



UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS REITOR JOÃO DAVID FERREIRA LIMA
CURSO DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Telmo Henrique Valverde da Silva

Dango: Ferramenta para Projeto de Bancos de Dados NoSQL de Grafos a partir de Diagramas Entidade-Relacionamento Estendido

Florianópolis
2020

Telmo Henrique Valverde da Silva

Dango: Ferramenta para Projeto de Bancos de Dados NoSQL de Grafos a partir de Diagramas Entidade-Relacionamento Estendido

Trabalho de Conclusão de Curso do Curso de Graduação em Ciências da Computação do Campus Reitor João David Ferreira Lima da Universidade Federal de Santa Catarina para a obtenção do título de bacharel em Ciências da Computação.
Orientador: Prof. Ronaldo dos Santos Mello, Dr.

Florianópolis
2020

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

Silva, Telmo Henrique Valverde da

Dango: Ferramenta para Projeto de Bancos de Dados NoSQL de Grafos a partir de Diagramas Entidade-Relacionamento Estendido / Telmo Henrique Valverde da Silva ; orientador, Ronaldo dos Santos Mello, 2020.

108 p.

Trabalho de Conclusão de Curso (graduação) -
Universidade Federal de Santa Catarina, Centro Tecnológico,
Graduação em Ciências da Computação, Florianópolis, 2020.

Inclui referências.

1. Ciências da Computação. 2. Banco de dados de grafos.
3. Modelo EER. 4. NoSQL. I. Mello, Ronaldo dos Santos. II.
Universidade Federal de Santa Catarina. Graduação em
Ciências da Computação. III. Título.

Telmo Henrique Valverde da Silva

Dango: Ferramenta para Projeto de Bancos de Dados NoSQL de Grafos a partir de Diagramas Entidade-Relacionamento Estendido

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de “bacharel em Ciências da Computação” e aprovado em sua forma final pelo Curso de Graduação em Ciências da Computação.

Florianópolis, 24 de novembro de 2020.

Prof. Alexandre Gonçalves Silva, Dr.
Coordenador do Curso

Banca Examinadora:

Prof. Ronaldo dos Santos Mello, Dr.
Orientador

Prof. Antonio Carlos Mariani, Me.
Avaliador
Universidade Federal de Santa Catarina

Profa. Carina Friedrich Dorneles, Dra.
Avaliadora
Universidade Federal de Santa Catarina

Este trabalho é dedicado a todos que me auxiliaram na jornada árdua, mas recompensadora, de me tornar um cientista da computação.

AGRADECIMENTOS

Eu gostaria de agradecer aos meus pais, Telmo e Nilza, que se esforçaram para que eu pudesse ter um bom estudo e seguir a carreira que eu queria. Também às minhas irmãs, Fernanda e Renata, que sempre me deram todo o seu apoio e me ajudaram a lidar com as adversidades da vida. À minha namorada Luiza, cujo apoio foi de vital importância para que eu conseguisse concluir este trabalho e sem a qual essa jornada teria sido bem menos divertida. Aos grandes amigos que fiz durante a graduação, cujas amizades pretendo carregar pelo resto da vida. Aos professores que tive que realmente se importam com os seus alunos e os cativam a atingir o melhor de si. Este trabalho é para todos vocês.

RESUMO

O mundo moderno é composto por dados altamente interligados: conjuntos de estradas conectadas em pontos específicos, com sentidos que mudam em alguns trechos; redes elétricas compostas por postes, cabos e transformadores que precisam formar linhas de abastecimento para garantir a alta disponibilidade de energia ou mesmo redes sociais onde amigos se conectam, curtem postagens e as compartilham entre si. Porém como armazenar esses dados a fim de consultá-los de forma rápida e eficiente? Eles poderiam ser colocados em Bancos de Dados (BDs) convencionais no padrão relacional, mas isso implicaria em um número grande de custosas operações de junção e não permitiria a versatilidade de consultar relacionamentos em ambos os sentidos a menos que esses relacionamentos fossem explicitamente modelados dessa forma, tornando a modelagem mais sobrecarregada. A solução então vem por meio dos BDs de grafos, que utilizam da teoria de grafos para modelar esses problemas no mundo real. Como relacionamentos são um conceito central no modelo de dados de grafos, não há a necessidade de inferir conexões entre entidades através da utilização de chaves estrangeiras. Apesar da possibilidade de se modelar problemas reais diretamente através de nós e suas conexões, a forma mais amplamente utilizada de se modelar BDs em nível conceitual ainda é através do modelo Entidade-Relacionamento Estendido (EER). Dessa forma, esse trabalho propõe uma ferramenta que, recebendo uma modelagem EER de entrada, é capaz de transformá-la em um esquema para um BD de grafo. Esse esquema é gerado através de uma série de instruções escritas na linguagem Cypher (compatível com o BD Neo4j), que é montada conforme a modelagem EER provida. Na prática, a ferramenta pretende facilitar a adoção dos BDs de grafos por profissionais que já possuem o entendimento do modelo relacional e também permitir o reaproveitamento de modelos EER feitos previamente na construção desses BDs. Trabalhos relacionados propõem mapeamentos de outros modelos conceituais para BDs de grafos, até mesmo do próprio Entidade-Relacionamento (ER). Entretanto, nenhum fornece regras para o mapeamento de todos os conceitos presentes no modelo EER.

Palavras-chave: Banco de dados de grafos 1. Modelo EER 2. NoSQL 3.

ABSTRACT

The modern world is composed by highly connected data: sets of roads connected at specific points, which MAY change directions; electric grids composed by light poles, cables and converters that form supply lines to guarantee the high availability of electric power, and even social networks where friends are connected, like each other's posts and share them. But how could one store this kind of data in order to query it efficiently and easily? It could be stored inside a conventional relational database, but that would require a large number of costly JOIN operations and would not provide the versatility of querying these relations in both directions unless the data was explicitly modeled in that way, which would add a considerable overhead to the database physical modeling. The solution then comes through graph databases, which use graph theory to model and store this kind of real world problem in an easy way. Since relationships are a central concept in the graph model, there's no need to infer connections between the data through the usage of foreign keys. The connections can be traversed in both ways with inexpensive operations. Even though the usage of nodes and their relationships to model real world problems is fairly straightforward, the most adopted way of creating a conceptual model of a database is the Enhanced Entity-Relationship (EER) model. Given this motivation, this work proposes a tool that accepts an EER diagram as input and generates a schema for a graph database representing this modeling. The schema is generated through a series of Cypher instructions ready to run into a Neo4j database instance. In practice, this tool is intended to facilitate the adoption of graph databases by professionals that already have an understanding of the relational paradigm, and also allow them to reuse previous EER diagrams to generate their graph databases. Related works proposed mappings from other conceptual models to graph databases, even from the Entity-Relationship (ER) model, but none provided rules to map all concepts existing in the EER model.

Keywords: Graph database 1. EER diagram 2. NoSQL 3.

LISTA DE FIGURAS

Figura 1 – Variação de popularidade por tipo de BD de 2013 à 2020, mostrando BDs relacionais mantendo uma popularidade quase constante e a popularidade de BDs de grafos aumentando.	13
Figura 2 – Exemplo de um grafo simples simulando uma rede social.	14
Figura 3 – Exemplo de uma modelagem Entidade-Relacionamento (ER) na notação de Chen	19
Figura 4 – Exemplo de uma modelagem ER utilizando conceitos avançados	20
Figura 5 – Modelagem EER representando uma generalização e suas especializações.	21
Figura 6 – Modelagem EER representando uma união de entidades.	22
Figura 7 – Exemplo de uma modelagem lógica de BD relacional	23
Figura 8 – Modelagem conceitual na brModeloWeb	24
Figura 9 – Representação gráfica da modelagem lógica gerada pelo código DBML	25
Figura 10 – Modelagem ER simples descrito em “formato ER”	26
Figura 11 – Uma modelagem ER simples e um LPG equivalente no Neo4j	27
Figura 12 – SGBDs de grafos populares e suas abordagens quanto a processamento e armazenamento de grafos	29
Figura 13 – Meta-modelagem conceitual EER contendo estimativas quanto aos quatro V’s	31
Figura 14 – Comandos Cypher propostos por trabalho relacionado	33
Figura 15 – Comparação entre os trabalhos relacionados e o trabalho atual	34
Figura 16 – Arquitetura da ferramenta	35
Figura 17 – Tela principal da ferramenta	36
Figura 18 – Tela de ajuda da aplicação	38
Figura 19 – Representação JSON de uma modelagem EER	39
Figura 20 – Sequência de comandos Cypher representando o esquema do BD	40
Figura 21 – Tela para configuração da conexão com a instância local Neo4j	41
Figura 22 – Notificações de sucesso da execução da operação	41
Figura 23 – Declaração de uma entidade	45
Figura 24 – Inserção de um nó representando uma instância da entidade “Librarians”	45
Figura 25 – Declaração de um relacionamento	46
Figura 26 – Inserção do relacionamento representando uma instância do relacionamento “enrollment”	47
Figura 27 – Declaração de um relacionamento contendo timestamp	49
Figura 28 – Declaração de um relacionamento n-ário onde $n = 3$	51
Figura 29 – Inserção de um relacionamento n-ário com $n = 3$	52
Figura 30 – Declaração de uma entidade associativa	54
Figura 31 – Inserção de uma instância de uma entidade associativa	55

Figura 32 – Declaração de um atributo composto	56
Figura 33 – Inserção de uma instância de uma entidade e seu atributo composto . .	57
Figura 34 – Declaração de uma entidade com atributo multivalorado	58
Figura 35 – Inserção de uma instância de uma entidade com atributo multivalorado	58
Figura 36 – Declaração de uma modelagem EER contendo uma entidade fraca . . .	59
Figura 37 – Inserção de uma entidade fraca	60
Figura 38 – Declaração de uma especialização	61
Figura 39 – Inserção de dois nós representando entidades especializadas	61
Figura 40 – Declaração de um auto-relacionamento	63
Figura 41 – Inserção de nós demonstrando um auto-relacionamento	63
Figura 42 – Declaração de uma união	64
Figura 43 – Inserção de nós representando instâncias das entidades “Car” e “Truck”, respectivamente	65
Figura 44 – Modelagem EER contendo atributos compostos e relacionamentos . . .	66
Figura 45 – Representação gráfica da modelagem EER para o estudo de caso	68
Figura 46 – LPG respeitando o esquema definido pela modelagem EER para o estudo de caso	70
Figura 47 – Tempo de processamento da etapa ER to JSON nos diferentes cenários	71
Figura 48 – Tempo de processamento da etapa JSON to Cypher nos diferentes cenários	72
Figura 49 – Tempo médio de processamento de cada etapa por cenário	72
Figura 50 – Análise do tempo de processamento para múltiplos cenários de teste . .	72

LISTA DE ABREVIATURAS E SIGLAS

ANSI	American National Standards Institute
API	Application Programming Interface
APOC	Awesome Procedures On Cypher
BDs	Bancos de Dados
DBML	Database Markup Language
DSL	Domain-Specific Language
EER	Enhanced Entity-Relationship
ER	Entidade-Relacionamento
JSON	JavaScript Object Notation
LPG	Labeled Property Graph
OCL	Object Constraint Language
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
PPGCC	Programa de Pós-Graduação em Ciência da Computação
RDF	Resource Description Framework Schema
SGBD	Sistema de Gestão de Banco de Dados
SQL	Structured Query Language
UML	Unified Modeling Language

SUMÁRIO

1	INTRODUÇÃO	13
1.1	OBJETIVOS	16
1.1.1	Objetivo Geral	16
1.1.2	Objetivos Específicos	16
1.2	ESCOPO DO TRABALHO	16
1.3	MÉTODO DE PESQUISA	17
1.4	ESTRUTURA DO TRABALHO	17
2	FUNDAMENTAÇÃO TEÓRICA	18
2.1	BANCOS DE DADOS RELACIONAIS	18
2.2	PROJETO DE BANCO DE DADOS	18
2.2.1	Modelagem Conceitual	18
2.2.1.1	Modelo Entidade-Relacionamento	19
2.2.1.2	Modelo Entidade-Relacionamento Estendido	20
2.2.2	Modelagem Lógica	22
2.2.3	Modelagem Física	23
2.3	FERRAMENTAS PARA MODELAGEM DE BANCOS DE DADOS	24
2.3.1	brModeloWeb	24
2.3.2	dbdiagram.io	24
2.3.3	ERD: An Entity-Relationship diagram generator written in Haskell	26
2.3.4	Store and Visualize EER in Neo4j	26
2.4	NOSQL	27
2.5	BANCOS DE DADOS DE GRAFO	28
3	TRABALHOS RELACIONADOS	30
3.1	UMLTOGRAPHDB	30
3.2	A FOUR V'S DESIGN APPROACH OF NOSQL GRAPH DATABASES	30
3.3	MODELLING OF GRAPH DATABASES	32
3.4	MODEL-DRIVEN DESIGN OF GRAPH DATABASES	32
3.5	PROJETO LÓGICO DE BANCO DE DADOS NOSQL DE GRAFOS A PARTIR DE UM MODELO CONCEITUAL BASEADO NO MODELO ER	33
3.6	COMPARAÇÃO DOS TRABALHOS EXISTENTES	34
4	FERRAMENTA PROPOSTA: DANGO	35
4.1	ARQUITETURA	35
4.2	INTERFACE COM O USUÁRIO E FUNCIONALIDADES DA FERRAMENTA	36
4.3	MAPEAMENTO CONCEITUAL-GRAFO	41
4.3.1	Direção dos Relacionamentos	43
4.4	NOTAÇÃO EER EM TEXTO E REGRAS DE MAPEAMENTO	44
4.4.1	Entidade	44

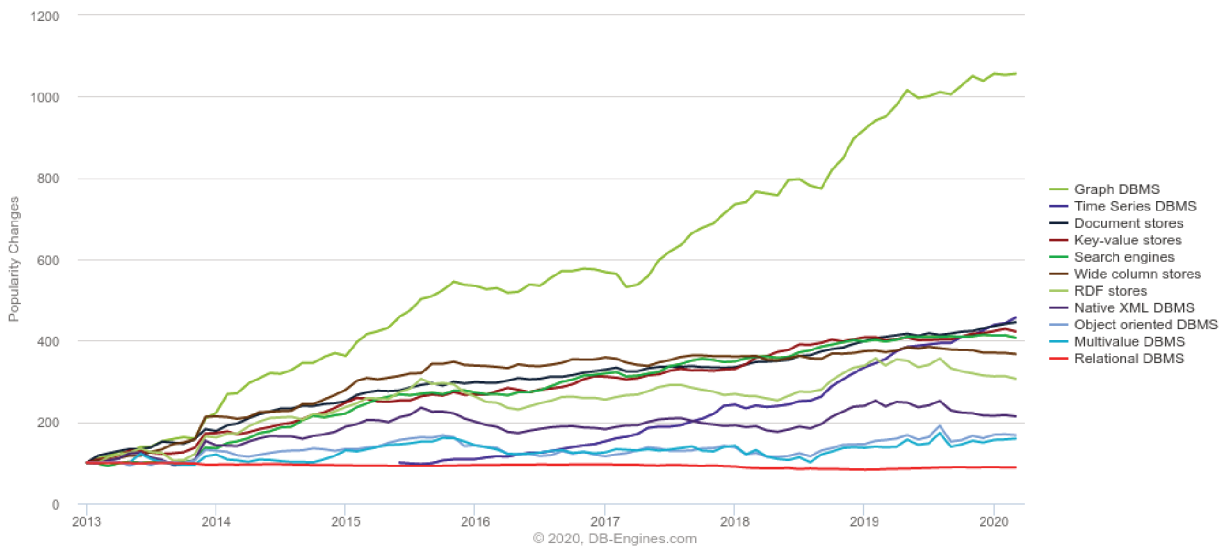
4.4.2	Relacionamento	46
4.4.3	Relacionamento entre três ou mais entidades	50
4.4.4	Entidade Associativa	53
4.4.5	Atributos Compostos	56
4.4.6	Atributos Multivalorados	58
4.4.7	Entidades Fracas	59
4.4.8	Especializações e Generalizações	60
4.4.9	Auto-Relacionamento	62
4.4.10	União	64
4.4.11	Strict mode	66
5	ESTUDO DE CASO: SISTEMA DE INFORMAÇÃO PARA UMA BIBLIOTECA	67
6	ANÁLISE DE DESEMPENHO	71
7	CONCLUSÃO	74
	REFERÊNCIAS	76
	APÊNDICE A – MODELAGEM EER REFERENTE AO ESTUDO DE CASO (NA NOTAÇÃO EER EM TEXTO)	80
	APÊNDICE B – ESQUEMA NEO4J REFERENTE AO ESTUDO DE CASO	83
	APÊNDICE C – LPG REFERENTE AO ESTUDO DE CASO	95
	APÊNDICE D – ARTIGO NO FORMATO SBC	97

1 INTRODUÇÃO

Bancos de Dados (BDs) relacionais são conhecidos pelos seus rígidos controles de integridade dos dados. Esses controles limitam a sua eficiência em situações com um número de dados muito grande (*big data*). Ainda, o seu modelo organizacional dificulta o armazenamento de dados sem esquema rígido e o de dados que têm muitas ligações entre si. Na intenção de contornar esses problemas surgiram os BDs NoSQL, que oferecem alternativas ao modelo relacional para o armazenamento e gerência de dados (CATTELL, 2011).

