DE]->(amigo)
WHERE p.nome = 'Alice'
RETURN amigo.nome
```

A consulta acima busca os nomes de todos os amigos de uma pessoa chamada Alice. A notação com '()' representa nós, e '[:RELACAO]' representa arestas, o que facilita a compreensão visual da topologia da consulta (ROBINSON; WEBBER; EIFREM, 2015).

Já o Gremlin, por outro lado, é uma linguagem imperativa baseada em caminhos, parte do projeto Apache TinkerPop. Ela é projetada para ser multi-modelo, podendo ser usada com diversos bancos como JanusGraph, Amazon Neptune, entre outros. Gremlin adota um estilo de encadeamento de operações (pipeline), como exemplificado abaixo:

```
g.V().has('nome', 'Alice').out('AMIGO_DE').values('nome')
```

Neste exemplo, a consulta busca os nomes dos amigos de Alice, a partir de uma travessia orientada por vértices ('V()'), arestas de saída ('out()'), e filtragem por propriedade ('has()') (GROVER, 2015).

Enquanto Cypher prioriza legibilidade e facilidade de uso, sendo adequado para analistas e cientistas de dados, Gremlin é mais flexível e programável, sendo muitas vezes preferido em contextos com lógica de travessia mais complexa ou integrada a aplicações.

Ambas as linguagens possuem suporte a operações fundamentais de grafos como filtros, agregações, padrões recursivos, e subconsultas. Recentemente, esforços vêm sendo feitos para padronizar essas linguagens, como por exemplo a proposta do *GQL* (Graph Query Language) como padrão ISO (FRANCIS, 2022).

2.4.4 Vantagens do Modelo de Grafos

Os bancos de dados orientados a grafos têm se consolidado como uma alternativa eficaz para aplicações que exigem a modelagem e análise de estruturas altamente conectadas.

Sua principal força reside na capacidade de representar entidades e seus relacionamentos como elementos de primeira classe — nós e arestas — refletindo de forma direta os princípios da teoria de grafos (ANGLES; GUTIERREZ, 2008; ROBINSON; WEBBER; EIFREM, 2015).

Ao contrário dos bancos relacionais, que dependem de junções (*joins*) entre múltiplas tabelas para recuperar dados relacionados, os bancos de grafos armazenam as conexões diretamente nas arestas. Isso permite que operações de travessia — como encontrar vizinhos de um nó ou seguir cadeias relacionais — sejam realizadas com menor complexidade algorítmica e maior eficiência computacional, especialmente em grafos densos e com múltiplos níveis de relacionamento (ANGLES et al., 2020).

Essas, que podemos chamar de consultas exploratórias, são aquelas em que o usuário não define antecipadamente todo o caminho de navegação entre os dados, mas deseja explorar conexões e vizinhanças a partir de um determinado ponto no grafo. Diferente das consultas tradicionais — que seguem uma estrutura fixa baseada em `JOINs` — as consultas exploratórias permitem descobrir relações emergentes, muitas vezes desconhecidas previamente.

Em domínios como redes sociais, sistemas de recomendação e detecção de fraudes, onde o foco está mais nas conexões do que nas entidades isoladas, os bancos de grafos oferecem desempenho significativamente superior na execução de consultas de múltiplos saltos (*multi-hop queries*) (CHAPPELL, 2019).

A estrutura do modelo de grafos proporciona uma forma intuitiva e expressiva de representar dados conectados, eliminando a rigidez dos esquemas pré-definidos dos bancos relacionais. A adição de novos tipos de nós e relacionamentos pode ser realizada de forma incremental, sem necessidade de migrações estruturais, tornando o modelo altamente adaptável a sistemas em constante evolução (ROBINSON; WEBBER; EIFREM, 2015).

Essa flexibilidade é especialmente útil em domínios com semântica aberta e em cenários com dados heterogêneos, como na Web Semântica, redes de transporte, biologia computacional e cadeias de suprimentos.

Ferramenta de bancos de grafos modernos suportam, de forma nativa, algoritmos de análise topológica como *PageRank*, *Shortest Path*, *Betweenness Centrality* e *Community Detection*, essenciais para identificar padrões, influenciadores e agrupamentos em grafos reais (SUN; LI et al., 2021).

Além disso, linguagens como Cypher e Gremlin permitem expressar consultas com padrões recorrentes e recursivos de forma declarativa, possibilitando a identificação de ciclos, caminhos alternativos, e relações indiretas com clareza e concisão sintática (FRANCIS et al., 2018).

2.4.5 Limitações e Desafios do Modelo de Grafos

Apesar das vantagens expressivas dos bancos de dados orientados a grafos em contextos de alta conectividade, sua adoção ainda apresenta desafios técnicos, operacionais e conceituais. Tais limitações devem ser cuidadosamente consideradas durante a escolha do modelo de dados mais adequado para cada aplicação.

Uma das barreiras mais recorrentes na adoção de bancos de grafos é a curva de aprendizado associada a esse paradigma. Profissionais com forte formação em bancos relacionais podem enfrentar dificuldades na transição para o modelo de grafos, que exige repensar tanto a modelagem conceitual quanto a forma de estruturar consultas (ANGLES; GUTIERREZ, 2012). Linguagens como Cypher ou Gremlin possuem sintaxes e operadores distintos, e muitas vezes carecem de padronização plena, o que pode impactar a interoperabilidade entre ferramentas e fornecedores (FRANCIS et al., 2018).

Embora os bancos de grafos sejam extremamente eficientes em consultas locais, a escalabilidade para grafos de grandes dimensões e altamente distribuídos ainda representa um desafio técnico considerável. A fragmentação eficiente do grafo entre diferentes nós de um cluster — mantendo a performance em travessias entre partições — é uma tarefa complexa e ainda objeto de intensa pesquisa (SUN; LI et al., 2021). Em comparação com modelos chave-valor ou orientados a documentos, que oferecem mecanismos maduros de particionamento e replicação, os bancos de grafos enfrentam limitações em ambientes de alta disponibilidade com baixa latência.

Outra limitação prática está relacionada ao ecossistema de ferramentas. Apesar dos avanços recentes, os bancos de grafos ainda não dispõem do mesmo nível de maturidade e variedade de ferramentas de suporte que os bancos relacionais, especialmente no que se refere a ETL, análise de desempenho e integração com sistemas corporativos legados (ROBINSON; WEBBER; EIFREM, 2015). Além disso, a escassez de profissionais com experiência em modelagem de grafos pode restringir sua adoção em organizações com infraestrutura já consolidada em tecnologias relacionais ou NoSQL tradicionais.

2.5 Comparativo entre Modelos de Banco de Dados

A escolha do modelo de banco de dados mais adequado depende de diversos fatores, como a natureza dos dados, o padrão de acesso, os requisitos de escalabilidade, e o nível de complexidade nas relações entre entidades. Os modelos relacionais, abordagens NoSQL e orientção a grafos apresentam características distintas que influenciam diretamente o desempenho e a flexibilidade de aplicações modernas. Esta seção apresenta uma análise comparativa entre esses modelos, destacando suas diferenças estruturais, linguagens de consulta, desempenho, escalabilidade e adequação a diferentes cenários.

2.5.1 Diferenças Estruturais

O modelo relacional organiza os dados em tabelas com linhas e colunas, estabelecendo relacionamentos por meio de chaves primárias e estrangeiras. Já os bancos NoSQL adotam estruturas mais flexíveis como documentos, pares chave-valor, colunas e grafos, permitindo a modelagem de dados sem a rigidez dos esquemas relacionais (MONIRUZZAMAN; HOSSAIN, 2013).

Nos bancos orientados a grafos, os dados são representados por nós e arestas, permitindo uma modelagem natural de entidades e suas conexões. Essa estrutura torna os bancos de grafos mais expressivos em cenários onde as relações são tão importantes quanto os dados em si (ANGLES; GUTIERREZ, 2008).

2.5.2 Consultas e Linguagens

A linguagem padrão para bancos relacionais é o SQL, amplamente difundida, padronizada e com forte suporte por ferramentas e profissionais. Em contrapartida, os bancos NoSQL não seguem uma linguagem única; cada categoria possui sua própria API ou linguagem de consulta. Os bancos de grafos, por sua vez, adotam linguagens específicas como Cypher (Neo4j), Gremlin (Apache TinkerPop) ou GQL (Graph Query Language), que permitem expressar padrões relacionais de forma mais concisa (FRANCIS et al., 2018).

2.5.3 Desempenho e Escalabilidade

Bancos relacionais oferecem desempenho consistente em transações ACID e consultas com filtros definidos, mas apresentam gargalos em consultas com muitas junções. Bancos NoSQL se destacam em escalabilidade horizontal e desempenho em grandes volumes de dados sem estrutura rígida (POKORNY, 2013).

As transações em bancos de dados relacionais são regidas pelas propriedades conhecidas como ACID, acrônimo para Atomicidade, Consistência, Isolamento e Durabilidade. A atomicidade garante que uma transação seja executada integralmente ou não produza nenhum efeito no banco de dados em caso de falha, assegurando que operações parciais não comprometam o estado do sistema. A consistência assegura que toda transação leva o banco de dados de um estado válido para outro igualmente válido, respeitando todas as restrições de integridade definidas no esquema. O isolamento define que transações executadas simultaneamente não interfiram umas nas outras, fazendo com que o resultado final seja equivalente a uma execução sequencial das operações. Por fim, a durabilidade garante que, após a confirmação de uma transação, seus efeitos sejam permanentemente armazenados, mesmo na ocorrência de falhas no sistema ou desligamentos inesperados. Essas propriedades tornam os bancos de dados relacionais especialmente adequados para sistemas que exigem alto grau de confiabilidade e integridade das informações.

Os Bancos de Dados grafos, embora menos eficientes em operações massivas de agregação, superam os relacionais em consultas que envolvem múltiplas relações entre entidades, graças à persistência nativa das conexões (SUN; LI et al., 2021).

2.5.4 Adequação a Diferentes Cenários

O Modelo relacional continua sendo a escolha ideal para sistemas com dados bem estruturados e lógica transacional forte, como ERPs e sistemas financeiros. Bancos NoSQL são apropriados para aplicações que exigem flexibilidade, disponibilidade e alta escalabilidade, como sistemas de conteúdo, logs e análise em tempo real.

Bancos de dados de grafos se destacam em domínios como redes sociais, sistemas de recomendação, análise de fraudes, IoT e qualquer aplicação onde a relação entre dados seja mais relevante que os dados isoladamente (ROBINSON; WEBBER; EIFREM, 2015).

2.5.5 Considerações sobre Transformação de Modelos

Com a crescente diversidade dos sistemas de informação e a necessidade de interoperabilidade entre aplicações, a transformação entre modelos de banco de dados tem se tornado uma prática estratégica. Converter um modelo relacional para um modelo orientado a grafos exige não apenas mapeamento estrutural, mas também a preservação de semântica, relações e cardinalidades. Essa transição será abordada no Capítulo 3, onde são apresentadas técnicas, algoritmos e decisões de projeto utilizadas para realizar tal conversão.

2.6 Conversão de Dados entre Modelos

A migração de dados entre diferentes modelos de banco, especialmente do relacional para o orientado a grafos, tornou-se uma necessidade cada vez mais comum em contextos que demandam análise de conexões complexas e alta expressividade semântica. Essa conversão, no entanto, não se resume a um simples processo técnico de transposição de dados, mas envolve uma reestruturação conceitual que respeite a semântica original do domínio, preserve relacionamentos e minimize perdas de informação.

Como dito nas seções anteriores, o modelo relacional é baseado em estruturas tabulares e normalização, enfatizando consistência e integridade referencial. Já o modelo de grafos adota uma estrutura mais dinâmica, baseada em nós e arestas com propriedades, projetada para representar de forma explícita relações entre entidades. Assim, a transição entre esses paradigmas requer um entendimento profundo das diferenças estruturais, das implicações de cardinalidade e da semântica implícita nas chaves e tabelas relacionais (ANGLES et al., 2020; PAN et al., 2012).

Nos tópicos seguintes, serão explorados os principais desafios que emergem nesse processo de transformação, bem como as abordagens já existentes na literatura e os mecanismos utilizados para garantir a preservação de significado e da integridade relacional durante a conversão.

2.6.1 Desafios da Conversão entre Modelos

A conversão de um banco de dados relacional para um grafo envolve diversos desafios técnicos e conceituais. Um dos principais entraves reside nas diferenças fundamentais entre os modelos: enquanto o relacional organiza os dados em relações normalizadas, o grafo valoriza a explicitação de relacionamentos e a navegação entre conexões.

Dentre os principais desafios, destacam-se:

- **Diferenças conceituais**: Tabelas, tuplas e chaves precisam ser reinterpretadas como nós, arestas e propriedades. A estrutura relacional não possui uma representação nativa de relacionamento como entidade de primeira classe.
- **Perdas de semântica**: Informações implícitas em tabelas de junção, chaves compostas ou restrições de integridade podem ser perdidas ou mal interpretadas se não forem explicitadas na modelagem em grafo (GIUNCHIGLIA; KHARLAMOV; ZHELEZNYAKOV, 2021).
- **Conversão de relacionamentos complexos**: Relações n-para-n, atributos multivalorados ou entidades dependentes (tabelas fracas) requerem mapeamentos cuidadosos para não distorcer a estrutura conceitual do domínio.
- **Preservação da cardinalidade**: A correta interpretação das chaves primárias e estrangeiras é essencial para garantir que a nova estrutura mantenha a coerência das relações originais entre entidades.
- **Escalabilidade e performance**: Estratégias de transformação devem considerar o custo de geração e carregamento de grafos grandes, além da capacidade de consulta posterior sobre os dados convertidos.

2.6.2 Abordagens Existentes para Conversão

A literatura apresenta diversas propostas para a conversão automática ou semiautomática entre modelos relacionais e grafos. Dentre elas, destaca-se o *ThrusterDB*, um sistema que realiza o mapeamento conceitual do esquema relacional, gera a estrutura de grafo correspondente e otimiza as consultas com base em padrões de acesso e semântica preservada (VICENTE et al., 2019).

Outras abordagens incluem frameworks baseados em *Graph-Relational Mapping* (GRM), nos quais o modelo relacional é analisado para identificar entidades, relacionamentos e hierarquias a fim de gerar a estrutura equivalente em grafos. Trabalhos como os de Alobaid et al. (2020) e Ali et al. (2019) investigam algoritmos de mapeamento estrutural e heurísticas para detecção automática de chaves e relacionamentos implícitos.

Esses sistemas diferem, em geral, quanto à profundidade do mapeamento semântico, à capacidade de lidar com aninhamentos complexos, à customização por parte do usuário e ao grau de automação. Uma característica desejável em abordagens modernas é a preservação da navegabilidade sem comprometer a semântica original do esquema relacional.

