



UNIVERSIDADE FEDERAL DE SANTA CATARINA  
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA  
CURSO DE SISTEMAS DE INFORMAÇÃO

Lucas Santos de Oliveira  
Pablo Vicente

**UMA SOLUÇÃO PARA MAPEAMENTO DE BANCO DE DADOS  
RELACIONAL PARA NOSQL GRAPH: SQLTONOSQL GRAPH**

Florianópolis  
2022

Lucas Santos de Oliveira  
Pablo Vicente

**UMA SOLUÇÃO PARA MAPEAMENTO DE BANCO DE DADOS  
RELACIONAL PARA NOSQL GRAPH: SQLTONOSQL GRAPH**

Trabalho de Conclusão de Curso do Curso de Sistemas de Informação do Departamento de Informática e Estatística da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Ronaldo dos Santos Mello, Dr.

Coorientador: Prof. Geomar André Schreiner, Dr.

Florianópolis

2022

Ficha de identificação da obra elaborada pelo autor,  
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Vicente, Pablo

UMA SOLUÇÃO PARA MAPEAMENTO DE BANCO DE DADOS  
RELACIONAL PARA NOSQL GRAPH : UMA SOLUÇÃO PARA MAPEAMENTO  
DE BANCO DE DADOS RELACIONAL PARA NOSQL GRAPH / Pablo  
Vicente, Lucas Oliveira ; orientador, Ronaldo Mello,  
coorientador, Geomar Schreiner, 2022.

78 p.

Trabalho de Conclusão de Curso (graduação) -  
Universidade Federal de Santa Catarina, Centro Tecnológico,  
Graduação em Sistema de Informação, Florianópolis, 2022.

Inclui referências.

1. Sistema de Informação. 2. Mapeamento. Modelo  
relacional. 3. NoSQL. 4. Modelagem em grafos. 5. Neo4j. I.  
Oliveira, Lucas. II. Mello, Ronaldo. III. Schreiner,  
Geomar. IV. Universidade Federal de Santa Catarina.  
Graduação em Sistema de Informação. V. Título.

Lucas Santos de Oliveira

Pablo Vicente

**UMA SOLUÇÃO PARA MAPEAMENTO DE BANCO DE DADOS  
RELACIONAL PARA NOSQL GRAPH: SQLTONOSQL GRAPH**

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de “Bacharel em Sistemas de Informação” e aprovado em sua forma final pelo Curso de Sistemas de Informação.

Florianópolis, 23 de Dezembro de 2022.

---

Prof. Álvaro Junio Pereira Franco, Dr.  
Coordenador do Curso

**Banca Examinadora:**

---

Prof. Ronaldo dos Santos Mello, Dr.  
Orientador

---

Prof. Geomar André Schreiner, Dr.  
Coorientador

---

Prof.(a) Carina Friedrich Dorneles, Dr(a).  
Avaliador(a)  
Instituição Universidade Federal de Santa  
Catarina

---

Prof Angelo Augusto Frozza, Dr.  
Avaliador(a)  
Instituição Instituto Federal Catarinense  
(IFC) - Campus Camboriú

## **AGRADECIMENTOS**

Somos gratos a nossas mães pela vida, sem elas nossa existência não seria possível. Somos gratos pelo carinho, apoio e amor incondicional depositados em nós todos os dias, e desta forma, podemos ser quem somos hoje. Por nos instruir nos bons caminhos e por ser nosso exemplo de esforço e dedicação, por nos mostrar que somos capazes de trilhar o caminho que desejamos e conquistar o que almejamos.

Somos gratos aos amigos, pelo companheirismo, alegrias, tristeza e principalmente pela amizade. Foi a cumplicidade do dia-a-dia de vocês e o nosso esforço como equipe que nos ajudaram a chegar aonde chegamos.

Somos gratos à universidade pela oferta do curso e à oportunidade de fazer parte da família nesse período de aprendizado.

Somos gratos aos professores, pelo tempo e esforço em compartilhar todo o seu conhecimento, em especial ao nosso orientador, pelo empenho dedicado à elaboração deste trabalho.

Enorme gratidão ao nosso amigo e colega Vinícius Biajoni Braga Flor pelas dicas e ajudas.

*“O espaço entre a teoria e a prática não é tão grande como é, a teoria na prática.”*  
*(Lucas Bispo de Oliveira)*

## RESUMO

Atualmente a sociedade está passando pela revolução industrial 4.0 e com ela diversas mudanças estão acontecendo no cenário tecnológico: *big data*, inteligência artificial, computação em nuvem e a *Internet of Things (IoT)*. Com objetivo de manipular grandes volumes de dados, não estruturados, de forma rápida e constante, foram criados os Bancos de Dados (BDs) *Not only SQL (NoSQL)*, a fim de garantir maior disponibilidade e escalabilidade nas aplicações. Neste contexto de BDs *NoSQL*, os desenvolvedores são desafiados a lidar com diferentes linguagens para acesso e manipulação entre suas bases de dados relacionais e não relacionais. Para contornar esta problemática algumas soluções têm sido desenvolvidas, com o objetivo de intermediar os dois modelos de Banco de Dados (BD), para permitir que o usuário consiga lidar com as diferentes especificidades de forma mais amigável. Entretanto, essas abordagens costumam ser limitadas, não dando suporte a todos os modelos de BD *NoSQL* em uma única aplicação. Dada esta problemática, propõe-se a criação de um conjunto de regras para o mapeamento de esquemas de Bancos de Dados Relacionais (BDRs) para BDs *NoSQL* orientado a grafos através da extensão da solução *SQLToKeyNoSQL*. Nesta solução foi inicialmente implementado o suporte para os modelos chave-valor, orientado a colunas e orientado a documentos. As instruções *Structured Query Language (SQL)* são mapeadas para um modelo canônico em uma camada intermediária e posteriormente são traduzidos para a linguagem de acesso dos BDs *NoSQL*. Os experimentos realizados sobre a extensão proposta demonstram que através desta ferramenta é possível manipular BDs não relacionais sem a necessidade de aprender a linguagem de acesso para este modelo.

**Palavras-chave:** Mapeamento. Modelo relacional. *NoSQL*. Modelagem em grafos. *Neo4j*.

## ABSTRACT

*Currently, society is going through the industrial revolution 4.0, and with it, several changes are occurring in the technological scenario: big data, artificial intelligence, cloud computing, and IoT. In order to manipulate large volumes of unstructured data quickly and constantly, NoSQL databases were created to guarantee greater availability and scalability in applications. In this context of NoSQL databases, developers are challenged to deal with different languages for access and manipulation between their relational and non-relational databases. Some solutions have been developed to treat this problem, aiming to intermediate the two DB models and allow the user to deal with the different specificities more friendly. However, these approaches are often limited, not supporting all NoSQL DB models in a single application. Given this issue, we propose creating a set of rules for mapping schemas from BDRs to graph-oriented NoSQL databases through the extension of the SQLToKeyNoSQL solution. This solution initially supports the mapping to key-value, column-oriented, and document-oriented models. The SQL statements are mapped to a canonical model in an intermediate layer and later translated into NoSQL databases' access language. The experiments accomplished over the proposed extension demonstrate that through this tool, it is possible to manipulate non-relational DBs without learning the access language for this model.*

**Keywords:** *Mapping. Relational Model. NoSQL. Neo4j.*

## LISTA DE FIGURAS

Figura 1 – Exemplo de banco de dados orientado a grafos. . . . .	13
Figura 2 – Exemplo de banco de dados <i>NoSQL</i> chave-valor . . . . .	17
Figura 3 – Exemplo de banco de dados <i>NoSQL</i> orientado a colunas . . . . .	18
Figura 4 – Exemplo de banco de dados <i>NoSQL</i> orientado a documentos . . . . .	19
Figura 5 – Exemplo de conexões do mundo real . . . . .	20
Figura 6 – Exemplo de relacionamentos no modelo relacional . . . . .	21
Figura 7 – Exemplo de relacionamentos no modelo orientado a grafos . . . . .	21
Figura 8 – Popularidade de bancos de dados . . . . .	22
Figura 9 – Comparação de SGBDs orientados a grafos entre armazenamento e processamento . . . . .	22
Figura 10 – Exemplo de representação gráfica de um grafo . . . . .	23
Figura 11 – Exemplo de relacionamentos em um grafo . . . . .	24
Figura 12 – Exemplo de <i>CONSTRAINT</i> com propriedade única . . . . .	24
Figura 13 – Exemplo de consulta na linguagem <i>Cypher</i> . . . . .	25
Figura 14 – Exemplo de equivalência de uma consulta em <i>SQL</i> e <i>Cypher</i> . . . . .	25
Figura 15 – Exemplo do comando <i>MATCH</i> na linguagem <i>Cypher</i> . . . . .	26
Figura 16 – Exemplo do comando <i>OPTIONAL MATCH</i> na linguagem <i>Cypher</i> . . . . .	26
Figura 17 – Exemplo do comando <i>RETURN</i> na linguagem <i>Cypher</i> . . . . .	26
Figura 18 – Exemplo do comando <i>CREATE</i> na linguagem <i>Cypher</i> . . . . .	26
Figura 19 – Exemplo do comando <i>SET</i> na linguagem <i>Cypher</i> . . . . .	27
Figura 20 – Exemplo do comando <i>DELETE</i> na linguagem <i>Cypher</i> . . . . .	27
Figura 21 – Exemplo do comando <i>WHERE</i> na linguagem <i>Cypher</i> . . . . .	27
Figura 22 – Exemplo do comando <i>WITH</i> na linguagem <i>Cypher</i> . . . . .	28
Figura 23 – Exemplo do comando <i>REMOVE</i> na linguagem <i>Cypher</i> . . . . .	28
Figura 24 – Exemplo do comando <i>CALL</i> na linguagem <i>Cypher</i> . . . . .	28
Figura 25 – Estratégia de extração de esquema genérico . . . . .	29
Figura 26 – Tabela comparativa entre a proposta e os trabalhos correlatos . . . . .	31
Figura 27 – Modelo canônico . . . . .	32
Figura 28 – Arquitetura da solução <i>SQLToKeyNoSQL</i> . . . . .	33
Figura 29 – Exemplo da instrução <i>UPDATE</i> convertida em métodos <i>GetN</i> , <i>Delete</i> e <i>Put</i> . . . . .	35
Figura 30 – Exemplo de instruções <i>SELECT</i> convertidas em métodos <i>GetN</i> . . . . .	35
Figura 31 – Exemplo do mapeamento <i>SQL</i> para <i>Neo4j</i> . . . . .	38
Figura 32 – Exemplo de criação da <i>CONSTRAINT</i> para o método <i>CREATE</i> . . . . .	39
Figura 33 – Exemplo do comando <i>DROP COLUMN</i> para o método <i>ALTER</i> . . . . .	39
Figura 34 – Exemplo do comando <i>RENAME COLUMN</i> para o método <i>ALTER</i> . . . . .	40
Figura 35 – Exemplo da instrução <i>DROP TABLE</i> . . . . .	40

Figura 36 – Exemplo de mapeamento da instrução <i>INSERT INTO</i> . . . . .	41
Figura 37 – Exemplo de mapeamento da instrução <i>UPDATE</i> . . . . .	42
Figura 38 – Exemplo de mapeamento da instrução <i>DELETE</i> . . . . .	43
Figura 39 – Exemplo de mapeamento da instrução <i>GETN</i> . . . . .	43
Figura 40 – Tela para inserção das credenciais de acesso ao BD <i>Neo4j</i> . . . . .	44
Figura 41 – Tela para execução de <i>queries</i> . . . . .	44
Figura 42 – Componente para retorno da <i>query</i> . . . . .	45
Figura 43 – Seleção de visualização das tabelas . . . . .	45
Figura 44 – Esquema do BDR utilizado nos experimentos . . . . .	46
Figura 45 – Gerador sintético de instruções de inserção . . . . .	47
Figura 46 – Resultados do <i>CREATE</i> . . . . .	49
Figura 47 – <i>Overhead</i> para a instrução <i>CREATE</i> . . . . .	50
Figura 48 – Resultados da instrução <i>ALTER</i> . . . . .	51
Figura 49 – <i>Overhead</i> para a instrução <i>ALTER</i> . . . . .	52
Figura 50 – Resultados da instrução <i>DROP</i> . . . . .	53
Figura 51 – <i>Overhead</i> para a instrução <i>DROP</i> . . . . .	54
Figura 52 – Resultados da instrução <i>INSERT</i> . . . . .	55
Figura 53 – <i>Overhead</i> para a instrução <i>INSERT</i> . . . . .	55
Figura 54 – Resultados da instrução <i>INSERTN</i> . . . . .	56
Figura 55 – <i>Overhead</i> para a instrução <i>INSERTN</i> . . . . .	57
Figura 56 – Resultados da instrução <i>UPDATE</i> . . . . .	58
Figura 57 – <i>Overhead</i> para a instrução <i>UPDATE</i> . . . . .	59
Figura 58 – Resultados da instrução <i>DELETE</i> . . . . .	60
Figura 59 – <i>Overhead</i> para a instrução <i>DELETE</i> . . . . .	61
Figura 60 – Resultados da instrução <i>SELECT</i> . . . . .	62
Figura 61 – <i>Overhead</i> para a instrução <i>SELECT</i> . . . . .	63

## LISTA DE ABREVIATURAS E SIGLAS

ACID	Atomicidade, Consistência, Isolamento e Durabilidade
API	Application Programming Interface
BD	Banco de Dados
BDR	Banco de Dados Relacional
BDRs	Bancos de Dados Relacionais
BDs	Bancos de Dados
CRUD	Create, Read, Update, Delete
DDL	Data Definition Language
DML	Data Manipulation Language
ETL	Extract, Transform, Load
IoT	Internet of Things
NoSQL	Not only SQL
SGBD	Sistema de Gerenciamento de Banco de Dados
SGBDs	Sistemas de Gerenciamento de Bancos de Dados
SQL	Structured Query Language

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO</b>	<b>12</b>
1.1	OBJETIVOS	14
1.2	METODOLOGIA	14
1.3	ESTRUTURA DO TRABALHO	15
<b>2</b>	<b>FUNDAMENTAÇÃO TEÓRICA</b>	<b>16</b>
2.1	BANCOS DE DADOS RELACIONAIS	16
2.2	BANCO DE DADOS <i>NOSQL</i>	16
2.3	BANCO DE DADOS ORIENTADO A GRAFOS	19
<b>2.3.1</b>	<b><i>Neo4j</i></b>	<b>21</b>
<b>2.3.2</b>	<b>Cypher</b>	<b>24</b>
<b>3</b>	<b>TRABALHOS CORRELATOS</b>	<b>29</b>
3.1	<i>A UNIFIED METAMODEL FOR NOSQL AND RELATIONAL DATABASES</i>	29
3.2	<i>AUTOMATIC NOSQL TO RELATIONAL DATABASE TRANSFORMATION WITH DYNAMIC SCHEMA MAPPING</i>	30
3.3	<i>TOWARDS UNIFIED MODELING FOR NOSQL SOLUTION BASED ON MAPPING APPROACH</i>	31
3.4	CONSIDERAÇÕES FINAIS	31
<b>4</b>	<b><i>SQLTOKEYNOSQL</i></b>	<b>32</b>
<b>5</b>	<b><i>SQLTONOSQL-GRAPH</i></b>	<b>36</b>
5.1	REGRAS DE MAPEAMENTO PARA O MODELO ORIENTADO A GRAFOS	37
5.2	MAPEAMENTO DE INSTRUÇÕES <i>DDL</i>	38
5.3	MAPEAMENTO DE INSTRUÇÕES <i>DML</i>	40
5.4	INTERFACE DO USUÁRIO E FUNCIONALIDADES DA FERRAMENTA	43
<b>6</b>	<b>ANÁLISE DE DESEMPENHO E RESULTADOS</b>	<b>46</b>
6.1	DATASET	46
6.2	CONFIGURAÇÃO DO AMBIENTE	47
6.3	MÉTRICAS DE AVALIAÇÃO	47
6.4	EXPERIMENTOS COM INSTRUÇÕES <i>SQL DDL</i>	48
6.5	EXPERIMENTOS COM INSTRUÇÕES <i>SQL DML</i>	54
<b>7</b>	<b>CONCLUSÃO</b>	<b>64</b>
	<b>REFERÊNCIAS</b>	<b>66</b>
	<b>APÊNDICE A – CÓDIGO FONTE</b>	<b>69</b>
	<b>APÊNDICE B – ARTIGO NO FORMATO SBC</b>	<b>70</b>

## 1 INTRODUÇÃO

Atualmente o mundo está passando pela revolução industrial 4.0 e com ela diversas mudanças estão acontecendo no cenário tecnológico. Esta nova realidade dá espaço a novas tecnologias, tais como, *big data*, inteligência artificial, computação em nuvem e a *IoT* (INDUSTRIA, 2022). Nesse contexto, aplicações geram um grande volume de dados que na maior parte das vezes são heterogêneos e não possuem esquemas rígidos (SCHREINER, 2016). Dentre as principais características do *big data* pode-se elencar a flexibilidade de representação, a alta velocidade de geração e o grande volume de dados (BERMAN, 2013).

Ao longo dos anos, os BDRs se tornaram os mais populares e disseminados sistemas de gerenciamento de dados, pois atendiam as necessidades da maior parte das aplicações, entregando alto desempenho, simplicidade e confiabilidade na manipulação dos dados (STONEBRAKER, 2012). No entanto, para grandes volumes de dados acabam não sendo compatíveis, pois precisam gerenciar e manter a sua consistência sobre os dados, além do esquema lógico inflexível, comprometendo o desempenho (ABADI, 2009). Para suprir essas necessidades, novos modelos de dados e novos Sistemas de Gerenciamento de Bancos de Dados (SGBDs) estão sendo empregados (ZHANG; CHENG; BOUTABA, 2010). As arquiteturas destes sistemas utilizam recursos de alta disponibilidade, distributividade e escalabilidade atrelados à capacidade de processamento e armazenamento de grandes volumes de dados em *data centers* alocados na Internet, conceito esse denominado *cloud computing* (ABOUZEID *et al.*, 2009).

Esses novos SGBDs são denominados *NoSQL* (STONEBRAKER, 2012). Eles oferecem suporte a novos modelos de dados flexíveis (ABADI, 2009). Os BDs *NoSQL* são classificados de acordo com o modelo de dados, sendo denominados como: chave-valor, orientado a colunas, orientado a documentos e orientado a grafos, esse último representando dados através de nodos e relacionamentos entre nodos.

Nesse contexto, diversas aplicações atualmente desejam migrar seus dados relacionais para soluções *NoSQL*. Porém, usuários e desenvolvedores estão familiarizados com a interface de acesso *SQL* enquanto BDs *NoSQL* definem interfaces de acesso mais simples, geralmente utilizando *Application Programming Interface (API)* e/ou linguagem específica, como por exemplo a linguagem *Cypher*<sup>1</sup>, que lida com as instruções declarativas no BD orientado a grafos *Neo4j*<sup>2</sup>.

O foco deste trabalho é o mapeamento de dados de BDRs para BDs *NoSQL* orientado a grafos. O mapeamento do modelo relacional para o modelo de grafos é uma alternativa para otimizar principalmente consultas que devem acessar diversos relacionamentos entre os dados. Isso porque, em um grafo, os relacionamentos são definidos de forma mais natural. As entidades chamadas de vértices, que são interligadas pelas are-

<sup>1</sup> <https://neo4j.com/docs/cypher-manual/current/>

<sup>2</sup> <https://neo4j.com/>

tas, podem guardar dados entre os relacionamentos em diferentes direções, criando assim uma estrutura mais orgânica. Como exemplifica a Figura 1. Cada elipse representa uma instância de dado e os seus relacionamentos são caracterizados pelas arestas, indicando que há interação entre as instâncias.

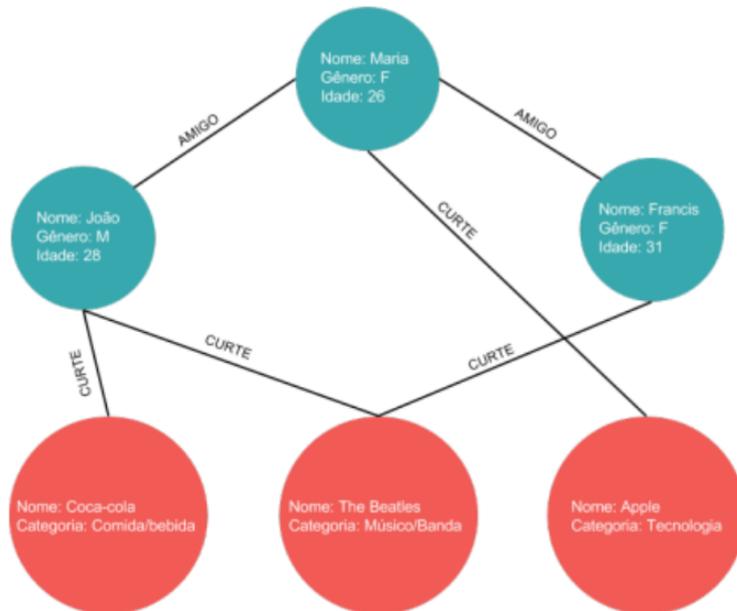


Figura 1 – Exemplo de banco de dados orientado a grafos.

Fonte: Lab Codes (Nicolle Cysneiros - 2016).

Alguns dos trabalhos relacionados apontam vantagens em utilizar BDs *NoSQL* para a realização de consultas sobre grandes volumes de dados com esquemas flexíveis e com alta escalabilidade (BRITO, 2010) (NICOLE MELO E LEANDRO DOS SANTOS E WELLINGTON DE OLIVEIRA, 2018), (NADINE ANDERLE E DALVAN GRIEBLER E SAMUEL SOUZA, 2018), (HOMRICH; MERGEN, 2018). A solução *SQLToKeyNoSQL* (SCHREINER, 2016) é um exemplo que permite realizar operações *Create, Read, Update, Delete (CRUD)* em BDs *NoSQL* utilizando *SQL*. Ela suporta um conjunto restrito de instruções *Data Definition Language (DDL)* e *Data Manipulation Language (DML)*. Ela também realiza o mapeamento do esquema relacional para um modelo canônico intermediário, e deste modelo canônico para três modelos de dados *NoSQL*: chave-valor, orientado a colunas e orientado a documentos. O acesso aos BDs *NoSQL* suportados é realizado através de um conector genérico pelo qual os métodos de manipulação de dados são baseados em verbos da *API REST*. Este trabalho estende a *SQLToKeyNoSQL* propondo uma camada externa para o mapeamento de esquemas relacionais e operações *SQL* restritas para esquemas em grafo e operações sobre um BD orientado a grafos.

## 1.1 OBJETIVOS

O objetivo principal deste trabalho é desenvolver uma solução de mapeamento de esquemas relacionais e instruções *SQL* para BDs *NoSQL* orientado a grafos. Adicionalmente as regras são implementadas na forma de uma extensão da solução *SQLToKeyNoSQL* (SCHREINER, 2016), para o BD *NoSQL* orientado a grafos *Neo4j*.

Para tanto, os seguintes objetivos específicos são definidos:

- Estudar estratégias de mapeamento de BDRs para BDs orientado a grafos;
- Propor regras de mapeamento de BDRs para BDs orientado a grafos;
- Desenvolver uma extensão do *SQLToKeyNoSQL* capaz de realizar o mapeamento direto de instruções *CRUD* da *SQL* para instruções do BD *NoSQL* orientado a grafos *Neo4j*;
- Avaliar a qualidade dos mapeamentos e o desempenho da aplicação.

## 1.2 METODOLOGIA

Esse estudo tem por finalidade realizar um projeto de pesquisa aplicada, uma vez que utiliza conhecimento da pesquisa básica para resolver problemas práticos. Para um melhor tratamento dos objetivos e melhor apreciação desta pesquisa, observa-se que ela é classificada como pesquisa qualitativa e exploratória (WAZLAWICK, 2017). Detectou-se também a necessidade da pesquisa bibliográfica, uma vez que, a pesquisa bibliográfica implica em que os dados e informações necessárias para realização da pesquisa sejam obtidos a partir de abordagens já trabalhadas por outros autores através de livros, artigos, *surveys*, revistas especializadas, documentos eletrônicos e enciclopédias.

Neste trabalho foi feito o uso e coleta de dados de material já publicado, constituído principalmente de teses, dissertações e artigos científicos na busca e alocação de conhecimento sobre mapeamento de BDRs para BDs *NoSQL* orientado a grafos. Como estratégia de busca por estas publicações foi utilizada a revisão bibliográfica sistemática, selecionando os trabalhos correlatos que compartilhavam os termos chave relacionados ao tema abordado neste trabalho, tais como, *mapping*, *NoSql Graph*, *mapping relational to not relational*. O trabalho divide-se em cinco etapas. Na primeira etapa foi elaborada a fundamentação teórica do trabalho, pelo qual foram relacionados os assuntos de BDRs, não relacionais, *Neo4j* e *Cypher* para melhor compreensão dos conceitos. Na segunda etapa foi feita a revisão literária para análise de similaridades em outras publicações como foco no mapeamento de BDRs para BDs *NoSQL*. Na terceira etapa foram selecionadas as publicações correlatas para identificar os trabalhos que se aproximam da proposta sugerida e validar o desenvolvimento desta solução. Na quarta etapa foi realizado o desenvolvimento da ferramenta *SqlToNoSql-Graph*, para validação das regras de mapeamento propostas

neste trabalho. Na quinta etapa foi realizada a análise de desempenho com o objetivo de avaliar a eficiência da ferramenta e o *overhead* envolvido em todo o processo.