BDs de grafos são BDs NoSQL que permitem o armazenamento e acesso de forma eficiente à relacionamentos complexos e dinâmicos em dados altamente conectados (ROBINSON; WEBBER; EIFREM, 2015), tornando-os bastante adequados para dados de aplicações como redes sociais, sistemas de recomendações, sistemas de informação geográfica, entre outros. De fato, apesar de os BDs relacionais ainda serem mais populares, o interesse em BDs de grafos tem crescido rapidamente nos últimos anos, como indica o site *DB-Engines* (Figura 1).

Figura 1 – Variação de popularidade por tipo de BD de 2013 à 2020, mostrando BDs relacionais mantendo uma popularidade quase constante e a popularidade de BDs de grafos aumentando.

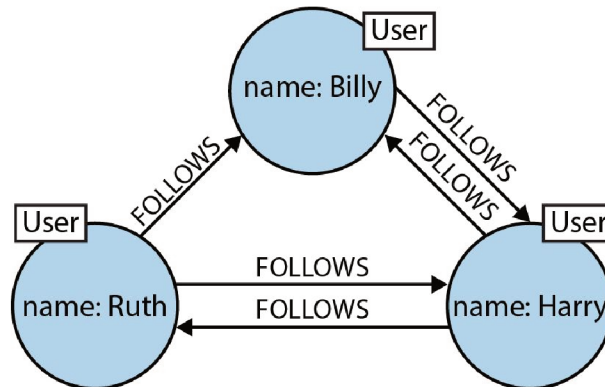


Fonte: (DB-ENGINES, 2020)

Mas afinal, o que são grafos? Grafos são um conceito proveniente da matemática, mais especificamente da chamada teoria dos grafos (WEST, 2002). São estruturas compostas de dois conjuntos: o dos vértices (aqui chamados de nós) e o das arestas (aqui chamadas de relacionamentos). Nós podem ser imaginados como uma espécie de objeto ou indivíduo e os seus relacionamentos são conexões que se estabelecem entre eles. A partir

dessa definição, muitas situações reais se tornam fáceis de modelar, como por exemplo, um conjunto de pessoas que seguem umas às outras em uma rede social (Figura 2).

Figura 2 – Exemplo de um grafo simples simulando uma rede social.



Fonte: (ROBINSON; WEBBER; EIFREM, 2015)

O grafo da Figura 2 possui algumas características interessantes:

- Como qualquer outro grafo, ele possui pelo menos um nó e zero ou mais relacionamentos;
- Os nós possuem propriedades;
- Os nós podem ter etiquetas;
- Os relacionamentos têm nomes e direções;
- Apesar de não mostrado na Figura 2, os relacionamentos podem ter também propriedades.

Essas características fazem com que esse grafo seja um *Labeled Property Graph (LPG)* (ROBINSON; WEBBER; EIFREM, 2015) ou ainda um *Property Graph* (RODRIGUEZ; NEUBAUER, 2010), ou seja, um grafo etiquetado e com propriedades. Esse tipo de grafo é a base para os BDs de grafos, pois apesar de conceitualmente simples, têm um poder de expressividade muito alto.

É importante perceber, porém, que a definição de um LPG não impede que nós com a mesma etiqueta possuam propriedades diferentes entre si ou que propriedades iguais tenham tipos diferentes em cada nó. Isso demonstra a flexibilidade do modelo. Grafos do tipo *Resource Description Framework Schema (RDF)*, utilizados por aplicações da *Web semântica*, são uma especificação mais detalhada desse tipo de grafo (RODRIGUEZ; NEUBAUER, 2010).

Apesar da forma intuitiva como grafos podem ser utilizados para representar diversos tipos de situações, a forma de representação conceitual predominante na área de BD

ainda é o modelo ER. Dessa forma, se faz bastante necessária uma ferramenta capaz de não só transformar quaisquer modelagens ER convencionais em grafos, mas também em BDs de grafo. Dessa forma será possível transformar modelagens de dados de alto nível, como é o caso de modelagens ER, diretamente em BDs de grafo.

Este trabalho se propõe a desenvolver uma ferramenta para projeto conceitual, lógico e físico de BDs NoSQL de grafos e mapeamento de modelagens ER e Enhanced Entity-Relationship (EER) para LPG, incluindo a implementação de restrições de integridade no Neo4j para garantir que os dados se comportarão conforme as restrições impostas pela modelagem. O Neo4j foi o BD escolhido para esse trabalho por ser o BD de grafo mais popular na atualidade. Dessa forma, o formato de saída da ferramenta proposta deve ser um série de comandos na linguagem *Cypher*, utilizada por esse BD. Essa série de comandos deve ser capaz de gerar o esquema Neo4j resultante da modelagem EER dada como entrada. Para esse fim, são propostas uma série de mapeamentos dos conceitos existentes no EER para LPG.

Alguns dos trabalhos relacionados, como (DE VIRGILIO; MACCIONI; TORLONE, 2014), (AKOKA; COMYN-WATTIAU; PRAT, 2017), (POKORNY, 2017) e (SOUSA, 2018) sugerem abordagens para realizar o mapeamento ER para LPG. Todos eles, entretanto, propõem variações próprias dos modelos ER ou EER diminuindo o poder de expressividade do modelo para facilitar o seu mapeamento para um LPG. Alguns deles buscam ser mais agnósticos em relação ao Sistema de Gestão de Banco de Dados (SGBD) nas suas soluções, como (DE VIRGILIO; MACCIONI; TORLONE, 2014) e (POKORNY, 2017), e outros mais específicos, como (AKOKA; COMYN-WATTIAU; PRAT, 2017) e (SOUSA, 2018), que implementam soluções voltadas ao *Neo4j*.

O trabalho de (DE VIRGILIO; MACCIONI; TORLONE, 2014) realiza o controle das restrições de integridade do dado a nível da aplicação e não do SGBD. Dessa forma, as restrições só são mantidas enquanto as interações com o BD forem realizadas exclusivamente pela ferramenta fornecida por ele. O trabalho de (AKOKA; COMYN-WATTIAU; PRAT, 2017) fornece algoritmos para mapear alguns dos conceitos do EER para LPG, entretanto ele não se preocupa em realizar o controle das cardinalidades dos relacionamentos durante a utilização do BD. O trabalho de (POKORNY, 2017), por ser muito agnóstico, não fornece restrições de integridade suficientes para que qualquer um dos seus SGBDs alvo respeite todas as restrições da modelagem conceitual realizada. O trabalho de (SOUSA, 2018) propõe extensões para a linguagem *Cypher* do *Neo4j* para lidar com as situações que não são suportadas nativamente pelo SGBD. O trabalho, entretanto, não fornece implementações para essas extensões.

Nenhum desses trabalhos fornece uma solução que contemple simultaneamente as funcionalidades de especializações e de uniões do modelo EER e que contenha uma implementação garantindo que o SGBD alvo (nesse caso o *Neo4j*) respeite todas as restrições presentes na modelagem. O trabalho aqui proposto utiliza uma combinação das restrições

de integridade de dados disponibilizadas pelo *Neo4j* e da interface de *triggers* fornecidas pelo plugin *Awesome Procedures On Cypher (APOC)* para realizar todo o controle de integridade dos dados a nível do SGBD. Dessa forma, são propostos mapeamentos para todos os conceitos do modelo EER que podem ser executados diretamente em uma instância do *Neo4j* para que ela passe a reforçar as regras da modelagem EER.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Este trabalho tem como objetivo principal o desenvolvimento de uma ferramenta que transforma modelagens ER e EER em esquemas para o BD de grafos Neo4j. O seu foco está em facilitar a utilização de BDs de grafos por projetistas de BDs relacionais, que já estão habituados a utilizar esses modelos para o projeto conceitual do BD. A ferramenta tem uma interface web através da qual a modelagem EER pode ser inserida e então mapeada para uma sequência de comandos na linguagem Cypher contendo o esquema do BD.

1.1.2 Objetivos Específicos

Os objetivos específicos deste trabalho são os seguintes:

- Desenvolvimento de uma notação para especificação textual de modelagens EER;
- Desenvolvimento do módulo que realiza o *parsing* da modelagem EER construindo uma estrutura intermediária no formato *JavaScript Object Notation (JSON)*, a fim de facilitar o consumo das informações da modelagem pelos demais módulos;
- Desenvolvimento do módulo que consome a estrutura intermediária para gerar a visualização gráfica da modelagem EER;
- Desenvolvimento do módulo que consome a estrutura intermediária para gerar uma sequência de instruções Cypher que definem o esquema do BD.

1.2 ESCOPO DO TRABALHO

O produto final deste trabalho é uma ferramenta que aceita modelagens ER e EER e permite a conversão dessas modelagens em esquemas para o BD de grafo Neo4j. Essa aplicação deve:

- Permitir a construção de uma modelagem ER ou EER válida do zero;
- Permitir o carregamento de modelagens ER ou EER existentes como entrada para mapeamento para BD de grafo;

- Implementar regras para o mapeamento de modelagens ER e EER para um LPG equivalente;
- Gerar a sequência de comandos Cypher que faz a construção do esquema para o BD Neo4j e reforça as restrições de integridade no LPG.

É importante ressaltar que as regras propostas para o mapeamento ER e EER para LPG podem ser utilizadas em qualquer BD de grafo. Entretanto, as restrições de integridade implementadas são compatíveis apenas com o Neo4j.

1.3 MÉTODO DE PESQUISA

A pesquisa realizada nesse trabalho classifica-se como aplicada e pretende utilizar os conhecimentos estabelecidos no estado da arte, relacionados ao mapeamento de modelagens conceituais para BDs de grafos, para implementar uma ferramenta que converte uma modelagem EER para uma série de comandos *Cypher* capaz de gerar um esquema para o BD de grafo *Neo4j*, de forma que os dados inseridos respeitarão as restrições de integridade impostas pela modelagem.

1.4 ESTRUTURA DO TRABALHO

O presente trabalho se divide em seis capítulos. Este primeiro capítulo apresenta a introdução do trabalho, seus objetivos, escopo e método de pesquisa. O capítulo 2 apresenta a fundamentação teórica, incluindo os conceitos de BDs relacionais, projeto de BDs, NoSQL, BDs de grafo e descrições de algumas ferramentas de modelagem de BDs, incluindo suas similaridades e diferenças com a ferramenta aqui proposta. O capítulo 3 apresenta os trabalhos relacionados, focando principalmente em trabalhos que realizam mapeamentos de modelagens conceituais para BDs de grafo e esclarecendo as suas vantagens e desvantagens em relação às técnicas de mapeamento apresentadas nesse trabalho. O capítulo 4 apresenta a ferramenta proposta, sua arquitetura, a interface gráfica da aplicação, a notação EER em texto criada para uso na ferramenta e todas as regras de mapeamento EER para LPG compatível com Neo4j. O capítulo 5 apresenta um estudo de caso demonstrando a utilização da ferramenta para realizar o mapeamento EER para LPG de uma modelagem que faz uso de todas as funcionalidades implementadas. O capítulo 6 apresenta uma análise de desempenho dos algoritmos implementados. O capítulo 7 apresenta a conclusão do trabalho, incluindo suas contribuições para o estado da arte e também possibilidades de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Essa seção fornece o embasamento necessário para compreender o contexto em que está inserida a ferramenta proposta.

2.1 BANCOS DE DADOS RELACIONAIS

O modelo relacional é uma forma de representação de dados através de tabelas, onde cada registro é uma tupla. Ele permite relações de ordem n entre os dados. Para tal, utiliza chaves primárias, que identificam unicamente os registros, e chaves estrangeiras, que indicam a chave primária de um registro em outra tabela permitindo a definição de relacionamentos entre registros (CODD, 1983). A maioria dos BDs utilizados atualmente são BDs relacionais, ou seja, se baseiam nesse modelo (PRATT; ADAMSKI, 2011). A linguagem de consulta considerada padrão para realizar operações em um BD relacional é a *Structured Query Language (SQL)*, como padronizado pelo *American National Standards Institute (ANSI)*.

BDs relacionais seguem as propriedades ACID (ROE, 2013):

- **Atomicidade:** Ou todas as tarefas em uma transação são realizadas ou nenhuma é. Nenhuma transação é executada parcialmente;
- **Consistência:** Nenhuma transação pode levar o BD para um estado inconsistente;
- **Isolamento:** Nenhuma transação pode acessar dados de uma transação que ainda não tenha sido terminada;
- **Durabilidade:** Depois de uma transação ter sido executada, suas ações sobre os dados devem ser persistidas no BD e não podem ser desfeitas.

2.2 PROJETO DE BANCO DE DADOS

Nas subseções seguintes são descritas as três etapas do projeto clássico de um BD.

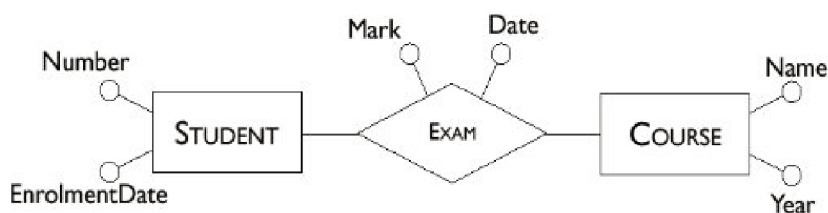
2.2.1 Modelagem Conceitual

A modelagem conceitual é a primeira etapa na construção de um BD. Esta é a etapa com maior nível de abstração onde são representados os requisitos de dados do domínio de forma independente do modelo adotado por um SGBD (ELMASRI; NAVATHE, 2015). O modelo de dados mais amplamente utilizado para a definição de uma modelagem conceitual é o modelo Entidade-Relacionamento (ER). Esse modelo tem como objetivo representar a semântica dos dados no mundo real de forma intuitiva e permitir uma forma unificada de representá-los (CHEN, 1976).

2.2.1.1 Modelo Entidade-Relacionamento

No modelo ER, os dados são divididos em *entidades*, que classificam os objetos de interesse, e *relacionamentos*, que descrevem as associações entre eles. Na notação de *Chen* (CHEN, 1976), as entidades são representadas como retângulos e as relações que as conectam como losangos. Os atributos das entidades e das relações são representados através de círculos ou ovais, como é o caso do exemplo mostrado na Figura 3.

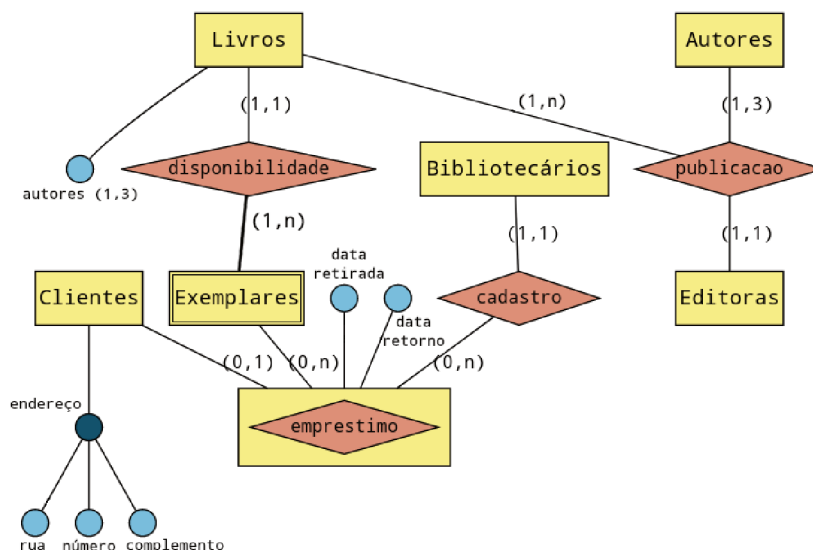
Figura 3 – Exemplo de uma modelagem ER na notação de Chen



Fonte: (EASTERBROOK, 2005)

Em (ELMASRI; NAVATHE, 2015), são atribuídos também ao modelo ER os conceitos de atributo composto (atributo que contém múltiplos campos), atributo multivalorado (atributo que pode conter mais de um valor), entidade fraca (entidade que depende da existência de outra entidade para existir), entidade associativa (entidade que representa uma associação entre entidades através de um relacionamento) e relacionamento n-ário (relacionamento entre três ou mais entidades). A Figura 4 mostra uma modelagem ER contendo todos os conceitos mencionados.