2.6.3 Aninhamento, Cardinalidade e Preservação de Semântica

Para garantir uma conversão eficiente e fiel entre modelos, é fundamental considerar o tratamento adequado de aspectos como:

- **Aninhamento:** Refere-se à inclusão de entidades dependentes como subcomponentes de entidades principais, de modo a evitar sua representação como nós independentes no grafo. Em bancos relacionais, isso se manifesta comumente em tabelas cuja chave primária também é uma chave estrangeira para outra tabela — o que indica uma dependência funcional total, típica de relações 1:1. Já no modelo em grafos, essas estruturas podem ser incorporadas como *propriedades aninhadas* simplificando a estrutura e reduzindo a complexidade de navegação.
- **Cardinalidade**: Deve-se garantir que os relacionamentos 1:1, 1:N ou N:M sejam corretamente representados por arestas ou nós auxiliares no grafo. A não observância dessas regras pode levar à perda de precisão semântica ou à criação de estruturas inadequadas para consultas posteriores.
- **Preservação de Semântica**: A semântica do modelo relacional — expressa em restrições, nomes de chaves, regras de integridade e domínios — deve ser preservada ou explicitada na transição. Em muitos casos, é necessário enriquecer o modelo de grafo com metadados ou anotações que contextualizem as conexões criadas.

A correta gestão desses elementos é essencial para que o grafo resultante não apenas reflita a estrutura do banco relacional original, mas também otimize o acesso a dados conectados, garanta integridade e mantenha a fidelidade ao domínio modelado.

2.6.4 Estudos de Estratégias de Conversão: Aninhamento, Cardinalidade e Grau de Relações

Diversas abordagens têm sido propostas na literatura para realizar a conversão entre modelos relacionais e orientados a grafos. Embora compartilhem fundamentos conceituais, essas implementações adotam diferentes estratégias para lidar com aspectos críticos como o mapeamento estrutural, a preservação de cardinalidade e o tratamento de aninhamentos. Esta seção analisa algumas dessas estratégias em profundidade, com foco especial no ThrusterDB e outras propostas contemporâneas.

Extração e Interpretação do Modelo Relacional

O ponto de partida comum às abordagens analisadas é a extração do esquema relacional, o que inclui a identificação de tabelas, chaves primárias, chaves estrangeiras e restrições de integridade. Algumas propostas, como o R2GMapper (ALI et al., 2019), automatizam esse processo a partir do metadado do SGBD, enquanto outras utilizam arquivos de definição externa ou análise semântica do modelo lógico.

No ThrusterDB, por exemplo, a estrutura relacional é extraída diretamente do dicionário de dados da base, com especial atenção para identificar relações n-árias e atributos compostos. Tabelas com chaves compostas ou relacionamentos implícitos são reinterpretadas em uma estrutura intermediária, que permite classificá-las segundo seu papel no grafo final (VICENTE et al., 2019).

As abordagens analisadas convertem tabelas que representam entidades (com chaves primárias simples e atributos descritivos) em nós do grafo. Já as tabelas de junção (com múltiplas chaves estrangeiras e poucos ou nenhum atributo) são geralmente convertidas em arestas.

No ThrusterDB, a definição do que constitui um nó ou uma aresta é feita com base no grau relacional: um parâmetro que quantifica o nível de conectividade da tabela com outras entidades. Tabelas com grau elevado e sem atributos descritivos são candidatas naturais a arestas, enquanto aquelas com menor grau ou significado semântico autônomo tendem a ser representadas como nós.

O tratamento de aninhamento é uma das inovações importantes do ThrusterDB. O sistema introduz o conceito de *nível de aninhamento* para classificar tabelas que devem ser embutidas como propriedades ou subestruturas de nós principais.

Além disso, o grau de relação é utilizado para decidir quando uma tabela intermediária (como uma entidade fraca) deve ser convertida em uma aresta com propriedades ou em um nó independente conectado por múltiplas arestas. Essa decisão é essencial para manter a navegabilidade sem comprometer a semântica relacional.

Para a preservação da cardinalidade e semântica, a correta conversão das cardinalidades é abordada de maneira diferenciada nas propostas existentes. No ThrusterDB, os relacionamentos são enriquecidos com metadados que registram a cardinalidade original, permitindo futuras validações e reconstruções. A cardinalidade influencia diretamente na escolha entre criar múltiplas arestas, uma única aresta ou até mesmo um nó auxiliar para representar associações complexas.

Outras abordagens, como a de Alobaid et al. (2020), propõem heurísticas baseadas em nomes de chaves e análise de colunas para inferir cardinalidades, embora reconheçam a dificuldade de manter precisão em bases com modelagem fraca ou inconsistente.

2.6.5 Outras Estratégias Complementares de Conversão

Além das abordagens analisadas nas seções anteriores, a literatura apresenta outras estratégias relevantes para a conversão de dados relacionais para estruturas de grafos. Embora nem todas sejam voltadas diretamente a bancos de grafos de propriedade, essas propostas contribuem com técnicas, critérios e mecanismos que enriquecem o panorama de soluções disponíveis.

O Ontop é uma ferramenta para mapeamento virtual de grafos RDF sobre bases relacionais, permitindo que consultas SPARQL sejam executadas diretamente sobre dados SQL, sem necessidade de migração física (CALVANESE et al., 2017). É especialmente útil em ambientes onde a conversão precisa ser não intrusiva e orientada a ontologias.

O GraphGen é uma ferramenta voltada à geração automática de grafos a partir de bancos relacionais, com base em inferência heurística de entidades, relacionamentos e atributos (BARBOSA et al., 2015). A proposta permite a descoberta de grafos interessantes de forma iterativa, com suporte a filtros e análise de conectividade.

A abordagem GR2Graph foca na criação de grafos de propriedade com base em um mapeamento semântico explícito, empregando ontologias e transformações parametrizáveis (SONG et al., 2022). A ferramenta permite alto grau de customização e reutilização de relacionamentos, destacando-se pelo suporte a atributos compostos e consultas otimizadas.

O RML (*RDF Mapping Language*) é uma linguagem genérica baseada no padrão RDF para mapeamento de dados relacionais (e outros formatos, como CSV e JSON) em grafos RDF (DIMOU et al., 2014). Embora mais usada na Web Semântica, sua estrutura declarativa de mapeamento e tratamento de aninhamento são relevantes para estratégias de transformação.

2.6.5.1 Critérios de Comparação

Os critérios utilizados para análise das abordagens de conversão entre bancos de dados relacionais e grafos foram definidos com base nas características mais recorrentes da

literatura, bem como nas necessidades práticas de projetos de transformação estrutural de dados. A seguir, detalha-se o significado de cada um deles:

- **Extração automática de esquema**: Refere-se à capacidade da ferramenta de analisar o esquema do banco de dados relacional (tabelas, colunas, chaves primárias e estrangeiras) de forma automatizada, sem depender de entrada manual ou configurações externas. Essa funcionalidade é essencial para escalabilidade e reutilização da ferramenta em diferentes domínios de dados.

- **Tratamento de cardinalidade**: Diz respeito à habilidade da abordagem em identificar e preservar relações entre entidades com diferentes graus de associação, como um-para-um (1:1), um-para-muitos (1:N) e muitos-para-muitos (N:M). Essa preservação é fundamental para manter a semântica original do modelo relacional e garantir integridade lógica no grafo gerado.

- **Aninhamento de entidades**: Representa a capacidade de embutir estruturas de dados secundárias (como tabelas com dependência funcional de outra) dentro de uma entidade principal no grafo. Por exemplo, atributos multivalorados ou entidades fracas podem ser transformados em propriedades compostas ou subestruturas embutidas nos vértices, reduzindo o número de nós e facilitando a consulta.

- **Grau de relação como critério de transformação**: Este critério avalia se a abordagem considera o grau de conectividade das tabelas (isto é, quantas outras tabelas se relacionam com uma determinada tabela) como fator determinante para classificar seu papel no grafo — como vértice, aresta ou atributo. Essa estratégia é comum em abordagens como o ThrusterDB, em que tabelas com alto grau tendem a ser convertidas em vértices.

- **Arestas com propriedades**: Verifica se a ferramenta suporta a criação de arestas que carregam atributos próprios, como datas, pesos, valores ou status de relacionamento. Isso é relevante para representar relações enriquecidas sem inflar o número de nós ou deformar o modelo com estruturas auxiliares.

- **Customização pelo usuário**: Mede o grau de flexibilidade que o sistema oferece ao usuário durante o mapeamento, como definir regras personalizadas, sobrescrever inferências automáticas ou modificar tipos de transformação. Essa capacidade é essencial em cenários onde a semântica do domínio não pode ser inferida apenas pela estrutura do banco.

- **Tipo de grafo gerado**: Classifica o tipo final de grafo produzido pela abordagem. Os principais tipos são:

- *Grafo de propriedade*: Modelo orientado a vértices e arestas com atributos, como o utilizado por Neo4j e JanusGraph - tipo escolhido para essa implementação;
- *Grafo RDF*: Modelo baseado em triplas sujeito-predicado-objeto, típico da Web Semântica;
- *Grafo RDF virtual*: Grafo RDF construído logicamente sobre um banco relacional, sem migração física dos dados.

Essa classificação impacta diretamente nas linguagens de consulta, ferramentas de visualização e casos de uso viáveis para cada abordagem.

As Tabelas 1 e 2 apresentam um resumo das principais estratégias analisadas. A análise dessas estratégias fornece subsídios diretos para o projeto apresentado nesta monografia. A decisão de incorporar conceitos como aninhamento por profundidade, preservação semântica via metadados e classificação de tabelas por grau relacional está fortemente ancorada nas boas práticas identificadas nas abordagens estudadas.

A análise dessas abordagens complementares permite compreender diferentes paradigmas de conversão, inclusive em contextos além dos bancos de grafos de propriedade, e oferece subsídios para decisões de projeto mais embasadas.

Tabela 1 – Comparativo de estratégias de conversão entre modelos relacionais e grafos

Aspecto	**ThrusterDB**	**R2GMapper**	**Ontop**
Extração automática de esquema	Sim	Sim	Sim
Tratamento de cardinalidade	Metadados explícitos	Heurístico	Não
Aninhamento de entidades	Sim (nível 1 e 2)	Não	Parcial
Grau de relação como critério	Sim	Não	Não
Arestas com propriedades	Sim	Sim	Não
Customização pelo usuário	Sim	Baixa	Alta
Tipo de grafo gerado	Propriedade	Propriedade	RDF Virtual

Tabela 2 – Comparativo de abordagens complementares de conversão relacional para grafos

Aspecto	**GraphGen**	**GR2Graph**	**RML**
Extração automática de esquema	Sim	Sim	Parcial
Tratamento de cardinalidade	Heurístico	Explícito	Não
Aninhamento de entidades	Parcial	Sim	Sim
Grau de relação como critério	Não	Parcial	Não
Arestas com propriedades	Sim	Sim	Não
Customização pelo usuário	Média	Alta	Alta
Tipo de grafo gerado	Propriedade	Propriedade	RDF

3 Desenvolvimento

Este capítulo descreve o algoritmo proposto para a solução do mapeamento e conversão de um banco de dados no modelo relacional, para o modelo orientado a grafos. Será explicítado a arquitetura pensada para a solução desde a extração dos metadados relacional até a exportação para o modelo em grafos.

3.1 Arquitetura Geral da Solução

A arquitetura do sistema foi organizada em etapas modulares que abrangem desde a extração de metadados do banco relacional até a geração dos elementos que compõem o grafo: nós e arestas. O sistema foi desenvolvido em linguagem Python, devido à sua expressividade e grande disponibilidade de bibliotecas para manipulação de dados e conexão com bancos relacionais, como `pyodbc` e `sqlalchemy`.

A Figura 6 é a pipeline que representa o fluxo do sistema, ou seja, a implementação deste projeto:

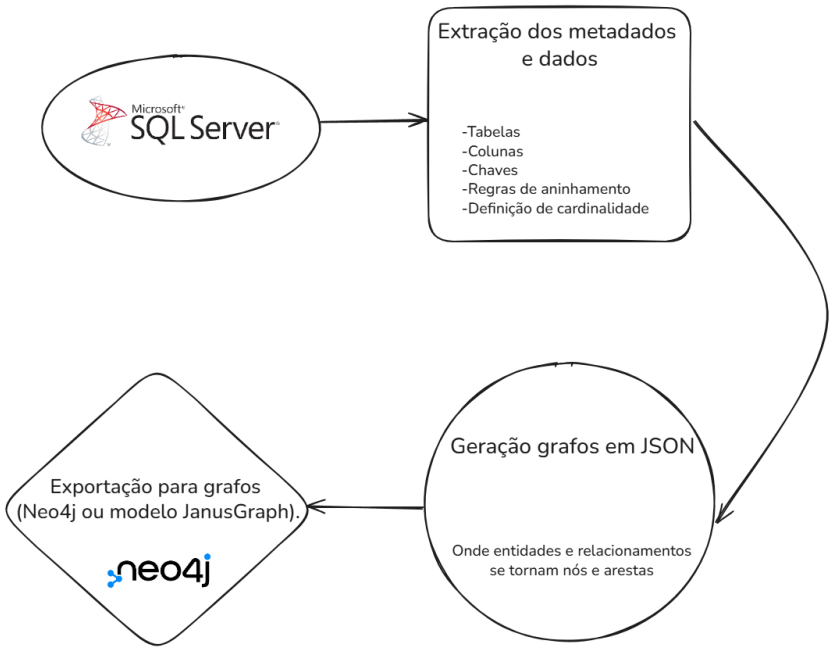


Figura 6 – Fluxo geral da arquitetura da solução

O processo inicia-se com a leitura da estrutura do banco de dados, no qual são identificados os esquemas, tabelas, colunas, chaves primárias e estrangeiras. Em seguida, aplica-se uma lógica de aninhamento para reorganizar as relações de acordo com a proximi-

dade estrutural entre as tabelas — especialmente em cenários com grau de relacionamento baixo ou relações auxiliares.

Posteriormente, o sistema realiza a coleta dos registros propriamente ditos no banco relacional, incorporando os dados aninhados aonde for apropriado. A etapa final consiste na geração de nós e arestas do grafo de forma genérica, respeitando as estruturas e chaves previamente mapeadas.

Essa abordagem modular permite flexibilidade na extensão e manutenção do sistema, além de favorecer a adaptação a diferentes bancos relacionais e estruturas de dados. A transformação é feita de forma a preservar a semântica dos dados, algo essencial para aplicações em grafos, como análise de redes, mineração de relacionamentos e exploração de conexões latentes.

A arquitetura proposta se inspira em trabalhos prévios, como o ThrusterDB (VICENTE et al., 2019) e o R2GMapper (ALI et al., 2019), mas propõe inovações na forma de parametrização, na incorporação automática de relações aninhadas e na generalização da geração de elementos do grafo.

3.2 Extração de Metadados do Banco Relacional

O primeiro passo para a transformação de dados relacionais em grafos consiste na extração dos metadados do banco de dados de origem. Esses metadados incluem informações estruturais sobre as tabelas, colunas, chaves primárias, chaves estrangeiras, tipos de dados e esquemas presentes no banco. A recuperação automatizada dessas informações é essencial para permitir a conversão sem a necessidade de intervenção manual, garantindo escalabilidade e aplicabilidade a diferentes domínios.

Nesta etapa, foi utilizado o SQLAlchemy, uma biblioteca Python amplamente adotada para abstração e manipulação de bancos de dados relacionais. Por meio dela é possível recuperar a estrutura completa do banco de forma programática. A função central `refletir_todos_os_schemas` percorre todos os esquemas não pertencentes ao sistema, coletando as definições de tabelas e seus vínculos.

Uma vez refletido o banco, os metadados são convertidos para uma estrutura JSON padronizada, contendo campos como `nome da tabela`, `esquema`, `colunas`, `chave primaria` e `chaves estrangeiras`. Cada coluna possui seus atributos como nome, tipo e nulabilidade. As chaves estrangeiras também são armazenadas com os respectivos relacionamentos, apontando para a tabela e coluna referenciada. O formato desse primeiro *JSON* é a dada na Figura 7 e é dado um exemplo na Figura 8