### 1.3 ESTRUTURA DO TRABALHO

O presente trabalho se divide em sete capítulos:

- 1°. Capítulo apresenta a introdução do trabalho, seus objetivos, escopo e método de pesquisa;
- 2°. Capítulo apresenta a fundamentação teórica, incluindo os conceitos de BDRs, *NoSQL*, BD orientado a grafos, *Neo4j* e *Cypher*;
- 3°. Capítulo apresenta os trabalhos relacionados, focando principalmente em trabalhos que realizam mapeamentos para BDs orientado a grafos e esclarecendo as suas vantagens e desvantagens em relação às técnicas de mapeamento apresentadas nesse trabalho;
- 4°. Capítulo apresenta a ferramenta a *SQLToKeyNoSQL* (SCHREINER, 2016), sua arquitetura, a interface gráfica da aplicação, o modelo canônico e as regras de mapeamento de esquemas e de instruções *SQL* e quais BDs são suportados;
- 5°. Capítulo apresenta o andamento do desenvolvimento da ferramenta proposta, mostrando as regras de mapeamento de esquemas e de instruções *SQL* compatíveis com *Neo4j*;
- 6°. Capítulo apresenta uma análise de desempenho e resultado da proposta;
- 7°. Capítulo apresenta a conclusão do trabalho e também possibilidades de trabalhos futuros.

## 2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta uma revisão dos conceitos essenciais para o entendimento e desenvolvimento do presente trabalho. São descritos de maneira breve conceitos de BDRs e BDs *NoSQL*. Na sequência, BD orientado a grafos, detalhando o *Neo4j* e a sua linguagem de consulta, a *Cypher*.

### 2.1 BANCOS DE DADOS RELACIONAIS

Um Banco de Dados Relacional (BDR) é caracterizado pela capacidade de organizar uma coleção de dados de forma estruturada e geralmente administrado por um Sistema de Gerenciamento de Banco de Dados (SGBD). Em 1970 foi desenvolvida a linguagem *SQL*, utilizada pela maioria dos BDRs. Seu propósito é aplicado na consulta, definição, manipulação e controle de dados, dentre outros (ORACLE, s.d.). A utilização da *SQL* é vasta devido ao seu padrão que atinge a maior parte do mercado de SGBD. Ela entrega grande poder de consulta e facilita a migração entre outros sistemas. Explorando as categorias de instruções *SQL* pode-se elencar duas principais: a *DML*, utilizada em consultas e manipulação dos dados armazenados, e a *DDL* para a criação e modificação das estruturas dos objetos de um BD (NETO, 2017).

Os BDRs se destacam por manter a consistência dos dados entre as aplicações e suas instâncias. Eles possuem quatro propriedades para as suas transações, conhecidas por Atomicidade, Consistência, Isolamento e Durabilidade (ACID), sendo responsáveis por garantir a confiabilidade das transações mesmo que ocorram de maneira concorrente.

### 2.2 BANCO DE DADOS *NOSQL*

O mundo está constantemente passando por transformações, se atualizando e encontrando novas formas de atender as necessidades que vão surgindo ao longo do tempo. No que tange ao cenário tecnológico, as revoluções são muito mais intensas: padrões e paradigmas são reinventados a todo momento. Os dados que compõem a maior parte dos sistemas informacionais vêm aumentando seu volume de forma exponencial. Deste modo, novas abordagens são necessárias para atender as necessidades computacionais (ROMÃO, 2002).

Frente a esses cenário surgiu um novo conceito de BD, o *NoSQL*. Ele possui a capacidade de manipulação de dados de maneira não relacional. Deste modo é possível contornar as particularidades de cada aplicação e entregar uma solução mais ajustada à natureza dos dados manipulada por ela (CATTELL, 2011). Para lidar com dados heterogêneos e complexos foram criados diversos modelos de dados para BD *NoSQL*. De modo geral, são categorizados quatro modelos de dados (SADALAGE P. J.; FOWLER, 2012), sendo eles:

**Chave-Valor:** Caracteriza-se por ser um modelo simples, similar a uma estrutura de indexação, em que uma chave é responsável por identificar determinado conteúdo. A ausência de definição de esquemas fez com que se tornasse o mais famoso dentre os BDs *NoSQL*. No entanto, não suporta relacionamento entre dados, e não possui linguagem de consulta. Um exemplo é o SGBD *Voldemort*.

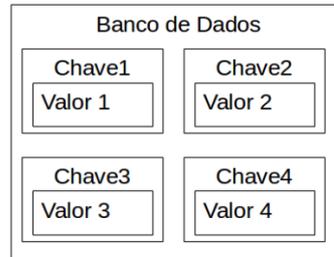


Figura 2 – Exemplo de banco de dados *NoSQL* chave-valor

Fonte: (SCHREINER, 2016)

A Figura 2 apresenta um exemplo de BD que segue o modelo chave-valor. O valor armazena determinado conteúdo, sendo ele complexo ou simples, pois o modelo não avalia esta diferenciação. O valor é manipulado como um conteúdo atômico, mesmo que o foco seja em manter um conjunto de valores com diversas propriedades (SCHREINER, 2016).

**Orientado a colunas:** Os dados são tratados de forma similar a uma tabela. Ele apresenta maior complexidade em relação ao modelo chave-valor. Possui linguagem de consulta simples e permite colunas multivaloradas e supercolunas. Sua deficiência se dá pela falta de suporte a relacionamentos entre os dados. Um exemplo é o SGBD *HBase*.

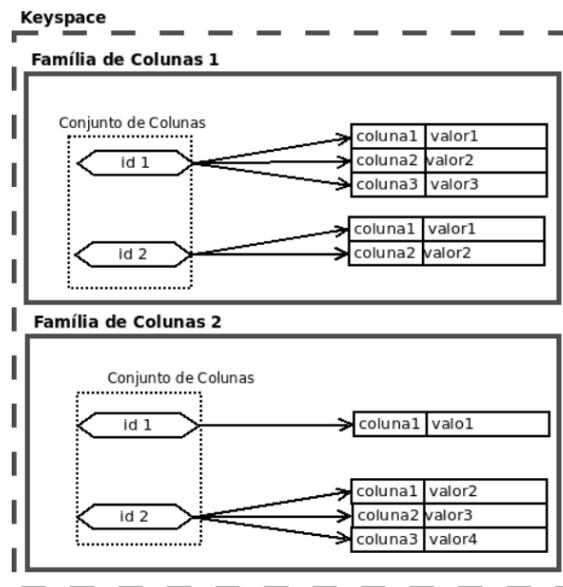


Figura 3 – Exemplo de banco de dados *NoSQL* orientado a colunas

Fonte: (SCHREINER, 2016)

A Figura 3 exemplifica o modelo orientado a colunas. Cada identificador de um conjunto de colunas pode ter uma quantidade diferente de colunas, pois se trata de itens de dados que podem ter propriedades distintas.

**Orientado a documentos:** Este modelo é destinado à representação de objetos complexos. Cada objeto possui uma chave e um conjunto de atributos, que por sua vez, podem ser atômicos ou complexos, apresentando linguagens de consulta simples ou *API* de acesso. Seus pontos fracos estão associados à falta de suporte a relacionamentos entre os dados e pela falta de padronização. Um exemplo é o SGBD *CouchDB*.

A Figura 4 exemplifica um esquema que segue o modelo orientado a documentos. A organização hierárquica das suas definições aborda a definição do BD, a coleção de documentos e seus respectivos atributos.

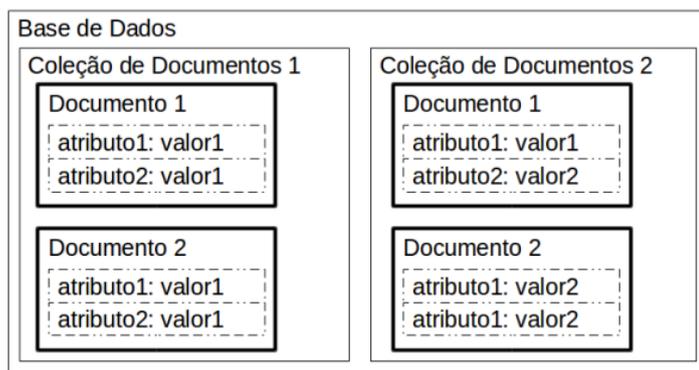


Figura 4 – Exemplo de banco de dados *NoSQL* orientado a documentos

Fonte: (SCHREINER, 2016)

**Orientado a Grafos:** Modelo composto por nodos (itens de dados compostos por atributos) e arestas que relacionam nodos e possuem um rótulo. Tanto nodos quanto arestas podem ter atributos, compostos por nome e valor, podendo ser de domínio atômico ou multivalorado. Diferentemente dos outros modelos citados anteriormente, este se destaca pelo suporte a relacionamentos entre dados. Um exemplo é o SGBD *Neo4j*. BDs orientado a grafos são o foco deste trabalho, sendo detalhado a seguir.

### 2.3 BANCO DE DADOS ORIENTADO A GRAFOS

BD orientado a grafos não armazenam tabelas como BDRs. Eles armazenam nodos e relacionamentos e não possuem restrições pré-definidas de esquema, tornando-se assim altamente flexível. Para se compreender um determinado assunto ou comportamento muitas vezes as conexões entres os dados são mais importantes do que o próprio dado em si (NEO4J, 2022g). Com o propósito de tornar os relacionamentos mais evidentes, os BDs orientado a grafos foram criados.

A Figura 5 exemplifica um modelo orientado a grafos representando conexões e relacionamentos presentes no mundo real. É possível observar as entidades (nodos) e seus relacionamentos (arestas). Tal representação é muito semelhante à armazenada ou visualizada em BDs orientado a grafos.

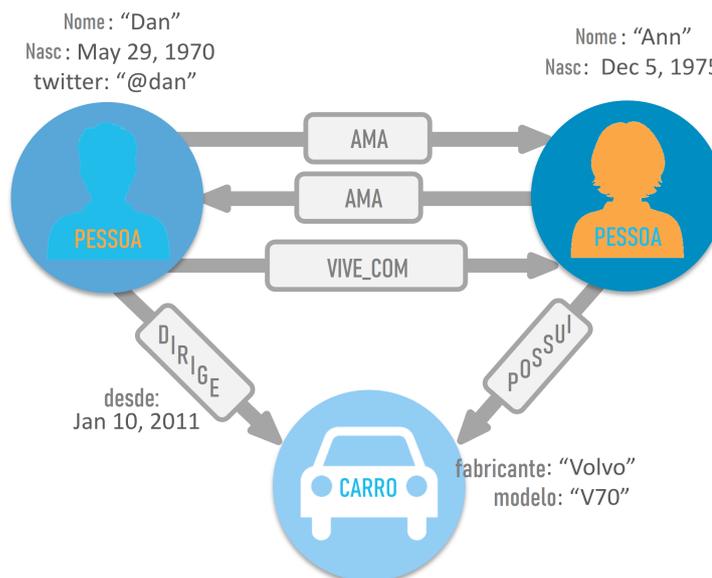


Figura 5 – Exemplo de conexões do mundo real

Fonte: (NEO4J, 2022g)

BDRs podem armazenar relacionamentos, mas para navegar entre eles é necessário utilizar junções entre as tabelas. Este processo envolve a busca de chaves de uma tabela em outra tabela. Isso significa que BDRs não conseguem lidar com relacionamentos de maneira satisfatória, pois não são performáticos nessas operações e as consultas realizadas podem ser tornar complexas (NEO4J, 2022g). Em um BD orientado a grafos não há junções. Os relacionamentos são armazenados juntamente com os dados. BDs *NoSQL* orientado a grafos foram criados para representar problemas do mundo real que envolvem relacionamentos entre dados, como relacionamentos muitos-para-muitos, navegação em hierarquias e inter-relacionamentos entre eles (NEO4J, 2022g).

A Figura 6 apresenta o mapeamento de um relacionamento muitos-para-muitos entre duas entidades. Para isso é necessário criar uma tabela de relacionamento na qual é armazenada uma chave estrangeira para cada entidade. Para executar uma consulta envolvendo este relacionamento é necessário utilizar uma junção entre as três entidades, relacionando a chave primária com as suas respectivas chaves estrangeiras na tabela associativa.

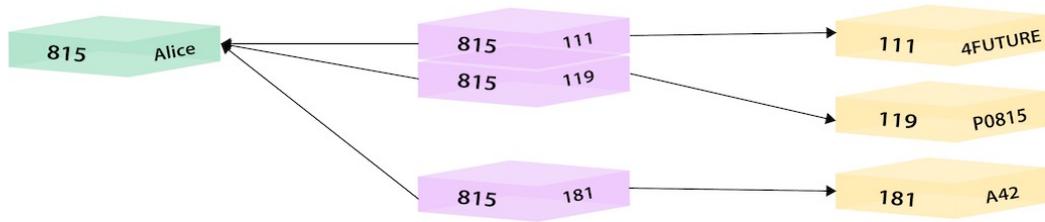


Figura 6 – Exemplo de relacionamentos no modelo relacional

Fonte: (NEO4J, 2022b)

A Figura 7 apresenta o mesmo mapeamento em BD orientado a grafos. Neste modelo tem-se um nodo *Alice*, o qual possui três relacionamentos com outros nodos. As relações entre eles tornam-se mais explícitas, facilitando inclusive a busca, pois para retornar essa relação precisa-se apenas pesquisar pela entidade e nome do relacionamento. Não é necessário manter chaves estrangeiras e tabelas de relacionamentos.

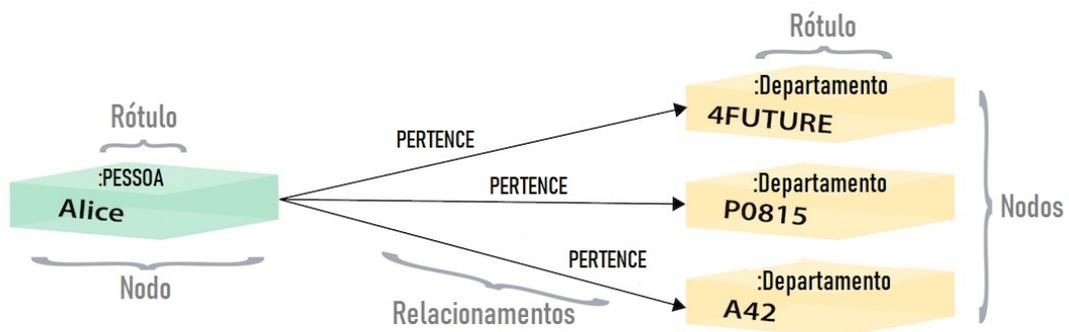


Figura 7 – Exemplo de relacionamentos no modelo orientado a grafos

Fonte: (NEO4J, 2022b)

### 2.3.1 Neo4j

O interesse por BDs *NoSQL* vem crescendo muito nos últimos anos, especialmente BD orientado a grafos, como mostra a Figura 8.

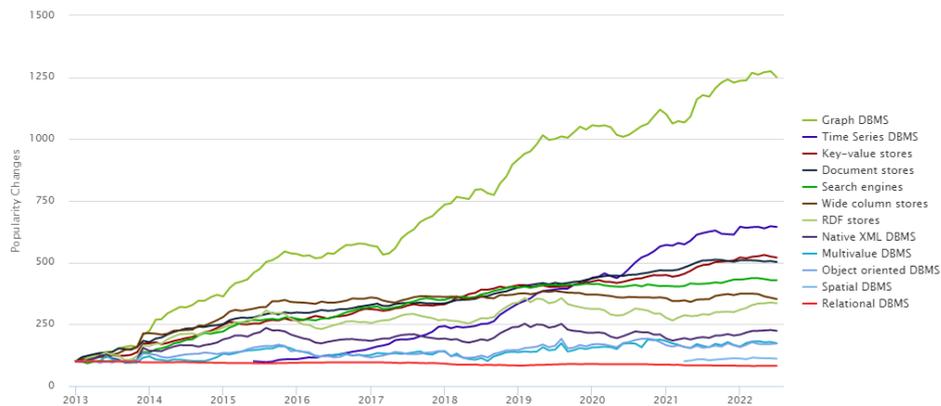


Figura 8 – Popularidade de bancos de dados

Fonte: (DB-ENGINES.COM, 2022)

A Figura 9 demonstra um comparativo entre SGBDs populares que possuem orientação a grafos, com destaque para o armazenamento e processamento de grafos. Dentre eles, o *Neo4j* se destaca, pois implementa o modelo de grafo no nível de armazenamento, diferentemente de outros SGBDs que entregam somente a visualização no formato de grafo.



Figura 9 – Comparação de SGBDs orientados a grafos entre armazenamento e processamento

Fonte: (RIAN ROBINSON; EIFREM, 2015)

O *Neo4j* é um BD *NoSQL*, *open-source*, orientado a grafos que permite o controle

das propriedades ACID, *procedures*, controle de usuários, funções e permissões. Está disponível no mercado desde 2007 e possui duas versões: *Community Edition* e *Enterprise Edition*. A versão *Enterprise* oferece alguns recursos extras como *backups* e *clustering* (NEO4J, 2022b). Além disso, o *Neo4j* é um BD orientado a grafos nativo, ou seja, implementa o modelo orientado a grafos ao nível de armazenamento, não apenas como abstração de grafo. Por esta razão possui um desempenho melhor que outros BDs.

As entidades no *Neo4j* são representadas através de nodos, sendo a representação escrita muito similar à representação visual. Nodos contém uma propriedade denominada *label* que são como rótulos e permitem especificar o tipo de entidade (NEO4J, 2022e). Apesar de ser permitido realizar consultas sem especificar o rótulo, é recomendado utilizá-lo, pois caso contrário a pesquisa é realizada em todo o BD.

A Figura 10 representa um grafo. Nela os rótulos são de três tipos: Pessoa, Companhia e Tecnologia. Por ser um esquema flexível, nem todos os nodos apresentam as mesmas propriedades. Tipo e nome não são propriedades comuns a todos os nodos.

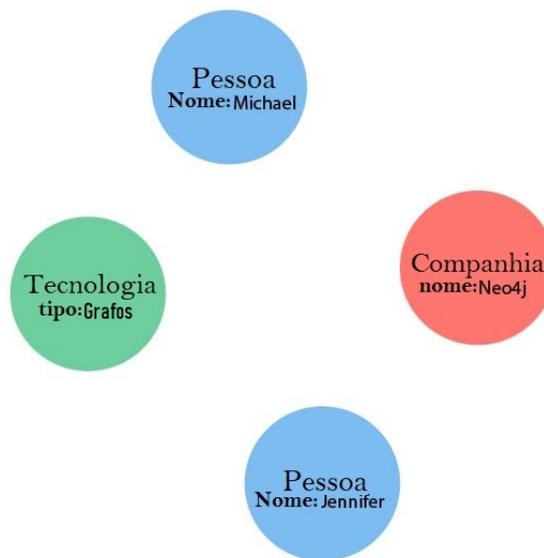


Figura 10 – Exemplo de representação gráfica de um grafo

Fonte: (NEO4J, 2022e)

O relacionamento entre as entidades se dá pela utilização de setas  $\rightarrow$  ou  $\leftarrow$ , a direção da seta indica a direção do relacionamento. O relacionamento entre os nodos suporta a criação de propriedades exclusivas da relação, como representado na Figura 11 (NEO4J, 2022e).

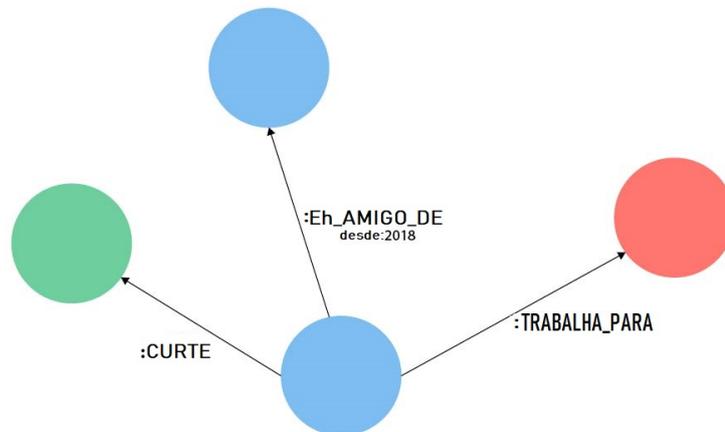


Figura 11 – Exemplo de relacionamentos em um grafo

Fonte: (NEO4J, 2022e)

Para manter a integridade dos dados e otimizar o desempenho das consultas, o *Neo4j* permite a criação de quatro tipos de *Constraints* (NEO4J, 2022c), sendo três delas disponíveis apenas na versão *Enterprise*. Um exemplo é a *Propriedade Única*, que garante que o valor da propriedade é único entre todos os nodos com o mesmo rótulo. Ela permite a combinação de múltiplas propriedades. Um exemplo é mostrado na Figura 12.

```
1 CREATE CONSTRAINT constraint_name
2 IF NOT EXISTS FOR (book:Book)
3 REQUIRE book.isbn IS UNIQUE
```

Figura 12 – Exemplo de *CONSTRAINT* com propriedade única

Fonte: (NEO4J, 2022c)

### 2.3.2 Cypher

*Cypher* é a linguagem declarativa de consulta do *Neo4j*. Ela é muito semelhante a *SQL*, pois é inspirada nela, sendo focada nos dados que se deseja retirar do grafo não em como retirá-los (NEO4J, 2022d). A Figura 13 apresenta uma consulta escrita na linguagem *Cypher* que retorna as pessoas que "Dan" ama.