Figura 4 – Exemplo de uma modelagem ER utilizando conceitos avançados



Fonte: Elaboração própria

Nessa modelagem:

- O item “endereço” é um atributo composto contendo “rua”, “número” e “complemento”. Ele é denotado por um círculo conectado a outros círculos;
- O item “autores” (com “a” minúsculo) é um atributo multivalorado, podendo ter de 1 a 3 valores. Ele é denotado por um círculo cujo nome é seguido das suas cardinalidades;
- O item “Exemplares” é uma entidade fraca, cuja existência depende de um “Livro”. Ele é denotado por um retângulo com um contorno interno e conectado a um relacionamento através de uma linha mais grossa;
- O item “empréstimo” é uma entidade associativa que representa o relacionamento entre um “cliente” e um “exemplar” com a qual “Bibliotecários” se relaciona. Ele é denotado por um retângulo com um losango dentro;
- O item “publicação” é um relacionamento n-ário entre de 1 a 3 “Autores”, uma “Editora” e 1 ou mais “Livros”. Ele é denotado por um losango conectado a três ou mais entidades.

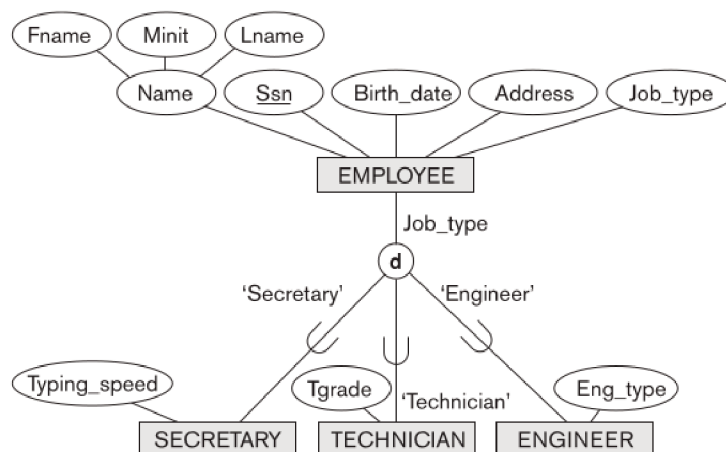
2.2.1.2 Modelo Entidade-Relacionamento Estendido

O modelo *Enhanced Entity-Relationship* (EER), ou ainda modelo Entidade-Relacionamento Estendido, é descrito por (ELMASRI; NAVATHE, 2015) como uma tentativa de aumentar o poder de expressividade do modelo ER para permitir a

modelagem de aplicações mais complexas. O modelo conta com todas as características originais do modelo ER com a adição dos seguintes aspectos:

- **Especializações e generalizações:** através de especializações (subclasses) e generalizações (superclasses), o modelo EER permite a representação do conceito de herança de tipos. Os atributos modelados para as generalizações tornam-se também atributos das suas especializações. Dessa forma, é possível representar entidades que compartilham uma estrutura de base, porém apresentam características próprias. A Figura 5 mostra uma modelagem EER onde se define a relação semântica entre a generalização “*Employee*” (funcionário) com as suas especializações “*Secretary*” (secretário), “*Technician*” (técnico) e “*Engineer*” (engenheiro).

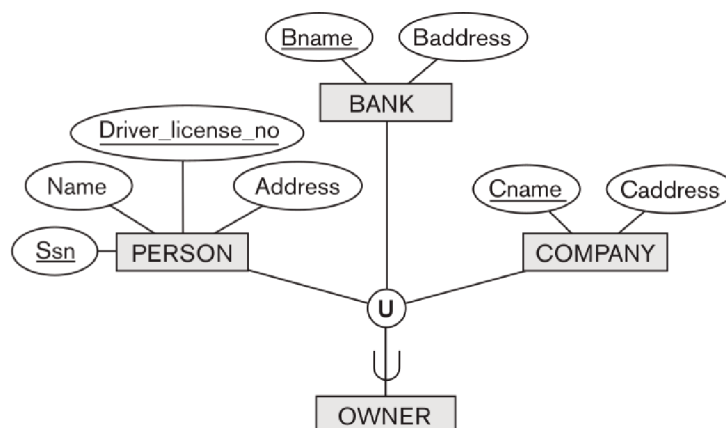
Figura 5 – Modelagem EER representando uma generalização e suas especializações.



Fonte: (ELMASRI; NAVATHE, 2015)

- **Uniões:** uniões permitem que entidades que não necessariamente têm estruturas similares sejam agrupadas em categorias. A Figura 6 mostra uma modelagem EER na qual entidades com estruturas completamente diferentes entre si (“*Person*” (pessoa), “*Bank*” (banco) e “*Company*” (empresa)) são agrupadas em “*Owner*” (dono) indicando que, caso houvesse uma entidade que se relacionasse com um “*Owner*”, essa entidade poderia estar se relacionando com uma entidade de qualquer um desses três tipos.

Figura 6 – Modelagem EER representando uma união de entidades.



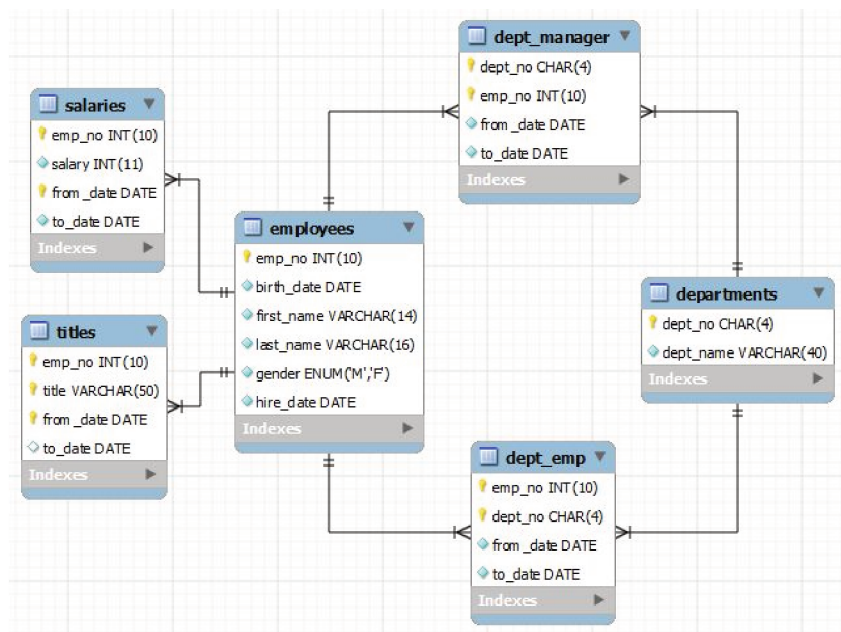
Fonte: (ELMASRI; NAVATHE, 2015)

2.2.2 Modelagem Lógica

A modelagem lógica realiza a adequação da modelagem conceitual para um modelo de dados adotado por um SGBD, como por exemplo, o modelo relacional. Nessa etapa, a ênfase está em tomar decisões que permitam a forma mais eficiente de dispor esses dados em um BD.

No caso do modelo relacional, há um compromisso entre evitar um grande número de tabelas, evitar atributos opcionais e evitar muitos controles de integridade do BD (ELMASRI; NAVATHE, 2015). A Figura 7 contém um exemplo de modelo lógico de um BD relacional. Os atributos marcados com uma chave amarela fazem parte da chave primária de cada tabela, enquanto que os atributos *emp_no* nas tabelas *salaries*, *titles*, *dept_manager* e *dept_emp* são chaves estrangeiras. O mesmo vale para *dept_no* nas tabelas *dept_manager* e *dept_emp*.

Figura 7 – Exemplo de uma modelagem lógica de BD relacional



Fonte: (HOCK-CHUAN, 2012)

2.2.3 Modelagem Física

A modelagem física é a última etapa do projeto de um BD, onde a partir da modelagem lógica, é gerada a definição do esquema do BD na linguagem de consulta aceita pelo SGBD alvo. No caso dos BDs relacionais, essa etapa produz instruções de criação de tabelas na linguagem SQL. Note que apesar da existência do padrão ANSI SQL, diferentes SGBDs relacionais (como *Oracle*, *MySQL*, *Microsoft SQL Server* e *PostgreSQL*) implementam a linguagem de maneiras distintas, contendo geralmente dialetos e funcionalidades incompatíveis entre si. O Trecho de Código 1 mostra a declaração SQL compatível com o SGBD *MySQL* para gerar a tabela “employees” da modelagem lógica mostrada na Figura 7.

Trecho de Código 1 – Consulta SQL (MySQL) para gerar a tabela “employee”. Fonte: (HOCK-CHUAN, 2012)

```

1 CREATE TABLE employees (
2     emp_no      INT          NOT NULL,
3     birth_date  DATE         NOT NULL,
4     first_name  VARCHAR(14)  NOT NULL,
5     last_name   VARCHAR(16)  NOT NULL,
6     gender     ENUM ('M','F') NOT NULL,
7     hire_date  DATE         NOT NULL,
8     PRIMARY KEY (emp_no)
9 );

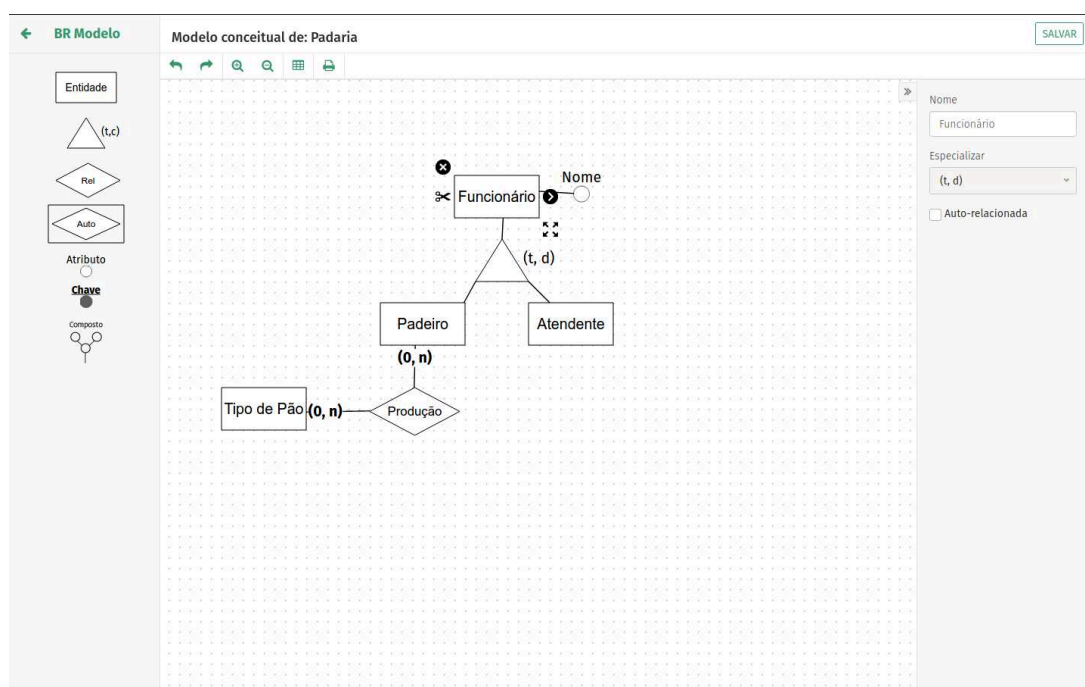
```


2.3 FERRAMENTAS PARA MODELAGEM DE BANCOS DE DADOS

2.3.1 brModeloWeb

A *brModeloWeb* (SOUZA NETO, 2016) é uma aplicação web específica para modelagem conceitual e lógica de BDs relacionais. Através dela é possível construir uma modelagem ER e convertê-la para uma modelagem lógica de BD relacional. Ainda, a ferramenta permite converter essa modelagem lógica em uma modelagem física através de instruções SQL contendo o esquema do BD projetado. É possível também entrar diretamente com a modelagem lógica para gerar a modelagem física. A Figura 8 mostra a interface gráfica para modelagem conceitual da ferramenta *brModeloWeb*.

Figura 8 – Modelagem conceitual na brModeloWeb



Fonte: (SOUZA NETO, 2016)

Essa aplicação, entretanto, não fornece opções para geração do esquema SQL específico para o dialeto de um SGBD relacional específico. Os únicos tipos aceitos na modelagem lógica são DATE, FLOAT, VARCHAR(n), CHAR(n) e INT, deixando de lado muitos dos tipos existentes no padrão ANSI SQL.

2.3.2 dbdiagram.io

O dbdiagram.io (HOLISTICS, 2020) é uma aplicação web voltada também para a modelagem lógica de BDs relacionais. A aplicação se baseia na utilização de uma *Domain-Specific Language (DSL)* chamada *Database Markup Language (DBML)* através da qual

se pode modelar tabelas, atributos e relações com chaves primárias e estrangeiras (PHAN, 2020).

Ao final da modelagem lógica é possível gerar o esquema SQL de forma compatível com os SGBDs *MySQL* e *PostgreSQL*. Pode-se também importar esquemas extraídos de BDs já existentes para utilização na modelagem. Essa aplicação porém, não suporta a etapa de modelagem conceitual. O Trecho de Código 2 mostra um exemplo de modelagem lógica descrita em DBML e a Figura 9 mostra a representação gráfica dessa modelagem.

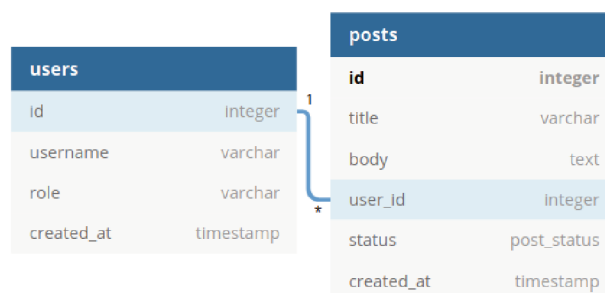
Trecho de Código 2 – Exemplo de modelagem lógica descrita em DBML. Fonte: (PHAN, 2020)

```

1 Table users {
2   id integer
3   username varchar
4   role varchar
5   created_at timestamp
6 }
7
8 Table posts {
9   id integer [primary key]
10  title varchar
11  body text [note: 'Content of the post']
12  user_id integer
13  status post_status
14  created_at timestamp
15 }
16
17 Enum post_status {
18  draft
19  published
20  private [note: 'visible via URL only']
21 }
22
23 Ref: posts.user_id > users.id // many-to-one

```

Figura 9 – Representação gráfica da modelagem lógica gerada pelo código DBML

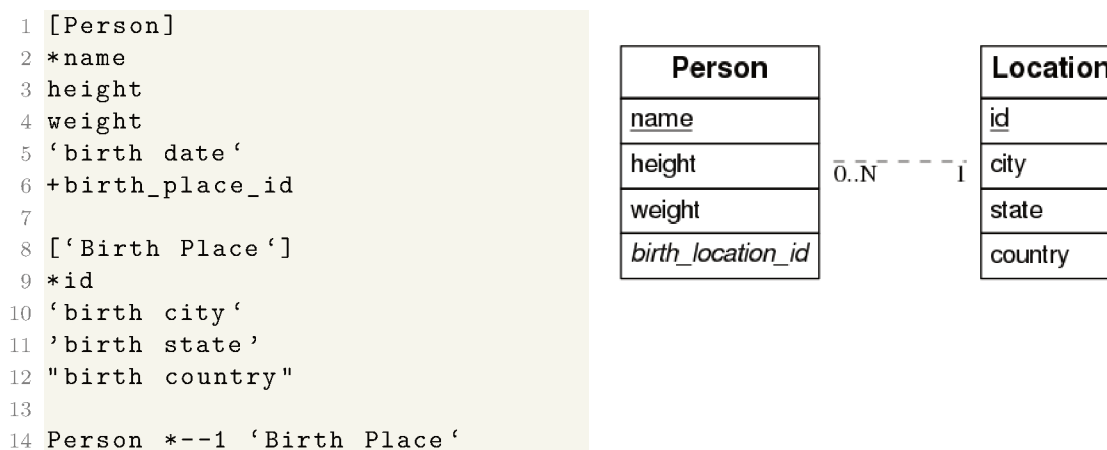


Fonte: (PHAN, 2020)

2.3.3 ERD: An Entity-Relationship diagram generator written in Haskell

O ERD (GALLANT, 2020) é uma ferramenta para geração de modelagens ER a partir de um texto de entrada. A ferramenta introduz uma DSL chamada de “formato ER”, através do qual esquemas ER podem ser descritos em texto, o qual pode ser utilizado para gerar uma visualização gráfica da modelagem. A Figura 10 mostra uma modelagem ER descrita nessa notação e a representação gráfica dessa modelagem gerada pela ferramenta.