```
{
    "nome_tabela": Nome da tabela (string),
    "esquema": Esquema da tabela (string),
    "chave_primaria": [
        Nome da 1° chave primária (string),
        Nome da 2° chave primária (string),
        ...
    ],
    "colunas": [
        {
            "nome": Nome da coluna (string),
            "tipo": Tipo da coluna (string),
            "nullable": Aceita valores nulos (boolean)
        }
    ],
    "chaves_estrangeiras": [
        {
            "nome_fk": Nome da constraint da FK (string),
            "coluna": Nome da coluna (string),
            "target": Caminho da referencia - Esquema.tabela.coluna (string),
            "tabela_referenciada": Nome da tabela referenciado da FK (string),
            "schema_referenciado": Nome do esquema referenciado da FK (string),
            "coluna_referenciada": Nome da coluna referenciado da FK (string)
        }
    ]
}
```

Figura 7 – Modelo do JSON da estrutura da tabela

Essa padronização dos metadados permite o processamento posterior das tabelas de forma genérica, independentemente da modelagem do banco original. Além disso, a separação clara entre colunas comuns e colunas que representam relacionamentos é um fator-chave para os passos seguintes do processo de mapeamento e transformação.

```
{
    "nome_tabela": "Materia",
    "esquema": "Universidade",
    "chave_primaria": [
        "codigo"
    ],
    "colunas": [
        {
            "nome": "codigo",
            "tipo": "string",
            "nullable": false
        },
        {
            "nome": "nome",
            "tipo": "string",
            "nullable": false
        },
        {
            "nome": "id_departamento",
            "tipo": "int",
            "nullable": false
        }
    ],
    "chaves_estrangeiras": [
        {
            "nome_fk": "FK_departamento_id",
            "coluna": "id_departamento",
            "target": Caminho da referencia - Universidade.Departamento.id_departamento,
            "tabela_referenciada": "Departamento",
            "schema_referenciado": "Universidade",
            "coluna_referenciada": "id_departamento"
        }
    ]
}
```


Figura 8 – Exemplo do JSON da estrutura para a tabela Materia

3.2.1 Critérios para Aninhamento

Uma etapa importante na conversão de dados relacionais para o modelo de grafos consiste na identificação de tabelas que podem ser incorporadas como estruturas aninhadas (*nested structures*) em nós principais, em vez de se tornarem nós independentes. Essa estratégia reduz o número de entidades explícitas no grafo e melhora a navegabilidade, especialmente em cenários no qual tabelas possuem baixa relevância semântica de forma isolada.

No sistema desenvolvido, o processo de aninhamento é realizado de forma automática a partir da análise dos metadados das tabelas extraídas. Cada tabela recebe uma classificação no campo `aninhamento`, sendo:

- `1` – quando a tabela deve ser incorporada como estrutura aninhada no nó principal que a referencia;
- `0` – quando a tabela deve ser mantida como nó independente no grafo.

Os critérios utilizados para definir se uma tabela será aninhada foram estabelecidos com base em características estruturais do modelo relacional:

1. **Chave primária simples**: tabelas com apenas uma coluna como chave primária são candidatas a aninhamento, pois geralmente representam entidades dependentes ou atributos multivalorados.

2. **Poucos atributos próprios**: tabelas com número reduzido de colunas (até duas, neste projeto) que não são chaves primárias ou estrangeiras, indicam baixa granularidade de informação e, portanto, podem ser incorporadas ao nó que as referencia.

3. **Baixa referência por outras tabelas**: se uma tabela é referenciada por no máximo uma outra tabela, seu impacto estrutural no grafo é reduzido, favorecendo a inclusão como subestrutura.

Além disso, durante o processo de aninhamento, cada tabela recebe um campo adicional denominado `grau_relacao`, que representa o número de chaves estrangeiras que ela possui em direção a outras tabelas. Esse campo auxilia na compreensão da conectividade da tabela e é utilizado em etapas posteriores para determinar a ordem de geração de nós e arestas.

A Figura 9 apresenta um exemplo simplificado do processo de aninhamento: uma tabela *Telefone*, contendo poucos atributos e referenciando exclusivamente *Cliente*, é incorporada como subestrutura dentro do nó *Cliente* no grafo resultante.

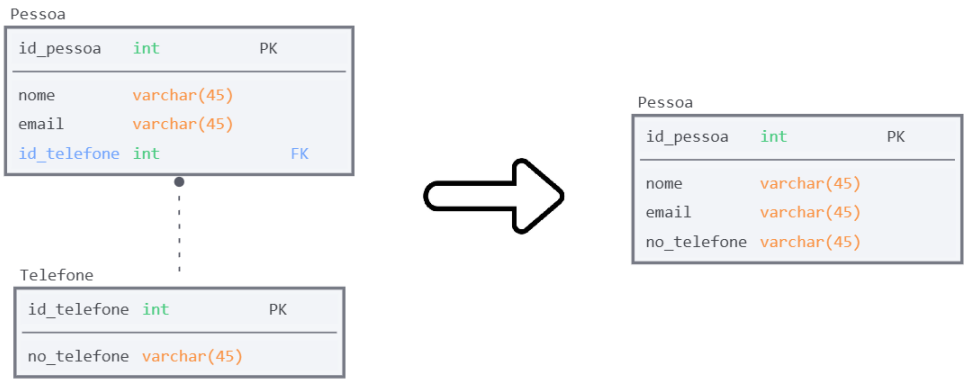


Figura 9 – Exemplo de aninhamento de tabela dependente em nó principal.

Segundo Vicente et al. (2019), a aplicação de aninhamento em processos de transformação de dados relacionais para grafos permite reduzir a fragmentação estrutural, otimizando consultas que envolvem relações de dependência direta. Esse tipo de abordagem também é citado em frameworks como o ThrusterDB, que utiliza critérios semelhantes para classificar entidades fracas e incorporá-las em nós principais.

3.2.2 Cardinalidade das Relações e Classificação de Tabelas

A identificação correta da **cardinalidade das relações** entre entidades é essencial para transformar um modelo relacional em um modelo orientado a grafos, onde os relacionamentos podem representar conexões do tipo *um-para-um* (1:1), *um-para-muitos* (1:N) ou *muitos-para-muitos* (N:N). A definição dessas cardinalidades no banco de dados relacional é derivada das *foreign keys* (chaves estrangeiras), juntamente com restrições adicionais como chaves primárias e índices únicos.

Neste trabalho, a cardinalidade de uma relação é inferida a partir da seguinte lógica:

- Relações **1:1** ocorrem quando um registro em uma tabela A está associada, no máximo, a uma única linha em uma tabela B, e vice-versa. Em termos práticos, isso significa que a chave estrangeira em uma das tabelas também é uma chave primária (ou possui uma restrição de unicidade), garantindo que não haja duplicidade de associações.
- Relações **1:N** são assumidas como padrão quando a chave estrangeira não é única nem chave primária.
- Relações **N:N** são inferidas por meio de tabelas intermediárias que contêm exclusivamente (ou majoritariamente) chaves estrangeiras para duas outras tabelas distintas, na qual essas colunas em conjunto formam a chave primária ou compõem todas as colunas da tabela.

Esse tipo de inferência é alinhado com as práticas descritas por Silberschatz, Korth e Sudarshan (2006), onde a cardinalidade de relacionamentos pode ser deduzida por restrições impostas ao esquema relacional. A identificação automática dessas relações permite que a estrutura de grafos preserve a semântica original do modelo relacional, mantendo a integridade dos vínculos.

Para continuação desse projeto é interessante que nesse ponto da conversão o usuário possa intervir e alterar as cardinalidades manualmente para melhor conversão de tabelas, utilizando o contexto que usuário possuí sobre a tabela.

Além disso, a análise da estrutura da tabela permite classificá-la em dois tipos principais:

- **Tabelas normais:** Representam entidades com atributos próprios e, em geral, possuem uma chave primária definida, com possíveis chaves estrangeiras.
- **Tabelas de associação (N:N):** São estruturas intermediárias cuja principal função é representar relacionamentos muitos-para-muitos. Essas tabelas geralmente não

possuem atributos próprios além das chaves estrangeiras que referenciam duas outras tabelas, e sua chave primária é composta justamente por essas chaves estrangeiras (*FKs*).

A implementação desenvolvida percorre cada tabela extraída do metadado do banco de dados relacional e aplica a lógica de inferência de cardinalidade para cada chave estrangeira encontrada. A Figura 10 demonstra uma tabela (que será um nó) que tem um relacionamento 1:1, pois a *foreign key* é a primary key, logo única.