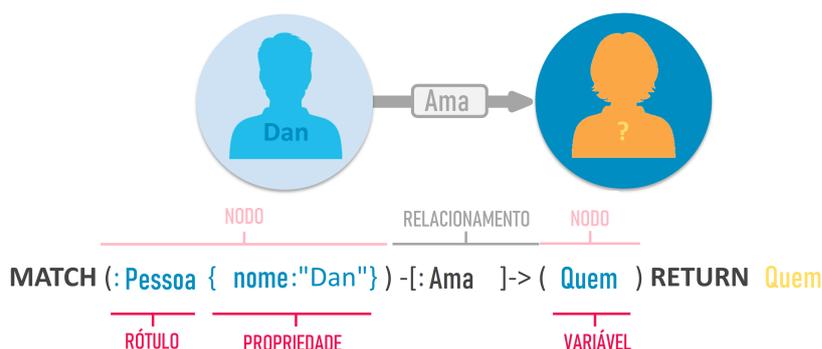


Figura 13 – Exemplo de consulta na linguagem *Cypher*

Fonte: (NEO4J, 2022f)

Consultas com *SQL* que possuem grande quantidade de *joins* tornam-se complexas, dificultando a compreensão e o que realmente pretende-se recuperar. Com a *Cypher* a sintaxe se mantém concisa e focada nos domínios dos dados e nas conexões entre eles (NEO4J, 2022e). A Figura 14 apresenta a equivalência de uma mesma consulta em ambas as linguagens.

```

1 // SQL
2 SELECT name FROM Person
3 LEFT JOIN Person_Department
4     ON Person.Id = Person_Department.PersonId
5 LEFT JOIN Department
6     ON Department.Id = Person_Department.DepartmentId
7 WHERE Department.name = "IT Department"
8
9 // Cypher
10 MATCH (p:Person) -[:WORKS_AT] ->(d:Dept)
11 WHERE d.name = "IT Department"
12 RETURN p.name

```

Figura 14 – Exemplo de equivalência de uma consulta em *SQL* e *Cypher*

Fonte: (NEO4J, 2022f)

*Cypher* é única pois fornece uma maneira visual para encontrar padrões e relacionamentos. Utiliza-se da sintaxe *ASCII-art* para isso, os **nodos** (declarados com parênteses) são conectados a **outros nodos** através de expressões de caminho no formato **-[:ARROWS]->** para relacionamentos. Assim como a *SQL*, há algumas palavras reservadas para a realização de funções básicas como *CRUD* (NEO4J, 2022f), (NEO4J, 2022a). As mais comuns são:

***MATCH***: Equivalente ao *SELECT* em *SQL*, é utilizado para pesquisar nodos, relacionamentos, rótulos, propriedades ou padrões no BD. Em uma consulta é possível

retornar: todos nodos com um rótulo específico, um nodo em particular, todos os nodos com determinado relacionamento, entre outros. A Figura 15 representa a seleção de nodos com o rótulo *Pessoa*.

```
1 MATCH (:Pessoa)
```

Figura 15 – Exemplo do comando *MATCH* na linguagem *Cypher*

Fonte: Elaborado pelo Autor

**OPTIONAL MATCH:** Possui o mesmo comportamento do *MATCH*. A diferença se dá caso o padrão especificado não seja encontrado, ele é desconsiderado pelo *MATCH*. A Figura 16 exemplifica a seleção de um nodo com a propriedade *nome Homer Simpsons*, que possui ou não o relacionamento com o rótulo: *Pai*.

```
1 MATCH (n:Pessoa {Nome: 'Homer Simpsons'})
2 OPTIONAL MATCH (n)-[:Pai]->(x)
```

Figura 16 – Exemplo do comando *OPTIONAL MATCH* na linguagem *Cypher*

Fonte: Elaborado pelo Autor

**RETURN:** Especifica qual valor é retornado na consulta. Utilizado para retornar nodos, relacionamentos e propriedades. Obrigatório para leitura de informações. A Figura 17 exemplifica o retorno de todos os nodos que possuem o rótulo *Pessoa*, cuja propriedade *nome* é *Homer Simpsons*.

```
1 MATCH (p:Pessoa {nome: 'Homer Simpsons'})
2 RETURN p
```

Figura 17 – Exemplo do comando *RETURN* na linguagem *Cypher*

Fonte: Elaborado pelo Autor

**CREATE:** Equivalente ao *INSERT* em *SQL*. Utilizado para criar nodos e relacionamentos. A Figura 18 exemplifica a criação de um nodo com o rotulo *Pessoa*, cuja propriedade *nome* é *Homer Simpsons*.

```
1 CREATE (:Pessoa {nome: 'Homer Simpsons'})
```

Figura 18 – Exemplo do comando *CREATE* na linguagem *Cypher*

Fonte: Elaborado pelo Autor

**SET**: Equivalente ao *UPDATE* em *SQL*. Utilizado em conjunto com o **MATCH**, pois é necessário encontrar o nodo ou relacionamento a ser atualizado. Permite a criação ou atualização de propriedades. A Figura 19 exemplifica a atualização da propriedade *nascimento* do nodo com rótulo *Pessoa*, cuja propriedade *nome* é *Homer Simpsons*.

```
1 MATCH (p:Pessoa {nome: 'Homer Simpsons'})
2 SET p.nascimento = date('1956-03-12')
```

Figura 19 – Exemplo do comando *SET* na linguagem *Cypher*

Fonte: Elaborado pelo Autor

**DELETE**: Equivalente ao *DELETE* em *SQL*. Utilizado para deletar nodos e relacionamentos. Devido às restrições *ACID*, um nodo com relacionamentos não pode ser deletado. Para isso é necessário utilizar o comando *DETACH* para deletar ambos. Este segundo comando é equivalente ao *CASCADE* em *SQL*. A Figura 20 exemplifica a exclusão de todos os nodos com rótulo *Pessoa*, cuja propriedade *nome* é *Homer Simpsons* e seus respectivos relacionamentos.

```
1 MATCH (p:Pessoa {nome: 'Homer Simpsons'})
2 DETACH DELETE p
```

Figura 20 – Exemplo do comando *DELETE* na linguagem *Cypher*

Fonte: Elaborado pelo Autor

**WHERE**: Equivalente ao *WHERE* em *SQL*. Faz parte do **MATCH** e suporta operadores *booleanos* (*AND*, *OR*, *XOR*, *NOT*), operações com intervalos, pesquisa e verificação de *strings*, entre outros. A Figura 21 exemplifica a filtragem dos nodos cuja propriedade *nome* é *Homer Simpsons*.

```
1 MATCH (n)
2 WHERE n.nome = 'Homer Simpsons'
3 RETURN n
```

Figura 21 – Exemplo do comando *WHERE* na linguagem *Cypher*

Fonte: Elaborado pelo Autor

**WITH**: Permite aproveitar o resultado de consultas em outras consultas. Permite subdividir uma grande consulta em consultas menores. A Figura 22 exemplifica a reutilização do resultado da filtragem de nodos que possuem a propriedade *sobrenome Simpsons*.

```
1 MATCH (p:Pessoa)
2 WHERE p.sobrenome = 'Simpsons'
3 WITH p
4 ORDER BY p.nome DESC
5 LIMIT 3
6 RETURN p.nome
```

Figura 22 – Exemplo do comando *WITH* na linguagem *Cypher*

Fonte: Elaborado pelo Autor

**REMOVE:** Utilizada para remover propriedades dos nodos, relacionamentos e os rótulos dos nodos. A Figura 23 exemplifica a remoção da propriedade *idade* de todos os nodos com rótulo *Pessoa*, cuja propriedade *nome* é *Homer Simpsons*.

```
1 MATCH (p:Pessoa {nome: 'Homer Simpsons'})
2 REMOVE p.idade
```

Figura 23 – Exemplo do comando *REMOVE* na linguagem *Cypher*

Fonte: Elaborado pelo Autor

**CALL:** Permite executar subconsultas dentro de outras consultas. Subconsultas permitem compor consultas quando envolvem uniões e agregações. A Figura 24 exemplifica a união de duas consultas diferentes em um único resultado.

```
1 CALL {
2   MATCH (p:Pessoa {nome: 'Homer Simpsons'})
3   RETURN p
4 UNION
5   MATCH (p:Pessoa {nome: 'Ned Flanders'})
6   RETURN p
7 }
8 RETURN p.nome, p.idade
```

Figura 24 – Exemplo do comando *CALL* na linguagem *Cypher*

Fonte: Elaborado pelo Autor

### 3 TRABALHOS CORRELATOS

Este capítulo apresenta, de forma sucinta, soluções existentes para o mapeamento de BDRs para BD orientado a grafos.

#### 3.1 A UNIFIED METAMODEL FOR NOSQL AND RELATIONAL DATABASES

Esta abordagem é a que mais se aproxima da solução desenvolvida neste trabalho. O autor propõe a criação de um metamodelo unificado, denominado *U-Schema*, capaz de representar esquemas lógicos para os quatro modelos de dados para BDs *NoSQL*. O trabalho apresenta a definição formal dos mapeamentos entre *U-Schema* e os modelos de dados (CANDEL; SEVILLA RUIZ; GARCÍA-MOLINA, 2022). Utiliza-se uma correspondência entre cada elemento de um modelo de dados e os elementos do *U-Schema*.

A aplicação do metamodelo unificado tem o objetivo de dar suporte a interface de diversos SGBDs. A Figura 25 exemplifica as etapas de extração de esquemas genéricos.

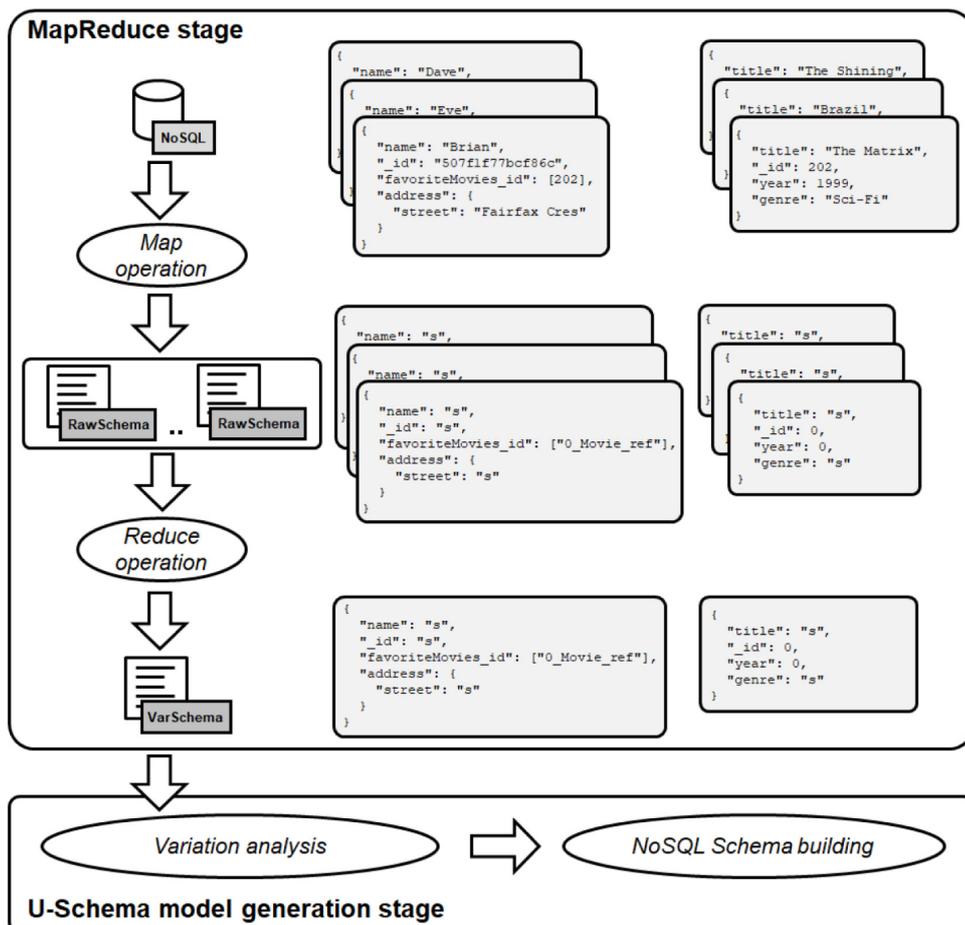


Figura 25 – Estratégia de extração de esquema genérico

Fonte: (CANDEL; SEVILLA RUIZ; GARCÍA-MOLINA, 2022)

Na operação *map*, um esquema bruto é obtido para cada objeto armazenado no BD. A definição de esquema bruto é uma representação intermediária no formato *JSON* que descreve a estrutura de dados. Ela é constituída por um conjunto de pares formado pelo nome de uma propriedade e seu tipo. Dado um objeto armazenado no BD de um tipo de entidade, seu esquema bruto é obtido.

Uma vez que a função *map* é executada, a função *reduce* coleta todos os esquemas brutos idênticos, gerando um único esquema bruto para cada variação estrutural.

Na segunda etapa, os esquemas de variação são analisados para construir o modelo *U-Schema*. Os esquemas de variação são analisados para identificar suas partes constituintes: propriedades e relacionamentos, bem como o tipo de entidade (ou tipo de relacionamento) ao qual eles pertencem. Sempre que o analisador reconhece uma parte, ele a passa para um construtor encarregado de criar o esquema.

Um construtor foi implementado para cada modelo de dados, que captura como as partes são mapeadas para *U-Schema*. O mesmo analisador é usado para todos os modelos de dados, pois sua entrada são esquemas de variação. No caso de BDRs, apenas esta segunda etapa é necessária, pois os esquemas já estão previamente definidos.

Embora o foco da abordagem seja muito parecido, este trabalho se diferencia pelo modelo de mapeamento que atua sobre a modelagem lógica do BD. A solução proposta neste trabalho tem como abordagem a tradução da modelagem física *SQL* para a linguagem *Cypher*.

### 3.2 AUTOMATIC NOSQL TO RELATIONAL DATABASE TRANSFORMATION WITH DYNAMIC SCHEMA MAPPING

Nesta solução os autores propõem um modelo de *Extract, Transform, Load (ETL)*. A ferramenta desenvolvida, *Schema Analyzer*, faz a extração dos dados de origem e, em seguida, define dinamicamente o esquema para preenchê-lo no SGBD de destino (AFTAB *et al.*, 2020). Uma vez realizada esta tarefa, o sistema transforma os dados no formato de destino e carrega os dados em lotes.

A ferramenta proposta pode ser facilmente usada por profissionais que não atuam na área e que precisam realizar a transformação de dados *NoSQL* para *SQL* com baixo custo, pouco esforço e em pouco tempo. A avaliação experimental utiliza diferentes tamanhos de BDs *NoSQL* e apresenta um excelente desempenho em comparação com o *Talend Open Studio*<sup>1</sup>, uma ferramenta de última geração em termos de BDs *NoSQL* para tarefas *ETL* de BDRs.

Embora essa abordagem se aproxime deste trabalho no que tange a terminologia, seu propósito é o oposto do pretendido por este trabalho, ou seja, a solução *Schema Analyzer* realiza a criação de um esquema relacional a partir de um esquema *NoSQL*.

<sup>1</sup> <https://www.talend.com/products/talend-open-studio/>

### 3.3 TOWARDS UNIFIED MODELING FOR NOSQL SOLUTION BASED ON MAPPING APPROACH

Neste artigo o autor apresenta uma técnica de mapeamento de esquemas relacionais para os modelos de dados *NoSQL*. Em relação ao mapeamento para grafos, o autor define um conjunto de regras. Definem-se os rótulos que correspondem aos nomes das tabelas, cada tupla no BDR corresponde a um nodo no esquema de grafo, as colunas das tabelas relacionais podem ser armazenadas como propriedades, ignorando-se quaisquer colunas que tenham valores nulos. Chaves primárias podem ser impostas usando restrições exclusivas e chaves estrangeiras são substituídas por arestas que representam o relacionamento entre tabelas (GUEIDI; GHARSELLAOUI; AHMED, 2021). Seu ponto fraco é que ele apresenta uma proposta teórica, ou seja, não foi desenvolvida uma aplicação para validar a técnica.

### 3.4 CONSIDERAÇÕES FINAIS

Após pesquisas para correlacionar trabalhos que atuassem na mesma abordagem que a solução proposta, identificou-se que esta é a primeira aplicação que engloba de maneira unificada o mapeamento de esquemas relacionais, baseando-se no mapeamento físico, para os quatro modelos de dados para BDs *NoSQL* (CATTELL, 2011), além de suportar a conversão de um subconjunto de instruções *SQL* para instruções equivalentes nos BDs *NoSQL*. Este trabalho, em particular, provê esse mapeamento para BDs *NoSQL* orientado a grafos, modelo *NoSQL* ainda não suportado pela abordagem *SQLToKeyNoSQL*. Constata-se também a falta de soluções comerciais que resolvam a mesma problemática abordada neste trabalho. A Figura 26 sintetiza as principais semelhanças e diferenças entre os trabalhos publicados e a solução proposta neste trabalho.

	Proposta Teórica	Modelos NoSQL	Mapeamento para Grafo	Mapeamento Lógico	Mapeamento de Instrução SQL
A Unified Metamodel For Nosql And Relational Databases	✓	✓	✓	✓	✗
Automatic Nosql To Relational Database Transformation With Dynamic Schema Mapping	✓	✓	✓	✗	✗
Towards Unified Modeling For Nosql Solution Based On Mapping Approach	✓	✗	✗	✗	✗
Sqltonokey-graph	✓	✓	✓	✓	✓

Figura 26 – Tabela comparativa entre a proposta e os trabalhos correlatos

Fonte: Elaborado pelo Autor

## 4 SQLTOKEYNOSQL

*SQLToKeyNoSQL* (SCHREINER, 2016) fornece uma camada de abstração através de um modelo canônico hierárquico pelo qual é possível realizar operações de *CRUD* em BDs *NoSQL* utilizando *SQL*. Ele realiza o mapeamento do esquema relacional para um esquema canônico intermediário, e deste para cada um os esquemas dos BDs *NoSQL*. A abordagem suporta um conjunto restrito de instruções *DDL* e *DML* que são também mapeadas para instruções correspondentes nos BDs *NoSQL* suportados através de um conector genérico baseado em verbos da *API REST*.

A camada prove suporte a três tipos de BDs *NoSQL*: chave-valor, orientado a colunas e orientado a documentos. Todos eles são BDs baseados em chave de acesso e não possuem rigidez no esquema de dados. Suas interfaces de acesso são simples, permitindo inserção, exclusão e busca por chave. Geralmente cada modelo implementa a sua própria interface não havendo um padrão. Apesar da alta disponibilidade, escalabilidade e flexibilidade, esses BDs *NoSQL* não oferecem suporte a *SQL*. Assim sendo, a camada propõe a manipulação de dados relacionais quando a intenção é portá-los para BD *NoSQL*.

A abstração da camada proposta se dá pela utilização de um *modelo canônico*. Ele consiste em um conjunto de chaves e valores organizados hierarquicamente, e possui três níveis a partir da raiz, sendo capaz de representar um esquema de BD. Para mais detalhes e definições de mapeamento consultar a dissertação *SQLToKeyNoSQL* (SCHREINER, 2016). A Figura 27 apresenta um exemplo de abstração feita no modelo canônico (B) a partir do modelo relacional (A), para posteriormente ser utilizada na representação da estrutura *NoSQL* destino.

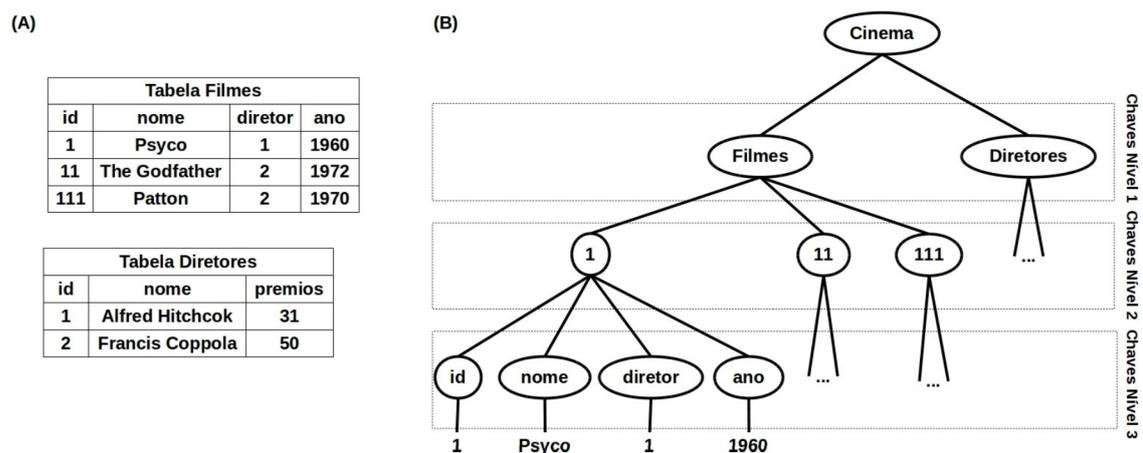


Figura 27 – Modelo canônico

Fonte: (SCHREINER, 2016)

A arquitetura proposta pela abordagem *SQLToKeyNoSQL* é mostrada na Figura 28. Seus componentes são:

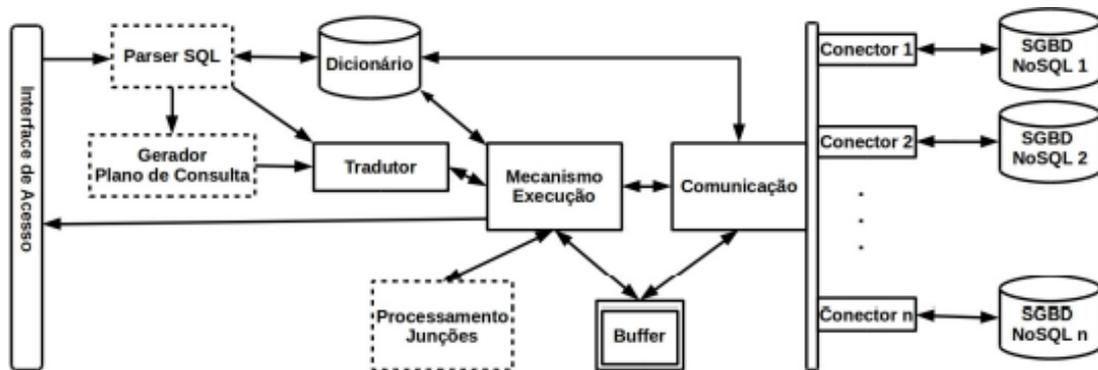


Figura 28 – Arquitetura da solução *SQLToKeyNoSQL*

Fonte: (SCHREINER, 2016)

- **Interface de Acesso:** ambiente gráfico desenvolvido em *Java* utilizado pelo usuário como forma de acesso. Através dela o usuário pode fornecer instruções *SQL*;
- **Parser SQL:** verificação sintática e semântica da instrução *SQL* recebida como entrada;
- **Tradutor:** para instruções do tipo *DDL* ou do tipo *INSERT*, ele realiza o seu mapeamento para as interfaces de acesso (*Get*, *Put*, *Delete* e *GetN*);
- **Gerador de Plano de Consulta:** para instruções tipo *SELECT*, *UPDATE* ou *DELETE*, ele otimiza o processamento de consultas que possuem filtros conectados exclusivamente pelo operador lógico *AND* através da execução antecipada de cada filtro;
- **Mecanismo de Execução:** recebe os métodos gerados pelo módulo **Tradutor** e os executa por meio do módulo de **Comunicação**. Também é responsável por executar cada filtro e enviar conjuntos de dados ao módulo de **Processamento de Junções**;
- **Comunicação:** Executa as solicitações do **Mecanismo de Execução**, através do envio de métodos para os **Conectores**, e encaminha a resposta obtida ao **Mecanismo de Execução** através do **Buffer**;
- **Processamento de Junções:** junções são executadas exclusivamente pela *SQLtoKeyNoSQL*. Ele oferece flexibilidade para otimização e inserção de outros algoritmos;
- **Buffer:** utilizado apenas quando o tamanho da tabela não é comportado na memória. Neste caso, utiliza-se o disco para salvar os arquivos;

- **Dicionário:** armazena metadados relativos aos esquemas relacionais de origem (BDs, tabelas, atributos, chaves primárias e chaves estrangeiras);
- **Conector:** fornece a conexão entre o módulo de **Comunicação** e a interface específica do BD *NoSQL*.

Como comentado anteriormente, a camada funciona primeiramente mapeando as instruções *DDL* para o modelo canônico (esquema rígido), e posteriormente mapeando as instruções *DML* do esquema canônico para o esquema do BD *NoSQL* através de comandos baseados nos verbos na *API REST*. As instruções *DDL* suportadas são:

- **CREATE TABLE:** cria as definições da tabela relacional no dicionário, mantendo nome da tabela, atributos, chaves primárias e estrangeiras;
- **ALTER TABLE:** considera a alteração de nome de coluna, bem como a adição e remoção de uma coluna. As respectivas definições são atualizadas no dicionário. Não são permitidas alterações de chaves primárias.
- **DROP TABLE:** apaga as informações da tabela no dicionário.

As instruções *DML* são decompostas em um ou mais métodos baseados nos verbos da *API REST*:

- **Put:** armazena o registro no BD;
- **Get:** busca o registro do BD;
- **Delete:** deleta um registro do BD;
- **GetN:** decomposto em uma série de métodos *Get*.

As instruções *DML* suportadas são as seguintes:

- **INSERT:** é mapeado para o método *Put*. Ele usa como base as informações presentes no dicionário e não suporta subconsultas;
- **UPDATE:** decomposto em métodos *Get*, *Delete* e *Put*. A atualização de uma ou várias tuplas é possível através de filtros simples. Um exemplo de mapeamento da instrução *UPDATE* é mostrado na Figura 29;

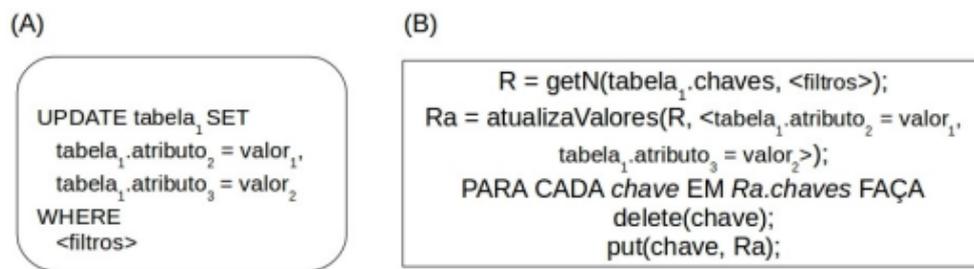


Figura 29 – Exemplo da instrução *UPDATE* convertida em métodos *GetN*, *Delete* e *Put*

Fonte: (SCHREINER, 2016)

- **DELETE**: decomposto em métodos *Get* e *Delete*. A exclusão de uma ou várias tuplas é possível através de filtros simples;
- **SELECT**: decomposto em uma série de métodos *GetN*. O retorno é um conjunto de tuplas recuperadas a partir de filtros simples. Há o suporte para junções. A Figura 30 apresenta exemplos de conversão da instrução *SELECT* sem (A) e com (B) junção.

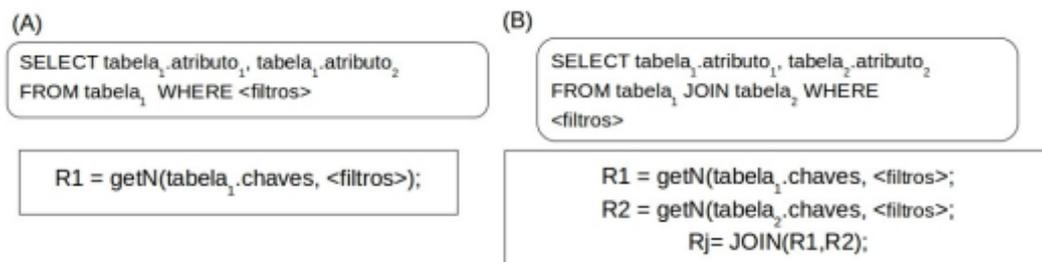


Figura 30 – Exemplo de instruções *SELECT* convertidas em métodos *GetN*

Fonte: (SCHREINER, 2016)

Este trabalho estende a *SQLtoKeyNoSQL* oferecendo suporte ao mapeamento de esquemas relacionais para esquemas *NoSQL* orientados a grafos, com foco no SGBD *Neo4j*, bem como o mapeamento das instruções *SQL* supracitadas para instruções *Cypher* correspondentes. A proposta deste trabalho é detalhada no próximo capítulo.