Figura 10 – Modelagem ER simples descrito em “formato ER”



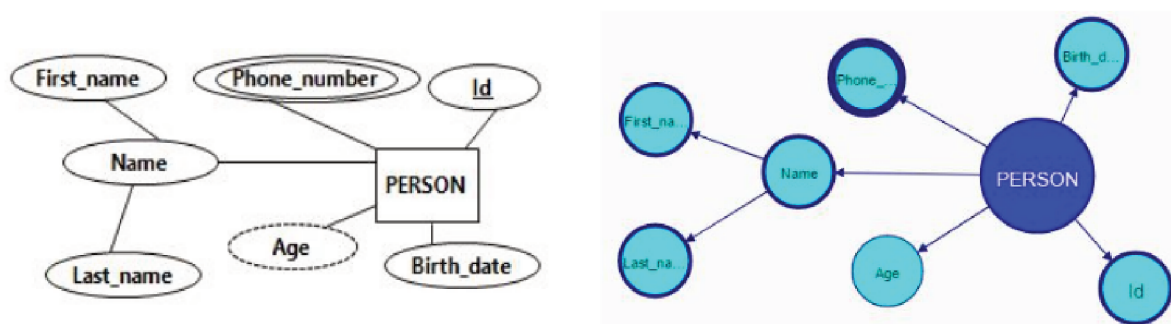
Fonte: (GALLANT, 2020)

A notação “formato ER”, entretanto, não possui a capacidade de representar todos os conceitos dos modelos ER e EER como: atributos compostos, relacionamento n-ários, especializações e uniões. Dessa forma, o trabalho aqui proposto introduz uma notação própria, chamada “EER em texto”, detalhada na seção 4.4.

2.3.4 Store and Visualize EER in Neo4j

Esse trabalho (VÁGNER, 2018) fornece uma série de comandos Cypher para construção e armazenamento de uma modelagem ER em uma instância do BD de grafo Neo4j, ou seja, ele representa a modelagem em um LPG aceito pelo Neo4j. A Figura 11 mostra à esquerda uma modelagem ER e à direita o LPG gerado pelo algoritmo de mapeamento proposto.

Figura 11 – Uma modelagem ER simples e um LPG equivalente no Neo4j



Fonte: (VÁGNER, 2018)

O uso de *constraints* do Neo4j é sugerido para garantir algumas propriedades do modelo, como por exemplo, a unicidade do nome de uma entidade no modelo e a unicidade do nome de um relacionamento entre as mesmas entidades.

O escopo do trabalho não engloba a implementação de um BD de grafo utilizando a modelagem ER construída, apenas como armazenar uma representação dessa modelagem dentro do BD. Dessa forma, ele é diferente do trabalho aqui proposto cujo foco é a construção do esquema para que os dados no BD de grafo respeitem a modelagem conceitual realizada.

2.4 NOSQL

Com o desenvolvimento da Internet e da computação na nuvem, o número de dados cresceu muito rápido. Dessa forma, surgiu a necessidade dos sistemas computacionais lidarem com *Big Data* de forma eficiente. Percebeu-se que as restrições impostas pelo princípio ACID carregam um peso muito grande sob a eficiência do banco e que para muitos casos as garantias que ele oferece simplesmente não são necessárias (JING HAN *et al.*, 2011). Além disso, os BDs relacionais não são muitas vezes adequados para a representação de dados cujas instâncias são complexas e heterogêneas.

Dessa forma, surgiram os BDs NoSQL. NoSQL não significa que eles não entendem consultas em SQL (apesar de muitas vezes ser o caso), mas sim que eles não são relacionais. Os tipos mais populares de BDs NoSQL são:

- **Documento:** São orientados à representação de objetos complexos e caracterizam-se pela organização de dados livre de esquema, o que significa que dados de vários formatos podem ser guardados juntos. Ainda, os atributos de um objeto podem ter domínios complexos, ou seja, manter estruturas complexas. Um exemplo é o SGBD MongoDB¹;

¹ <https://www.mongodb.com/>

- **Chave-valor:** É o tipo mais simples de modelo de BD, sendo os dados guardados em pares chave-valor. A chave funciona como um índice para encontrar o valor. Podem ser pensados como um grande dicionário. Um exemplo é o SGBD Redis²;
- **Colunares:** Trabalham com o conceito de tabelas mas, diferentemente dos BDs relacionais, dados na mesma tabela não precisam seguir a mesma estrutura, ou seja, apresentar o mesmo conjunto de colunas. Um exemplo é o SGBD Cassandra³;
- **Grafo:** São orientados a grafos, ou seja, os dados são organizados entre nós e arestas, o que torna a consulta a relacionamentos entre dados muito barato. São geralmente utilizados em aplicações onde os dados do domínio apresentam muitos relacionamentos. Um exemplo é o SGBD Neo4j⁴;

Os BDs NoSQL, em sua maioria, adotam o princípio BASE (ROE, 2013):

- **Basically Available:** Haverá uma resposta para toda requisição, porém o dado pode estar temporariamente em um estado inconsistente;
- **Soft state:** O estado do sistema pode mudar mesmo sem haver transações do usuário devido à propriedade de consistência eventual;
- **Eventual consistency:** O sistema eventualmente se tornará consistente após o fim de uma transação. Os dados atualizados se propagarão para todos os lugares nos quais devem ser atualizados ou cadastrados cedo ou tarde, tornando o BD consistente.

2.5 BANCOS DE DADOS DE GRAFO

Conforme comentado anteriormente, BDs de grafo são BDs NoSQL que se baseiam no modelo de grafos para representar os seus dados. Mais especificamente, os seus dados estão modelados em um LPG, o que possibilita que informações sejam guardadas tanto nos nós quanto nos seus relacionamentos e que os relacionamentos sempre terão direção.

É válido ressaltar que nem todo BD de grafos guarda os seus dados em um grafo (armazenamento nativo). Alguns serializam os grafos em dados relacionais ou orientados a objetos para fins de armazenamento (armazenamento não-nativo). Da mesma forma, nem todo BD de grafos faz com que nós que estão relacionados apontem fisicamente um para o outro (processamento nativo) (ROBINSON; WEBBER; EIFREM, 2015). BDs de grafo que não o fazem não suportam adjacência livre de índice e tendem a ter um desempenho de consulta mais lento, pois na medida que os dados crescem, torna-se mais custoso consultar a lista de índices para acessar os relacionamentos. Em BDs de grafos com adjacência

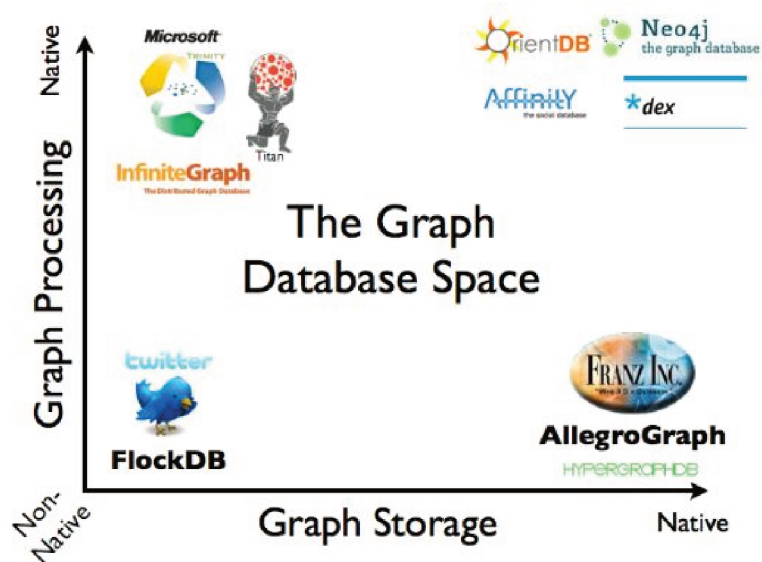
² <https://redis.io/>

³ <https://cassandra.apache.org/>

⁴ <https://neo4j.com/>

livre de índice (processamento nativo) o tempo de consulta de um relacionamento não depende do número de dados e tem um tempo constante (RODRIGUEZ; NEUBAUER, 2010). A Figura 12 mostra uma comparação entre os SGBDs de grafo populares e as suas abordagens quanto à processamento e armazenamento de grafos.

Figura 12 – SGBDs de grafos populares e suas abordagens quanto a processamento e armazenamento de grafos



Fonte: (ROBINSON; WEBBER; EIFREM, 2015)

Mais detalhes sobre o modelo de grafo estão descritos no início do Capítulo 1.

3 TRABALHOS RELACIONADOS

Este capítulo apresenta, de forma sucinta, soluções existentes para o mapeamento de modelagens conceituais para BDs de grafo.

3.1 UMLTOGRAPHDB

O framework *UMLtoGraphDB* (DANIEL; SUNYÉ; CABOT, 2016) fornece um método para o mapeamento de um modelo conceitual *Unified Modeling Language (UML)* contendo restrições de regra de negócio na linguagem *Object Constraint Language (OCL)* para BDs de grafos. O modelo UML é transformado em um grafo a partir de uma série de regras e as restrições OCL são transformadas em código Java para impor as regras de negócio sobre o BD. O esquema do BD pode ser gerado sobre uma instância Neo4j ou OrientDB. Dessa forma, a abordagem desse trabalho consiste em criar uma aplicação que intermedeia as interações com o BD, não permitindo que os dados deixem de estar consistentes com o esquema proposto.

O esquema do BD é gerado na linguagem de consulta *Gremlin*, assim como as consultas por trás dos métodos da aplicação Java. O Gremlin faz parte do projeto Apache *TinkerPop*, que é um framework para computação de grafos para processamento *Online Transaction Processing (OLTP)* e *Online Analytical Processing (OLAP)*. O benefício de utilizá-lo é que, através de extensões, ele é compatível com vários BDs de grafo existentes, como Neo4j, OrientDB, Amazon Neptune e Titan.

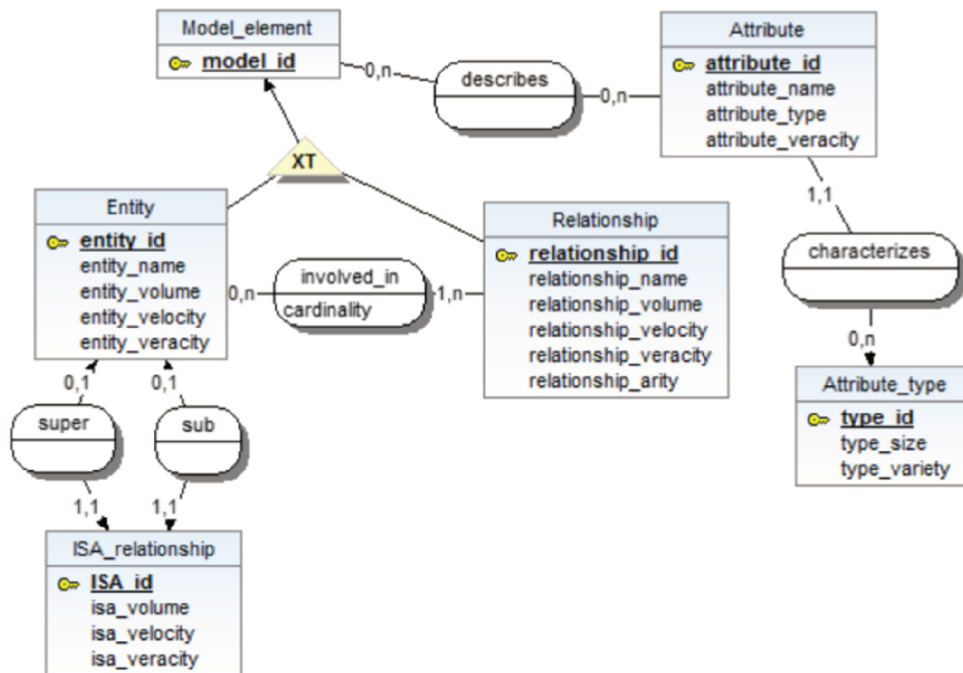
Esse trabalho se diferencia da ferramenta aqui proposta em alguns aspectos:

- Conversão ER para grafos e não UML para grafos;
- Controle de integridade em nível do SGBD;
- Além da solução de conversão para grafos, este trabalho propõe uma ferramenta com interface gráfica para construção do modelo conceitual.

3.2 A FOUR V'S DESIGN APPROACH OF NOSQL GRAPH DATABASES

Este trabalho propõe uma meta-modelagem na qual o projetista do BD descreve um EER incluindo estimativas úteis para análise de *Big Data*: volume, variedade, velocidade e veracidade de cada dado (AKOKA; COMYN-WATTIAU; PRAT, 2017). Volume se refere à quantidade do dado; Variedade se refere aos diferentes tipos que esse dado pode assumir; Velocidade se refere à velocidade na qual o dado é acessível; Veracidade se refere à incerteza em relação ao dado (devido à fatores como inconsistência de informação ou incompletude). A Figura 13 mostra uma meta-modelagem conceitual EER contendo estimativas quanto aos quatro Vs.

Figura 13 – Meta-modelagem conceitual EER contendo estimativas quanto aos quatro V's



Fonte: (AKOKA; COMYN-WATTIAU; PRAT, 2017)

Este trabalho se inspira no trabalho de (DANIEL; SUNYÉ; CABOT, 2016) para representar um LPG através da sua meta-modelagem. Ele argumenta que a modelagem conceitual EER possui o mesmo poder de expressão de uma modelagem LPG e, dessa forma, é possível escrever regras de transformação traduzindo todos os elementos conceituais nos elementos lógicos equivalentes. Entretanto, o modelo EER considerado no trabalho não engloba todas as funcionalidades definidas por (ELMASRI; NAVATHE, 2015). As regras de mapeamento definidas tratam de entidades, relacionamentos entre duas entidades, relacionamentos entre três ou mais entidades, atributos, tipos de atributos e especializações. Ainda, apesar de fornecer algoritmos para os mapeamentos sugeridos, o trabalho não conta com uma implementação demonstrando o seu uso prático.

O trabalho fornece métricas para que o projetista do banco possa:

- Avaliar qual SGBD de grafos é mais apropriado para o seu conjunto de dados (entre Neo4j e OrientDB);
- Estimar o tamanho dos arquivos para decidir se armazenamento em memória é uma opção;
- Averiguar a conectividade do grafo para saber se a base de dados é eficiente para o processamento das informações.

Diferente do trabalho aqui proposto, este trabalho não se preocupa em garantir que o esquema do BD seja respeitado, apenas sugerindo como os dados da modelagem ER

poderiam ter sido representados em um LPG. Ainda, como comentado anteriormente, ele não fornece regras de mapeamento para todas as funcionalidades do modelo EER.