```
{
    "nome_tabela": "Employee",
    "esquema": "HumanResources",
    "chave_primaria": [
        "BusinessEntityID"
    ],
    "colunas": [
        {
            "nome": "BusinessEntityID",
            "tipo": "int",
            "nullable": false
        },
        {
            "nome": "NationalIDNumber",
            "tipo": "string",
            "nullable": false
        },
        {
            "nome": "LoginID",
            "tipo": "string",
            "nullable": false
        },
        {...}
        {
            "nome": "rowguid",
            "tipo": "UNIQUEIDENTIFIER",
            "nullable": false
        },
        {
            "nome": "ModifiedDate",
            "tipo": "datetime",
            "nullable": false
        }
    ],
    "chaves_estrangeiras": [
        {
            "nome_fk": "FK_Employee_Person_BusinessEntityID",
            "coluna_local": "BusinessEntityID",
            "tabela_referenciada": "Person",
            "schema_referenciado": "Person",
            "coluna_referenciada": "BusinessEntityID",
            "cardinalidade": "1:1"
        }
    ],
    "colunas_unicas": [
        [
            "LoginID"
        ],
        [
            "rowguid"
        ],
        [
            "NationalIDNumber"
        ]
    ],
    "tipo": "Tabela Normal"
}
```


Figura 10 – Exemplo de aplicação da cardinalidade.

Essa inferência é essencial para uma correta tradução entre os modelos, pois a forma

como as relações são expressas no grafo depende diretamente da sua cardinalidade. Por exemplo, relações 1:1 e 1:N são representadas como arestas direcionais simples, enquanto relações N:N geralmente exigem a modelagem de uma aresta com propriedades adicionais, ou até a representação da tabela de associação como um vértice adicional.

Angles e Gutierrez (2008) e Vicente et al. (2019) destacam que a correta identificação dessas relações é um fator determinante para garantir a coerência da estrutura de grafos gerada, especialmente em aplicações de análise de dados, onde a semântica das conexões entre entidades é crítica. Como todo o algoritmo é genérico, essas informações são utilizadas para a melhor tentativa de conversão sem a necessidade do contexto da arquitetura de dados.

Dessa forma, a lógica implementada automatiza não apenas a identificação de atributos e chaves, mas também a classificação estrutural e relacional de cada tabela, o que fornece subsídios fundamentais para as próximas etapas da conversão para grafos.

3.3 Coleta dos Dados no Banco Relacional

A extração dos dados do banco relacional é uma etapa essencial no processo de conversão para o modelo orientado a grafos. Essa fase visa não apenas recuperar os registros de cada tabela, mas também estruturá-los de forma que reflitam corretamente os vínculos lógicos estabelecidos pelas chaves estrangeiras e pelos níveis de aninhamento. Nesta seção, são apresentadas as estratégias adotadas para coletar os dados diretamente do banco, armazená-los em formato JSON e realizar o aninhamento automático com base nas dependências definidas no esquema relacional.

3.3.1 Extração dos Registros por Tabela

O algoritmo implementado contém uma função responsável por realizar consultas SQL diretamente ao banco relacional, utilizando os metadados extraídos previamente. Essa operação é feita individualmente para cada tabela, considerando o esquema e as colunas relevantes. O resultado da consulta é processado e transformado em uma lista de dicionários Python, onde cada dicionário representa uma linha da tabela, com os nomes das colunas como chaves.

Esse processo é aplicado a todas as tabelas do banco de dados, retornando uma estrutura padronizada contendo os dados organizados por tabela.

3.3.2 Estruturação dos Dados em JSON

A estrutura escolhida para representar os dados extraídos é o formato `JSON` (JavaScript Object Notation), que permite uma representação hierárquica e de fácil manipulação,

adequada para a posterior transformação em grafos.

Cada tabela é convertida em um objeto contendo seu nome, o esquema correspondente e a lista de registros. Essa estruturação é importante para manter a rastreabilidade entre os dados extraídos e sua origem relacional, além de facilitar o mapeamento para os vértices e arestas do modelo de grafos.

3.3.3 Aninhamento dos Dados com Base no Modelo Conceitual

Além da extração simples dos dados, o algoritmo já realiza automaticamente o aninhamento das tabelas que possuem essa característica definida. Essa rotina consiste em incorporar, nos registros de uma tabela *pai*, os dados de uma tabela referenciada por chave estrangeira e marcada com `aninhamento = 1` nos metadados.

Diferentemente do que ocorre apenas em relações do tipo 1:1, o aninhamento não está restrito a aspectos de cardinalidade. Ele é orientado pela análise conceitual do domínio dos dados, buscando evitar a criação de nós desnecessários no grafo quando uma entidade pode ser representada como subestrutura de outra. Tabelas como endereços, contatos, ou configurações, por exemplo, são candidatas comuns a esse tipo de incorporação.

O processo de aninhamento segue as seguintes etapas:

1. Extração dos dados de todas as tabelas;
2. Identificação das tabelas com `aninhamento = 1`;
3. Localização da tabela-pai que referencia a tabela aninhável;
4. Criação de um dicionário de `lookup` com base na chave primária da tabela aninhável;
5. Incorporação dos dados da tabela aninhada nos registros da tabela-pai;
6. Remoção das tabelas aninhadas do conjunto final de resultados.

Esse mecanismo permite gerar uma estrutura de dados mais concisa e representativa, evitando que entidades auxiliares ocupem o papel de nós independentes no grafo. Dessa forma, as informações são incorporadas a um nó mais relevante do ponto de vista semântico e relacional, o que contribui para a simplicidade da estrutura final do grafo.

3.4 Conversão de Modelos

Uma vez coletados e organizados os dados provenientes do banco relacional, a próxima etapa consiste na conversão desses dados para uma estrutura intermediária de grafo. Essa estrutura representa os dados como **nós** e **arestas**, abstraindo os registros

relacionais em uma forma que possa ser compreendida por modelos orientados a grafos. Nesta seção, descrevemos a abordagem utilizada para realizar essa conversão de maneira genérica e flexível, com base nos metadados extraídos e nos dados aninhados das tabelas.

3.4.1 Geração dos Nós

A criação dos nós é feita exclusivamente para tabelas que não são de associação (ou seja, cujo tipo não é `N:N`) e que não estão marcadas com aninhamento. Cada registro da tabela origina um nó no grafo, contendo suas propriedades e um identificador único. Esse identificador é construído a partir do nome da tabela e do valor da chave primária do registro.

Além disso, os dados passam por um processo de normalização, no qual são tratados tipos como `Decimal`, `bytes`, `datetime`, valores nulos, entre outros, para garantir compatibilidade e padronização na representação do grafo, visto que esses tipos podem gerar problema para a inserção direta no *Neo4j*.

Cada nó gerado contém as seguintes informações:

- **id:** identificador único, formado por `<tabela>:<valor da chave primária>`;
- **type:** nome da tabela original;
- **scheme:** esquema da tabela;
- **properties:** dicionário com os atributos do registro.

3.4.2 Geração das Arestas

As conexões entre os nós são representadas por arestas. Existem dois casos distintos para sua geração:

3.4.2.1 Arestas a partir de Tabelas de Associação (N:N)

Tabelas do tipo `N:N` geralmente representam relacionamentos muitos-para-muitos. Quando identificadas, essas tabelas, no algoritmo apresentado nesse projeto, não geram nós próprios; em vez disso, cada registro gera uma aresta conectando dois nós principais, conforme as chaves estrangeiras presentes na tabela.

Uma ideia de implementação é permitir que o usuário defina as tabelas que ela quer como nós ou arestas, ideias como essa serão melhor abordadas no Capítulo 5, de conclusão.

Essas arestas podem conter propriedades adicionais oriundas das colunas da tabela de associação que não fazem parte das chaves estrangeiras, permitindo enriquecer o relacionamento no grafo com atributos relevantes.

3.4.2.2 Arestas a partir de Chaves Estrangeiras

Para tabelas normais que possuem chaves estrangeiras, é criada uma aresta para cada vínculo entre o registro atual (origem) e o registro referenciado (destino). A relação é inferida a partir das correspondências entre chaves primárias e estrangeiras.

As arestas são nomeadas de acordo com o padrão `<tabela de origem>_TO_<tabela de destino>` e não possuem propriedades adicionais, salvo no caso das tabelas `N:N`, conforme descrito anteriormente.

Cada aresta gerada contém as seguintes informações:

- **type:** nome identificador da relação, formado por `<tabela de origem>_TO_<tabela de destino>`;
- **scheme:** nome do esquema da relação;
- **from:** nome da tabela origem;
- **to:** nome da tabela destino;
- **properties:** dicionário com os atributos do registro, caso relação N:N.

3.4.3 Estrutura Intermediária do Grafo

Ao final do processo, é gerada uma estrutura intermediária contendo dois conjuntos principais:

- **nodes:** lista de todos os nós gerados;
- **relationships:** lista de todas as arestas criadas.

Essa estrutura serve como base para as etapas subsequentes, onde o grafo poderá ser exportado para bancos como *Neo4j* ou transformado para o padrão do *JanusGraph*, conforme abordado nas próximas seções.

3.5 Exportação para o Modelo de Grafo

Após a conversão dos dados relacionais em uma estrutura intermediária de grafo contendo nós e relacionamentos, o sistema implementado oferece duas formas de exportação:

uma para um modelo genérico compatível com o banco de grafos *JanusGraph*, e outra para a inserção direta em um banco *Neo4j*, utilizando comandos Cypher.

3.5.1 Geração do Modelo para JanusGraph

Embora o JanusGraph não possua uma linguagem própria de inserção como o Cypher do Neo4j, ele exige que os dados estejam estruturados de forma adequada para a importação via APIs ou ferramentas auxiliares (como TinkerPop ou Gremlin Server).

No algoritmo proposto por este projeto, há uma função que é responsável por transformar os nós gerados anteriormente em um formato simples que contém:

- `label`: o tipo da entidade, correspondente ao nome da tabela original;
- `properties`: as propriedades do nó, como um dicionário chave-valor.

Da mesma forma, a função `gerar_edges_janus` trata os relacionamentos gerados, criando estruturas com os seguintes campos:

- `from`: identificador do nó de origem;
- `to`: identificador do nó de destino;
- `label`: tipo do relacionamento;
- `properties`: propriedades associadas à aresta, se houver.

O resultado dessas funções é uma estrutura JSON compatível com ferramentas auxiliares de inserção para o JanusGraph, que podem utilizar essas informações para popular a base de grafos.

3.5.2 Exportação Direta para Neo4j

O sistema também permite a exportação direta da estrutura de grafo para o banco *Neo4j*, utilizando a linguagem Cypher. Para isso, são utilizados métodos que operam em **lotes**, tanto para a criação de nós quanto de relacionamentos. Isso garante eficiência e evita estouros de memória ou timeouts durante a escrita no banco.

Antes da inserção de nós e arestas, foi utilizada a tática de criação de índices para que as querys fossem mais otimizadas. Em testes anteriores à criação de índices foram obtidos péssimos resultados de tempo para a criação de nós e arestas com bases com registros acima de 100 mil.

Listing 3.1 – Criação de índices no Neo4j