## 5 SQLTONOSQL-GRAPH

A proposta deste trabalho consiste em acoplar mais um conector à camada *SQLToKeyNoSQL*, utilizando a mesma arquitetura e os mesmos módulos. O novo conector fornece a ligação entre módulo de **Comunicação** existente e a interface específica do BD orientado a grafos *Neo4j*. Essa comunicação se dá pela implementação da interface fornecida pela camada *SQLToKeyNoSQL*, cujos métodos são mapeados para comandos na linguagem *Cypher*, suportada pelo *Neo4j*.

Conforme descrito no Capítulo 4, a interface de comunicação da camada *SQLToKeyNoSQL* com os BDs *NoSQL* disponibiliza apenas a manipulação de instruções *DML* através dos quatro comandos: *Get*, *GetN*, *Put* e *Delete*. Instruções *DDL* apenas manipulam as informações presentes no dicionário. Apesar da camada suportar as instruções *DDL ALTER* e *DROP TABLE*, o código fonte disponibilizado não dispõe de tais implementações. A fim de suprir estas lacunas é proposta a implementação destas instruções com melhorias:

- **CREATE TABLE**: cria as definições da tabela relacional no dicionário, considerando o nome da tabela, atributos, chaves primárias e estrangeiras. Não é permitido executar duas instruções para a mesma tabela. Adicionalmente, é disponibilizado o comando *CREATE* na interface de comunicação com o BD para que o BD possa realizar parametrizações e ajustes se necessário.
- **ALTER TABLE**: contempla a alteração de nome, adição e remoção de uma coluna. Não são permitidas alterações de chaves primárias. As novas definições são atualizadas no dicionário. Na operação de adição é permitido apenas a inclusão de colunas simples, sem propriedades com chave estrangeira. A operação de remoção contempla a exclusão de uma coluna existente e suas respectivas chaves estrangeiras. É possível também alterar o nome de uma coluna para outra ainda não existente na tabela, assim como suas respectivas chaves estrangeiras. Além disso, é permitido alterar múltiplas colunas com a mesma instrução. Adicionalmente é disponibilizado o comando *ALTER* na interface de comunicação com o BD para que as alterações sejam executadas no BD.
- **DROP TABLE**: remove as informações referentes à tabela armazenadas no dicionário: nome, atributos, chaves primárias e estrangeiras. Adicionalmente é disponibilizado o comando *DROP* na interface de comunicação com o BD para que as deleções sejam refletidas no BD.

Para as instruções *DML* previstas pela camada *SQLToKeyNoSQL* na interface de comunicação com os BDs é previsto manter os mesmos comportamentos com as seguintes modificações:

- **PUT**: este comando inicialmente previa o suporte para inserção de somente uma chave. Nesta modificação é proposto o suporte à inserção de uma ou mais chaves através da conexão com o BD, possibilitando a execução de forma mais performática com a utilização da função *InsertN*;
- **DELETE**: O comando inicialmente previa o suporte para a exclusão de somente uma chave. Nesta modificação, é proposto o suporte a exclusão de uma ou mais chaves através da conexão com o BD, possibilitando a execução de forma mais performática.
- **UPDATE**: comando inicialmente realizado através da operação de exclusão, seguida de uma inserção. Tal comportamento foi movido para a interface de comunicação com o BD, pois alguns SGBDs não possuem suporte nativo a esta instrução, permitindo, assim, que os que possuem o suporte à instrução possam realizá-la de forma mais performática. Além disso, é disponibilizada a alteração de múltiplas chaves na interface de comunicação com o BD.

## 5.1 REGRAS DE MAPEAMENTO PARA O MODELO ORIENTADO A GRAFOS

A inspiração para o conjunto de regras de mapeamento proposto neste trabalho é fruto do conhecimento de *SQL* e BDRs, bem como o funcionamento e restrições deste modelo, associado ao processo empírico na transformação de dados relacionais para a estrutura equivalente no novo formato de dados. Deste modo, foram definidas as seguintes regras de mapeamento entre os dois modelos:

- **Regra 1 (Mapeamento de Tabela)**: dada uma tabela com nome *T*, gera-se um tipo de nodo com rótulo *T* no *BD* orientado a grafos;
- **Regra 2 (Mapeamento de Registros)**: dada uma tupla em um BD relacional, gera-se um nodo no *BD* orientado a grafos;
- **Regra 3 (Mapeamento de Atributos)**: para cada coluna com nome *NC* de um registro *R* em uma tabela de um BDR, gera-se uma propriedade com nome *NC* no nodo gerado para *R* no *BD* orientado a grafos;
- **Regra 4 (Mapeamento de Chave Primária)**: dada uma chave primária de uma tabela em um BDR, gera-se uma propriedade cujo valor é único para todos os nodos que possuem o mesmo rótulo no *BD* orientado a grafos;
- **Regra 5 (Mapeamento de Chave Estrangeira)**: dada uma coluna com nome *NC* que atua como chave estrangeira de uma tabela em um BDR, gera-se uma aresta entre os nodos com rótulo *NC*;

- **Regra 6 (Mapeamento de Relacionamentos):** dado um relacionamento entre duas tabelas, ambos os nodos desta relação devem estar previamente criados no BD orientado a grafos.

A Figura 31 representa um conjunto de dados relacionais e ao lado o mapeamento para a estrutura equivalente em um BD *NoSQL* orientado a grafos. De acordo com a regra de mapeamento 2, cada nodo amarelo é o resultado da conversão de uma tupla relacional presente na tabela *Personagens*, assim como a tupla da tabela *Familias*, que dá origem ao nodo azul. As arestas criadas são resultantes da aplicação da regra número 5. A coluna *FamiliaId* é uma chave estrangeira. Deste modo, o nome da coluna foi mapeado para o rótulo da aresta.

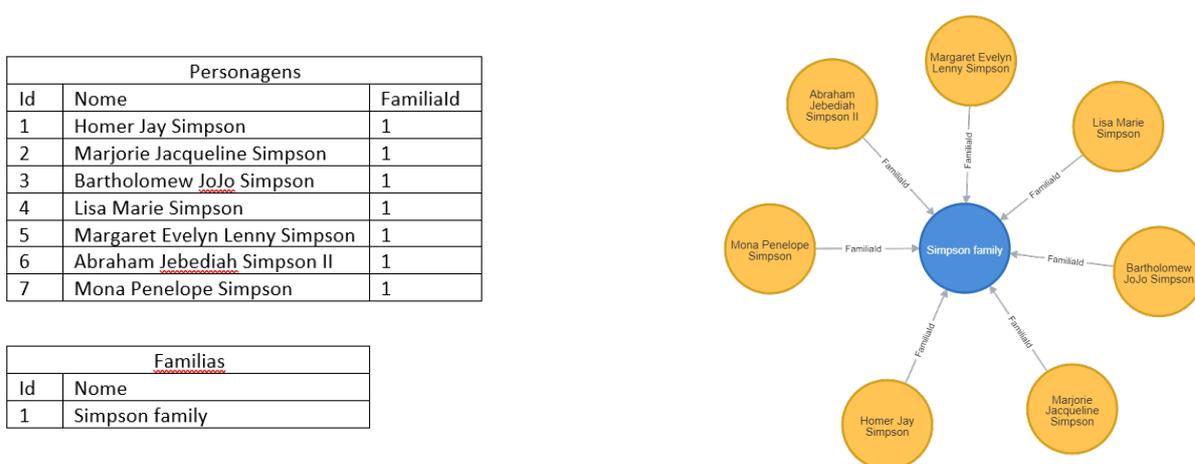


Figura 31 – Exemplo do mapeamento *SQL* para *Neo4j*

Fonte: Elaborado pelo Autor

## 5.2 MAPEAMENTO DE INSTRUÇÕES *DDL*

Comandos *DDL* foram adicionados à interface de comunicação com o BD com o objetivo de garantir a integridade dos dados. Para contemplar esta premissa é realizada a tradução para os comandos na linguagem *Cypher* das seguintes instruções:

- **CREATE:** uma *CONSTRAINT* do tipo chave única é criada para garantir a aplicação da regra de mapeamento número 4. Esta chave é definida a partir de uma propriedade sintética denominada *NODE\_KEY*, cujo valor é resultante da combinação de chaves primarias definidas no esquema relacional.

A Figura 32 exemplifica a criação da *CONSTRAINT funcao\_NODE\_KEY*, na qual os nodos do tipo *funcao* possuem o valor da propriedade *NODE\_KEY* definida como única para todos os nodos de mesmo tipo, presentes no BD *Neo4j*.

```

1 CREATE CONSTRAINT funcao_NODE_KEY
2 IF NOT EXISTS FOR (n:funcao)
3 REQUIRE (n.NODE_KEY) IS UNIQUE

```

Figura 32 – Exemplo de criação da *CONSTRAINT* para o método *CREATE*

Fonte: Elaborado pelo Autor

- **ALTER**: esta operação suporta apenas a alteração de colunas (adição, modificação ou exclusão). A adição não apresenta alteração imediata no BD, pois o modelo é flexível, não sendo necessária a declaração de todas as propriedades em um nodo. A operação *DROP COLUMN* é dividida em simples (quando a coluna não é uma chave estrangeira) e complexa (quando a coluna é uma chave estrangeira).

A Figura 33 exemplifica a remoção de duas colunas, uma com chave estrangeira e outra simples, sendo feita a seleção de todos os nodos (*n*) do tipo *funcionario* que possuem ou não (*OPTIONAL*) o relacionamento do tipo *banco\_id*. Em seguida, as propriedades *nome* e *banco\_id* são removidas do nodo, em seguida executa-se a deleção do relacionamento *banco\_id*.

```

1 MATCH (n:funcionario)
2 OPTIONAL MATCH (n:funcionario) -[banco_id:banco_id]-> (nbanco_id)
3 REMOVE n.nome
4 REMOVE n.banco_id
5 DELETE banco_id

```

Figura 33 – Exemplo do comando *DROP COLUMN* para o método *ALTER*

Fonte: Elaborado pelo Autor

No caso de um *RENAME COLUMN*, as propriedades não podem ser renomeadas no *Neo4j*. Neste caso, é necessário criar uma nova propriedade com o novo nome, mantendo o valor e em seguida excluir a propriedade antiga. Para propriedades com chave estrangeira, é feito o mesmo processo. Neste caso, é criado um novo relacionamento com o novo tipo e excluído o relacionamento com o tipo antigo.

A Figura 34 exemplifica a renomeação de duas colunas, uma com chave estrangeira e outra sem, sendo feita a cópia das propriedades *nome* e *banco\_id* para as propriedades *nome\_novo* e *banco\_id\_novo*, respectivamente. Em seguida, é feita a remoção das propriedades *nome* e *banco\_id*. Na sequência é criado o relacionamento *banco\_id\_novo* e removido o relacionamento antigo *banco\_id*.

```

1 MATCH(n:funcionario)
2 OPTIONAL MATCH(n:funcionario) -[banco_id:banco_id]-> (nbanco_id)
3 SET n.nome_novo = n.nome
4 REMOVE n.nome
5 SET n.banco_id_novo = n.banco_id
6 REMOVE n.banco_id
7 CREATE (n)-[:banco_id_novo]->(nbanco_id)
8 DELETE banco_id

```

Figura 34 – Exemplo do comando *RENAME COLUMN* para o método *ALTER*

Fonte: Elaborado pelo Autor

- ***DROP***: o *Neo4j* apresenta um comportamento *default* similar aos BDRs, ou seja, não é permitido a remoção de nodos que são referenciados por outros nodos. A exclusão é permitida somente nos casos em que o nodo não seja apontado por outro. Neste caso, são excluídos os nodos e seus respectivos relacionamentos. Após a remoção de todos os nodos é feita a exclusão da *CONSTRAINT* de chave única. A Figura 35 exemplifica a exclusão de todos os nodos (*n*) do tipo *cliente* que possuam ou não (*OPTIONAL*) quaisquer tipos de relacionamento (*chave\_estrangeira*) com outros nodos (*ce*). Em seguida, é feita a exclusão da *CONSTRAINT* (*cliente\_NODE\_KEY*) caso exista.

```

1 MATCH(n:cliente)
2 OPTIONAL MATCH(n:cliente) -[chave_estrangeira]-> (ce)
3 DELETE n, chave_estrangeira;
4
5 DROP CONSTRAINT cliente_NODE_KEY IF EXISTS

```

Figura 35 – Exemplo da instrução *DROP TABLE*

Fonte: Elaborado pelo Autor

### 5.3 MAPEAMENTO DE INSTRUÇÕES *DML*

Conforme descrito no Capítulo 5.2, além dos comandos mencionados no Capítulo 4 foram adicionados novos métodos à interface de comunicação com o BD. As operações de filtragem e junções são realizadas pela camada *SQLToKeyNoSQL*. Nos exemplos de cada comando é descrito a instrução *SQL* e seu respectivo mapeamento para a linguagem *Cypher*. Para manipular os dados é realizada a tradução das seguintes instruções:

- ***PUT (INSERT)***: Diferentemente de outros BDs *NoSQL* suportados, o *Neo4j* gerencia relacionamentos entre os dados armazenados. A inserção de dados é dividida em dois tipos: simples e complexa. A inserção simples ocorre quando o nodo não tem relacionamento com outro nodo. Neste caso, é realizado o mapeamento direto

para a instrução *CREATE* na linguagem *Cypher*, definindo o tipo do nodo e seus atributos.

A inserção complexa ocorre quando há relacionamentos entre os nodos. Neste caso, a criação do nodo só pode ser feita se ambos os nodos da relação já estiverem inseridos no BD. Após a execução da instrução é verificado se o nodo e seus respectivos relacionamentos foram criados como esperado. Do contrário, o comando é invalidado.

A Figura 36 exemplifica o mapeamento da instrução *INSERT*. Pressupõe-se que existe uma tabela *item\_venda* que possui uma chave estrangeira com a tabela *item\_produto*. Inicialmente é verificado se os nodos da relação existem. Em seguida é realizada a criação do nodo com o tipo *item\_venda*.

```
1 //SQL
2 INSERT INTO Item_venda (id_Item,id_produto,quant_vendida, prec_unitario)
3 VALUES (1,1,1,100);
4
5 // CYPHER
6 MATCH
7 (produto1:produto{NODE_KEY:1})
8 CREATE
9 (item_venda1:item_venda {
10     NODE_KEY:1,
11     id_item:1,
12     id_produto=1
13     quant_vendida:1,
14     prec_unitario:100
15     }),
16 (item_venda1)-[id_produto:id_produto]->(produto1)
```

Figura 36 – Exemplo de mapeamento da instrução *INSERT INTO*

Fonte: Elaborado pelo Autor

- **UPDATE**: A alteração de dados também é dividida em dois tipos: simples e complexa. A alteração simples ocorre quando o nodo não tem relacionamento com outro nodo. Neste caso, o valor da propriedade é alterado diretamente. Nos casos complexos, em que há chave estrangeira, é verificado se o novo nodo do relacionamento já existe no BD. Em seguida são deletados e recriados todos os seus relacionamentos.

A Figura 37 exemplifica o mapeamento da instrução *UPDATE*. Pressupõe-se que existe uma tabela *usuario* que possui uma chave estrangeira com a tabela *funcionario*. Inicialmente é verificado se existe um nodo com o rótulo *funcionario* que possua a propriedade *id '1'*. Caso tenha sucesso, todas as propriedades são alteradas através do comando *SET+*. Seus relacionamentos são deletados e recriados em seguida com os novos valores.

```

1 //SQL
2 UPDATE usuario
3 SET func_id=1,
4     user_log='adm (Atualizado)',
5     user_pwd='adm';
6
7 //CYPHER
8 //1 VERIFICATION
9 MATCH (n:funcionario)
10 WHERE n.id = 1
11 RETURN (n)
12
13 //2 INSERTION
14 MATCH (n:usuario)
15 WHERE n.NODE_KEY in [1, 2]
16 SET n +={
17     user_log:'adm (atualizado)',
18     func_id:1,
19     user_pwd:'adm'
20 }
21
22 WITH n
23 CALL
24 {
25     WITH n
26     MATCH (n)-[relacionamento]->(relacionamento_apontado)
27     WHERE type(relacionamento) in ["func_id"]
28     DELETE relacionamento
29 }
30
31 WITH (n)
32 MATCH (funcionario_id_func_id:funcionario)
33 WHERE funcionario_id_func_id.id = 1
34 CREATE (n)-[:func_id]->(funcionario_id_func_id)

```

Figura 37 – Exemplo de mapeamento da instrução *UPDATE*

Fonte: Elaborado pelo Autor

- **DELETE**: Pelo fato de um BD orientado a grafos conter relacionamentos entre os dados, não é possível excluir um nodo que mantém relacionamentos ativos. Conforme descrito no início deste capítulo, é habilitado o suporte para o envio de múltiplas chaves. Desta forma, é possível executar a exclusão em uma única consulta especificando apenas as chaves a serem excluídas. Seu funcionamento é similar ao comportamento do comando *DROP*, no entanto, não é necessário realizar a remoção da *CONSTRAINT* de chave única.

A Figura 38 exemplifica o mapeamento da instrução *DELETE*. A exclusão dos nodos é realizada utilizando a chave sintética *NODE\_KEY*. São selecionados todos os nodos do tipo *item\_venda* que possuem ou não (*OPTIONAL*) algum tipo de relacionamento, cujas chaves são fornecidas pela camada *SQLToKeyNoSQL*. Em seguida são excluídos os nodos e seus respectivos relacionamentos.

```
1 //SQL
2 DELETE FROM Item_venda;
3
4 /CYPHER
5 MATCH(n:item_venda)
6 OPTIONAL MATCH(n:item_venda) -[arestas]-> (ce)
7 WHERE n.NODE_KEY in [1, 2]
8 DELETE arestas,n
```

Figura 38 – Exemplo de mapeamento da instrução *DELETE*

Fonte: Elaborado pelo Autor

- ***GET (SELECT)***: Esta operação apresenta comportamento de baixa complexidade, responsável pela recuperação de determinado nodo a partir da chave sintética *NODE\_KEY*.
- ***GETN (SELECT)***: É a sobre-escrita do método *GETN* disponibilizado pela interface de comunicação com o BD da camada *SQLToKeyNoSQL*. Em uma única consulta, são recuperados todos os nodos do mesmo tipo. Desta forma, o método *GET* deixou de ser utilizado.

A Figura 39 exemplifica o mapeamento da instrução *SELECT*. É realizada a seleção de todos os nodos do tipo *vendas*. Embora sua execução seja simples, ela entrega o resultado de forma mais performática se comparada com a solução inicial prevista na camada de origem *SQLToKeyNoSQL*.

```
1 //SQL
2 SELECT * FROM vendas;
3
4 /CYPHER
5 MATCH(n:vendas) RETURN (n)
```

Figura 39 – Exemplo de mapeamento da instrução *GETN*

Fonte: Elaborado pelo Autor

## 5.4 INTERFACE DO USUÁRIO E FUNCIONALIDADES DA FERRAMENTA

Esta seção descreve a interface gráfica e as funcionalidades da ferramenta desenvolvida para dar suporte a este trabalho. A Figura 40 mostra a tela inicial na qual o usuário insere as informações necessárias para a conexão com um BD *Neo4j*. Cada funcionalidade possui uma descrição ao lado para facilitar a utilização da ferramenta.



**Credenciais de Acesso**

Conector: NEO4j

Usuário: neo4j

Senha: .....

URL: bolt://localhost:7687

Banco de Dados

Editar Salvar

**DESCRIÇÃO**

OBS: É necessário ter uma conexão com MONGO com usuário root e senha root. Esta conexão pode ser utilizada como target.  
Criar conexão com SGBD (exemplo Neo4j)  
Conector: "NEO4j",  
Usuário: "neo4j",  
Senha: "pASsw0rD",  
URL: "bolt://localhost:7687",  
Criar o(s) banco(s) de dados, este será associado ao conector selecionado

Figura 40 – Tela para inserção das credenciais de acesso ao BD *Neo4j*

Fonte: Elaborado pelo Autor

A Figura 41 representa a funcionalidade central da ferramenta: a execução de *queries*. Nesta tela o usuário pode escolher o BDR, selecionar um arquivo ou digitar as instruções *SQL* a serem executadas. Em caráter complementar foi acrescentado um componente que retorna o tempo de execução das instruções. Por fim, foram indicados os comandos *DML* e *DDL* suportados pela ferramenta.

**Queries**

Banco de Dados: matconstru (NEO4j)

Choose File: No file chosen

Executar

Escreva uma expressão SQL

```
SELECT * FROM funcão;  
SELECT * FROM Banco;  
SELECT * FROM funcionario;  
SELECT * FROM usuário;
```

Tempo de Execução: 1,7094s | 0,0285m | 0,0005h

**Realizando o Transferência para Banco NoSQL**

OBS: A camada não faz a leitura automática das definições das tabelas de um

- 1º Importar script de criação.
- 2º Importar script de inserção.

Comandos Suportados:

DDL: CREATE | ALTER | DROP

DML: INSERT | UPDATE | SELECT | DELETE

O Parse SQL utilizado suporta um conjunto limitado de sintaxe, para mais det

Figura 41 – Tela para execução de *queries*

Fonte: Elaborado pelo Autor

Quando o usuário executa mais de um comando do tipo *SELECT*, as informações recuperadas são carregadas em várias tabelas. A Figura 42 exemplifica o componente responsável por exibir aos usuários as informações retornadas pela instrução.

ID	FUNCAO_ID	BANCO_ID	NOME	DATA_NASC	TELEFONE	EMAIL	T_PESSOA	RAZAO_SOCIAL	RG	CPF	ENDERECO
1	1	1	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	peessoa fisica	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
2	2	2	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	peessoa fisica	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
3	3	3	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	peessoa fisica	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
4	4	4	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	peessoa fisica	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
5	5	5	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	peessoa fisica	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
6	6	6	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	peessoa fisica	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
7	7	7	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	peessoa fisica	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
8	8	8	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	peessoa fisica	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
9	9	9	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	peessoa fisica	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
10	10	10	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	peessoa fisica	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
11	11	11	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	peessoa fisica	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae

Mostrando de 1 até 11 de 1.000 registros

Figura 42 – Componente para retorno da query

Fonte: Elaborado pelo Autor

Nos casos em que são retornadas mais de uma tabela é possível selecionar a visualização através do componente de seleção, localizado acima da própria tabela. A Figura 43 exemplifica tal comportamento, possibilitando ao usuário a opção de escolher qual tabela deseja visualizar.

ID	FUNCAO_ID	BANCO_ID	NOME	DATA_NASC	TELEFONE	EMAIL	T_PESSOA	RAZAO_SOCIAL	RG	CPF	ENDERECO
1	1	1	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	peessoa fisica	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
2	2	2	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	peessoa fisica	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
3	3	3	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	peessoa fisica	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
4	4	4	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	peessoa fisica	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
5	5	5	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	peessoa fisica	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
6	6	6	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	peessoa fisica	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
7	7	7	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	peessoa fisica	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
8	8	8	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	peessoa fisica	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
9	9	9	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	peessoa fisica	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
10	10	10	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	peessoa fisica	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
11	11	11	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	peessoa fisica	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae

Mostrando de 1 até 11 de 1.000 registros

Figura 43 – Seleção de visualização das tabelas

Fonte: Elaborado pelo Autor

## 6 ANÁLISE DE DESEMPENHO E RESULTADOS

Este capítulo descreve os experimentos realizados sobre a camada *SQLtoKeyNoSQL* estendida por este trabalho, a fim de avaliar o tempo de *overhead* introduzido pela camada para o processamento e execução de instruções *DDL* e *DML*.

### 6.1 DATASET

O BD utilizado é *GerenciadorMatConstru*<sup>1</sup>, esquema com 17 tabelas baseado em um comércio de materiais de construções e lida com as cadeias de controle de cadastros, acessos e estrutura de venda (Figura 44). Ele foi escolhido por ser um BD com acesso livre e permitir a aplicação das regras de mapeamento propostas na seção 5.2.