3.3 MODELLING OF GRAPH DATABASES

Este trabalho (POKORNY, 2017) propõe uma variação do modelo ER chamada de ER Binário, na qual apenas relacionamentos binários (envolvendo duas entidades) são permitidos, com cardinalidades (1,1), (1,n) e (m,n). Essa variação contém as seguintes funcionalidades: entidades fortes, entidades fracas, relacionamentos binários, atributos, identificadores, identificadores parciais (utilizados pelas entidades fracas), hierarquias do tipo “é um” e *constraints* inerentes (restrições de formato e comportamento dos dados que são assumidas durante a utilização do BD, mas que não são reforçadas a nível do SGBD).

A partir do modelo ER Binário são propostos algoritmos para realizar o seu mapeamento para LPG. O trabalho fornece também códigos com formas de trazer algumas das *constraints* inerentes para restrições em nível do BD, a depender das funcionalidades do SGBD utilizado. Ele apresenta alguns códigos para *Neo4j* (inserção de *constraints* de formato, também utilizadas no trabalho aqui proposto), *Titan* (inserção de *constraints* para controle de cardinalidade) e *OrientDB* (criação de classes para controlar o comportamento dos nós e relacionamentos).

Diferentemente do trabalho aqui proposto, o trabalho de (POKORNY, 2017) opta por diminuir a expressividade da modelagem para facilitar o seu mapeamento para um LPG. E por não focar em um único SGBD, não fornece uma solução que consiga garantir todas as restrições em nível do BD. Para o *Neo4j* são utilizadas apenas as *constraints* de existência e unicidade de propriedades, sem prover garantias de formato dos relacionamentos e das suas cardinalidades.

3.4 MODEL-DRIVEN DESIGN OF GRAPH DATABASES

Este trabalho propõe um framework que apresenta um método para transformar uma modelagem ER sem especializações em um BD de grafo (DE VIRGILIO; MACCIONI; TORLONE, 2014). A ideia geral é transformar a modelagem ER em uma modelagem de uma notação estendida chamado *Oriented ER (O-ER)*, que é um grafo etiquetado, orientado e com pesos. O trabalho descreve um algoritmo para gerar esse novo modelo.

Uma vez construído o modelo O-ER, o algoritmo utiliza uma série de passos para particioná-lo a fim de juntar dados que têm uma probabilidade alta de aparecerem na mesma consulta. O grafo resultante é então transformado em um template para o BD, que define o formato que o LPG deve ter.

Esse trabalho, diferentemente da aplicação aqui proposta, não faz nenhuma tentativa de garantir a integridade dos dados no BD após a geração do BD seguindo o template gerado. Assume-se que os dados serão inseridos em acordo com a modelagem,

caso contrário o BD não respeitará as regras definidas na modelagem. Também, como dito anteriormente, a notação O-ER não é capaz de lidar com o conceito de especializações de uma modelagem EER.

3.5 PROJETO LÓGICO DE BANCO DE DADOS NOSQL DE GRAFOS A PARTIR DE UM MODELO CONCEITUAL BASEADO NO MODELO ER

Esse trabalho (SOUSA, 2018) propõe um conjunto de extensões para a linguagem Cypher (Figura 14) que permite a definição de comportamentos mais rígidos tanto em nível do SGBD quanto do esquema do BD.

Figura 14 – Comandos Cypher propostos por trabalho relacionado

Restrição de Rótulo de Aresta	Comando ou extensão para o Cypher
exists (e, t)	CREATE CONSTRAINT ON ()-[e]-() ASSERT exists (e,t)
disjoint (e, S)	CREATE CONSTRAINT ON ()-[e]-() ASSERT DISJOINT (e.S ₁ , e.S ₂ , ..., e.S _N)
unique (e, t)	CREATE CONSTRAINT ON ()-[e]-() ASSERT e.t IS UNIQUE
identifier (e, t)	CREATE CONSTRAINT ON ()-[e]-() ASSERT e.t IS EDGE KEY

Fonte: (SOUSA, 2018)

Da tabela anterior, apenas o comando “exists” é nativo do Cypher. Os demais comandos são propostas do trabalho para extensão da linguagem. O trabalho, entretanto, falha em perceber que a existência do plugin *APOC* permite a definição de *triggers* em nível do BD que podem ser utilizados para reforçar a estrutura dos dados armazenados. Dessa forma, o trabalho deixa a sua solução proposta sem implementação, o que o torna pouco útil em cenários do mundo real.

Ainda, ele optou por implementar uma versão limitada do modelo ER a fim de simplificar o mapeamento para BD de grafos, deixando de fora conceitos como: entidades fracas, entidades associativas, relacionamentos n-ários, especializações, atributos compostos e multivalorados. A ferramenta aqui proposta busca implementar todas as funcionalidades esperadas de um esquema EER.

3.6 COMPARAÇÃO DOS TRABALHOS EXISTENTES

A Figura 15 apresenta uma comparação entre os trabalhos relacionados e o trabalho atual no que se refere aos conceitos dos modelos ER e EER mapeados para BDs de grafo e às restrições de integridade implementadas. Note que as demais contribuições realizadas pelos trabalhos relacionados não foram incluídas na lista, como por exemplo, a inclusão de estimativas relacionadas à *Big Data* na modelagem proposta por (AKOKA; COMYN-WATTIAU; PRAT, 2017) e a abordagem para otimização do grafo proposta por (DE VIRGILIO; MACCIONI; TORLONE, 2014).

Figura 15 – Comparação entre os trabalhos relacionados e o trabalho atual

Trabalho	Conceitos do ER	Conceitos do EER	Restrições de integridade	BDs de grafos suportados
(VIRGILIO; MACCIONI; TORLONE, 2014)	Apenas o básico (entidades, relacionamentos e atributos)	X	Não possui	Qualquer um
(AKOKA; COMYN-WATTIAU; PRAT, 2017)	Básico + relacionamentos n-ários	Especializações	Não possui	Qualquer um
(POKORNY, 2017)	Básico + entidades fracas	Especializações	Parcial (algumas restrições são suportadas, a depender dos recursos providos pelo SGBD)	Neo4j, Titan, OrientDB e Stardog
(SOUSA, 2018)	Básico + auto-relacionamentos	X	Parcial, propõe extensões para a linguagem Cypher para reforçar restrições em nível do SGBD	Neo4j
Trabalho atual	Todos	Todos (Especializações e uniões)	Total (em nível do SGBD)	Neo4j

Fonte: Elaboração própria

Conforme ilustra a Figura 15, o trabalho aqui proposto se destaca por suportar todos os conceitos dos modelos ER e EER. Além disso, ele garante que todas as restrições de integridade definidas em nível conceitual sejam respeitadas quando o esquema do BD é gerado no BD Neo4j.

4 FERRAMENTA PROPOSTA: DANGO

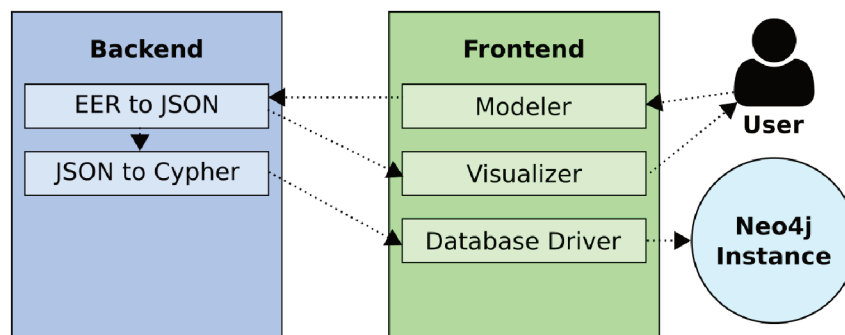
A ferramenta aqui proposta é uma aplicação web para modelagem de BDs de grafo. O seu nome (*Dango*) vem da sobremesa japonesa feita de bolinhas de farinha de arroz ligadas por um espeto de madeira, cuja estrutura lembra a de um grafo. O seu enfoque está em permitir que os usuários modelem um BD através de uma modelagem ER ou EER (normalmente utilizadas para modelar BDs relacionais) e possam gerar uma sequência de comandos Cypher que constrói o esquema desse BD em uma instância Neo4j.

No intuito de facilitar a introdução das modelagens na ferramenta, buscou-se alguma notação existente para representação em texto de modelagens ER e EER. Entretanto, as únicas alternativas encontradas, *ERD* (GALLANT, 2020) e *Erwiz* (SCHMITZ, 2011), não suportam os conceitos do modelo EER. Dessa forma, este trabalho introduziu uma linguagem própria para especificação textual de modelagens EER denominada *EER em texto*. Ela é detalhada na Seção 4.4. A sintaxe dessa linguagem tem como inspiração a da linguagem DBML, utilizada pela ferramenta *dbdiagram.io*¹ para modelagem lógica de BDs relacionais.

4.1 ARQUITETURA

A arquitetura da ferramenta possui 3 componentes. A Figura 16 apresenta essa arquitetura.

Figura 16 – Arquitetura da ferramenta



Fonte: Elaboração própria

Os 3 componentes são os seguintes:

- **Backend:** Esse componente oferece uma Application Programming Interface (API) capaz de:
 - Receber uma modelagem EER especificada textualmente (em uma notação breve e fácil de escrever) e transformá-la em uma representação intermediária

¹ Disponível em: <https://dbdiagram.io/>

no formato JSON (que é fácil de computar, pois interage facilmente com a linguagem *JavaScript*, porém é mais verbosa e difícil de escrever diretamente por incluir vários meta-dados sobre cada ítem da modelagem). (Módulo *EER to JSON*);

- Transformar um JSON representando uma modelagem EER em uma sequência de comandos Cypher (Módulo *JSON to Cypher*).

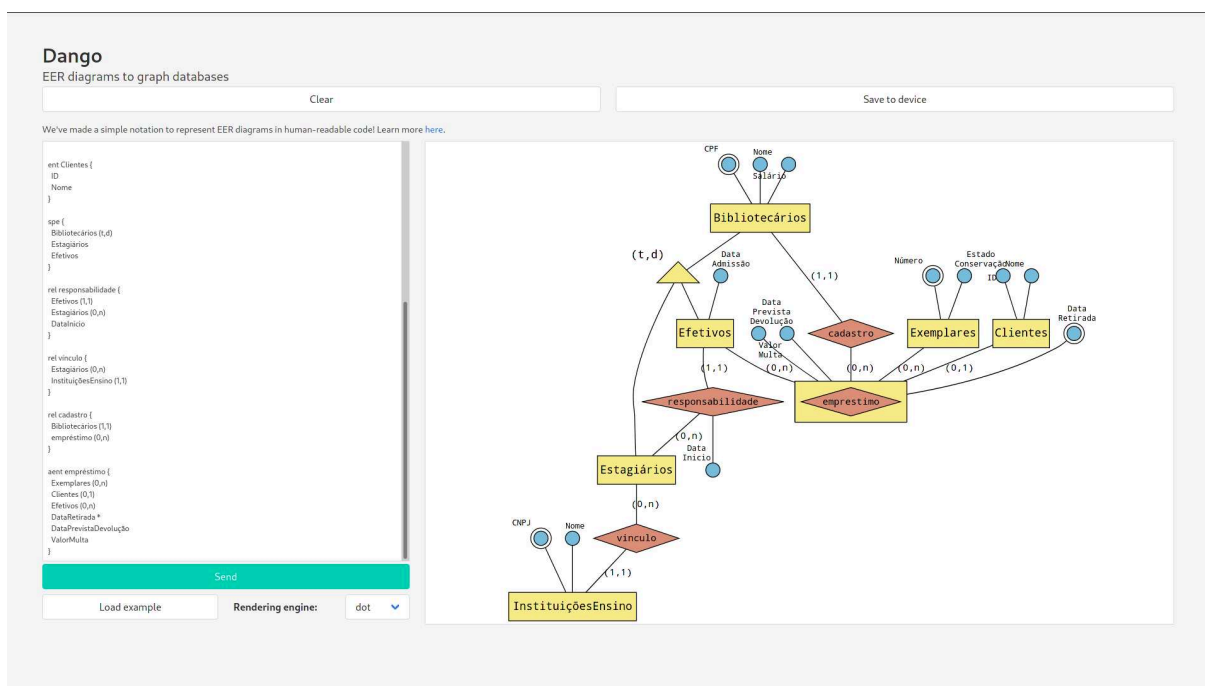
- **Frontend:** Esse componente oferece:

- Um módulo *Modeler* através do qual o usuário pode escrever uma modelagem EER através de texto;
- Um módulo *Visualizer* capaz de interpretar a entrada do *Modeler* e transformá-la em uma visualização gráfica do modelo;
- Um módulo *Database Driver* que permite que os comandos Cypher gerados pela ferramenta sejam executados em uma instância local do Neo4j.

4.2 INTERFACE COM O USUÁRIO E FUNCIONALIDADES DA FERRAMENTA

Esta seção descreve a interface gráfica da aplicação desenvolvida e suas funcionalidades. A Figura 17 mostra a sua tela principal.

Figura 17 – Tela principal da ferramenta



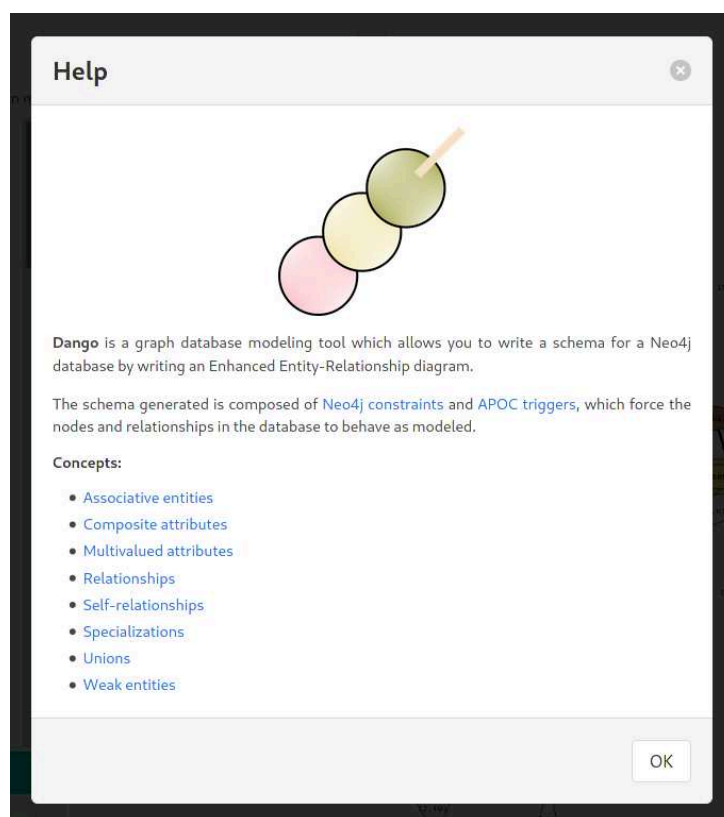
Fonte: Elaboração própria

A tela principal apresenta:

- o componente *Modeler* do lado esquerdo, que auxilia na definição de uma modelagem EER em texto. Ele possui funcionalidades básicas de indentação para tornar o processo mais simples. Por exemplo, ao iniciar a definição de uma entidade e avançar para a próxima linha, o *Modeler* automaticamente coloca dois caracteres de espaço para indentar a definição dos atributos dessa entidade.
- o componente *Visualizer* do lado direito, que gera uma visualização gráfica da modelagem EER em texto escrita pelo usuário;
- os botões “Clear” e “Save to device”, na parte superior, permitem, respectivamente, que o usuário apague a modelagem EER escrita e salve a modelagem em um arquivo de texto (.txt) no computador do usuário;
- o botão “Load example”, na parte inferior, permite que o usuário carregue uma modelagem EER exemplo para se familiarizar com a notação utilizada e com o funcionamento da ferramenta, podendo avançar para as etapas seguintes utilizando essa modelagem;
- o seletor “Rendering engine”, também na parte inferior, permite que o usuário altere o motor de renderização *Graphviz* utilizado para gerar a visualização da modelagem EER escrita.

A qualquer momento o usuário pode clicar no link “Learn more here”, que leva até a tela de ajuda da ferramenta (Figura 18) onde ele/ela pode ler sobre como utilizar a notação EER em texto para realizar a modelagem do seu BD.