```
CREATE INDEX node_id_index IF NOT EXISTS FOR (n:Node) ON (n.id)
```

A inserção dos nós é realizada por tipo, utilizando o comando `MERGE` para evitar duplicações:

Listing 3.2 – Inserção de nós no Neo4j em lotes

```
MERGE (n:Node:{safe_type} {id: node.id})
SET n += node.properties
```

Caso um erro ocorra durante a execução em lote, o sistema tenta inserir os nós individualmente, como forma de garantir resiliência.

A criação dos relacionamentos segue lógica similar, conectando os nós com base nos identificadores únicos gerados anteriormente:

Listing 3.3 – Inserção de relacionamentos no Neo4j

```
MATCH (from:Node {id: rel.from}), (to:Node {id: rel.to})
MERGE (from)-[r:{safe_rel_type}]->(to)
```

Essa estratégia garante uma exportação eficiente e segura dos dados convertidos para o banco Neo4j, mantendo a estrutura semântica de nós e arestas definida anteriormente. Além de tratar erros na inserção em lote e lidar com bancos com muitos registros.

3.6 Desafios Encontrados e Decisões de Implementação

Durante o processo de implementação da conversão de dados relacionais para o modelo orientado a grafos, diversos desafios técnicos e conceituais foram identificados. Esta seção apresenta os principais obstáculos enfrentados ao longo do desenvolvimento, bem como as decisões tomadas para solucioná-los, com base em critérios de coerência estrutural, viabilidade técnica e alinhamento com o modelo de domínio.

3.6.1 Definição de Tabelas Aninháveis

Uma das primeiras dificuldades enfrentadas foi a identificação de tabelas que poderiam ser aninhadas dentro de outras. Embora relações do tipo 1:1 pareçam inicialmente candidatas naturais ao aninhamento, essa decisão vai além da cardinalidade: envolve uma análise conceitual da função da tabela dentro do domínio da aplicação.

Para isso, optou-se por enriquecer os metadados das tabelas com um campo adicional denominado `aninhamento`, que assume valores como:

- `0` – Tabela principal (gera nó próprio);

- `1` – Tabela aninhável (será incorporada a outro nó);

Definir manualmente esse campo para cada tabela exigiu uma etapa de análise estrutural cuidadosa, além de testes empíricos com diferentes esquemas de banco, não existe uma forma errada ou certa para aninhar uma tabela, é uma escolha a depender do resultado que se quer obter, para esse projeto o aninhamento foi usado para reduzir tabelas que são desnecessárias na sua composição (por ter poucos dados) e não alterar muito o desenho base do banco de dados original. Essa abordagem permitiu maior controle sobre a conversão, mas também impôs uma responsabilidade adicional de validação por parte do usuário da ferramenta.

3.6.2 Aninhamento de Valores em Estruturas Já Extraídas

Outro desafio relevante foi o aninhamento dos dados já extraídos do banco relacional. Após a coleta dos registros em formato JSON, tornou-se necessário reorganizar os dados para incorporar as entidades aninháveis dentro das entidades principais que as referenciavam. Para isso, foi preciso:

- Localizar, por meio das chaves estrangeiras, os vínculos entre entidades principais e aninhadas;
- Utilizar dicionários e outras estruturas de dados para organizar as tabelas;
- Atualizar os registros principais com novos campos correspondentes às tabelas aninhadas;
- Remover as tabelas aninhadas da saída final.

Essa lógica, embora funcional, foi sensível a problemas como chaves compostas, ausência de dados, e duplicidade de referências. A decisão de restringir o suporte inicial a chaves primárias simples para o aninhamento foi uma simplificação proposital, a ser revisada em versões futuras.

3.6.3 Chaves compostas

Como dito na Subseção 3.6.2 as chaves compostas (*PK's* e *FK's*) não são tratadas definitivamente pela sua própria característica, elas são tratadas pelo conjunto a ser tratado e podem existir falhas (*gaps*) a depender do modelo de dados que utiliza dessas aplicações (não foi possível simular, mas é caso que não é possível afirmar nesse estudo). A complexidade de implementação de todas as etapas poderiam ser acrescidas, se comparada ao modelo atual do código, caso fossem tratados as chaves compostas de maneira específica.

3.6.4 Diferenciação entre Nós e Arestas

Uma decisão importante no processo de conversão envolveu a distinção entre quais estruturas do banco relacional deveriam ser representadas como nós e quais se encaixariam melhor como arestas no grafo. Embora no modelo relacional toda tabela contenha dados estruturados e relacionáveis, nem todas representam, de fato, entidades no sentido conceitual. Muitas vezes, existem tabelas auxiliares, associativas (para representar relações N:N) ou dependentes, cuja existência está atrelada à combinação de outras entidades principais.

No modelo de grafos, manter essa distinção é fundamental para evitar a criação excessiva de nós intermediários, que podem introduzir ruído visual e dificultar a execução de consultas. Por isso, o sistema proposto adota critérios como grau de relacionamento e dependência funcional para classificar automaticamente se uma tabela deve originar um nó ou ser incorporada como uma aresta com propriedades.

A lógica implementada seguiu os seguintes critérios:

- Tabelas do tipo `N:N` são tratadas como arestas, representando relacionamentos compostos entre dois nós principais;
- Tabelas com função meramente associativa ou descritiva são candidatas ao aninhamento;
- Apenas tabelas com aninhamento `0` e tipo diferente de `N:N` geram nós próprios.

Essa abordagem resultou em uma estrutura de grafo mais concisa e semanticamente expressiva, porém exigiu várias iterações para ser refinada, especialmente em bancos com esquemas complexos.

3.6.5 Tratamento de Tipos de Dados

Durante a normalização dos dados para o grafo, foi necessário tratar diferentes tipos de valores oriundos do banco relacional, como `Decimal`, `bytes`, `NULLs`, datas e strings. Isso foi feito por meio de uma função que converte os tipos para formatos compatíveis com o modelo JSON e, posteriormente, com a engine de grafos *Neo4j*.

Uma decisão importante foi manter os valores nulos como a string `"None"` em vez de `null`, evitando assim inconsistências em ferramentas que não tratam bem valores nulos em grafos.

3.6.6 Exportação em Lotes para Neo4j

Ao definir a exportação para o banco Neo4j, optou-se por um processo de inserção em lotes, agrupando os nós por tipo e os relacionamentos por label. Essa decisão foi tomada por motivos de performance e controle de erros, especialmente em bancos de dados com grande volume de registros.

A estratégia de *fallback* em caso de falha no lote, tentando inserir os registros individualmente, permitiu maior resiliência na carga dos dados.

3.6.7 Compatibilidade com Múltiplos Modelos de Grafo

A decisão de gerar uma estrutura genérica de grafo com nós e arestas separadas, e só posteriormente convertê-la para formatos específicos (como JanusGraph ou Cypher), permitiu maior modularidade e reuso do código. Com isso, a mesma base de dados pode ser adaptada para múltiplos sistemas de armazenamento de grafos com pequenas modificações.

Todas essas decisões, embora nem sempre ideais para todos os contextos, permitiram a construção de uma ferramenta genérica, flexível e extensível, capaz de atender diferentes cenários de conversão relacional-para-grafo.

4 Resultados

Este capítulo apresenta os resultados obtidos com a aplicação do sistema de conversão de bancos de dados relacionais para o modelo orientado a grafos. Foram utilizados dois bancos de dados amplamente conhecidos: *AdventureWorks*[1], para testes de desempenho e escalabilidade, e *Northwind*[2], para análise qualitativa e comparativa entre os modelos relacional e de grafos. Ambos os bancos a serem testados, são de fornecimento da *Microsoft*.

Os testes foram realizados localmente, utilizando um ambiente configurado com o banco SQL Server como fonte relacional e Neo4j como destino grafo. A base intermediária gerada também foi exportada para JSON e, opcionalmente, adaptada ao modelo do JanusGraph.

O algoritmo implementado está presente no github do Autor[3].

4.1 Ambiente de Execução dos Testes

Todos os testes foram realizados diretamente da maquina do Autor e as especificações do ambiente são:

- **Sistema Operacional:** Windows 10
- **Processador:** AMD Ryzen 5 5600 3.50 GHz
- **Memória RAM:** 16 GB
- **Banco relacional:** SQL Server 2022 (local)
- **Banco grafo:** Neo4j Community 5.24 (local)
- **Linguagem de implementação:** Python 3.9.13
- **Bibliotecas principais:** SQLAlchemy, pyodbc, neo4j-driver

4.2 Resultados com a Base AdventureWorks

A base *AdventureWorks* apresenta uma estrutura complexa e realista, projetada para simular os processos de uma empresa de médio porte do setor de manufatura, distribuição

[1] AdventureWorks: https://learn.microsoft.com/pt-br/sql/samples/adventureworks-install-configure

[2] Northwnd: https://learn.microsoft.com/pt-br/power-apps/maker/canvas-apps/northwind-install

[3] https://github.com/pedrevangelista/RelationalToGraph

e vendas de bicicletas e equipamentos esportivos. Ao contrário de bases mais simples como a *Northwind*, o *AdventureWorks* foi desenvolvido com foco em testes avançados de modelagem, consultas e integração com soluções empresariais completas.