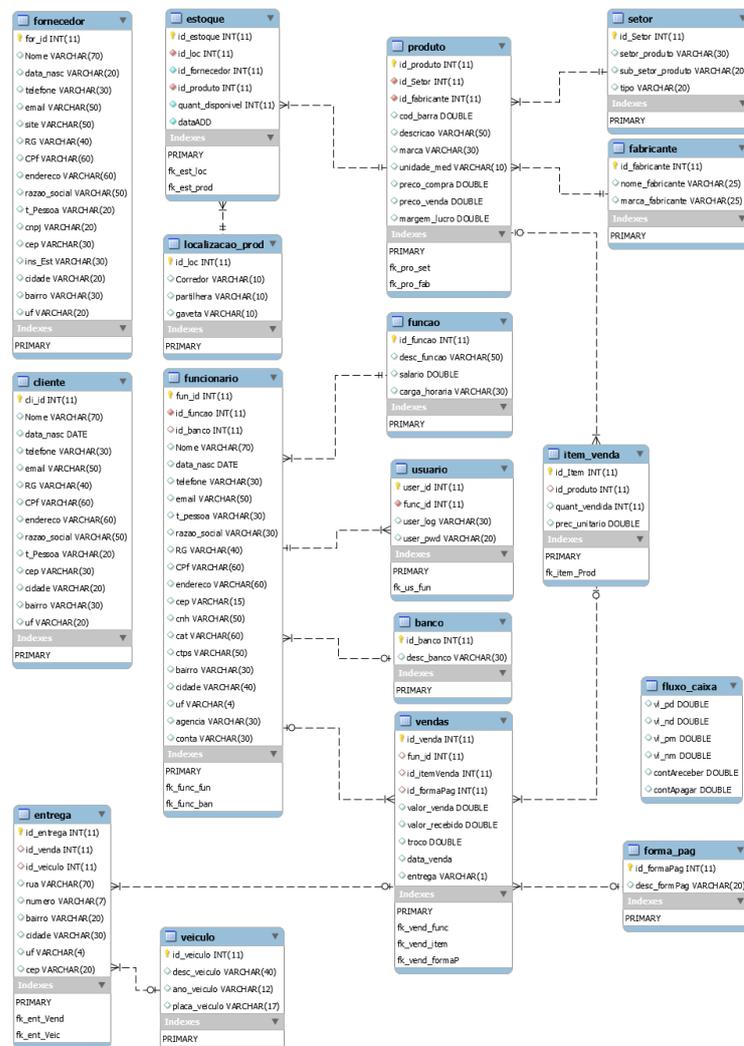


Figura 44 – Esquema do BDR utilizado nos experimentos

Fonte: Elaborado pelo Autor

<sup>1</sup> <https://github.com/lucasOliveira20/GerenciadorMatConstru>

## 6.2 CONFIGURAÇÃO DO AMBIENTE

Os testes foram realizados em um computador com um processador Intel(R) Core(TM) i5-7300HQ, 2.50GHz, 24 GB de RAM 2.400 MHz, GPU Nvidia GTX 1050 4GB, rodando o sistema operacional Windows(R) 11 Education 22H2. Para a análise do *overhead* foram utilizadas três medições: (i) o tempo gasto pela camada *SQLtoKeyNoSQL* para o processamento das instruções; (ii) o intervalo de tempo gasto para as verificações e construção das instruções e; (iii) o tempo de execução do comando no *Neo4j*.

## 6.3 MÉTRICAS DE AVALIAÇÃO

Com base no esquema original do BDR, foi criado um gerador sintético de instruções *SQL* do tipo *INSERT* e *INSERTN*, que permitem a parametrização do número de registros por tabela. A Figura 45 exemplifica a criação de 10 mil inserções do tipo *INSERTN*.

```
-----
GERADOR SQL BANCO DB_MATCONSTRU
-----
Quantidade CONSULTAS?
10_000
-----
Tipo de Consulta
[1] INSERT N
[2] INSERT
-----
1
Gerando consultas...
00:00:00.1478016
ARQUIVO: D:\TCC\SQLtoKeyNoSQL-Grafo\Extensions\GeradorSQL\bin\Debug\net6.0\Consultas\InsertN_170000.sql
-----
GERADOR SQL BANCO DB_MATCONSTRU
-----
Quantidade CONSULTAS?
```

Figura 45 – Gerador sintético de instruções de inserção

Fonte: Elaborado pelo Autor

Para realização dos testes foram construídos os seguintes cenários:

- Instruções com 1 mil registros para cada tabela (17 mil registros no total);
- Instruções com 3 mil registros para cada tabela (51 mil registros no total);
- Instruções com 5 mil registros para cada tabela (85 mil registros no total);
- Instruções com 7 mil registros para cada tabela (119 mil registros no total);
- Instruções com 10 mil registros para cada tabela (170 mil registros no total).

Para cada cenário foram executadas 20 vezes as seguintes instruções na seguinte ordem:

- *CREATE*: criação de todas as 17 tabelas do BD;
- *INSERT*: inserção de cada registro individualmente para as 17 tabelas;
- *SELECT*: recuperação de todos os registros para as 17 tabelas;
- *UPDATE*: atualização de todos os atributos para as 17 tabelas;
- *DELETE*: deleção de todos os registros para as 17 tabelas;
- *INSERTN*: inserção de todos os registros agrupados para as 17 tabelas;
- *ALTER*: renomeação de todas as colunas para as 17 tabelas;
- *DROP*: remoção das 17 tabelas.

Na sequência são descritos os resultados obtidos.

#### 6.4 EXPERIMENTOS COM INSTRUÇÕES SQL DDL

O comando *CREATE* não é afetado pela quantidade de dados. Por isso os tempos são praticamente os mesmos independentemente do tamanho do BD. No geral o tempo de execução varia entre 0,5 e 0,7 segundos, conforme apresentado na Figura 46. O *outlier* apresentado no cenário 12 é proveniente de um possível uso demasiado de memória *heap* do *Neo4j*. Já no cenário 1, os *outliers* apresentados são referentes ao tempo de criação do plano de consulta e demais pré-processamentos que são reutilizados em execuções futuras.

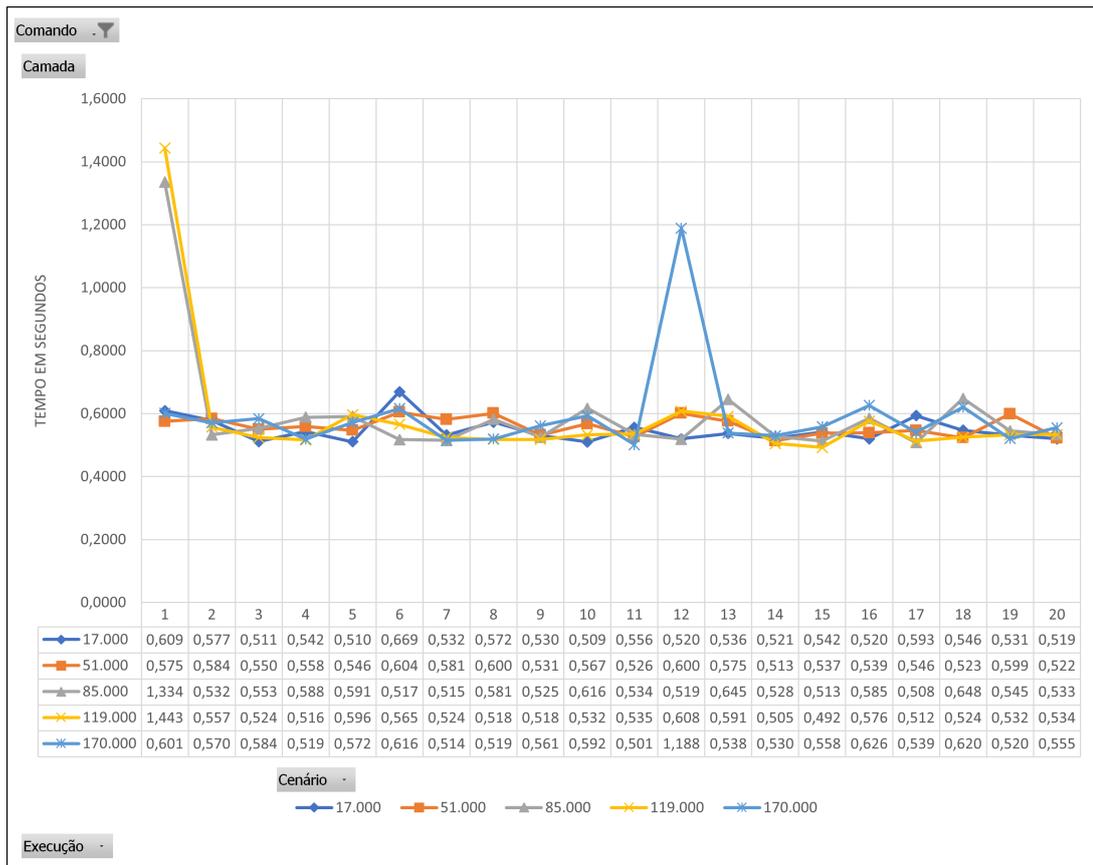


Figura 46 – Resultados do *CREATE*

Fonte: Elaborado pelo Autor

Analisando a Figura 47 pode-se identificar que o *overhead* é bastante pequeno independentemente do volume de dados, sendo em média 0,0326 segundos para conector, 0,0891 segundos para a camada e 0,4376 segundos para o *Neo4j*.

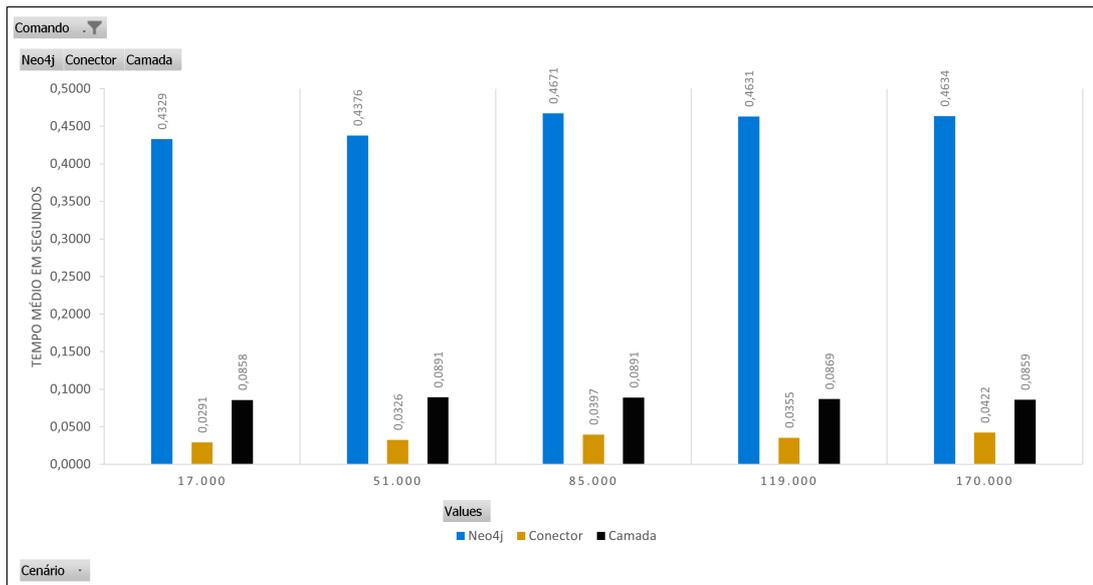


Figura 47 – *Overhead* para a instrução *CREATE*

Fonte: Elaborado pelo Autor

O desempenho do comando *ALTER* nos cenários até 85 mil registros apresenta um comportamento estável durante as 20 execuções. Nos testes a partir de 100 mil registros é possível identificar maior variância nos tempos de execução. A Figura 48 sintetiza os experimentos realizados nas 20 execuções com todos os cenários definidos para a análise de desempenho. Entre os cenários 11 e 12, pode-se notar um possível atingimento da memória *heap* seguido de uma liberação, causando uma variação acentuada nos tempos de execução.

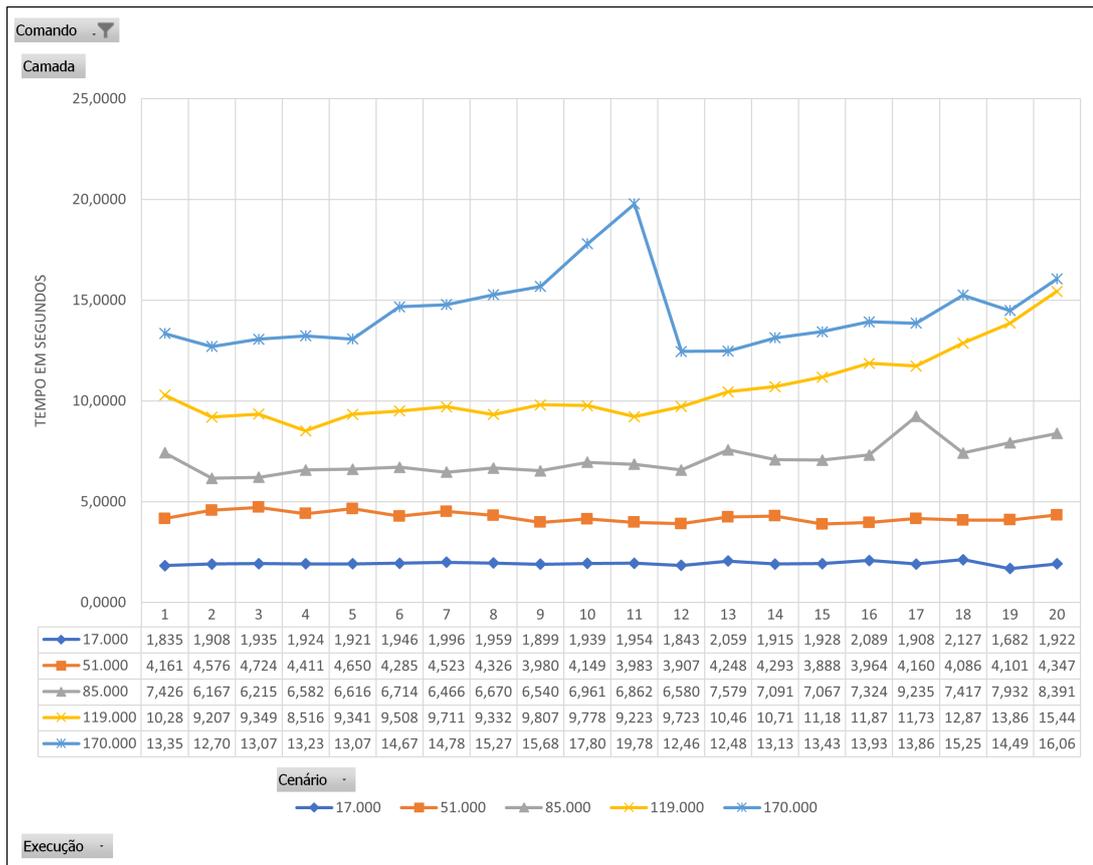
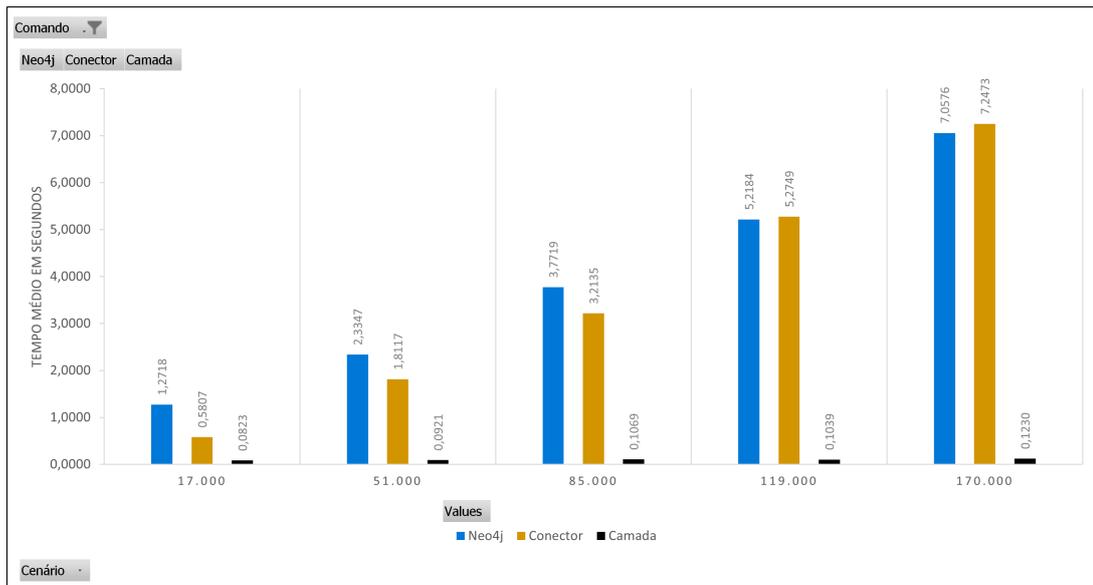


Figura 48 – Resultados da instrução ALTER

Fonte: Elaborado pelo Autor

Para os experimentos em todos os cenários, o tempo de *overhead* utilizado pela camada se manteve praticamente o mesmo, demandando em média 1 segundo para execução. A maior parte do tempo utilizado fica centralizado no *Neo4j* e no conector, como apresentado na Figura 49.

Figura 49 – *Overhead* para a instrução *ALTER*

Fonte: Elaborado pelo Autor

O desempenho do comando *DROP* em cenários até 85 mil registros também apresenta um comportamento estável durante as 20 execuções. Nos testes a partir de 100 mil registros é possível verificar variâncias mais agudas no tempo de execução. A Figura 50 sintetiza os experimentos realizados nas 20 execuções com todos os cenários definidos para a análise de desempenho. Entre os cenários 11 e 12, pode-se notar um possível atingimento da memória *heap* seguido de uma liberação, causando uma variação acentuada nos tempos de execução.

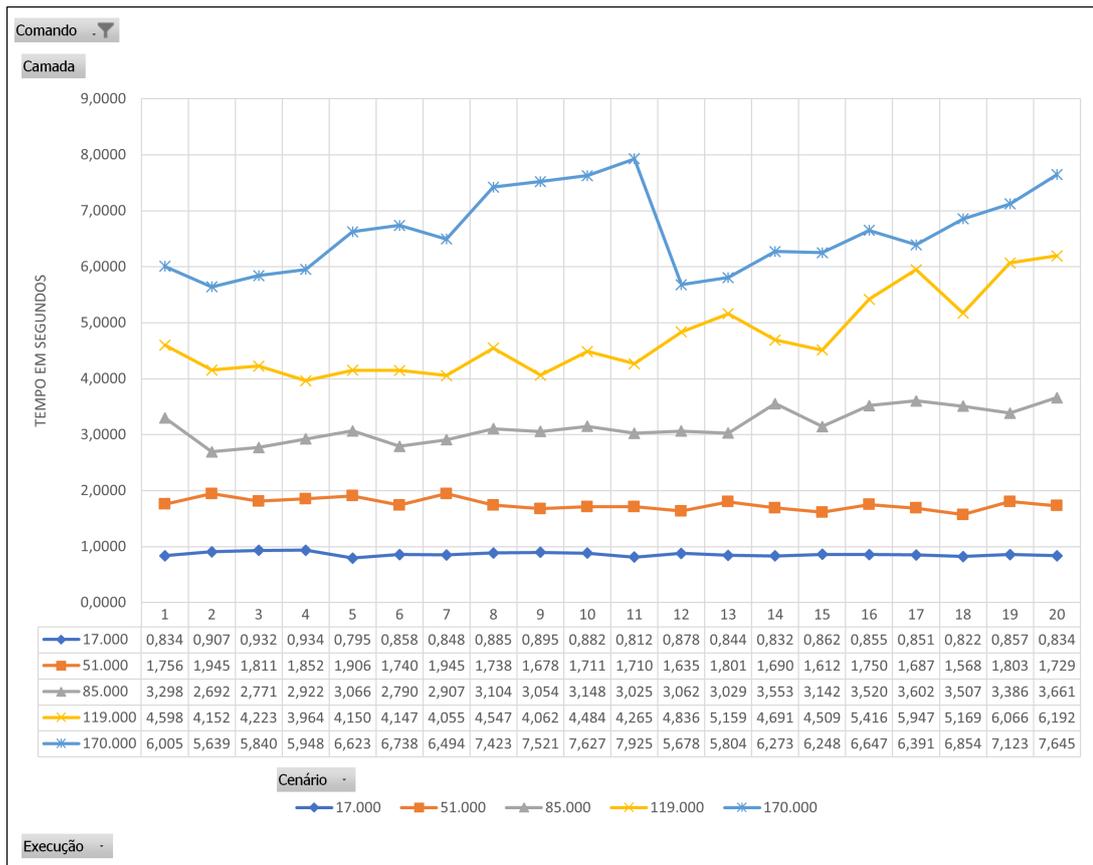
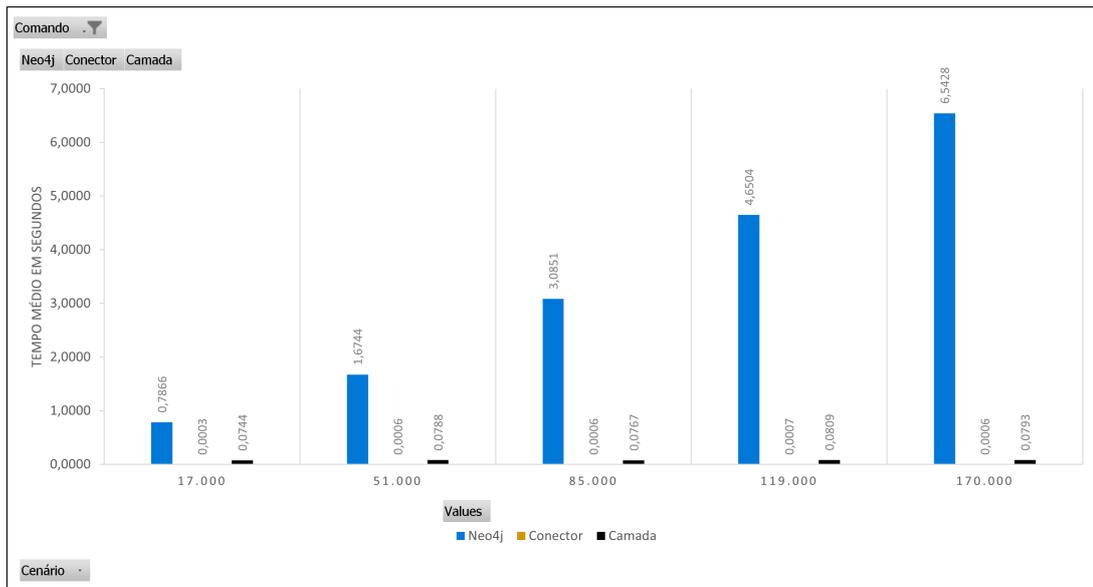


Figura 50 – Resultados da instrução *DROP*

Fonte: Elaborado pelo Autor

Para todos os experimentos o tempo de *overhead* utilizado pela camada se manteve praticamente o mesmo, demandando em média 0,079 segundos para execução. Analisando a Figura 51 é possível identificar que a maior parte do tempo utilizado fica centralizado no *Neo4j* e aumenta linearmente em função da quantidade de registros.

Figura 51 – *Overhead* para a instrução *DROP*

Fonte: Elaborado pelo Autor

## 6.5 EXPERIMENTOS COM INSTRUÇÕES *SQL DML*

O desempenho da instrução *INSERT* vai piorando em função do volume de dados. A Figura 52 sintetiza os resultados nas 20 execuções da instrução de inserção. A partir da análise da Figura 53 pode-se concluir que os tempos de *overhead* no processamento são afetados diretamente pela quantidade de registros e apresentam crescimento linear.