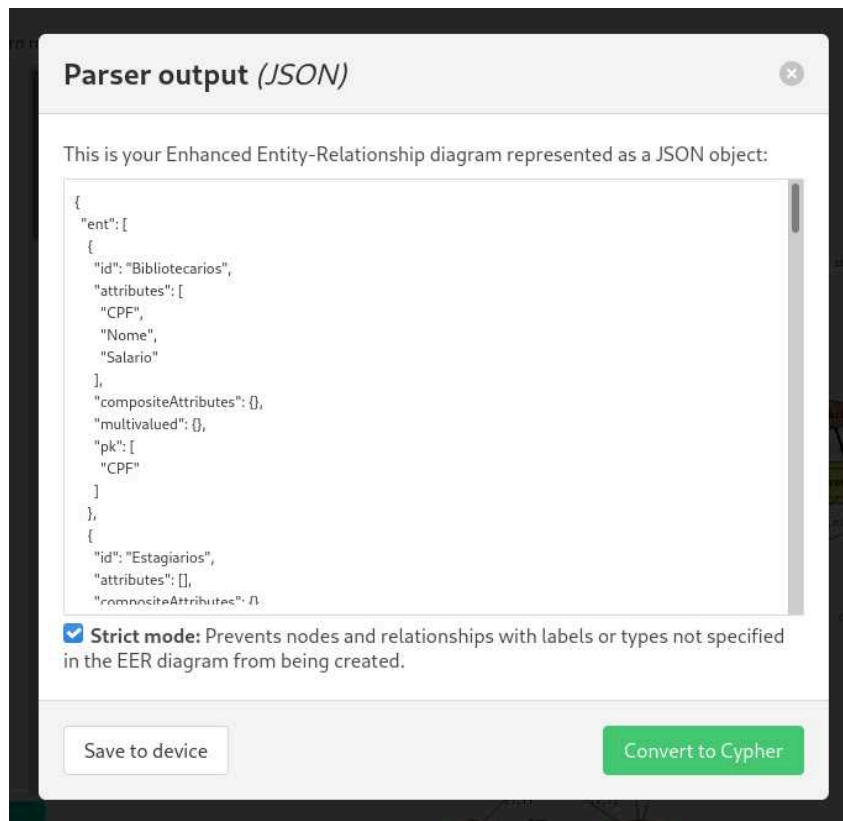
Figura 18 – Tela de ajuda da aplicação



Fonte: Elaboração própria

Quando o usuário entra com uma modelagem EER em texto, a sua visualização é automaticamente gerada. Nesse ponto, ele pode clicar no botão “Send”, que envia a modelagem para a API do *Backend*, e esse retorna uma representação dessa modelagem no formato JSON (Figura 19).

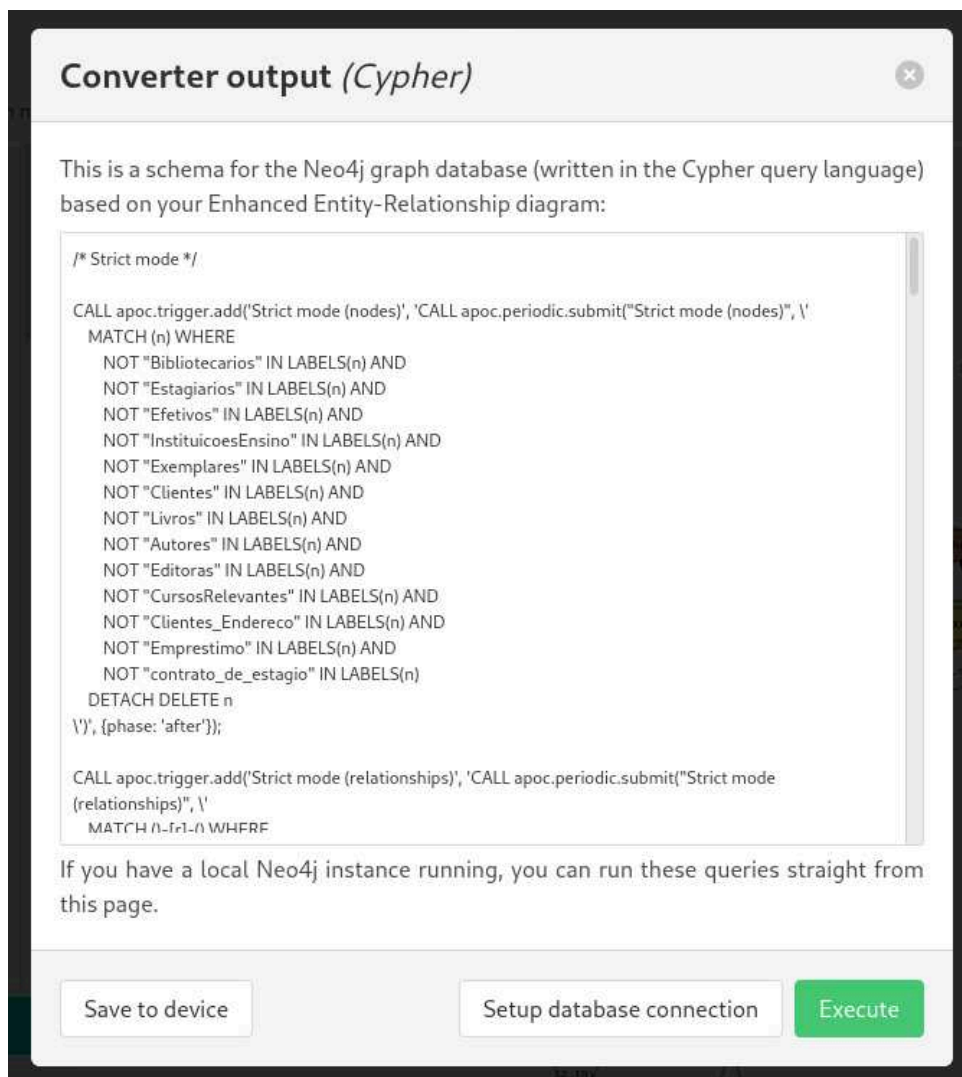
Figura 19 – Representação JSON de uma modelagem EER



Fonte: Elaboração própria

Na tela da Figura 19, a opção *Strict mode* (pré-habilitada) gera uma série de regras para impedir que entidades e relacionamentos não descritos na modelagem possam ser inseridos no BD. Mais detalhes sobre essa opção estão disponíveis na Seção 4.4.11. O botão “Convert to Cypher” faz com que a ferramenta gere a sequência de comandos Cypher que representa, através de *constraints* e *triggers*, o esquema Neo4j definido pela modelagem EER em texto (Figura 20).

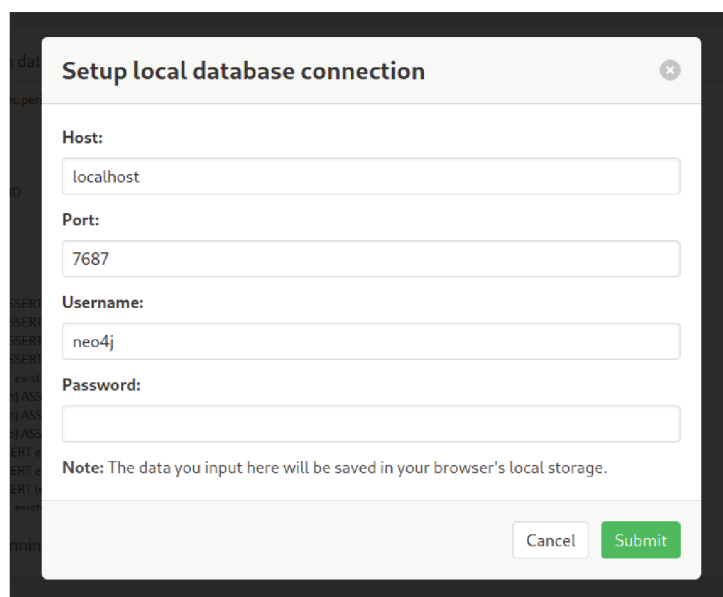
Figura 20 – Sequência de comandos Cypher representando o esquema do BD



Fonte: Elaboração própria

Para que o esquema gerado passe a valer, ele deve ser inserido em uma instância existente do Neo4j. Na tela da Figura 20, o usuário pode clicar em “Setup database connection” para configurar os dados para conexão na sua instância local do Neo4j ou “Execute” para executar os comandos na instância previamente configurada. Caso o usuário opte por configurar a conexão com o BD, ele/ela é levado(a) para a tela mostrada na Figura 21.

Figura 21 – Tela para configuração da conexão com a instância local Neo4j



Setup local database connection

Host:
localhost

Port:
7687

Username:
neo4j

Password:

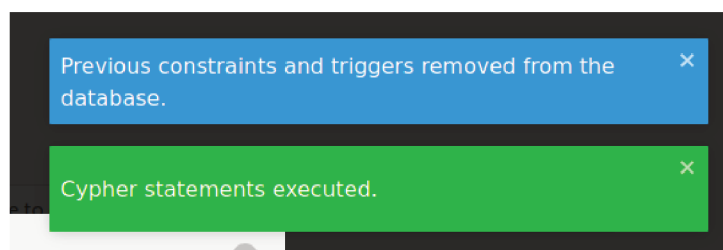
Note: The data you input here will be saved in your browser's local storage.

Cancel Submit

Fonte: Elaboração própria

Ao executar os comandos no BD, notificações aparecem no canto superior direito da tela informando o usuário sobre a execução da operação (Figura 22).

Figura 22 – Notificações de sucesso da execução da operação



Fonte: Elaboração própria

Uma vez finalizada com sucesso a operação, o esquema é inserido na instância Neo4j.

4.3 MAPEAMENTO CONCEITUAL-GRAFO

Enquanto BDs de grafo não precisam seguir um esquema para armazenar os dados, contanto que eles estejam dispostos em um LPG, é imprescindível que haja alguma espécie de esquema para que possamos aproveitar uma modelagem conceitual ER ou EER no projeto lógico e físico do nosso BD de grafo e garantir que os dados gerados estejam de acordo com esta modelagem. A abordagem adotada nesse trabalho é a de utilizar as

funcionalidades do SGBD de grafo Neo4j e do seu plugin APOC para forçar os dados a se comportarem conforme a modelagem realizada em nível do BD, sem a necessidade de qualquer controle externo.

O BD Neo4j oferece duas funcionalidades que permitem indiretamente a definição de um esquema para que o seu grafo tenha a estrutura desejada pelo projetista do BD: as etiquetas (*labels*) e as restrições (*constraints*). As etiquetas permitem a identificação de diferentes tipos de nós no BD. Pode-se interpretar as etiquetas como as entidades da modelagem ER. Entretanto, por si só, as etiquetas não fornecem qualquer garantia de formato do dado. Para isso, se faz necessária a utilização de restrições.

Três tipos de restrições estão presentes no Neo4j: propriedade única, existência de propriedade e chaves do nó. Na versão *Community* (gratuita) do BD apenas a de propriedade única está disponível. Entretanto, para projetos de código fonte aberto a versão *Enterprise* poder ser utilizada gratuitamente. Ambas são de código fonte aberto, porém com licenças de uso diferentes. Dessa forma, esse trabalho faz uso das restrições da versão *Enterprise* no seu algoritmo para mapeamento de modelagem conceitual EER para modelagem física Neo4j.

A fim de ilustrar essas restrições, essa seção traz exemplos referentes ao domínio de uma biblioteca. Livros (nós com a etiqueta “Book”) possuem um código ISBN único que os identifica. Não existe um livro que não contenha um código ISBN. Pessoas (nós com a etiqueta “Person”) possuem primeiro nome (“firstname”) e sobrenome (“surname”), e a combinação dos dois deve ser única.

A restrição de propriedade única (Trecho de Código 3) garante que, entre nós de uma dada etiqueta, uma propriedade não pode ter o mesmo valor em mais um de nó. Isso, porém, não significa que o nó não possa deixar de ter essa propriedade.

Trecho de Código 3 – Restrição para todo nó com etiqueta “Book”, caso tenha um código ISBN, esse código deverá ser único. Fonte: (NEO4J, 2019)

```
1 CREATE CONSTRAINT ON (book:Book) ASSERT book.isbn IS UNIQUE
```

A restrição de existência de propriedade (Trecho de Código 4) garante que todos os nós de uma mesma etiqueta devem ter uma dada propriedade. Comandos que tentarem criar novos nós ou relacionamentos desse tipo sem essa propriedade, ou que tentarem removê-la, falharão.

Trecho de Código 4 – Restrição de que todo nó com etiqueta “Book” deverá ter um código ISBN. Fonte: (NEO4J, 2019)

```
1 CREATE CONSTRAINT ON (book:Book) ASSERT exists(book.isbn)
```

A restrição de chaves do nó (Trecho de Código 5) garante que, para uma dada etiqueta e um dado conjunto de propriedades, todas as propriedades existem em um nó contendo essa etiqueta. A combinação dos valores das propriedades é única entre os nós contendo essa etiqueta.

Trecho de Código 5 – Restrição de que todo nó com etiqueta “Person” deve ter nome e sobrenome e a combinação dos dois deve ser única. Fonte: (NEO4J, 2019)

```
1 CREATE CONSTRAINT ON (n:Person) ASSERT (n.firstname, n.surname) IS NODE  
   KEY
```

A implementação do módulo *JSON to Cypher*, que converte a estrutura intermediária JSON em uma série de instruções Cypher que definem o esquema do DB Neo4j, faz uso das restrições de propriedade única e de existência de propriedade da versão *Enterprise* do Neo4j.

4.3.1 Direção dos Relacionamentos

Nos BDs de grafo os relacionamentos têm direção, mas o mesmo não pode ser dito para os modelos conceituais ER e UML. A solução proposta pelo trabalho UMLtoGraphDB é modelar todos os relacionamentos em ambas as direções no BD, e a solução proposta em (DE VIRGILIO; MACCIONI; TORLONE, 2014) implica na escolha das direções com base na cardinalidade dos relacionamentos que, em alguns casos, implica em relacionamentos bidirecionais. A existência de relacionamentos bidirecionais, porém, representa no Neo4j um excesso de notação, já que esse SGBD permite a travessia em ambos os sentidos independente da direção do relacionamento.

Dessa forma, a abordagem a ser utilizada na ferramenta proposta é a de decidir arbitrariamente de qual das entidades o relacionamento sai e em qual ele entra. Isso se torna uma alternativa interessante quando se percebe que a linguagem de consulta Cypher do Neo4j permite que a direção de um relacionamento seja ignorada completamente durante uma consulta. Se a direção do relacionamento na consulta não for especificada, o resultado obtido será o mesmo da consulta que faz a união das consultas em ambos os sentidos. O Trecho de Código 6 mostra duas consultas equivalentes para encontrar relacionamentos em ambas as direções. Fica evidente, assim, que não há agregação ao valor sintático da modelagem na definição de relacionamentos em ambos os sentidos ao invés de em apenas um.

Trecho de Código 6 – Comparação de consultas Cypher equivalentes para relacionamento unidirecional e bidirecional. Fonte: (BACHMAN, 2013)

```
1 // Unidirecional  
2 MATCH (neo)-[:PARTNER]-(partner)  
3  
4 // Bidirecional  
5 MATCH (neo)-[:PARTNER]->(partner)  
6 UNION ALL  
7 MATCH (neo)<-[:PARTNER]-(partner)
```

4.4 NOTAÇÃO EER EM TEXTO E REGRAS DE MAPEAMENTO

Uma das contribuições desse trabalho é a linguagem “EER em texto”, que permite a representação textual de modelagens EER em uma notação simples, com uma estrutura feita para se assemelhar ao JSON, formato amplamente utilizado em aplicações web na atualidade (LV; YAN; HE, 2018). Como explicado no início da Seção 4, essa linguagem foi criada na ausência de uma notação existente capaz de representar todos os conceitos do modelo EER, como definido por (ELMASRI; NAVATHE, 2015). Os blocos de construção básicos da linguagem são as declarações: “ent” (entidade), “rel” (relacionamento), “aent” (entidade associativa), “spe” (especialização) e “union” (união).

Nas subseções seguintes, após cada exemplo de conceito do modelo ER ou EER, representado na notação EER em texto, são descritas também as regras de mapeamento para representar uma estrutura equivalente no Neo4j. Essas regras estão implementadas na forma de *constraints* e *triggers* geradas pela aplicação. As *constraints* (cujas declarações começam com “CREATE CONSTRAINT”) são inseridas diretamente no BD, já os *triggers* (cujas declarações começam com “MATCH”) são encapsulados em um bloco de código que realiza a sua inserção no BD. Ao longo das subseções esse comando é abstraído para mostrar apenas as ações executadas pelo *trigger* a fim de facilitar a leitura do trabalho. Dessa forma, assume-se que qualquer *trigger* mostrada a partir de agora estará na prática encapsulada dentro de um comando no formato mostrado no trecho de código 7.