Sua modelagem é altamente normalizada e abrange múltiplos domínios organizacionais, como produção, vendas, recursos humanos, logística, compras, finanças e gerenciamento de pessoas. As tabelas estão distribuídas em diferentes esquemas lógicos, como `Person`, `Production`, `Sales`, `HumanResources`, `Purchasing` e `dbo`, o que favorece a organização modular do banco.

O banco faz uso intensivo de:

- **Chaves compostas**: várias tabelas possuem chaves primárias formadas por múltiplas colunas, exigindo atenção especial no mapeamento para grafos, onde identificadores únicos por nó são requeridos;
- **Relacionamentos complexos**: existem diversas relações muitos-para-muitos (`N:N`) intermediadas por tabelas de associação com atributos próprios;
- **Hierarquias e relacionamentos internos**: como as hierarquias de funcionários ou produtos, que exigem lógica recursiva em consultas no modelo relacional;
- **Dados com tipos complexos**: as colunas das tabelas possuem tipos diversos que necessitaram de tratamento especial;
- **Separação de responsabilidades entre entidades**: por exemplo, uma pessoa física pode assumir diferentes papéis no sistema (cliente, funcionário ou fornecedor), o que é modelado por meio de tabelas especializadas com relacionamentos a uma entidade comum.

Essa complexidade faz da base AdventureWorks um excelente caso de estudo para avaliar a robustez do processo de conversão. Sua estrutura proporciona cenários ricos para testar aninhamentos, cardinalidades diversas, e estratégias de transformação entre os modelos de dados, além de ser frequentemente utilizada por profissionais de dados e desenvolvedores para simulações realistas de ETL, BI e migração de modelos relacionais para NoSQL.

4.2.1 Características Estruturais

Uma das etapas do algoritmo proposto é a extração de metadados e os dados do banco no modelo relacional, nesta etapa foram obtidas as seguintes informações do banco:

- **Total de tabelas:** 71

- **Tabelas com aninhamento = 0:** Zero. Não houveram tabelas a serem aninhadas
- **Tabelas N:N:** 11 (relacionamentos compostos)
- **Total de relacionamentos (FKs):** 90
- **Total de chaves primárias (PKs):** 102
- **Total de registros encontrados:** 760.837

4.2.2 Resultados de Conversão

Na execução do algoritmo foram utilizadas funções para calcular o tempo de execução de cada etapa do processo, desde a extração dos metadados do banco relacional, até a exportação dos dados para o ***Neo4j*** no modelo em grafos.

Tabela 3 – Tempo de execução do algoritmo do banco AdventureWorks2022

Etapa	**Tempo médio (s)**
Extração de metadados	2.961
Verificar aninhamento de tabelas	0.019
Reescrever estrutura aninhada	0.008
Extração de dados do banco	8.034
Transformação para grafos em JSON	8.549
Exportação formato Janus em JSON	2.889
Exportação para o Neo4j	128.373

- **Nós gerados:** 570.757
- **Arestas geradas:** 867.252

4.2.3 Análise Qualitativa

Ao utilizar a base *AdventureWorks2022*, observou-se uma elevada complexidade tanto na estrutura dos dados quanto na realização de consultas no modelo de grafos. Essa dificuldade se deve, em grande parte, à baixa curva de aprendizado associada a bancos de dados desse porte e à relativa inexperiência de usuários iniciantes com tecnologias orientadas a grafos. Além disso, a presença de diversas tabelas com chaves primárias compostas e outras caracteristícas não mapeadas, impôs desafios significativos ao processo de mapeamento e conversão, uma vez que o algoritmo desenvolvido parte do pressuposto de identificadores únicos simples para cada nó.

A própria escala do banco, com grande quantidade de entidades e relações interconectadas, também compromete a compreensão teórica e a visualização clara da arquitetura de dados como um todo, exigindo abordagens mais sofisticadas de exploração e análise

visual. No quesito da conversão entre modelos, o algoritmo demorou bastante tempo para exportar os dados do **JSON** genérico para o ***Neo4j***.

4.3 Resultados com a Base Northwind

A base *Northwind* foi utilizada como caso de uso para validação da qualidade semântica da conversão. Com menos tabelas e volume de dados reduzido, ela permitiu comparar diretamente consultas no modelo relacional com consultas no modelo de grafos, além de analisar entidade por entidade cada transformação feita dada as características de arestas, nós, entidades aninhadas, realações N:N.

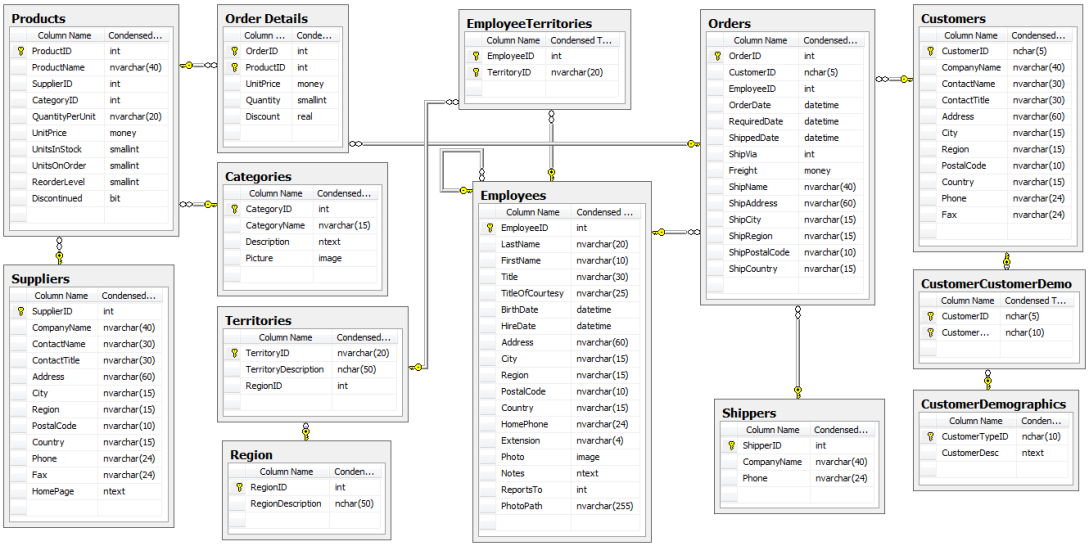


Figura 11 – Diagrama banco Northwnd.

4.3.1 Características Estruturais

- **Total de tabelas:** 14
- **Tabelas aninhadas:** 2 (Region, CustomerDemographics)
- **Tabelas N:N:** 3 (EmployeeTerritories, Order Details, CustomerCustomerDemo)
- **Total de relacionamentos (FKs):** 13
- **Total de chaves primárias (PKs):** 17
- **Total de registros encontrados:** 3.308

4.3.2 Resultados de Conversão

Como dito anteriormente, na execução do algoritmo foi possível obter o tempo de execução de cada etapa do processo, desde a extração dos metadados do banco relacional, até a exportação dos dados para o ***Neo4j*** no modelo em grafos.

Seguem os resultados obtidos:

Tabela 4 – Tempo de execução do algoritmo do banco Northwind

Etapa	**Tempo médio (s)**
Extração de metadados	1.011
Verificar aninhamento de tabelas	0.019
Reescrever estrutura aninhada	0.034
Extração de dados do banco	0,066
Transformação para grafos em JSON	0.019
Exportação formato Janus em JSON	0.003
Exportação para o Neo4j	2.412

- **Nós gerados:** 1.100
- **Arestas geradas:** 7.060

4.3.3 Observações de conversão

Tabela N:N

As tabelas `Orders` e *Products* foram convertidas para nós, enquanto a tabela `OrderDetails` foi convertida em arestas entre `Orders` e `Products`. Isso preserva a semântica original do relacionamento e permite consultas mais naturais no modelo grafo. Essa transformação de *Orders -> OrderDetails -> Products* respeita a implementação definida, colocando uma entidade que representa uma relação N:N como uma aresta e não como um nó.

Na figura 12 podemos observar no diagrama essa relação N:N entre *Orders* e *Products* e na Figura 13 podemos observar o resultado final de um registro de relação *Order* e *Product* onde o pedido *10254* foi feito com os produtos Longlife, Pâte chinois e Guaraná. As arestas podem possuir propriedades e para a Figura 13 foi colocada em exposição a quantidade de produtos comprada no pedido.

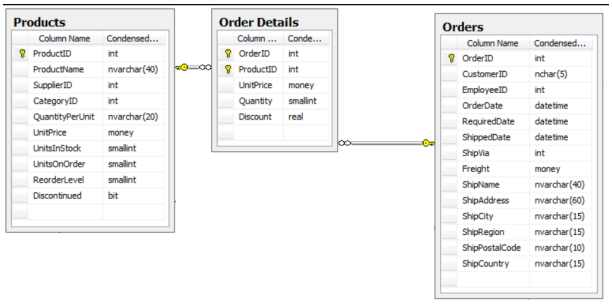


Figura 12 – Exemplificação da relação N:N e aninhamento juntos.

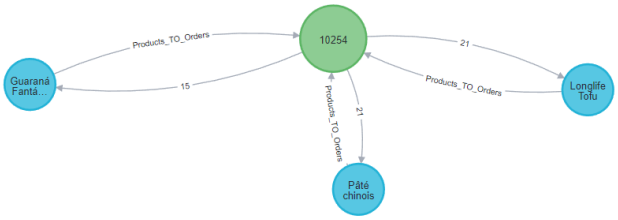


Figura 13 – Ligação de produto e pedido.

Aninhamento em uma Tabela N:N

Como podemos analisar na Figura 11 e na Figura 14 percebemos que existe uma relação N:N cuja entidade final de ligação pode ser aninhada àquela que é dita N:N, a problématica nessa história é que iriamos perder essas informações se não fosse tratado. Caso a tabela *CustomerDemographics* fosse aninhada à *CustomerCustomerDemo* (tabela de ligação N:N) e considerando que nossas relações N:N se tornam arestas, teriamos a perda das informações de *CustomerDemographics* que seria aninhada, pois a aresta de *CustomerCustomerDemo* necessitaria de um nó - nó este que seria removido.

Pelo desenvolvimento isso foi percebido e após o aninhamento das tabelas, a entidade *CustomerCustomerDemo* ficou com a coluna *CustomerDesc* e deixou de ter uma de suas chaves estrangeiras, logo ela deixou de ser uma tabela N:N.

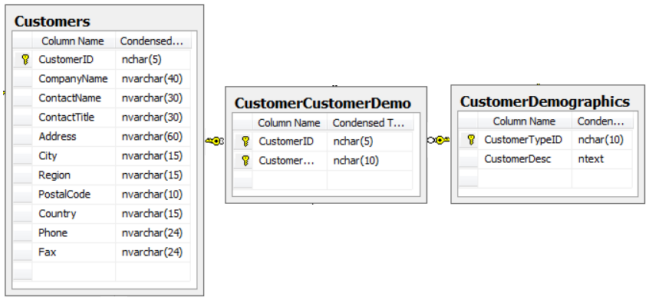


Figura 14 – Exemplificação da relação N:N e aninhamento juntos.

4.3.4 Comparação de Consultas

Com o objetivo de avaliar a expressividade, legibilidade e desempenho das abordagens relacional e orientada a grafos, esta seção apresenta um conjunto de consultas realizadas sobre a base de dados *Northwind*, utilizando as linguagens SQL e Cypher. As consultas selecionadas exploram diferentes aspectos dos dados, como relacionamentos diretos, relacionamentos muitos-para-muitos e filtragens condicionais.

Cada consulta é apresentada nas duas linguagens, seguida de uma análise comparativa destacando vantagens e limitações de cada modelo. Essa abordagem permite observar como a modelagem orientada a grafos pode oferecer em certas operações de consulta, especialmente aquelas que envolvem múltiplos saltos e estruturas relacionais complexas.

Considerando que essa tabela não possui grandeza de dados nas dezenas de milhões de dados, não temos alterações em tempo de execução, ambas os formatos e modelos são perfomáticos de forma ideal.

4.3.5 Consulta 1 — Clientes que compraram produtos da categoria "Beverages"

Essa consulta, para ambos os modelos, terão resultados em formato de texto, por estarmos fazendo consultas de dados específicos. No caso: Nome do cliente, nome do produto e nome da categoria.

Consulta SQL (Modelo Relacional)

Listing 4.1 – Consulta SQL

```sql
SELECT DISTINCT
    Customers.ContactName,
    Products.ProductName,
    Categories.CategoryName
FROM Customers
JOIN Orders ON Customers.CustomerID = Orders.CustomerID
JOIN [Order Details] ON Orders.OrderID = [Order Details].OrderID
JOIN Products ON [Order Details].ProductID = Products.ProductID
JOIN Categories ON Products.CategoryID = Categories.CategoryID
WHERE Categories.CategoryName = 'Beverages';
```

Consulta Cypher (Modelo em Grafos)

Listing 4.2 – Consulta Cypher

```
MATCH
  (c:Customers)<-[:Orders_TO_Customers]-(o:Orders)-[:Orders_TO_Products]->(p:
      Products)-[:Products_TO_Categories]->(cat:Categories)
WHERE cat.CategoryName = 'Beverages'
RETURN DISTINCT
```