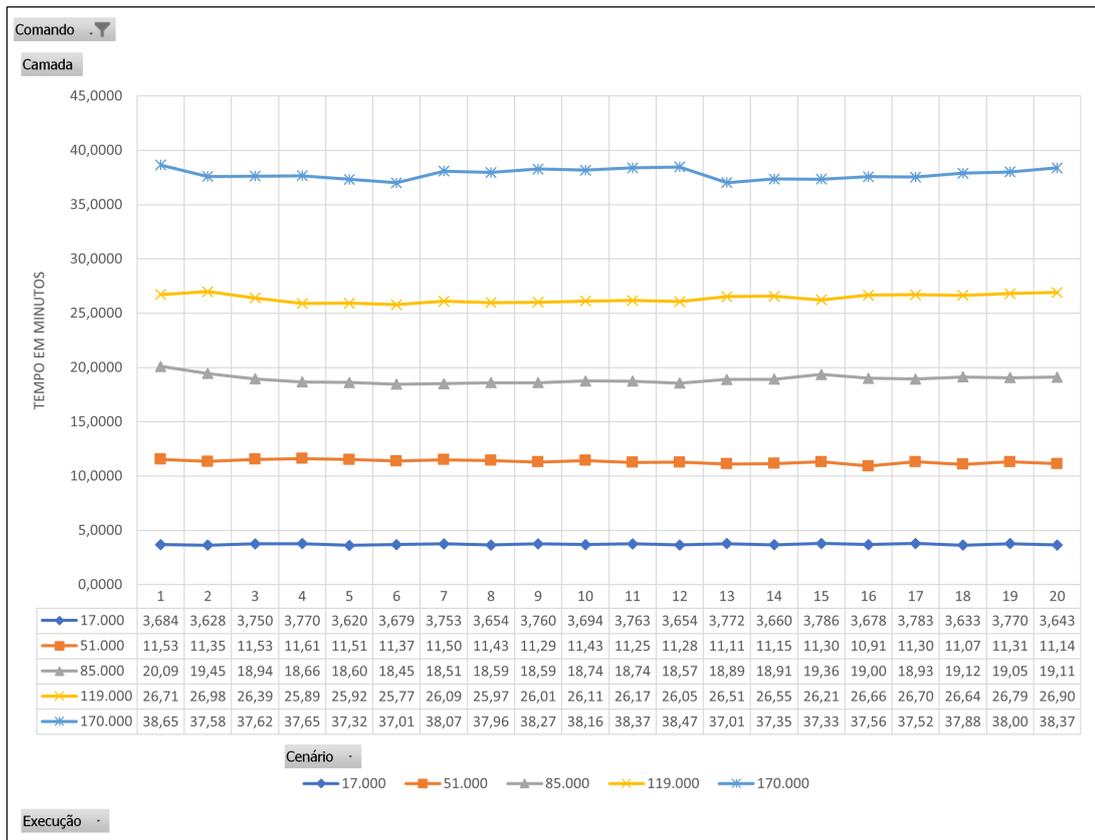


Figura 52 – Resultados da instrução *INSERT*

Fonte: Elaborado pelo Autor

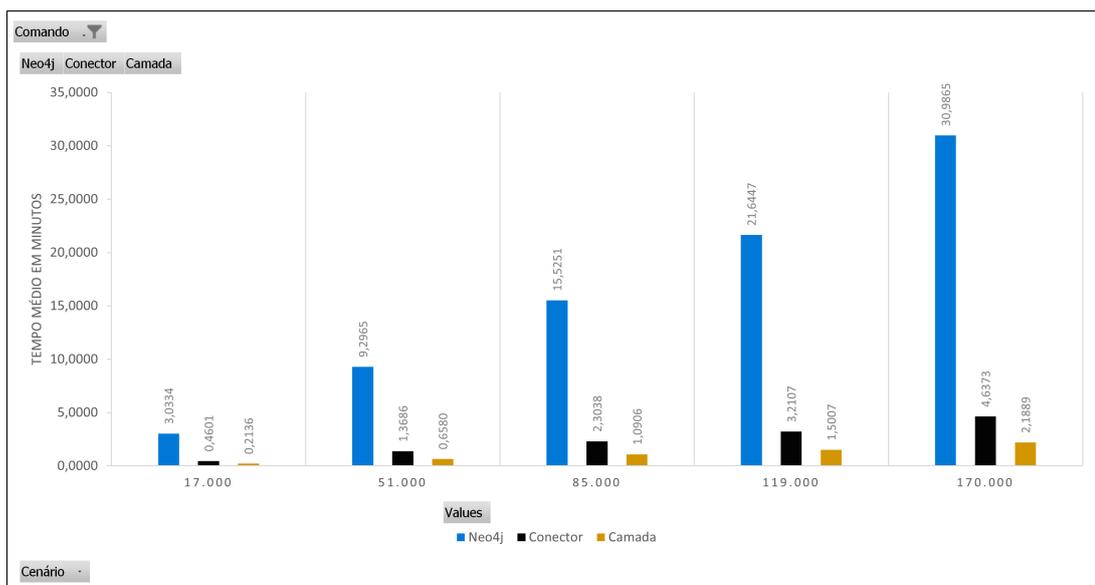


Figura 53 – *Overhead* para a instrução *INSERT*

Fonte: Elaborado pelo Autor

O desempenho do comando *INSERTN* apresenta também comportamento linear em todos os cenários. A Figura 54 sintetiza os resultados nas 20 execuções do comando para inserção. Em relação ao comando *INSERT* há uma redução de aproximadamente 5% no tempo de execução considerando os mesmos cenários.

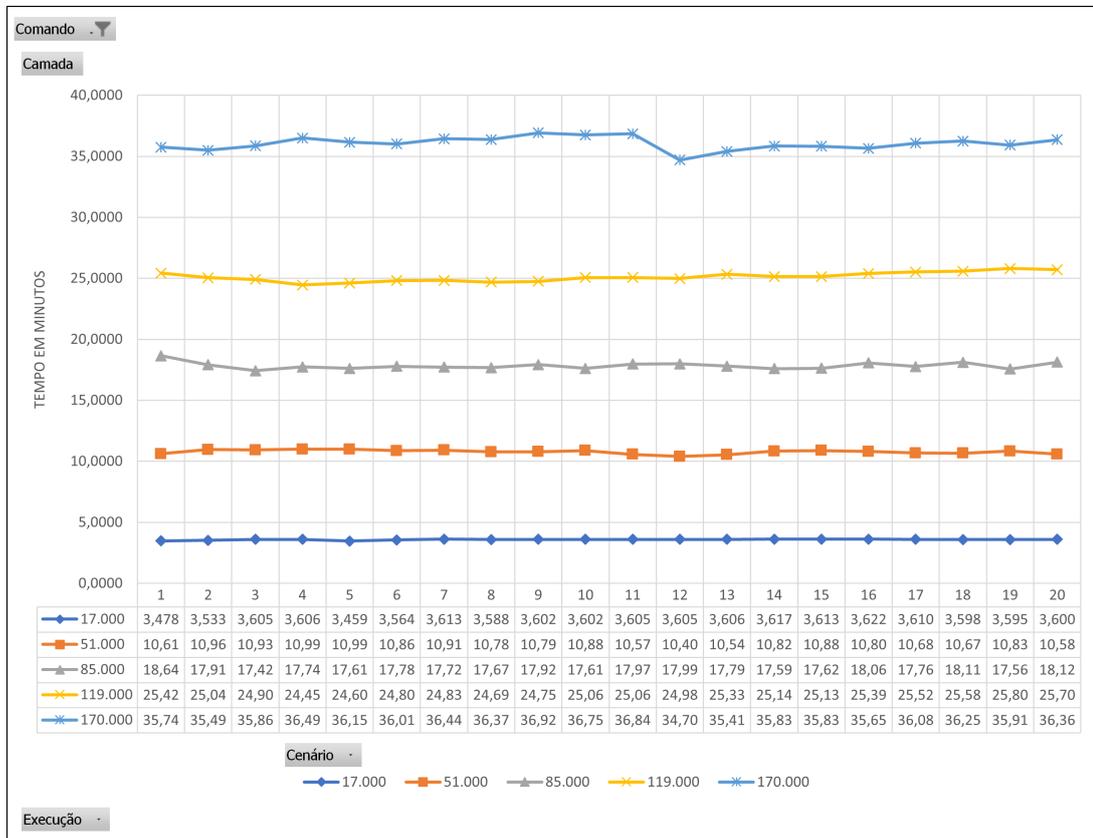


Figura 54 – Resultados da instrução *INSERTN*

Fonte: Elaborado pelo Autor

Pela Figura 55 pode-se também concluir que os tempos de *overhead* no processamento são afetados diretamente pela quantidade de registros e apresentam crescimento linear. Verifica-se ainda o ganho em eficiência pela redução do tempo utilizado pela camada em relação ao comportamento do comando *INSERT*.

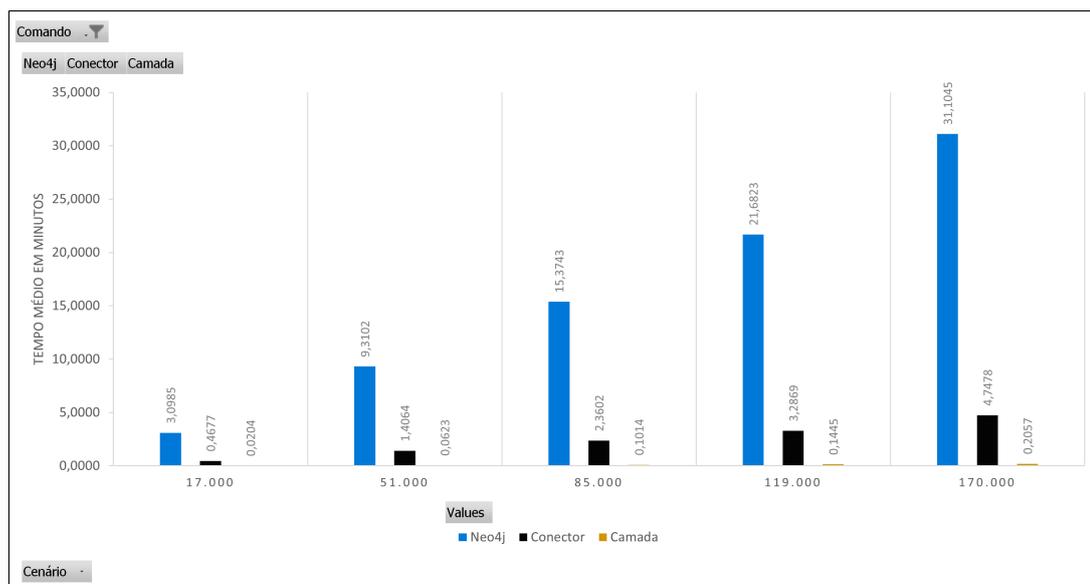


Figura 55 – *Overhead* para a instrução *INSERTN*

Fonte: Elaborado pelo Autor

O desempenho da instrução *UPDATE* apresenta variação no tempo de execução e cresce linearmente em função do número de registros. A Figura 52 sintetiza os resultados nas 20 execuções da instrução.

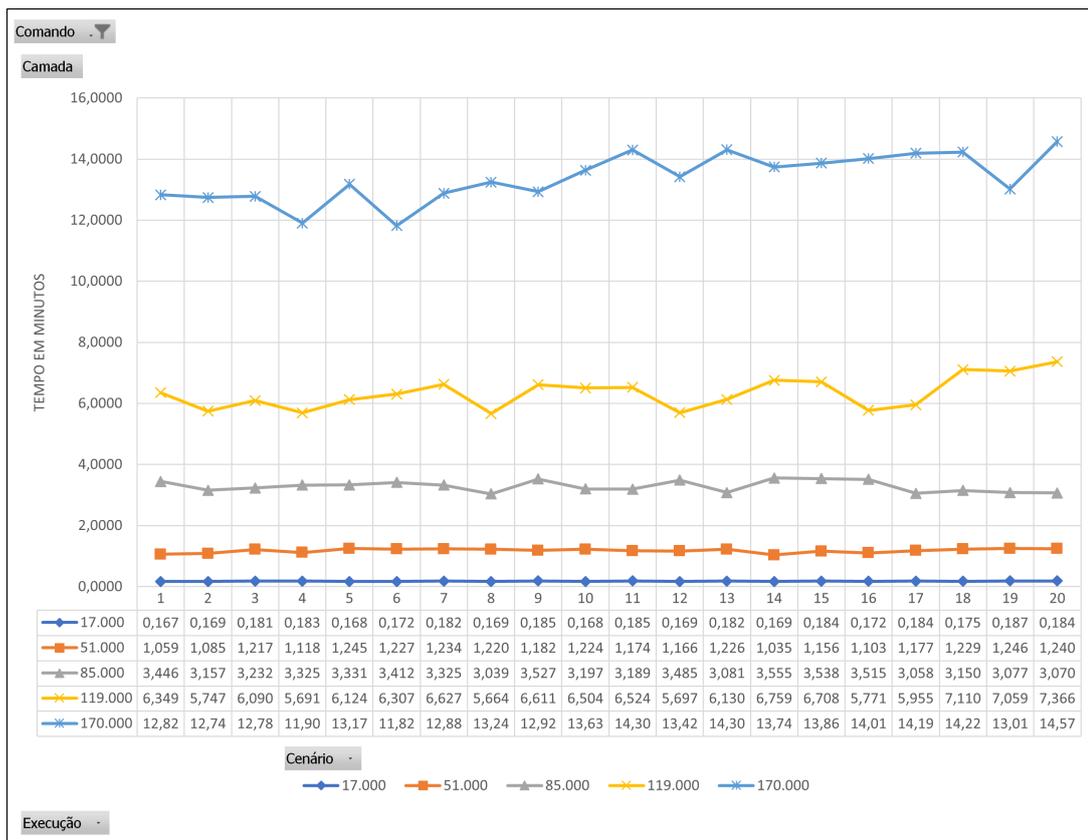


Figura 56 – Resultados da instrução *UPDATE*

Fonte: Elaborado pelo Autor

A partir da análise da Figura 57 pode-se concluir que os tempos de *overhead* no processamento são afetados diretamente pela quantidade de registros. Apresentam crescimento exponencial, levando a entender que grandes volumes de dados sejam inviáveis considerando o seu tempo de execução.

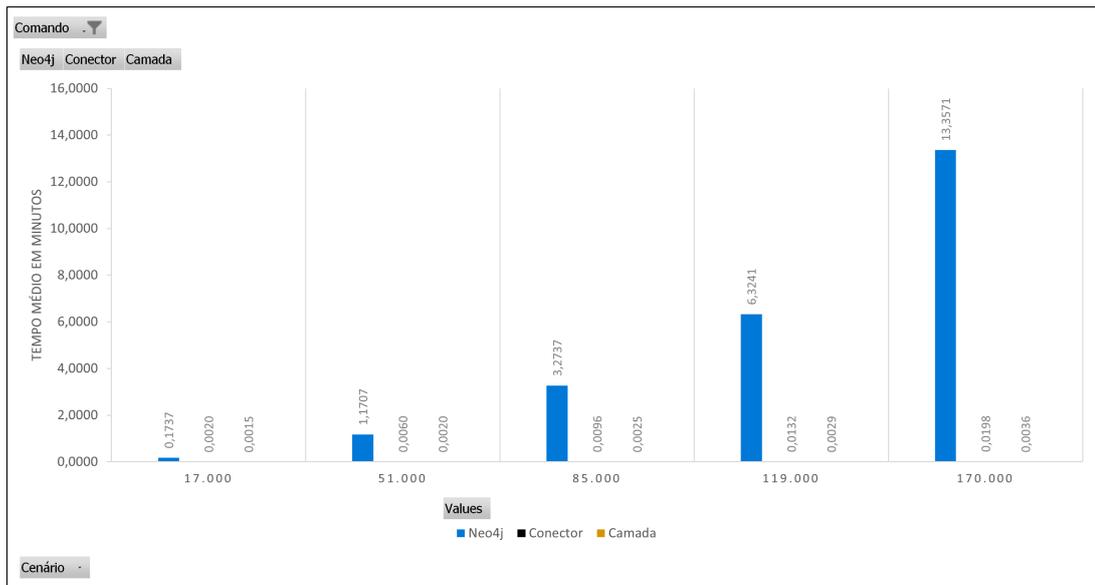


Figura 57 – *Overhead* para a instrução *UPDATE*

Fonte: Elaborado pelo Autor

O desempenho da instrução *DELETE* apresenta variação no tempo de execução que cresce linearmente em função do número de registros. A Figura 58 sintetiza os resultados nas 20 execuções da instrução. Pode-se observar a ocorrência de dois *outliers* na execução 11 e 20 em cenários diferentes, decorrente de uma possível maior uso da memória do *Neo4j*.

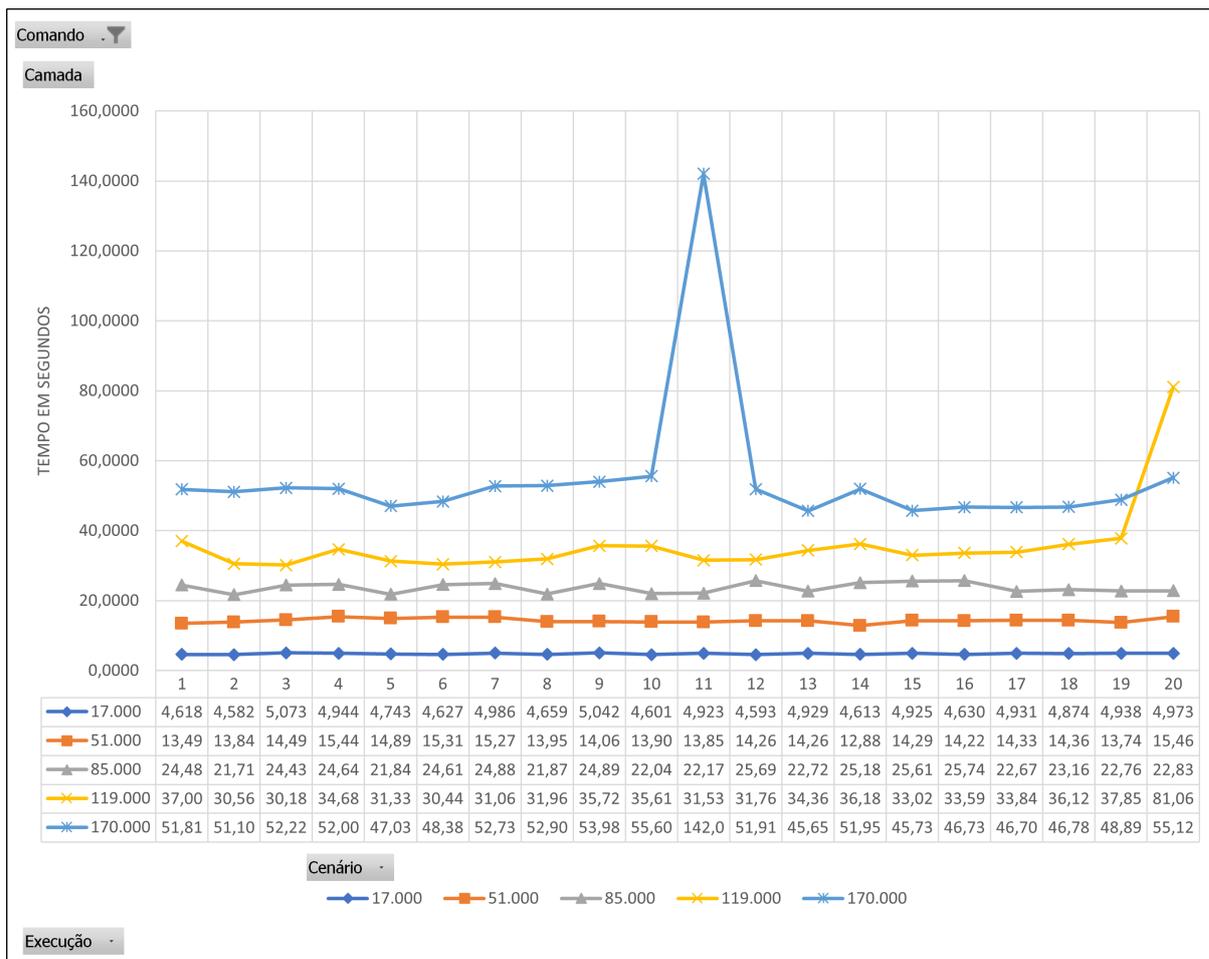


Figura 58 – Resultados da instrução *DELETE*

Fonte: Elaborado pelo Autor

A Figura 59 apresenta o *overhead* para a instrução *DELETE*. Pode-se concluir que os tempos de *overhead* no processamento são afetados diretamente pela quantidade de registros. O aumento linear do tempo de execução ocorre somente no *Neo4j*. Na camada e no conector é possível observar variações mais acentuadas em função do crescimento na quantidade de registros.

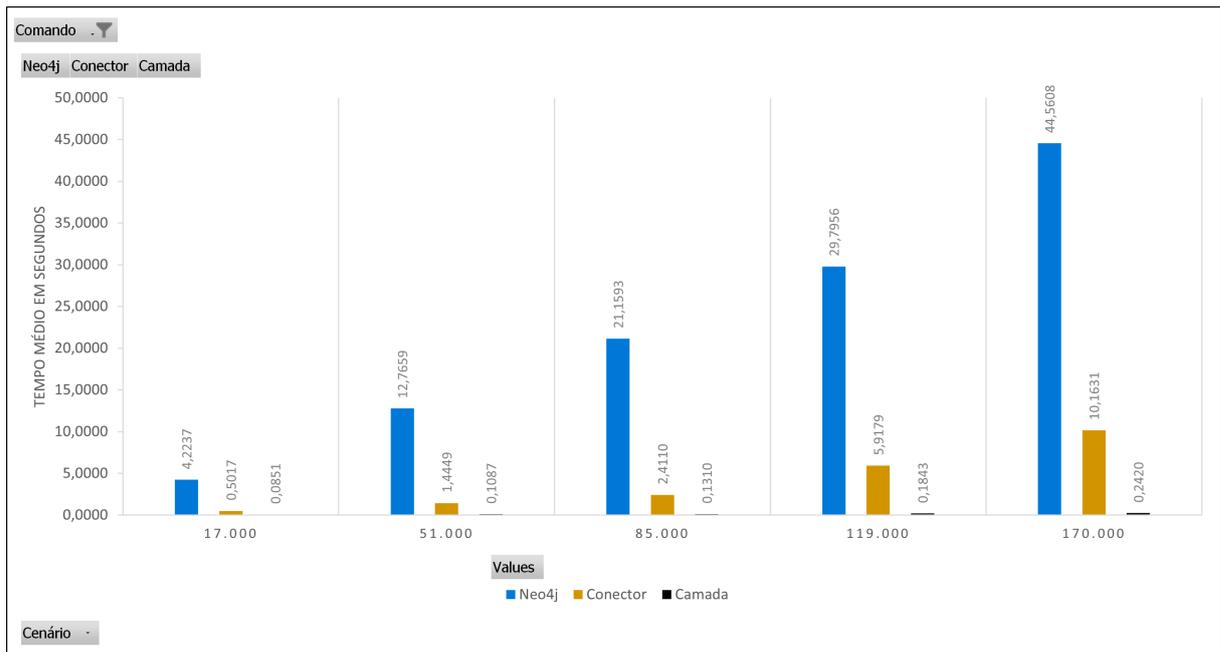


Figura 59 – *Overhead* para a instrução *DELETE*

Fonte: Elaborado pelo Autor

O desempenho da instrução *SELECT* apresenta variação no tempo de execução que cresce linearmente em função do número de registros. A Figura 60 sintetiza os resultados nas 20 execuções do comando para seleção, para consultas que retornam todos os registros das tabelas. Esse é o método com menor tempo de execução em todos os cenários avaliados levando em média 6 segundos para a execução em 170 mil registros.

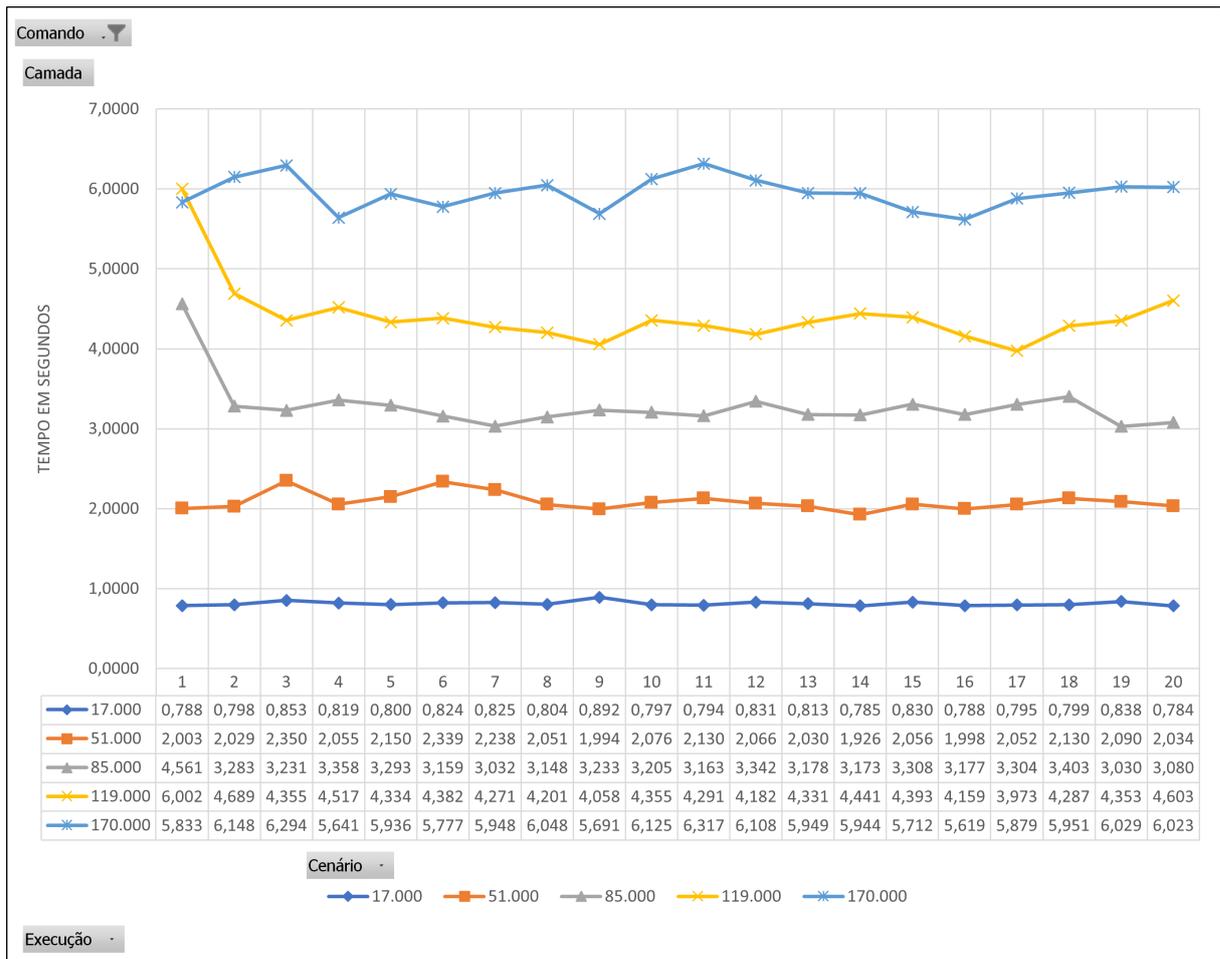


Figura 60 – Resultados da instrução *SELECT*

Fonte: Elaborado pelo Autor

A Figura 61 apresenta o *overhead* introduzido para a execução da instrução *SELECT*. Pode-se concluir que os tempos de *overhead* no processamento são afetados diretamente pela quantidade de registros. O aumento linear do tempo de execução ocorre no *Neo4j* e no conector. Já na camada pode se notar um comportamento estável, com variação quase nula em função do aumento no volume de registros processados.