Trecho de Código 7 – Comando para inserção de um trigger no Neo4j gerado pela aplicação

```
1 CALL apoc.trigger.add('NOME DO TRIGGER', 'CALL apoc.periodic.submit("
   NOME DO TRIGGER", \
2     COMANDO DO TRIGGER COMECANDO EM MATCH AQUI
3 \')', {phase: 'after'});
```

4.4.1 Entidade

A declaração de uma entidade deve conter o seu nome e seus atributos. Os atributos cujos valores devem ser únicos devem ser seguidos de um “*” (asterisco). A Figura 23 mostra um exemplo de declaração de entidade ao lado da sua representação gráfica.

Figura 23 – Declaração de uma entidade



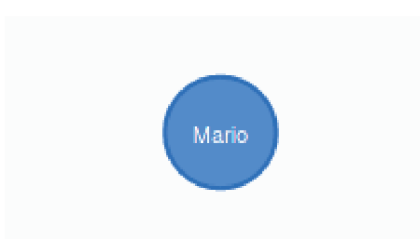
Fonte: Elaboração própria

O trabalho de (AKOKA; COMYN-WATTIAU; PRAT, 2017) sugere uma correspondência entre conceitos do modelo ER e de grafos: uma entidade seria um nó, um atributo seria uma propriedade de um nó, e um relacionamento seria uma aresta. Similarmente, o trabalho aqui proposto opta por representar em um LPG uma instância de uma entidade através de um nó marcado com a etiqueta referente a essa entidade e contendo propriedades equivalentes aos seus atributos. Dessa forma, uma instância da entidade mostrada na Figura 23 poderia ser representada no Neo4j através do comando Cypher mostrado na Figura 24.

Note que a Figura 24 inclui uma visualização do dado inserido no BD, gerada através da ferramenta *Neo4j Browser*. A ferramenta permite que o usuário personalize a visualização do conjunto de dados retornado por uma consulta, podendo escolher a cor que representa nós de cada etiqueta e também qual atributo representa o nó na visualização. O atributo escolhido para representar os nós com a etiqueta “Librarians” nesse caso foi “name”. Por isso, o nó mostrado contém o texto “Mario”. Essas escolhas de representação não afetam os dados no BD, servindo apenas para facilitar a visualização do resultado da consulta.

Figura 24 – Inserção de um nó representando uma instância da entidade “Librarians”

```
1 CREATE (1:Librarians {id: 1, name: "Mario", salary: 28000})
```



Fonte: Elaboração própria

Cabe observar que, para que haja a garantia de que qualquer nó marcado com a

etiqueta “Librarians” tenha as propriedades e os comportamentos esperados, a inserção de algumas *constraints* no BD se faz necessária, como mostrado no Trecho de Código 8.

Trecho de Código 8 – Regra de mapeamento para entidade exemplo

```

1 CREATE CONSTRAINT ON (l:Librarians) ASSERT exists(l.id);
2 CREATE CONSTRAINT ON (l:Librarians) ASSERT exists(l.name);
3 CREATE CONSTRAINT ON (l:Librarians) ASSERT exists(l.salary);
4 CREATE CONSTRAINT ON (l:Librarians) ASSERT (l.id) IS UNIQUE;

```

As declarações do Trecho de Código 8 garantem que qualquer nó que seja inserido com a etiqueta “Librarians” tenha obrigatoriamente as propriedades “id”, “name” e “salary” definidas. Ainda, a propriedade “id” deve ser única entre todos os nós que compartilharem dessa etiqueta. Caso alguma dessas características não se cumpra o Neo4j impede a inserção do dado.

4.4.2 Relacionamento

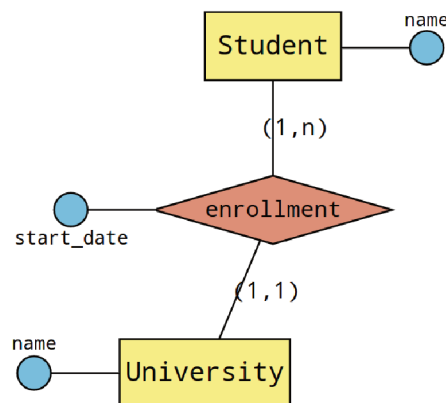
A declaração de um relacionamento deve conter o seu nome, o nome das entidades relacionados e suas respectivas cardinalidades, bem como os atributos do relacionamento. A Figura 25 contém um exemplo de declaração de relacionamento ao lado da sua representação gráfica.

Figura 25 – Declaração de um relacionamento

```

1 ent Student {
2   name
3 }
4
5 ent University {
6   name
7 }
8
9 rel enrollment {
10  Student (1,n)
11  University (1,1)
12  start_date
13 }

```



Fonte: Elaboração própria

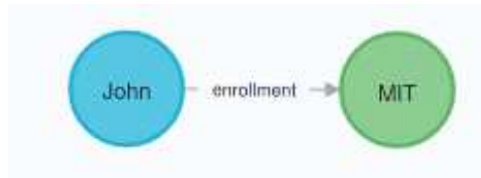
O trabalho aqui proposto, mantendo a abordagem proposta por (AKOKA; COMYN-WATTIAU; PRAT, 2017), opta por representar um relacionamento em um LPG como uma aresta que conecta as instâncias dos nós entre si. Uma instância do relacionamento mostrada na Figura 25 pode ser representada no Neo4j através do comando Cypher mostrado na Figura 26.

Figura 26 – Inserção do relacionamento representando uma instância do relacionamento “enrollment”

```

1 CREATE (s:Student {name: "John"})
2 CREATE (u:University {name: "MIT"})
3 CREATE (s)-[:enrollment {start_date: date() } ]->(u)

```



Fonte: Elaboração própria

Apesar de um relacionamento ser muito simples de representar em um LPG, as regras de comportamento que o acompanham em uma modelagem ER não são tão facilmente representadas. Para tal, se faz necessário o uso de tanto *constraints* quanto *triggers*. Dada a modelagem da Figura 25, para representar a existência de um atributo “start_date” em todo relacionamento do tipo “enrollment”, a *constraint* simples do Trecho de Código 9 é suficiente.

Trecho de Código 9 – Constraint garantindo a existência de uma propriedade em um relacionamento

```

1 CREATE CONSTRAINT ON ()-[e:enrollment]-() ASSERT exists(e.start_date);

```

Considerando ainda a modelagem EER da Figura 25, para que haja a garantia de que um relacionamento do tipo “enrollment” aconteça apenas entre nós com etiquetas “Student” e “University”, são necessários os três *triggers* definidos no Trecho de Código 10. Note que, conforme dito na seção 4.4, cada declaração iniciada em “MATCH” será encapsulada em um bloco de código (aqui abstraído) que realiza a sua inserção como *trigger* no BD.

Trecho de Código 10 – Triggers para garantir o formato de um relacionamento

```

1 MATCH (n)-[r:enrollment]-(:University) WHERE NOT "Student" IN LABELS(n)
  DELETE r;
2
3 MATCH (n)-[r:enrollment]-(:Student) WHERE NOT "University" IN LABELS(n)
  DELETE r;
4
5 MATCH (n)-[r:enrollment]-() WHERE NOT "Student" IN LABELS(n) AND NOT "
  University" IN LABELS(n) DELETE r;

```

O primeiro *trigger* remove qualquer relacionamento do tipo “enrollment” que esteja conectando um nó “University” a um nó que não tenha a etiqueta “Student”. O segundo *trigger* realiza a verificação equivalente na outra direção.

Nenhum dos dois *triggers* anteriores, entretanto, impede que nós com outras etiquetas se relacionem através de um relacionamento com a etiqueta “enrollment”. Dessa forma, o terceiro *trigger* remove qualquer relacionamento com a etiqueta “enrollment” que esteja conectando nós que não sejam nem “Student” e nem “University”.

Deve-se ainda controlar as cardinalidades do relacionamento. Na modelagem EER da Figura 25, um “Student” pode estar conectado a 1 e apenas 1 “University”, já uma “University” pode estar conectada a 1 ou n “Student”. O controle de cardinalidade mínima 1 já é garantido pelos *triggers* de formato do Trecho de Código 10, entretanto para garantir a cardinalidade máxima 1 o *trigger* do Trecho de Código 11 se faz necessário.

Trecho de Código 11 – Triggers reforçando o comportamento das cardinalidades

```

1 MATCH (n:Student)-[r:enrollment]-(:University)
2   WITH n, COLLECT(r) AS rs
3   WHERE SIZE(rs) > 1
4   FOREACH (r IN rs[1..] | DELETE r)

```

Nesse *trigger*, garante-se a cardinalidade máxima 1, de forma que um nó do tipo “Student” só poderá ter um “enrollment” com uma “University”. Caso essa condição não seja respeitada, os relacionamentos extras são removidos até que o esquema volte a estar consistente com a modelagem EER.

Note que para representar as cardinalidades mínimas 0 e 1 a cardinalidade máxima n não se faz necessária a inserção de novos *triggers* no BD. Para relacionamentos vários-para-vários, com cardinalidades mínimas e máximas com números arbitrários (n,m) , são sempre gerados *triggers* que contam o número de relacionamentos. Por exemplo, se as cardinalidades de “Student” fossem $(2,4)$ os *triggers* gerados teriam sido os apresentados no Trecho de Código 12.

Trecho de Código 12 – Triggers reforçando as cardinalidades $(2,4)$

```

1 MATCH (n:University)-[r:enrollment]-(:Student)
2   WITH n, COLLECT(r) AS rs
3   WHERE SIZE(rs) < 2
4   FOREACH (r IN rs | DELETE r)
5
6 MATCH (n:University)-[r:enrollment]-(:Student)
7   WITH n, COLLECT(r) AS rs
8   WHERE SIZE(rs) > 4
9   FOREACH (r IN rs[4..] | DELETE r)

```

O primeiro *trigger* remove todos os relacionamentos tipo “enrollment” para uma dada “University” caso a cardinalidade mínima não seja atendida. O segundo *trigger* remove os relacionamentos excedentes, caso o número passe de 4.

Deve-se ainda ressaltar que os *triggers* para controle de cardinalidades mostrados no Trecho de Código 12 não possuem informações suficientes para julgar quais relacionamentos deveriam ser mantidos por serem os mais antigos. Apesar do Neo4j inserir um id serial

único em cada nó e relacionamento, ele também reaproveita ids de nós removidos em novas inserções, o que impossibilita a utilização desse campo de forma confiável para julgar a ordem de inserção dos relacionamentos. Dessa forma, não há garantia de que os relacionamentos mantidos foram os primeiros a serem inseridos. Para contornar esse problema, esse trabalho propõe um atributo especial chamado *timestamp*.

O atributo *timestamp* é denotado por um círculo verde na representação gráfica da modelagem EER feita pela aplicação. Quando inserido pelo projetista do BD em um relacionamento (durante a etapa de modelagem), os *triggers* de controle de cardinalidade gerados assumem que esse atributo irá conter o resultado da função *timestamp()* do Neo4j (que retorna o momento atual do tempo) no momento da inserção do dado. Dessa forma, quando houver a necessidade de remover um relacionamento para respeitar as cardinalidades definidas na modelagem, essa informação pode ser utilizada para decidir qual relacionamento deve ser mantido no BD. A Figura 27 contém um exemplo da declaração de uma modelagem EER utilizando esse atributo.

Figura 27 – Declaração de um relacionamento contendo timestamp



Fonte: Elaboração própria

Com base na *timestamp*, é possível ordenar os relacionamentos encontrados por data crescente, garantindo que em um momento de inconsistência entre o Neo4j e a modelagem EER, os relacionamentos inseridos mais cedo serão os mantidos pelos *triggers*. O Trecho de Código 13 mostra um *trigger* de controle de cardinalidades utilizando o atributo *timestamp*.

Trecho de Código 13 – Triggers de cardinalidades utilizando timestamp

```

1 MATCH (n:Student)-[r:enrollment]-(:University)
2   WITH n, r
3   ORDER BY r.timestamp ASC
4   WITH n, COLLECT(r) AS rs
5   WHERE SIZE(rs) > 1
6   FOREACH (r IN rs[1..] | DELETE r)

```

Para melhor ilustrar essa situação, é possível utilizar a sequência de comandos Cypher presente no Trecho de Código 14. Note que cada bloco de código está sendo executado por uma transação separada.

Trecho de Código 14 – Inserção de dados controlados por timestamp

```
1 // Transaction 1
2 CREATE (s:Student)
3 CREATE (u1:University {name: "UFSC"})
4 CREATE (s)-[:enrollment {timestamp: timestamp()}]->(u1)
5
6 // Transaction 2
7 MATCH (s:Student)
8 CREATE (u2:University {name: "MIT"})
9 CREATE (s)-[:enrollment {timestamp: timestamp()}]->(u2)
10
11 // Transaction 3
12 MATCH (s:Student)
13 CREATE (u3:University {name: "Cambridge"})
14 CREATE (s)-[:enrollment {timestamp: timestamp()}]->(u3)
```

Dado esse trecho de código, é garantido que o relacionamento com a etiqueta “enrollment” a ser mantido é o que liga “Student” à “University” chamada “UFSC”, pois esse é o que contém a *timestamp* de valor mais baixo e, portanto, foi o primeiro a ser inserido. Os demais “enrollment” são removidos, tornando o estado do BD consistente com a modelagem EER.

4.4.3 Relacionamento entre três ou mais entidades

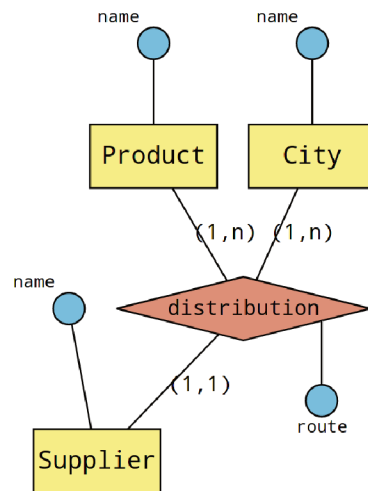
A declaração de um relacionamento entre três ou mais entidades é bastante similar a de um relacionamento simples, bastando adicionar mais entidades na declaração do relacionamento. A Figura 28 contém um exemplo de declaração de um relacionamento entre três entidades ao lado da sua representação gráfica.

Figura 28 – Declaração de um relacionamento n-ário onde $n = 3$

```

1 ent City {
2   name
3 }
4 ent Supplier {
5   name
6 }
7 ent Product {
8   name
9 }
10
11 rel distribution {
12   City (1,n)
13   Supplier (1,1)
14   Product (1,n)
15   route
16 }

```



Fonte: Elaboração própria

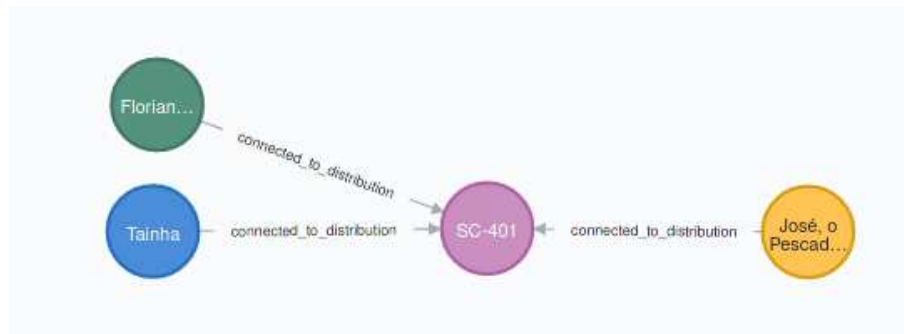
O trabalho aqui proposto opta por utilizar a abordagem de (AKOKA; COMYN-WATTIAU; PRAT, 2017) para representar em um LPG um relacionamento entre três ou mais entidades (relacionamento n-ário com $n \geq 3$). A abordagem consiste em transformar o relacionamento em um nó próprio e fazer com que as entidades envolvidas se relacionem com esse nó. O trabalho aqui proposto adota o padrão de nomear as etiquetas desses relacionamentos como “connected_to_relacionamento”. A Figura 29 mostra um comando para inserir no Neo4j um relacionamento entre três nós seguindo essa abordagem.