```
5   c.ContactName AS Cliente,
6   p.ProductName AS Produto,
7   cat.CategoryName AS Categoria
```

Comparativo entre os Modelos

Esta consulta tem como objetivo listar todos os clientes que realizaram pedidos de produtos pertencentes à categoria "Beverages".

No modelo relacional, a resposta exige múltiplos *joins* entre as tabelas `Customers`, `Orders`, `Order Details`, `Products` e `Categories`, exigindo conhecimento da estrutura do banco, incluindo nomes de chaves primárias e estrangeiras. A legibilidade da consulta pode ser comprometida conforme a complexidade das relações aumenta.

Em contraste, no modelo orientado a grafos, a consulta Cypher permite a navegação semântica direta entre entidades, através de relacionamentos nomeados. A estrutura da consulta é mais declarativa, focando no caminho entre os nós ao invés de combinar tabelas por chaves. Essa abordagem facilita a construção de consultas que envolvem múltiplas etapas de relacionamento, especialmente em bases com muitos relacionamentos indiretos ou N:N.

A modelagem em grafos pode oferecer vantagens específicas em consultas exploratórias ou com múltiplas relações encadeadas (*multi-hop*), pela simplicidade de expressar tais caminhos na linguagem Cypher. No entanto, sua efetividade depende diretamente de uma modelagem coerente e do grau de familiaridade do usuário com a estrutura de grafos, o que pode representar uma barreira em contextos com baixo domínio desse paradigma.

4.3.6 Consulta 2 — Caminho entre Cliente, Pedido, Produto, Categoria e Fornecedor

Nesta consulta, busca-se explorar visualmente o caminho de um cliente específico até os produtos da categoria *Confections* que ele adquiriu, incluindo os respectivos fornecedores desses produtos. A escolha por restringir a consulta a um cliente específico (*Alexander Feuer*) visa facilitar a visualização da estrutura do grafo e tornar mais clara a análise semântica dos relacionamentos. O grafo gerado conecta os nós `Customers`, `Orders`, `Products`, `Categories` e `Suppliers`, passando por múltiplos relacionamentos com nomeações descritivas.

Consulta SQL (Modelo Relacional)

```
1 SELECT DISTINCT
2   c.ContactName,
3   o.OrderID,
4   p.ProductName,
```

```
  cat.CategoryName,
  s.CompanyName
FROM Customers c
INNER JOIN Orders o ON c.CustomerID = o.CustomerID
INNER JOIN [Order Details] od ON o.OrderID = od.OrderID
INNER JOIN Products p ON od.ProductID = p.ProductID
INNER JOIN Categories cat ON p.CategoryID = cat.CategoryID
INNER JOIN Suppliers s ON p.SupplierID = s.SupplierID
WHERE
  cat.CategoryName = 'Confections'
  AND c.ContactName = 'Alexander Feuer'
```

Consulta Cypher (Modelo em Grafo)

```
MATCH
  (cust:Customers)<-[:Orders_TO_Customers]-(o:Orders)-[:Orders_TO_Products]->(p:
      Products),
  (p)-[:Products_TO_Categories]->(cat:Categories),
  (p)-[:Products_TO_Suppliers]->(s:Suppliers)
WHERE
  cat.CategoryName = 'Confections'
  AND cust.ContactName = "Alexander Feuer"
RETURN *
```

Análise Comparativa

A consulta no modelo relacional exige múltiplos *joins* explícitos, inclusive com uma tabela intermediária (`Order Details`), dificultando a interpretação direta da sequência semântica dos relacionamentos, além do resultado vir através de uma listagem de texto sendo menos intuitiva que uma relação em grafos. Por outro lado, a versão em grafo permite uma navegação mais fluida e semântica entre as entidades, evidenciando o caminho natural que conecta o cliente aos produtos adquiridos e seus fornecedores, sem necessidade de conhecer a estrutura relacional intermediária.

Essa abordagem é particularmente útil para consultas exploratórias e visuais, permitindo representar diretamente no grafo uma sequência significativa de nós e arestas. Apesar disso, exige familiaridade com o modelo orientado a grafos, o que pode representar uma barreira para usuários não acostumados a esse paradigma.

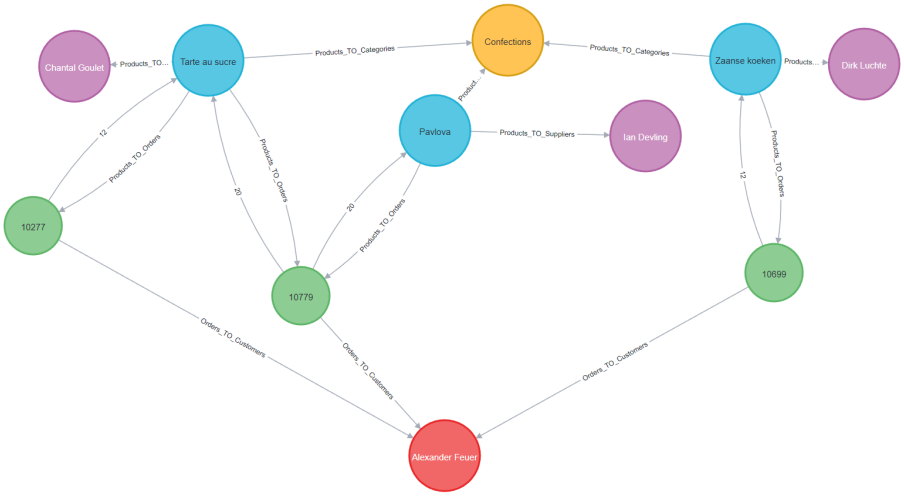


Figura 15 – Caminho entre Cliente, Pedido, Produto, Categoria e Fornecedor.

Na figura 15 podemos observar o caminho exato do cliente à categoria do produto comprado. Temos em amarelo o nó que representa a Categoria, em vermelho o nó do Cliente, em azul e rosa o Produto e o fornecedor e em verde o pedido feito. As arestas representam as ligações entre os nós, com exceção da aresta dupla encontrada entre os nós de Produto e Pedido que é uma aresta com propriedades (uma tabela relacional N:N). As arestas duplas, na figura, possuem o nome da conexão, em um delas, e em outra mostra a quantidade de itens foi feito do produto no pedido.

	ContactName	OrderID	ProductName	CategoryName	CompanyName
1	Alexander Feuer	10277	Tarte au sucre	Confections	Forêts d'érables
2	Alexander Feuer	10699	Zaanse koeken	Confections	Zaanse Snoepfabriek
3	Alexander Feuer	10779	Pavlova	Confections	Pavlova, Ltd.
4	Alexander Feuer	10779	Tarte au sucre	Confections	Forêts d'érables

Figura 16 – Resultado da query no Sql Server.

Já na figura 16 observamos a listagem simples das informações, que são corretas e de fácil visualização, mas em um modelo diferente.

4.3.7 Consulta 3 — Clientes que compraram produtos da categoria "Seafood" com preço elevado e fornecedores do mesmo país

Essa consulta busca todos os clientes que realizaram pedidos de produtos da categoria *Seafood*, com preço unitário superior a 20, e que possuem o mesmo país que o fornecedor do produto. A consulta destaca a capacidade de cruzamento de múltiplas entidades e condições no modelo grafo, mantendo a legibilidade mesmo com múltiplos relacionamentos.

Consulta SQL (Modelo Relacional)

```sql
SELECT DISTINCT
    Customers.ContactName,
    Orders.OrderID,
    Products.ProductName,
    Products.UnitPrice,
    Suppliers.CompanyName,
    Suppliers.Country
FROM Customers
JOIN Orders ON Customers.CustomerID = Orders.CustomerID
JOIN [Order Details] ON Orders.OrderID = [Order Details].OrderID
JOIN Products ON [Order Details].ProductID = Products.ProductID
JOIN Suppliers ON Products.SupplierID = Suppliers.SupplierID
JOIN Categories ON Products.CategoryID = Categories.CategoryID
WHERE Categories.CategoryName = 'Seafood'
  AND Products.UnitPrice > 20
  AND Customers.Country = Suppliers.Country;
```

Consulta Cypher (Modelo em Grafos)