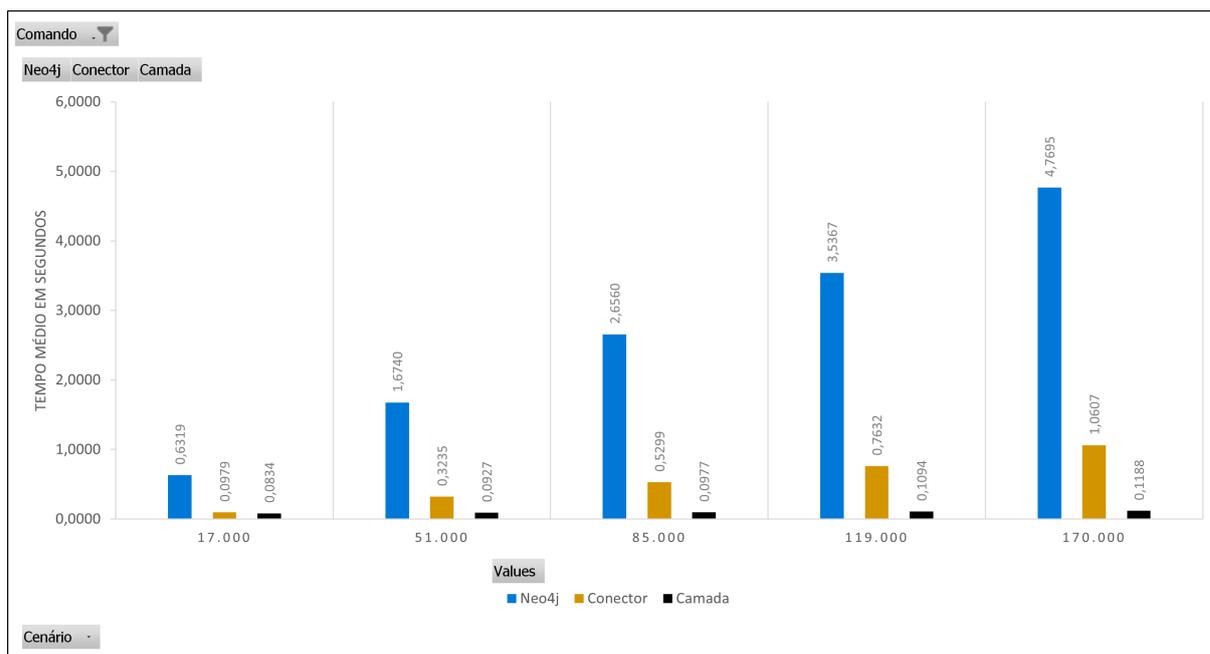


Figura 61 – *Overhead* para a instrução *SELECT*

Fonte: Elaborado pelo Autor

## 7 CONCLUSÃO

Acompanhar as mudanças e a evolução das tecnologias é essencial para se manter ativo no mercado. Assim como em qualquer área é preciso se reinventar e se adequar às necessidades de cada realidade. A solução *SqlToNoSql-Graph* compõe um objetivo maior, aproximar mundos distintos, *SQL* e *NoSQL*, mas que em determinados cenários se complementam. O foco primário é dar mais comodidade ao usuário no gerenciamento de BDs sem a necessidade de aprender uma nova linguagem ou formato de BD.

A principal motivação para realização deste trabalho é entregar uma ferramenta que facilite a interação dos profissionais de *SQL* com o universo *NoSQL*, sem que haja a necessidade de aprenderem novos modelos de dados e novos métodos de acesso. Através da ferramenta desenvolvida é possível realizar a execução de comandos *DDL* e *DML* da linguagem *SQL* em um BD *NoSQL* orientado a grafos.

Durante o desenvolvimento da ferramenta *SqlToNoSql-Graph* foi realizado um exaustivo trabalho de refino do seu código, otimizando os algoritmos de mapeamento com todos os recursos disponíveis para entregar o melhor desempenho.

As principais contribuições deste trabalho são:

- Regras para o mapeamento de BDRs para BD *NoSQL* orientados a grafos;
- Manipulação do BD *NoSQL* orientado a grafos *Neo4j* sem o conhecimento prévio da linguagem *Cypher*;
- A ferramenta *web SQLToNoSQL-Graph*, que recebe instruções *SQL* e realiza o mapeamento para a linguagem de acesso do BD *NoSQL* orientado a grafos *Neo4j*;
- Unificação do mapeamento de *SQL* para os principais modelos *NoSQL* (chave-valor, orientado a coluna, orientado a documento e orientado a grafos) em uma única solução.

Os resultados obtidos na análise de desempenho realizada no Capítulo 6 demonstram que, através desta ferramenta, é viável manipular um BD *NoSQL* orientado a grafos sem a necessidade de aprender a linguagem de acesso para este modelo, uma vez que o *overhead* de acesso com a utilização da ferramenta apresentou um comportamento linear para a grande maioria das instruções *SQL*, sendo considerado um tempo não proibitivo. Dentro do conjunto de instruções suportadas neste trabalho, afirma-se que a acurácia da eficiência de mapeamento, de estruturas relacionais para o modelo de grafos é totalmente confiável, pois em caso de falha não seria possível a execução por falha da sintaxe no comando executado. Deste modo, o profissional de BDRs pode aproveitar seus conhecimentos e desfrutar de um ambiente que explora e entrega excelentes resultados na recuperação de informações em BDs *NoSQL* orientados a grafos. Considera-se, assim, que o objetivo geral do trabalho foi atingido.

Como atividades futuras relacionadas a este trabalho considera-se:

- Leitura automática do esquema de um BDR e sua conversão para um BD *NoSQL*;
- Conversão de *joins* em consultas *SQL* diretamente para a linguagem *Cypher*, visando realizar o processamento de junções diretamente no *Neo4j*;
- Hospedagem da ferramenta na rede para livre utilização;
- Ampliar os comandos suportados pela ferramenta;
- Utilizar o potencial máximo de navegação entre as arestas provida pelo *Neo4j*.

Com esse trabalho complementa-se uma solução (*SQLToNoSQL*) que é capaz de mapear um conjunto restrito de instruções *DDL* e *DML* para três tipos de BDs *NoSQL* (chave-valor, orientado a colunas, orientado a documentos), permitindo também a criação e manipulação de dados em BDs *NoSQL* orientados a grafos. Unificar todas as subdivisões dos BDs *NoSQL* em uma única aplicação amplia as necessidades dos usuários pelo uso de BDs *NoSQL*.

O código fonte completo da aplicação desenvolvida encontra-se disponível em um repositório do GitHub<sup>1</sup> para um maior aprofundamento nos métodos e técnicas utilizados para viabilizar esta solução.

---

<sup>1</sup> <https://github.com/pablo-vicente/SQLToKeyNoSQL-Grafo>

## REFERÊNCIAS

- ABADI, Daniel. Data Management in the Cloud: Limitations and Opportunities. **IEEE Data Eng. Bull.**, v. 32, p. 3–12, jan. 2009.
- ABOUZEID, Azza *et al.* HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. **Proc. VLDB Endow.**, VLDB Endowment, v. 2, n. 1, p. 922–933, ago. 2009. ISSN 2150-8097. DOI: 10.14778/1687627.1687731. Disponível em: <https://doi.org/10.14778/1687627.1687731>.
- AFTAB, Zain *et al.* Automatic NoSQL to Relational Database Transformation with Dynamic Schema Mapping. **Scientific Programming**, Hindawi, v. 2020, p. 8813350, jul. 2020. ISSN 1058-9244. DOI: 10.1155/2020/8813350. Disponível em: <https://doi.org/10.1155/2020/8813350>.
- BERMAN, Jules J. **Principles of Big Data: Preparing, Sharing, and Analyzing Complex Information**. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013. ISBN 9780124045767.
- BRITO, Ricardo W. Bancos de Dados NoSQL x SGBDs Relacionais:Análise Comparativa\*, p. 6, 2010. Disponível em: <https://docplayer.com.br/433629-Bancos-de-dados-nosql-x-sgbds-relacionais-analise-comparativa.html>.
- CANDEL, Carlos J. Fernández; SEVILLA RUIZ, Diego; GARCÍA-MOLINA, Jesús J. A unified metamodel for NoSQL and relational databases. **Information Systems**, v. 104, p. 101898, 2022. ISSN 0306-4379. DOI: <https://doi.org/10.1016/j.is.2021.101898>. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0306437921001149>.
- CATTELL, Rick. Scalable SQL and NoSQL Data Stores. **SIGMOD Rec.**, Association for Computing Machinery, New York, NY, USA, v. 39, n. 4, p. 12–27, mai. 2011. ISSN 0163-5808. DOI: 10.1145/1978915.1978919. Disponível em: <https://doi.org/10.1145/1978915.1978919>.
- DB-ENGINES.COM. **Popularity changes per category, July 2022**. [*S.l.: s.n.*], 2022. Disponível em: [https://db-engines.com/en/ranking\\_categories](https://db-engines.com/en/ranking_categories).
- GUEIDI, Afef; GHARSELLAOUI, Hamza; AHMED, Samir Ben. Towards Unified Modeling for NoSQL Solution Based on Mapping Approach. **Procedia Computer Science**, v. 192, p. 3637–3646, 2021. Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 25th International Conference KES2021. ISSN 1877-0509. DOI: <https://doi.org/10.1016/j.procs.2021.09.137>. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877050921018767>.
- HOMRICH, Émerson P.; MERGEN, Sergio L. S. Comparação entre MySQL e Neo4J para o Acesso a Dados Complexos Usando Linguagens Declarativas. *In: ANAIS da XIV*

Escola Regional de Banco de Dados. Rio Grande: SBC, 2018. Disponível em:  
<https://sol.sbc.org.br/index.php/erbd/article/view/2827>.

INDÚSTRIA, Portal da. **Indústria 4.0: Entenda seus conceitos e fundamentos**. [S.l.: s.n.], mar. 2022. Disponível em:  
<https://www.portaldaindustria.com.br/industria-de-a-z/industria-4-0/>.

NADINE ANDERLE E DALVAN GRIEBLER E SAMUEL SOUZA, Dinei Rockenbach e. Estudo Comparativo de Bancos de Dados NoSQL. **Revista Eletrônica Argentina-Brasil de Tecnologias da Informação e da Comunicação**, v. 1, n. 8, 2018. ISSN 2446-7634. DOI: 10.5281/zenodo.1228503. Disponível em:  
<https://revistas.setrem.com.br/index.php/reabtic/article/view/286>.

NEO4J. **Clauses**. [S.l.: s.n.], 2022. Disponível em:  
<https://neo4j.com/docs/cypher-manual/current/clauses/>.

\_\_\_\_\_. **Concepts: Relational to Graph**. [S.l.: s.n.], 2022. Disponível em:  
<https://neo4j.com/developer/graph-db-vs-rdbms/>.

\_\_\_\_\_. **Constraints**. [S.l.: s.n.], 2022. Disponível em:  
<https://neo4j.com/docs/cypher-manual/current/constraints/>.

\_\_\_\_\_. **Cypher Query Language**. [S.l.: s.n.], 2022. Disponível em:  
<https://neo4j.com/developer/cypher/>.

\_\_\_\_\_. **Getting Started with Cypher**. [S.l.: s.n.], 2022. Disponível em:  
<https://neo4j.com/developer/cypher/intro-cypher/>.

\_\_\_\_\_. **Querying with Cypher**. [S.l.: s.n.], 2022. Disponível em:  
<https://neo4j.com/developer/cypher/querying/>.

\_\_\_\_\_. **What is a Graph Database?** [S.l.: s.n.], 2022. Disponível em:  
<https://neo4j.com/developer/graph-database/>.

NETO, Thomaz Antonio Rossito. **Comandos – DML, DDL, DCL, TCL – SQL Server**. [S.l.: s.n.], set. 2017. Disponível em:  
<https://www.thomazrossito.com.br/comandos-dml-ddl-dcl-tcl-sql-server/>.

NICOLE MELO E LEANDRO DOS SANTOS E WELLINGTON DE OLIVEIRA, Moacir Oliveira e. BANCO DE DADOS NO-SQL X BANCO DE DADOS SQL. **South American Development Society Journal**, v. 4, n. 11, p. 298, 2018. ISSN 2446-5763. DOI: 10.24325/issn.2446-5763.v4i11p298-320. Disponível em:  
<https://www.sadsj.org/index.php/revista/article/view/162>.

ORACLE. **O Que É um Banco de Dados?** [S.l.: s.n.]. Disponível em:  
<https://www.oracle.com/br/database/what-is-database/>.

RIAN ROBINSON, Jim Webber; EIFREM, Emil. **Graph Databases: New Opportunities for Connected Data**. [S.l.]: Pearson Education, 2015. ISBN 978-1-491-93089-2. Disponível em: <https://neo4j.com/graph-databases-book/>.

ROMÃO, Wesley. **Descoberta de conhecimento relevante em banco de dados sobre ciência e tecnologia**. 2002. F. 252. Pós-Graduação – Universidade Federal de Santa Catarina, Florianópolis. Disponível em: <https://repositorio.ufsc.br/handle/123456789/83503>.

SADALAGE P. J.; FOWLER, M. **NoSQL distilled: a brief guide to the emerging world of polyglot persistence**. [S.l.]: Pearson Education, 2012. ISBN 978-0321826626.

SCHREINER, Geomar André. **SQLTOKEYNOSQL: UMA CAMADA PARA MAPEAMENTO DE ESQUEMAS RELACIONAIS E DE OPERAÇÕES SQL PARA BANCOS DE DADOS NOSQL BASEADOS EM CHAVES DE ACESSO**. 2016. F. 79. Pós-Graduação – Universidade Federal de Santa Catarina, Florianópolis. Disponível em: <https://repositorio.ufsc.br/handle/123456789/167987>.

STONEBRAKER, Michael. New Opportunities for New SQL. **Commun. ACM**, Association for Computing Machinery, New York, NY, USA, v. 55, n. 11, p. 10–11, nov. 2012. ISSN 0001-0782. DOI: 10.1145/2366316.2366319. Disponível em: <https://doi.org/10.1145/2366316.2366319>.

WAZLAWICK, R. **Metodologia de Pesquisa para Ciência da Computação**. [S.l.]: Elsevier Brasil, 2017. ISBN 9788535277838. Disponível em: <https://books.google.com.br/books?id=BZioBQAAQBAJ>.

ZHANG, Qi; CHENG, Lu; BOUTABA, Raouf. Cloud computing: state-of-the-art and research challenges. **Journal of Internet Services and Applications**, v. 1, n. 1, p. 7–18, mai. 2010. ISSN 1869-0238. DOI: 10.1007/s13174-010-0007-6. Disponível em: <https://doi.org/10.1007/s13174-010-0007-6>.

## APÊNDICE A – CÓDIGO FONTE

O código fonte completo da aplicação desenvolvida encontra-se disponível em um repositório do GitHub ( <https://github.com/pablo-vicente/SQLToKeyNoSQL-Grafo>) para um maior aprofundamento nos métodos e técnicas utilizados para viabilizar esta solução.

**APÊNDICE B – ARTIGO NO FORMATO SBC**

# SqlToNoSql Graph: Uma solução para mapeamento de banco de dados relacional para NoSql Graph

Lucas Santos de Oliveira<sup>1</sup>, Pablo Vicente<sup>2</sup>  
Ronaldo dos Santos Mello<sup>1</sup>, Geomar Schreiner<sup>2</sup>

<sup>1</sup> Departamento de Informatica e Estatística (INE)

<sup>2</sup> Universidade Federal de Santa Catarina (UFSC)  
Florianopolis – SC – Brasil

{Oliveira, Vicente} Lucas.santos.mi@outlook.com , pablo.vicente@outlook.com.br

**Abstract.** Society is currently following several changes in the technological scenario: big data, artificial intelligence, cloud computing and the Internet of Things (IoT). In order to manipulate large volumes of unstructured data, quickly and constantly, the Not only SQL (NoSQL) Databases (DBs) were created. In this context of NoSQL databases, developers are challenged to deal with different languages for access and manipulation between their relational and non-relational databases. Given this issue, we propose the creation of a set of rules for mapping Relational Database schemas (BDRs) to graph-oriented NoSQL DBs through the extension of the SQLToKeyNoSQL solution. The instructions are mapped to a canonical model in an intermediate layer and are later translated into the access language of NoSQL databases. The experiments carried out demonstrate that through this tool it is possible to manipulate non-relational DBs without the need to learn the access language for this model.

**Resumo.** Atualmente a sociedade está acompanhando diversas mudanças no cenário tecnológico: big data, inteligência artificial, computação em nuvem e a Internet of Things (IoT). Com objetivo de manipular grandes volumes de dados, não estruturados, de forma rápida e constante, foram criados os Bancos de Dados (BDs) Not only SQL (NoSQL). Neste contexto de BDs NoSQL, os desenvolvedores são desafiados a lidar com diferentes linguagens para acesso e manipulação entre suas bases de dados relacionais e não relacionais. Dada esta problemática, propõe-se a criação de um conjunto de regras para o mapeamento de esquemas de Bancos de Dados Relacionais (BDRs) para BDs NoSQL orientado a grafos através da extensão da solução SQLToKeyNoSQL. As instruções são mapeadas para um modelo canônico em uma camada intermediária e posteriormente são traduzidos para a linguagem de acesso dos BDs NoSQL. Os experimentos realizados demonstram que através desta ferramenta é possível manipular BDs não relacionais sem a necessidade de aprender a linguagem de acesso para este modelo.

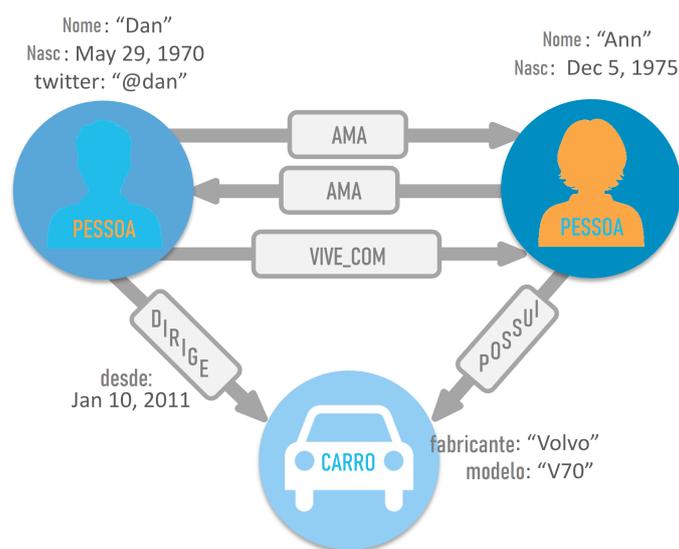
## 1. Introdução

Ao longo dos anos, os BDRs se tornaram os mais populares e disseminados sistemas de gerenciamento de dados, pois atendiam as necessidades da maior parte das aplicações, entregando alto desempenho, simplicidade e confiabilidade na manipulação

dos dados [Stonebraker 2012]. No entanto, para grandes volumes de dados acabam não sendo compatíveis, pois precisam gerenciar e manter a sua consistência sobre os dados, além do esquema lógico inflexível, comprometendo o desempenho [Abadi 2009]. Para suprir essas necessidades, novos modelos de dados e novos SGBDs estão sendo empregados [Zhang et al. 2010]. As arquiteturas destes sistemas utilizam recursos de alta disponibilidade, distributividade e escalabilidade atrelados à capacidade de processamento e armazenamento de grandes volumes de dados em *data centers* alocados na Internet, conceito esse denominado *cloud computing* [Abouzeid et al. 2009].

Esses novos SGBDs são denominados *NoSQL* [Stonebraker 2012]. Eles oferecem suporte a novos modelos de dados flexíveis [Abadi 2009]. Os BDs *NoSQL* são classificados de acordo com o modelo de dados, sendo denominados como: chave-valor, orientado a colunas, orientado a documentos e orientado a grafos, esse último representando dados através de nodos e relacionamentos entre nodos.

A Figura 1 exemplifica um modelo orientado a grafos representando conexões e relacionamentos presentes no mundo real. É possível observar as entidades (nodos) e seus relacionamentos (arestas). Tal representação é muito semelhante à armazenada ou visualizada em BDs orientado a grafos.



**Figura 1. Exemplo de conexões do mundo real**

O foco deste trabalho é o mapeamento de dados de BDRs para BDs *NoSQL* orientado a grafos. O mapeamento do modelo relacional para o modelo de grafos é uma alternativa para otimizar principalmente consultas que devem acessar diversos relacionamentos entre os dados. Isso porque, em um grafo, os relacionamentos são definidos de forma mais natural. As entidades chamadas de vértices, que são interligadas pelas arestas, podem guardar dados entre os relacionamentos em diferentes direções, criando assim uma estrutura mais orgânica.

Alguns dos trabalhos relacionados apontam vantagens em utilizar BDs *NoSQL* para a realização de consultas sobre grandes volumes de dados com esquemas flexíveis e com alta escalabilidade

[BRITO 2010] [e Nicole Melo e Leandro dos Santos e Wellington de Oliveira 2018], [e Nadine Anderle e Dalvan Griebler e Samuel Souza 2018], [Émerson P. Homrich and Mergen 2018]. A solução *SQLToKeyNoSQL* [Schreiner 2016] é um exemplo que permite realizar operações *CRUD* em BDs *NoSQL* utilizando *SQL*. Ela suporta um conjunto restrito de instruções *DDL* e *DML*. Ela também realiza o mapeamento do esquema relacional para um modelo canônico intermediário, e deste modelo canônico para três modelos de dados *NoSQL*: chave-valor, orientado a colunas e orientado a documentos. O acesso aos BDs *NoSQL* suportados é realizado através de um conector genérico pelo qual os métodos de manipulação de dados são baseados em verbos da *API REST*. Este trabalho estende a *SQLToKeyNoSQL* propondo uma camada externa para o mapeamento de esquemas relacionais e operações *SQL* restritas para esquemas em grafo e operações sobre um BD orientado a grafos.

### 1.1. Trabalhos Relacionados

Após pesquisas para correlacionar trabalhos que atuassem na mesma abordagem que a solução proposta, identificou-se que esta é a primeira aplicação que engloba de maneira unificada o mapeamento de esquemas relacionais, baseando-se no mapeamento físico, para os quatro modelos de dados para BDs *NoSQL* (CATTELL, 2011), além de suportar a conversão de um subconjunto de instruções *SQL* para instruções equivalentes nos BDs *NoSQL*. Este trabalho, em particular, provê esse mapeamento para BDs *NoSQL* orientado a grafos, modelo *NoSQL* ainda não suportado pela abordagem *SQLToKeyNoSQL*. Constata-se também a falta de soluções comerciais que resolvam a mesma problemática abordada neste trabalho. A Tabela a seguir sintetiza as principais semelhanças e diferenças entre os trabalhos publicados e a solução proposta neste trabalho.

	Proposta Teórica	Modelos NoSQL	Mapeamento para Grafo	Mapeamento Lógico	Mapeamento de Instrução SQL
A Unified Metamodel For Nosql And Relational Databases	✓	✓	✓	✓	✗
Automatic Nosql To Relational Database Transformation With Dynamic Schema Mapping	✓	✓	✓	✗	✗
Towards Unified Modeling For Nosql Solution Based On Mapping Approach	✓	✗	✗	✗	✗
Sqltonokey-graph	✓	✓	✓	✓	✓

Figura 2. Tabela comparativa entre a proposta e os trabalhos correlatos

## 2. Regras de Mapeamento

A inspiração para o conjunto de regras de mapeamento proposto neste trabalho é fruto do conhecimento de *SQL* e BDRs, bem como o funcionamento e restrições deste modelo, associado ao processo empírico na transformação de dados relacionais para a estrutura equivalente no novo formato de dados. Deste modo, foram definidas as seguintes regras de mapeamento entre os dois modelos:

- **Regra 1 (Mapeamento de Tabela):** dada uma tabela com nome *T*, gera-se um tipo de nodo com rótulo *T* no *BD* orientado a grafos;
- **Regra 2 (Mapeamento de Registros):** dada uma tupla em um *BD* relacional, gera-se um nodo no *BD* orientado a grafos;
- **Regra 3 (Mapeamento de Atributos):** para cada coluna com nome *NC* de um registro *R* em uma tabela de um BDR, gera-se uma propriedade com nome *NC* no nodo gerado para *R* no *BD* orientado a grafos;
- **Regra 4 (Mapeamento de Chave Primária):** dada uma chave primária de uma tabela em um BDR, gera-se uma propriedade cujo valor é único para todos os nodos que possuem o mesmo rótulo no *BD* orientado a grafos;
- **Regra 5 (Mapeamento de Chave Estrangeira):** dada uma coluna com nome *NC* que atua como chave estrangeira de uma tabela em um BDR, gera-se uma aresta entre os nodos com rótulo *NC*;
- **Regra 6 (Mapeamento de Relacionamentos):** dado um relacionamento entre duas tabelas, ambos os nodos desta relação devem estar previamente criados no *BD* orientado a grafos.

A Figura 3 representa um conjunto de dados relacionais e ao lado o mapeamento para a estrutura equivalente em um *BD NoSQL* orientado a grafos. De acordo com a regra de mapeamento 2, cada nodo amarelo é o resultado da conversão de uma tupla relacional presente na tabela *Personagens*, assim como a tupla da tabela *Familias*, que dá origem ao nodo azul. As arestas criadas são resultantes da aplicação da regra número 5. A coluna *FamilialId* é uma chave estrangeira. Deste modo, o nome da coluna foi mapeado para o rótulo da aresta.

Personagens		
Id	Nome	FamilialId
1	Homer Jay Simpson	1
2	Marjorie Jacqueline Simpson	1
3	Bartholomew JoJo Simpson	1
4	Lisa Marie Simpson	1
5	Margaret Evelyn Lenny Simpson	1
6	Abraham Jebediah Simpson II	1
7	Mona Penelope Simpson	1

Familias	
Id	Nome
1	Simpson family



Figura 3. Exemplo do mapeamento *SQL* para *Neo4j*

### 3. Ferramenta

A proposta deste trabalho consiste em acoplar mais um conector à camada *SQLToKeyNoSQL*, utilizando a mesma arquitetura e os mesmos módulos. O novo conector fornece a ligação entre módulo de **Comunicação** existente e a interface específica do BD orientado a grafos *Neo4j*. Essa comunicação se dá pela implementação da interface fornecida pela camada *SQLToKeyNoSQL*, cujos métodos são mapeados para comandos na linguagem *Cypher*, suportada pelo *Neo4j*.

A interface de comunicação da camada *SQLToKeyNoSQL* com os BDs *NoSQL* disponibiliza apenas a manipulação de instruções *DML* através dos quatro comandos: *Get*, *GetN*, *Put* e *Delete*. Instruções *DDL* apenas manipulam as informações presentes no dicionário. Apesar da camada suportar as instruções *DDL ALTER* e *DROP TABLE*, o código fonte disponibilizado não dispõe de tais implementações. A fim de suprir estas lacunas é proposta a implementação destas instruções com melhorias:

- **CREATE TABLE**: cria as definições da tabela relacional no dicionário, considerando o nome da tabela, atributos, chaves primárias e estrangeiras. Não é permitido executar duas instruções para a mesma tabela. Adicionalmente, é disponibilizado o comando *CREATE* na interface de comunicação com o BD para que o BD possa realizar parametrizações e ajustes se necessário.
- **ALTER TABLE**: contempla a alteração de nome, adição e remoção de uma coluna. Não são permitidas alterações de chaves primárias. As novas definições são atualizadas no dicionário. Na operação de adição é permitido apenas a inclusão de colunas simples, sem propriedades com chave estrangeira. A operação de remoção contempla a exclusão de uma coluna existente e suas respectivas chaves estrangeiras. É possível também alterar o nome de uma coluna para outra ainda não existente na tabela, assim como suas respectivas chaves estrangeiras. Além disso, é permitido alterar múltiplas colunas com a mesma instrução. Adicionalmente é disponibilizado o comando *ALTER* na interface de comunicação com o BD para que as alterações sejam executadas no BD.
- **DROP TABLE**: remove as informações referentes à tabela armazenadas no dicionário: nome, atributos, chaves primárias e estrangeiras. Adicionalmente é disponibilizado o comando *DROP* na interface de comunicação com o BD para que as deleções sejam refletidas no BD.

Para as instruções *DML* previstas pela camada *SQLToKeyNoSQL* na interface de comunicação com os BDs é previsto manter os mesmos comportamentos com as seguintes modificações:

- **PUT**: este comando inicialmente previa o suporte para inserção de somente uma chave. Nesta modificação é proposto o suporte à inserção de uma ou mais chaves através da conexão com o BD, possibilitando a execução de forma mais performática com a utilização da função *InsertN*;
- **DELETE**: O comando inicialmente previa o suporte para a exclusão de somente uma chave. Nesta modificação, é proposto o suporte a exclusão de uma ou mais chaves através da conexão com o BD, possibilitando a execução de forma mais performática.
- **UPDATE**: comando inicialmente realizado através da operação de exclusão, seguida de uma inserção. Tal comportamento foi movido para a interface de

comunicação com o BD, pois alguns SGBDs não possuem suporte nativo a esta instrução, permitindo, assim, que os que possuem o suporte à instrução possam realizá-la de forma mais performática. Além disso, é disponibilizada a alteração de múltiplas chaves na interface de comunicação com o BD.

### 3.1. Mapeamento de Instruções *DDL*

Comandos *DDL* foram adicionados à interface de comunicação com o BD com o objetivo de garantir a integridade dos dados. Para contemplar esta premissa é realizada a tradução para os comandos na linguagem *Cypher* das seguintes instruções:

- **CREATE**: uma *CONSTRAINT* do tipo chave única é criada para garantir a aplicação da regra de mapeamento número 4. Esta chave é definida a partir de uma propriedade sintética denominada *NODE\_KEY*, cujo valor é resultante da combinação de chaves primarias definidas no esquema relacional.

A Figura 4 exemplifica a criação da *CONSTRAINT funcao\_NODE\_KEY*, na qual os nodos do tipo *funcao* possuem o valor da propriedade *NODE\_KEY* definida como única para todos os nodos de mesmo tipo, presentes no BD *Neo4j*.

```
1 CREATE CONSTRAINT funcao_NODE_KEY
2 IF NOT EXISTS FOR (n:funcao)
3 REQUIRE (n.NODE_KEY) IS UNIQUE
```

Figura 4. Exemplo de criação da *CONSTRAINT* para o método *CREATE*

- **ALTER**: esta operação suporta apenas a alteração de colunas (adição, modificação ou exclusão). A adição não apresenta alteração imediata no BD, pois o modelo é flexível, não sendo necessária a declaração de todas as propriedades em um nodo. A operação *DROP COLUMN* é dividida em simples (quando a coluna não é uma chave estrangeira) e complexa (quando a coluna é uma chave estrangeira).

A Figura 5 exemplifica a remoção de duas colunas, uma com chave estrangeira e outra simples, sendo feita a seleção de todos os nodos (*n*) do tipo *funcionario* que possuem ou não (*OPTIONAL*) o relacionamento do tipo *banco\_id*. Em seguida, as propriedades *nome* e *banco\_id* são removidas do nodo, em seguida executa-se a deleção do relacionamento *banco\_id*.

```
1 MATCH (n:funcionario)
2 OPTIONAL MATCH (n:funcionario) -[banco_id:banco_id]-> (nbanco_id)
3 REMOVE n.nome
4 REMOVE n.banco_id
5 DELETE banco_id
```

Figura 5. Exemplo do comando *DROP COLUMN* para o método *ALTER*

No caso de um *RENAME COLUMN*, as propriedades não podem ser renomeadas no *Neo4j*. Neste caso, é necessário criar uma nova propriedade com o novo nome, mantendo o valor e em seguida excluir a propriedade antiga. Para propriedades com chave estrangeira, é feito o mesmo processo. Neste caso, é criado um novo relacionamento com o novo tipo e excluído o relacionamento com o tipo antigo.

A Figura 6 exemplifica a renomeação de duas colunas, uma com chave estrangeira e outra sem, sendo feita a cópia das propriedades *nome* e *banco\_id* para as propriedades *nome\_novo* e *banco\_id\_novo*, respectivamente. Em seguida, é feita a remoção das propriedades *nome* e *banco\_id*. Na sequência é criado o relacionamento *banco\_id\_novo* e removido o relacionamento antigo *banco\_id*.

```
1 MATCH (n:funcionario)
2 OPTIONAL MATCH (n:funcionario) -[banco_id:banco_id]-> (nbanco_id)
3 SET n.nome_novo = n.nome
4 REMOVE n.nome
5 SET n.banco_id_novo = n.banco_id
6 REMOVE n.banco_id
7 CREATE (n)-[:banco_id_novo]->(nbanco_id)
8 DELETE banco_id
```

**Figura 6. Exemplo do comando *RENAME COLUMN* para o método *ALTER***

- ***DROP***: o *Neo4j* apresenta um comportamento *default* similar aos BDRs, ou seja, não é permitido a remoção de nodos que são referenciados por outros nodos. A exclusão é permitida somente nos casos em que o nodo não seja apontado por outro. Neste caso, são excluídos os nodos e seus respectivos relacionamentos. Após a remoção de todos os nodos é feita a exclusão da *CONSTRAINT* de chave única.

A Figura 7 exemplifica a exclusão de todos os nodos (*n*) do tipo *cliente* que possuam ou não (*OPTIONAL*) quaisquer tipos de relacionamento (*chave\_estrangeira*) com outros nodos (*ce*). Em seguida, é feita a exclusão da *CONSTRAINT* (*cliente\_NODE\_KEY*) caso exista.

```
1 MATCH (n:cliente)
2 OPTIONAL MATCH (n:cliente) -[chave_estrangeira]-> (ce)
3 DELETE n, chave_estrangeira;
4
5 DROP CONSTRAINT cliente_NODE_KEY IF EXISTS
```

**Figura 7. Exemplo da instrução *DROP TABLE***

### 3.2. Mapeamento de Instruções *DML*

Conforme descrito no Capítulo 5.2, além dos comandos mencionados no Capítulo 4 foram adicionados novos métodos à interface de comunicação com o BD. As operações de filtragem e junções são realizadas pela camada *SQLToKeyNoSQL*. Nos exemplos de cada comando é descrito a instrução *SQL* e seu respectivo mapeamento para a linguagem *Cypher*. Para manipular os dados é realizada a tradução das seguintes instruções:

- ***PUT (INSERT)***: Diferentemente de outros BDs *NoSQL* suportados, o *Neo4j* gerencia relacionamentos entre os dados armazenados. A inserção de dados é dividida em dois tipos: simples e complexa. A inserção simples ocorre quando o nodo não tem relacionamento com outro nodo. Neste caso, é realizado o mapeamento direto para a instrução *CREATE* na linguagem *Cypher*, definindo o tipo do nodo e seus atributos.

A inserção complexa ocorre quando há relacionamentos entre os nodos. Neste caso, a criação do nodo só pode ser feita se ambos os nodos da relação já estiverem inseridos no BD. Após a execução da instrução é verificado se o nodo e seus respectivos relacionamentos foram criados como esperado. Do contrário, o comando é invalidado.

A Figura 8 exemplifica o mapeamento da instrução *INSERT*. Pressupõe-se que existe uma tabela *item\_venda* que possui uma chave estrangeira com a tabela *item\_produto*. Inicialmente é verificado se os nodos da relação existem. Em seguida é realizada a criação do nodo com o tipo *item\_venda*.

```
1 //SQL
2 INSERT INTO Item_venda (id_Item,id_produto,quant_vendida, prec_unitario
3 )
4 VALUES (1,1,1,100);
5 // CYPHER
6 MATCH
7 (produto1:produto{NODE_KEY:1})
8 CREATE
9 (item_venda1:item_venda {
10     NODE_KEY:1,
11     id_item:1,
12     id_produto=1
13     quant_vendida:1,
14     prec_unitario:100
15     }),
16 (item_venda1)-[id_produto:id_produto]->(produto1)
```

**Figura 8. Exemplo de mapeamento da instrução *INSERT INTO***

- **UPDATE:** A alteração de dados também é dividida em dois tipos: simples e complexa. A alteração simples ocorre quando o nodo não tem relacionamento com outro nodo. Neste caso, o valor da propriedade é alterado diretamente. Nos casos complexos, em que há chave estrangeira, é verificado se o novo nodo do relacionamento já existe no BD. Em seguida são deletados e recriados todos os seus relacionamentos.

A Figura 9 exemplifica o mapeamento da instrução *UPDATE*. Pressupõe-se que existe uma tabela *usuario* que possui uma chave estrangeira com a tabela *funcionario*. Inicialmente é verificado se existe um nodo com o rótulo *funcionario* que possua a propriedade *id '1'*. Caso tenha sucesso, todas as propriedades são alteradas através do comando *SET+*. Seus relacionamentos são deletados e recriados em seguida com os novos valores.

```

1 //SQL
2 UPDATE usuario
3 SET func_id=1,
4     user_log='adm (Atualizado)',
5     user_pwd='adm';
6
7 //CYPHER
8 //1 VERIFICATION
9 MATCH (n:funcionario)
10 WHERE n.id = 1
11 RETURN (n)
12
13 //2 INSERTION
14 MATCH (n:usuario)
15 WHERE n.NODE_KEY in [1, 2]
16 SET n +={
17     user_log:'adm (atualizado)',
18     func_id:1,
19     user_pwd:'adm'
20 }
21
22 WITH n
23 CALL
24 {
25     WITH n
26     MATCH (n)-[relacionamento]->(relacionamento_apontado)
27     WHERE type(relacionamento) in ["func_id"]
28     DELETE relacionamento
29 }
30
31 WITH (n)
32 MATCH (funcionario_id_func_id:funcionario)
33 WHERE funcionario_id_func_id.id = 1
34 CREATE (n)-[:func_id]->(funcionario_id_func_id)

```

**Figura 9. Exemplo de mapeamento da instrução UPDATE**

- **DELETE**: Pelo fato de um BD orientado a grafos conter relacionamentos entre os dados, não é possível excluir um nodo que mantém relacionamentos ativos. Conforme descrito no início deste capítulo, é habilitado o suporte para o envio de múltiplas chaves. Desta forma, é possível executar a exclusão em uma única consulta especificando apenas as chaves a serem excluídas. Seu funcionamento é similar ao comportamento do comando *DROP*, no entanto, não é necessário realizar a remoção da *CONSTRAINT* de chave única.

A Figura 10 exemplifica o mapeamento da instrução *DELETE*. A exclusão dos nodos é realizada utilizando a chave sintética *NODE\_KEY*. São selecionados todos os nodos do tipo *item\_venda* que possuem ou não (*OPTINONAL*) algum tipo de relacionamento, cujas chaves são fornecidas pela camada *SQLToKeyNoSQL*. Em seguida são excluídos os nodos e seus respectivos relacionamentos.

```

1 //SQL
2 DELETE FROM Item_venda;
3
4 /CYPHER
5 MATCH (n:item_venda)
6 OPTIONAL MATCH (n:item_venda) -[arestas]-> (ce)
7 WHERE n.NODE_KEY in [1, 2]
8 DELETE arestas,n

```

**Figura 10. Exemplo de mapeamento da instrução *DELETE***

- ***GET (SELECT)***: Esta operação apresenta comportamento de baixa complexidade, responsável pela recuperação de determinado nodo a partir da chave sintética *NODE\_KEY*.
- ***GETN (SELECT)***: É a sobre-escrita do método *GETN* disponibilizado pela interface de comunicação com o BD da camada *SQLToKeyNoSQL*. Em uma única consulta, são recuperados todos os nodos do mesmo tipo. Desta forma, o método *GET* deixou de ser utilizado.

A Figura 11 exemplifica o mapeamento da instrução *SELECT*. É realizada a seleção de todos os nodos do tipo *vendas*. Embora sua execução seja simples, ela entrega o resultado de forma mais performática se comparada com a solução inicial prevista na camada de origem *SQLToKeyNoSQL*.

```

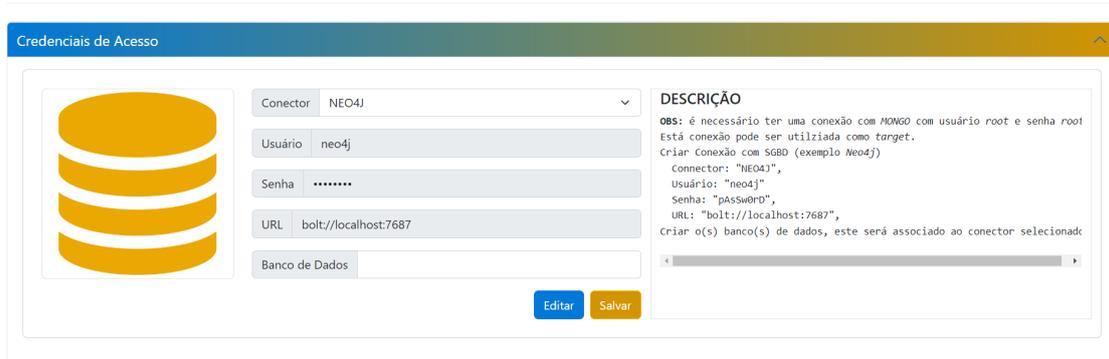
1 //SQL
2 SELECT * FROM vendas;
3
4 /CYPHER
5 MATCH (n:vendas) RETURN (n)

```

**Figura 11. Exemplo de mapeamento da instrução *GETN***

### 3.3. Interface do Usuário

Esta subseção descreve a interface gráfica e as funcionalidades da ferramenta desenvolvida para dar suporte a este trabalho. A Figura 12 mostra a tela inicial na qual o usuário insere as informações necessárias para a conexão com um BD *Neo4j*. Cada funcionalidade possui uma descrição ao lado para facilitar a utilização da ferramenta.



**Figura 12. Tela para inserção das credenciais de acesso ao BD Neo4j**

A Figura 13 representa a funcionalidade central da ferramenta: a execução de *queries*. Nesta tela o usuário pode escolher o BDR, selecionar um arquivo ou digitar as instruções *SQL* a serem executadas. Em caráter complementar foi acrescentado um componente que retorna o tempo de execução das instruções. Por fim, foram indicados os comandos *DML* e *DDL* suportados pela ferramenta.



**Figura 13. Tela para execução de queries**

Quando o usuário executa mais de um comando do tipo *SELECT*, as informações recuperadas são carregadas em várias tabelas. A Figura 14 exemplifica o componente responsável por exibir aos usuários as informações retornadas pela instrução.

Tabela: FUNCIONARIO

Pesquisar:  Buscar registros

ID	FUNCAO_ID	BANCO_ID	NOME	DATA_NASC	TELEFONE	EMAIL	T_PESSOA	RAZAO_SOCIAL	RG	CPF	ENDERECO
1	1	1	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	pessoa física	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
2	2	2	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	pessoa física	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
3	3	3	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	pessoa física	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
4	4	4	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	pessoa física	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
5	5	5	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	pessoa física	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
6	6	6	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	pessoa física	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
7	7	7	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	pessoa física	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
8	8	8	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	pessoa física	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
9	9	9	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	pessoa física	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
10	10	10	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	pessoa física	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
11	11	11	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	pessoa física	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae

Mostrando de 1 até 11 de 1.000 registros

**Figura 14. Componente para retorno da query**

Nos casos em que são retornadas mais de uma tabela é possível selecionar a visualização através do componente de seleção, localizado acima da própria tabela. A Figura 15 exemplifica tal comportamento, possibilitando ao usuário a opção de escolher qual tabela deseja visualizar.

Tabela: FUNCIONARIO

BANCO  
CLIENTE  
ENTREGA  
ESTOQUE  
FABRICANTE  
FLUXO\_CAIXA  
FORMA\_PAG  
FORNECEDOR  
FUNCAO  
**FUNCAO**  
ITEM\_VENDA  
LOCALIZACAO\_PROD  
PRODUTO  
SETOR  
USUARIO  
VEICULO  
VENDAS

ID	FUNCAO_ID	BANCO_ID	NOME	DATA_NASC	TELEFONE	EMAIL	T_PESSOA	RAZAO_SOCIAL	RG	CPF	ENDERECO
8	8	8	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	pessoa física	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
9	9	9	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	pessoa física	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
10	10	10	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	pessoa física	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae
11	11	11	lucas oliveira	1998-04-23	(19)3333-3333	lucas@com.net	pessoa física	num sei cara	45.343.942-1	450.802.308.86	rua: vinicius de morae

Mostrando de 1 até 11 de 1.000 registros

**Figura 15. Seleção de visualização das tabelas**

#### 4. Conclusão

Acompanhar as mudanças e a evolução das tecnologias é essencial para se manter ativo no mercado. Assim como em qualquer área é preciso se reinventar e se adequar às necessidades de cada realidade. A solução *SqlToNoSql-Graph* compõe um objetivo maior, aproximar mundos distintos, *SQL* e *NoSQL*, mas que em determinados cenários se complementam. O foco primário é dar mais comodidade ao usuário no gerenciamento de BDs sem a necessidade de aprender uma nova linguagem ou formato de BD.

A principal motivação para realização deste trabalho é entregar uma ferramenta que facilite a interação dos profissionais de *SQL* com o universo *NoSQL*, sem que haja a

necessidade de aprenderem novos modelos de dados e novos métodos de acesso. Através da ferramenta desenvolvida é possível realizar a execução de comandos *DDL* e *DML* da linguagem *SQL* em um BD *NoSQL* orientado a grafos.

Durante o desenvolvimento da ferramenta *SqlToNoSql-Graph* foi realizado um exaustivo trabalho de refino do seu código, otimizando os algoritmos de mapeamento com todos os recursos disponíveis para entregar o melhor desempenho.

Os resultados obtidos demonstram que, através desta ferramenta, é viável manipular um BD *NoSQL* orientado a grafos sem a necessidade de aprender a linguagem de acesso para este modelo, uma vez que o *overhead* de acesso com a utilização da ferramenta apresentou um comportamento linear para a grande maioria das instruções *SQL*, sendo considerado um tempo não proibitivo. Dentro do conjunto de instruções suportadas neste trabalho, afirma-se que a acurácia e a eficiência de mapeamento, de estruturas relacionais para o modelo de grafos é totalmente confiável, pois em caso de falha não seria possível a execução por falha da sintaxe no comando executado. Deste modo, o profissional de BDRs pode aproveitar seus conhecimentos e desfrutar de um ambiente que explora e entrega excelentes resultados na recuperação de informações em BDs *NoSQL* orientados a grafos.

Com esse trabalho complementa-se uma solução (*SQLToNoSQL*) que é capaz de mapear um conjunto restrito de instruções *DDL* e *DML* para três tipos de BDs *NoSQL* (chave-valor, orientado a colunas, orientado a documentos), permitindo também a criação e manipulação de dados em BDs *NoSQL* orientados a grafos. Unificar todas as subdivisões dos BDs *NoSQL* em uma única aplicação amplia as necessidades dos usuários pelo uso de BDs *NoSQL*.

## Referências

- Abadi, D. (2009). Data management in the cloud: Limitations and opportunities. *IEEE Data Eng. Bull.*, 32:3–12.
- Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D., Silberschatz, A., and Rasin, A. (2009). Hadoopdb: An architectural hybrid of mapreduce and dbms technologies for analytical workloads. *Proc. VLDB Endow.*, 2(1):922–933.
- BRITO, R. W. (2010). Bancos de dados nosql x sgbd's relacionais: análise comparativa\*. page 6.
- e Nadine Anderle e Dalvan Griebler e Samuel Souza, D. R. (2018). Estudo comparativo de bancos de dados nosql. *Revista Eletrônica Argentina-Brasil de Tecnologias da Informação e da Comunicação*, 1(8).
- e Nicole Melo e Leandro dos Santos e Wellington de Oliveira, M. O. (2018). Banco de dados no-sql x banco de dados sql. *South American Development Society Journal*, 4(11):298.
- Schreiner, G. A. (2016). *SQLTOKEYNOSQL: UMA CAMADA PARA MAPEAMENTO DE ESQUEMAS RELACIONAIS E DE OPERAÇÕES SQL PARA BANCOS DE DADOS NOSQL BASEADOS EM CHAVES DE ACESSO*. Pós-graduação, Universidade Federal de Santa Catarina, Florianópolis.
- Stonebraker, M. (2012). New opportunities for new sql. *Commun. ACM*, 55(11):10–11.

- Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18.
- Émerson P. Homrich and Mergen, S. L. S. (2018). Comparação entre mysql e neo4j para o acesso a dados complexos usando linguagens declarativas. In *Anais da XIV Escola Regional de Banco de Dados*, Porto Alegre, RS, Brasil. SBC.