```cypher
MATCH
  (c:Customers)<-[:Orders_TO_Customers]-(o:Orders)-[:Orders_TO_Products]->(p:
     Products),
  (p)-[:Products_TO_Suppliers]->(s:Suppliers),
  (p)-[:Products_TO_Categories]->(cat:Categories)
WHERE c.Country = s.Country AND cat.CategoryName = "Seafood" AND p.UnitPrice > 2
    0
RETURN *
```

Visualização e Interpretação dos Resultados

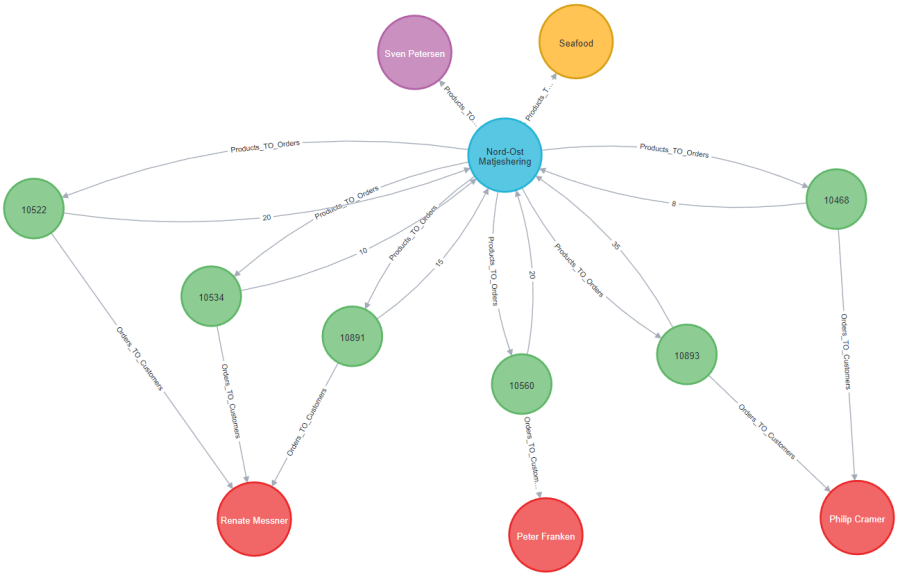


Figura 17 – Produtos do tipo "Seafood" comprados por clientes do mesmo país do fornecedor (com UnitPrice > 20).

A Figura 17 ilustra visualmente o caminho da consulta, conectando nós de clientes (em vermelho), pedidos (em verde), o produto (em azul), fornecedor (em rosa) e categoria (em amarelo). As arestas representam os relacionamentos entre essas entidades. A estrutura em grafo evidencia o fluxo de dados e a interconectividade entre as entidades envolvidas na consulta.

4.3.8 Validação da Consistência Semântica na Base Northwind

A validação da consistência semântica tem como objetivo verificar se as estruturas e relações originalmente representadas no modelo relacional da base *Northwind* foram corretamente transpostas e preservadas no modelo orientado a grafos, mantendo seu significado lógico e funcional.

4.3.8.1 Preservação dos Relacionamentos

A modelagem relacional da base *Northwind* apresenta diversas entidades conhecidas, como `Customers`, `Orders`, `Products`, `Suppliers`, `Categories` e tabelas de relacionamento como `OrderDetails`. No grafo gerado, observou-se que:

- Relacionamentos como *cliente realizou pedido*, *pedido inclui produtos*, *produto pertence a uma categoria* e *produto é fornecido por um fornecedor* foram corretamente preservados por meio de arestas com nomes como `Orders_TO_Customers`, `Orders_TO_Products`, `Products_TO_Categories` e `Products_TO_Suppliers`.
- Para tabelas de relacionamento (N:N), a direção das arestas foi ajustada para permitir flexibilidade de navegação, com inserções duplicadas em ambos os sentidos quando necessário (por exemplo, `Orders_TO_Products` e `Products_TO_Orders`), facilitando consultas exploratórias e análises semânticas complexas.

4.3.8.2 Ajustes e Interpretações Necessárias

Apesar da consistência geral, algumas interpretações exigem cuidados adicionais:

- É necessário conhecimento prévio da estrutura do banco relacional para sua utilização em grafos (nomes de tabelas e suas relações).
- O padrão de nomeação das arestas no grafo (`TabelaOrigem_TO_TabelaDestino`) é sistemático, mas pode não ser imediatamente intuitivo para usuários sem experiência técnica.
- A duplicação de algumas arestas (em direções opostas) deve ser compreendida como um recurso de usabilidade e não como duplicação semântica.

Dessa forma, conclui-se que a transformação preserva a semântica original dos dados, mantendo a capacidade expressiva e relacional da base, ao mesmo tempo em que viabiliza novas formas de consulta e exploração com maior fluidez no modelo em grafo.

4.3.9 Resultados Observados

A conversão do banco de dados relacional para o modelo orientado a grafos possibilitou uma análise prática e comparativa entre os dois paradigmas de consulta. Durante o processo, foi possível validar a eficácia do sistema implementado, observando como as informações se reorganizam em estruturas mais naturais para determinadas abordagens de exploração de dados.

Ao realizar as consultas em Cypher, especialmente aquelas que envolvem múltiplas junções no modelo relacional, foi notável a clareza e expressividade das instruções no modelo de grafos. Relações complexas como pedidos envolvendo múltiplos produtos, seus fornecedores e categorias tornaram-se mais acessíveis visualmente e semanticamente.

Outro resultado importante foi a facilidade na visualização gráfica das conexões entre os dados, o que permitiu identificar padrões, agrupar informações relacionadas e realizar inferências exploratórias com maior fluidez. A visualização dos caminhos no Neo4j, por exemplo, revelou-se uma ferramenta poderosa para compreender contextos como: clientes conectados a categorias específicas de produtos, fornecedores de mesma localidade, e a identificação de relações indiretas entre entidades distintas.

Entretanto, também se observou que o modelo orientado a grafos exige um conhecimento prévio da estrutura e da semântica das relações, o que pode apresentar uma curva de aprendizado maior para usuários acostumados exclusivamente ao paradigma relacional.

Adicionalmente, consultas consideradas triviais no modelo relacional — como a listagem direta de registros tabulares — podem exigir instruções mais verbosas em grafos, dependendo da forma como os dados foram modelados.

No geral, o sistema mostrou-se eficaz para Bancos com estrutura não tão complexas, como abordado em seções anteriores, relizando a transformação automática dos dados relacionais em grafos, respeitando relações de chave primária e estrangeira; Adaptando relações N:N como arestas com propriedades; Gerando uma estrutura explorável por meio de consultas Cypher; Possibilitando a visualização e análise exploratória de dados complexos e servindo como base para experimentos futuros de desempenho e escalabilidade em consultas de grafos.

5 Conclusão

Este trabalho apresentou o desenvolvimento de um sistema para conversão de bancos de dados relacionais para o modelo orientado a grafos, com foco em facilitar o entendimento e a execução de análises complexas sobre dados originalmente armazenados em estruturas tabulares. A solução proposta mostrou-se eficaz ao permitir que dados de modelos amplamente utilizados, como Northwind e AdventureWorks, fossem transformados em representações gráficas, favorecendo a execução de consultas exploratórias, análises multirrelacionais e visualização de conexões implícitas nos dados.

Ao longo do desenvolvimento, foi necessário lidar com desafios importantes, como a definição automática de nós e arestas, o tratamento de relacionamentos muitos-para-muitos e a aplicação de regras de aninhamento para simplificar a estrutura final do grafo. Estratégias como a incorporação de dados de tabelas aninhadas dentro dos próprios nós e o uso de uma estrutura intermediária genérica tornaram o processo mais flexível e extensível.

Os testes realizados com as bases *Northwind* e *AdventureWorks2022* evidenciaram a viabilidade do sistema proposto, tanto em termos funcionais quanto na representação dos relacionamentos. A base *Northwind*, por ser mais simples e didática, possibilitou uma conversão direta e facilitou a realização de comparações entre os modelos relacional (SQL) e orientado a grafos (Cypher).

Por outro lado, a base *AdventureWorks2022* apresentou desafios significativamente maiores: trata-se de um banco com mais de 70 tabelas distribuídas em múltiplos esquemas, com forte presença de chaves compostas, relacionamentos muitos-para-muitos e tipos de entidades e propriedades extremamente variados. Essa complexidade aumentou a dificuldade de definir o que deveria ser representado como nó ou aresta e os mapeamentos gerais da estrutura da tabela, exigindo ajustes e validações adicionais para garantir a integridade da estrutura gerada no grafo para uma próxima implementação. Ainda assim, a base serviu como um importante teste de estresse, validando a robustez e os limites da abordagem implementada.

Apesar dos avanços alcançados, o sistema ainda possui limitações e diversas oportunidades de expansão. Como trabalhos futuros, propõe-se:

- Suporte a bancos com chaves compostas, para melhorar a fidelidade de Bancos complexos.
- Suporte a outros bancos de entrada (modelos relacionais) e de saída (diversos bancos de grafos, como JanusGraph, Amazon Neptune e outros);

- Comparações sistemáticas de performance, com métricas de tempo de execução, custo computacional e uso de recursos;
- Desenvolvimento de uma interface gráfica para facilitar o uso da ferramenta por usuários não técnicos;
- Implementação de regras configuráveis para controle de aninhamento, definições do que constitui um nó ou uma aresta e configuração manual para nome de arestas (Exemplo: `Products_TO_Categories` ser configurado como `PERTENCE_A_CATEGORIA` ou `CATEGORIZADO_COMO`). .

Dessa forma, acredita-se que este sistema contribui para a aproximação entre os mundos relacional e orientado a grafos, promovendo maior acessibilidade ao paradigma de grafos para modelagem e análise de dados, e servindo como base para novas pesquisas e aplicações.

Referências

ALI, W. et al. R2gmapper: Automatic transformation of relational databases to property graphs. In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. [S.l.: s.n.], 2019. p. 1913–1916. Citado 3 vezes nas páginas 36, 37 e 42.

ALOBAID, A. et al. A survey on data modeling in graph databases. *Journal of Big Data*, 2020. Citado 2 vezes nas páginas 36 e 38.

ANGLES, R. et al. Foundations of modern graph query languages. *ACM Computing Surveys (CSUR)*, v. 53, n. 1, p. 1–30, 2020. Citado 4 vezes nas páginas 13, 29, 31 e 34.

ANGLES, R.; GUTIERREZ, C. A survey of graph database models. *ACM Computing Surveys (CSUR)*, v. 40, n. 1, p. 1–39, 2008. Citado 4 vezes nas páginas 28, 31, 33 e 48.

ANGLES, R.; GUTIERREZ, C. Foundations of modern graph query languages. *ACM Computing Surveys (CSUR)*, v. 50, n. 5, p. 1–40, 2012. Citado na página 32.

BARBOSA, D. et al. Graphgen: Exploring interesting graphs from relational data. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. [S.l.: s.n.], 2015. p. 77–90. Citado na página 38.

BORAD, P.; METAH, K.; CHAUHAN, P. A review on nosql databases. *International Journal of Science and Research*, v. 6, n. 11, p. 1–4, 2017. Disponível em: <https://www.researchgate.net/publication/327883334_A_Review_on_NoSQL_Databases>. Citado 3 vezes nas páginas 24, 26 e 27.

CALVANESE, D. et al. Ontop: Answering sparql queries over relational databases. *Semantic Web*, v. 8, n. 3, p. 471–487, 2017. Citado na página 38.

CATTELL, R. Scalable sql and nosql data stores. *ACM SIGMOD Record*, v. 39, n. 4, p. 12–27, 2011. Citado na página 26.

CHAPPELL, D. *A Developer's Guide to Graph Database Technology*. 2019. Disponível em: <https://neo4j.com/developer/graph-database/>. Citado na página 31.

CODD, E. F. A relational model of data for large shared data banks. *Communications of the ACM*, v. 13, n. 6, p. 377–387, 1970. Citado 4 vezes nas páginas 13, 18, 19 e 24.

CONNOLLY, T.; BEGG, C. *Database Systems: A Practical Approach to Design, Implementation, and Management*. [S.l.]: Pearson, 2015. Citado 2 vezes nas páginas 22 e 23.

CORONEL, C.; ROB, P. *Database Systems: Design, Implementation, Management*. [S.l.]: Cengage Learning, 2010. Citado 2 vezes nas páginas 16 e 22.

DIMOU, A. et al. Rml: A generic language for integrated rdf mappings of heterogeneous data. In: *Proceedings of the WWW2014 Developers Workshop*. [S.l.: s.n.], 2014. Citado na página 38.

ELMASRI, R.; NAVATHE, S. B. *Sistemas de bancos de dados*. 6. ed.. ed. São Paulo: Pearson, 2011. Citado 5 vezes nas páginas 16, 18, 19, 20 e 21.

FRANCIS, A. *GQL: ISO's Graph Query Language – A Standard in Progress*. 2022. Disponível em: <https://www.gqlstandards.org/>. Citado na página 30.

FRANCIS, N. et al. Cypher: An evolving query language for property graphs. In: *Proceedings of the 2018 International Conference on Management of Data*. [S.l.: s.n.], 2018. p. 1433–1445. Citado 3 vezes nas páginas 31, 32 e 33.

GIUNCHIGLIA, F.; KHARLAMOV, E.; ZHELEZNYAKOV, D. Knowledge-aware mappings for relational to graph model transformation. *Data Knowledge Engineering*, v. 132, 2021. Citado na página 35.

GROVER, K. R. L. *Practical Gremlin: An Apache TinkerPop Tutorial*. [S.l.]: Addison-Wesley Professional, 2015. Citado na página 30.

HECHT, R.; JABLONSKI, S. Nosql evaluation: A use case oriented survey. In: *Proceedings of the International Conference on Cloud and Service Computing (CSC)*. [S.l.: s.n.], 2011. p. 336–341. Citado na página 23.

HUNGER, M.; BOYD, R.; LYON, W. *Relational Database Modeling vs. Graph Database Modeling*. 2016. Acesso em: 06 out. 2025. Disponível em: <https://neo4j.com/blog/developer/rdbms-vs-graph-data-modeling/>. Citado na página 19.

MCMILLEN, C. *Advancements in Database Management Systems: A Comprehensive Review and Future Directions*. 2024. Disponível em: <https://www.researchgate.net/publication/ADVANCEMENTS_DBMS>. Citado na página 17.

MONIRUZZAMAN, A.; HOSSAIN, S. A. Nosql database: New era of databases for big data analytics - classification, characteristics and comparison. *International Journal of Database Theory and Application*, v. 6, n. 4, p. 1–14, 2013. Citado 4 vezes nas páginas 23, 24, 26 e 33.

NEWMAN, M. *Networks: An Introduction*. [S.l.]: Oxford University Press, 2010. Citado na página 28.

ORACLE. *What Is a Database?* 2020. Disponível em: <https://www.oracle.com/database/what-is-database/>. Citado na página 16.

PADHY, R.; PATRA, M.; SATAPATHY, S. Rdbms to nosql: Reviewing some next-generation non-relational database's. *International Journal of Advanced Engineering Sciences and Technologies*, v. 11, n. 1, p. 15–30, 2011. Citado na página 26.

PAN, J. T. et al. Traversing knowledge graphs in vector space. In: *Proceedings of the 21st ACM international conference on Information and knowledge management*. [S.l.: s.n.], 2012. p. 1481–1484. Citado na página 34.

POKORNY, J. Nosql databases: a step to database scalability in web environment. *International Journal of Web Information Systems*, 2013. Citado 2 vezes nas páginas 26 e 33.

ROBINSON, I.; WEBBER, J.; EIFREM, E. *Graph Databases: New Opportunities for Connected Data*. 2. ed. [S.l.]: O'Reilly Media, Inc., 2015. Citado 6 vezes nas páginas 27, 29, 30, 31, 32 e 34.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. *Sistemas de Banco de Dados*. 5. ed. Rio de Janeiro: Editora Campus, 2006. Citado 5 vezes nas páginas 17, 19, 20, 23 e 46.

SONG, J. et al. Gr2graph: A system for mapping relational databases into property graphs. In: *Proceedings of the 38th IEEE International Conference on Data Engineering (ICDE)*. [S.l.: s.n.], 2022. Citado na página 38.

SUN, Y.; LI, J. et al. Scalability challenges in graph databases. *IEEE Transactions on Knowledge and Data Engineering*, 2021. Citado 3 vezes nas páginas 31, 32 e 34.

VICENTE, A. T. d. O. et al. Mapeamento, conversão e migração automática de bancos de dados relacionais para orientados a grafos. In: *Anais do XXXIV Simpósio Brasileiro de Bancos de Dados*. [S.l.: s.n.], 2019. Citado 7 vezes nas páginas 13, 14, 35, 37, 42, 45 e 48.

WEST, D. B. *Introduction to Graph Theory*. 2. ed. [S.l.]: Prentice Hall, 2001. Citado 2 vezes nas páginas 13 e 28.