Figura 29 – Inserção de um relacionamento n-ário com $n = 3$

```

1 CREATE (c:City {name: "Florianopolis"})
2 CREATE (p:Product {name: "Tainha"})
3 CREATE (s:Supplier {name: "Jose, o Pescador"})
4 CREATE (d:distribution {route: "SC-401"})
5 CREATE (c)-[:connected_to_distribution]->(d)
6 CREATE (p)-[:connected_to_distribution]->(d)
7 CREATE (s)-[:connected_to_distribution]->(d)

```



Fonte: Elaboração própria

Na Figura 29, o relacionamento “distribution” foi modelado como um nó e as entidades que participam desse relacionamento se conectam a ele através de relacionamentos com a etiqueta “connected_to_distribution”. Para realizar os controles de formato e cardinalidade do relacionamento com três ou mais entidades, o algoritmo de mapeamento separa esse relacionamento em vários relacionamentos simples associando a entidade relacionada com o novo nó que representa o relacionamento. Dessa forma, as mesmas regras são geradas para criação dos *triggers*. Note, porém, que como os relacionamentos do tipo “connected_to_relacionamento” podem envolver nós de mais de duas etiquetas (na modelagem da Figura 28: “City”, “Product”, “Supplier” e “distribution”), é gerado um *trigger* que aceita qualquer uma das combinações de relacionamento, como pode ser visto no Trecho de Código 15.

Trecho de Código 15 – Triggers para controle de formato do relacionamento entre três entidades

```

1 MATCH (n)-[r:connected_to_distribution]-(:distribution) WHERE
2     NOT n:'City' AND
3     NOT n:'Supplier' AND
4     NOT n:'Product'
5 DETACH DELETE r;

```

Dessa forma, apenas nós com as etiquetas participantes do relacionamento podem se conectar a ele. Note que, como os relacionamentos n-ários são modelados como nós, os *triggers* para controle de cardinalidade mínima, quando necessário, realizam a remoção

desse nó (diferentemente do caso do relacionamento simples, onde o relacionamento é removido). O Trecho de Código 16 mostra todos os *triggers* de cardinalidade gerados para o relacionamento n-ário referente à modelagem da Figura 29.

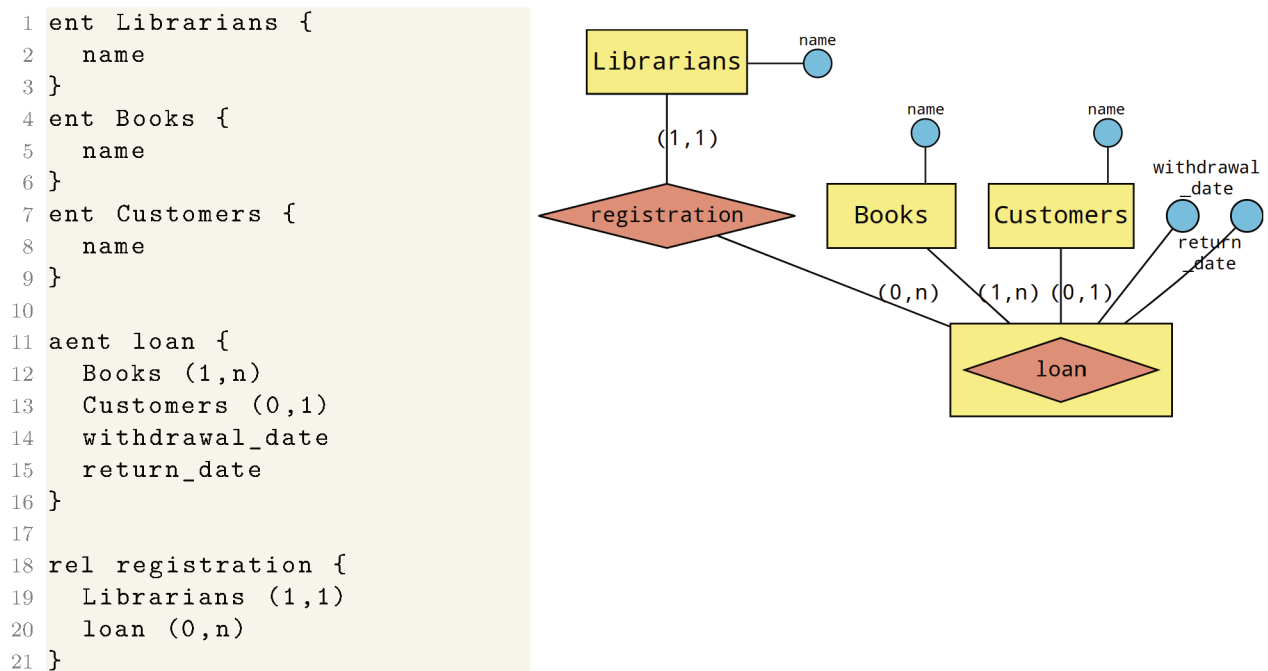
Trecho de Código 16 – Triggers para controle de formato do relacionamento entre três entidades

```
1 MATCH (n:distribution) WHERE NOT (:City)-[:connected_to_distribution]-(n
  ) DETACH DELETE n;
2
3 MATCH (n:distribution) WHERE NOT (:Supplier)-[:connected_to_distribution
  ]-(n) DETACH DELETE n;
4
5 MATCH (n:distribution)-[r:connected_to_distribution]-(:Supplier)
6   WITH n, COLLECT(r) AS rs
7   WHERE SIZE(rs) > 1
8   FOREACH (r IN rs[1..] | DELETE r);
9
10 MATCH (n:distribution) WHERE NOT (:Product)-[:connected_to_distribution
  ]-(n) DETACH DELETE n;
```

4.4.4 Entidade Associativa

A declaração de uma entidade associativa é muito similar a de um relacionamento: ela deve conter o seu nome, o nome das entidades que a constituem e suas respectivas cardinalidades e seus atributos. Entretanto, a diferença é que, caso alguma das entidades se conecte à entidade associativa através de um relacionamento, deve ser feita uma declaração própria para o relacionamento e o nome da entidade deve ser omitido da lista de entidades da entidade associativa. A Figura 30 contém um exemplo de declaração de uma entidade associativa ao lado da sua representação gráfica.

Figura 30 – Declaração de uma entidade associativa



Fonte: Elaboração própria

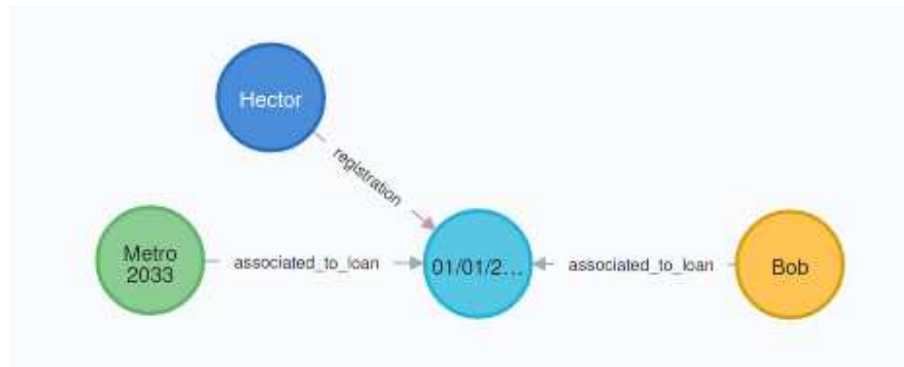
O trabalho aqui proposto opta por representar, em um LPG, uma entidade associativa como um nó que contém uma etiqueta própria. As entidades participantes da associação interna à entidade associativa se conectam a ela através de relacionamentos do tipo “associated_to_entidade” (para a modelagem da Figura 30: “associated_to_loan”). Já as demais entidades que se relacionam com a entidade associativa se conectam a ela como se ela fosse uma entidade comum. A Figura 31 apresenta um exemplo de inserção de dados no Neo4j respeitando a modelagem da entidade associativa apresentada na Figura 30.

Figura 31 – Inserção de uma instância de uma entidade associativa

```

1 CREATE (l:Librarians {name: "Hector"})
2 CREATE (b:Books {name: "Metro 2033"})
3 CREATE (c:Customers {name: "Bob"})
4 CREATE (lo:loan {withdrawal_date: "01/01/2021", return_date:
    "01/01/2022"})
5 CREATE (l)-[:registration]->(lo)
6 CREATE (b)-[:associated_to_loan]->(lo)
7 CREATE (c)-[:associated_to_loan]->(lo)

```



Fonte: Elaboração própria

As entidades externas que se relacionam com a entidade associativa mantêm as mesmas regras apresentadas na Seção 4.4.2 (Relacionamento) para geração dos *triggers*. Já as entidades que integram a entidade associativa (na modelagem da Figura 30: “Books” e “Customers”), se relacionam com ela através de relacionamentos do tipo “associated_to_entidade”.

O algoritmo de mapeamento separa a entidade associativa em relacionamentos simples (um a um) entre a entidade associativa e as entidades relacionadas e aplica as mesmas regras para controle de formato e cardinalidade de relacionamentos explicadas na Seção 4.4.2 (Relacionamento). Vale observar que, como relacionamentos do tipo “associated_to_entidade” podem envolver nós de mais de duas etiquetas (na modelagem da Figura 30: “loan”, “Books” e “Customers”), é gerado um *trigger* que aceita qualquer uma das combinações de relacionamento, como pode ser visto no Trecho de Código 17.

Trecho de Código 17 – Triggers para controle de formato da entidade associativa

```

1 MATCH (n)-[r:associated_to_loan]-(:Books) WHERE NOT "loan" IN LABELS(n)
  DELETE r;
2
3 MATCH (n)-[r:associated_to_loan]-(:Customers) WHERE NOT "loan" IN LABELS
  (n) DELETE r;
4
5 MATCH (n)-[r:associated_to_loan]-(:loan) WHERE
6   NOT n: 'Books' AND

```



```

7 NOT n: 'Customers '
8 DETACH DELETE r;

```

Os dois primeiros *triggers* são gerados utilizando as regras pré-existentes para mapeamento de relacionados. Já o terceiro, similar ao gerado para os relacionamentos n-ários, é gerado especialmente para o caso das entidades associativas, definindo que apenas nós participantes da entidade associativa se relacionem com ela utilizando relacionamentos do tipo “associated_to_entidade”. Para o controle das cardinalidades das entidades associativas, são utilizados *triggers* equivalentes aos dos relacionamentos n-ários (conforme descrito no final da seção 4.4.3).

4.4.5 Atributos Compostos

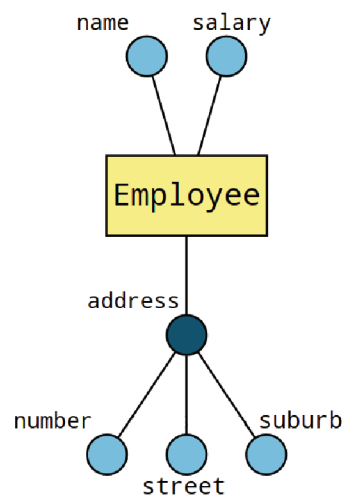
Um atributo composto (atributo que agrega outros atributos) pode ser definido através da utilização de colchetes na linguagem proposta neste trabalho. A Figura 32 contém um exemplo de declaração de entidade contendo um atributo composto ao lado da sua representação gráfica.

Figura 32 – Declaração de um atributo composto

```

1 ent Employee {
2   name
3   salary
4   address [
5     number
6     street
7     suburb
8   ]
9 }

```



Fonte: Elaboração própria

A forma como o trabalho aqui proposto representa em um LPG o conceito de um atributo composto é através da criação de uma etiqueta (uma instância de um nó) própria para o atributo. Dessa forma, é assumido que ao realizar a inserção de um nó representando uma entidade que contenha um atributo composto, o usuário do BD também irá inserir um nó representando o próprio atributo e um relacionamento associando os dois. O comando contido na Figura 33 insere no Neo4j uma entidade e seu atributo composto conforme as especificações definidas.

Figura 33 – Inserção de uma instância de uma entidade e seu atributo composto

```

1 CREATE (e:Employee {name: "Luiza", salary: 50000})
2 CREATE (ea:Employee_address {number: 8778, street: "Hamilton St.",
   suburb: "Arlington"})
3 CREATE (e)-[:has_employee_address]->(ea)

```



Fonte: Elaboração própria

Note que, para que o *trigger* funcione corretamente, a etiqueta para o nó representando o atributo composto deve ter o nome no padrão “*entidade_atributo*” (como em “Employee_address”) e a etiqueta para o relacionamento associando os dois deve ter o nome no padrão “*has_entidade_atributo*” (como em “has_employee_address”). Essa abordagem, entretanto, traz a limitação de que um relacionamento simples (entre duas entidades) não poderá conter um atributo composto. Como o atributo composto é modelado como um nó, isso implicaria que o relacionamento fosse um relacionamento envolvendo três entidades (detalhados na Seção 4.4.3), introduzindo uma maior complexidade no LPG resultante. Dessa forma, esse trabalho optou por não permitir esse cenário de modelagem. Caso o modelador do BD queira ter um atributo composto em um relacionamento, ele/ela deve explicitamente modelá-lo como um relacionamento entre três entidades.

O Trecho de Código 18 mostra todos os *triggers* gerados para garantir que todo nó “Employee” tenha um relacionamento “has_employee_address” com exatamente um “Employee_address” e vice-versa.

Trecho de Código 18 – Triggers para controle de cardinalidade de um atributo composto

```

1 MATCH (n:Employee_address) WHERE NOT (:Employee)-[:has_employee_address
   ]-(n) DETACH DELETE n;
2
3 MATCH (n:Employee_address)-[r:has_employee_address]-(:Employee)
4   WITH n, COLLECT(r) AS rs
5   WHERE SIZE(rs) > 1
6   FOREACH (r IN rs[1..] | DELETE r);
7
8 MATCH (n:Employee) WHERE NOT (:Employee_address)-[:has_employee_address
   ]-(n) DETACH DELETE n;
9
10 MATCH (n:Employee)-[r:has_employee_address]-(:Employee_address)
11   WITH n, COLLECT(r) AS rs
12   WHERE SIZE(rs) > 1

```

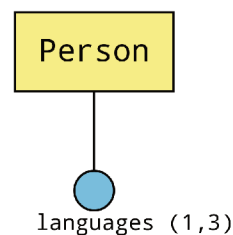
```
13 FOREACH (r IN rs[1..] | DELETE r);
```

4.4.6 Atributos Multivalorados

Um atributo multivalorado, ou seja, um atributo que contém múltiplos valores, pode ser representado através dos símbolos de menor (“<”) e maior (“>”) na linguagem proposta para representar a sua cardinalidade. A Figura 34 contém um exemplo de declaração de entidade com atributo multivalorado ao lado da sua representação gráfica.

Figura 34 – Declaração de uma entidade com atributo multivalorado

```
1 ent Person {
2   name
3   languages <1,3>
4 }
```



Fonte: Elaboração própria

O trabalho aqui proposto representa em um LPG um atributo multivalorado como um atributo contendo um *array*. Dessa forma, uma instância de uma entidade contendo um atributo multivalorado pode ser representada no Neo4j como na Figura 35.

Figura 35 – Inserção de uma instância de uma entidade com atributo multivalorado

```
1 CREATE (:Person {name: "Marco", languages: ["Portuguese", "English"]})
```



Fonte: Elaboração própria

Entretanto, para que haja a garantia de que as cardinalidades para o número de valores nesse atributo sejam respeitadas, a inclusão do *trigger* mostrado no Trecho de Código 19 se faz necessária.

Trecho de Código 19 – Trigger reforçando a quantidade de valores em um atributo multivalorado

```
1 MATCH (n:Person) WHERE
```

```

2   size(n.languages) < 1 OR size(n.languages) > 3
3 DETACH DELETE n

```

Caso o limite inferior da cardinalidade fosse 0 , o *trigger* para controle de cardinalidade não verificaria o limite inferior. Caso o limite superior da cardinalidade fosse n , o *trigger* para controle de cardinalidade não verificaria o limite superior. Dessa forma, um atributo multivalorado de cardinalidades $(0,n)$ não implica em um *trigger* de controle de cardinalidades.

4.4.7 Entidades Fracas

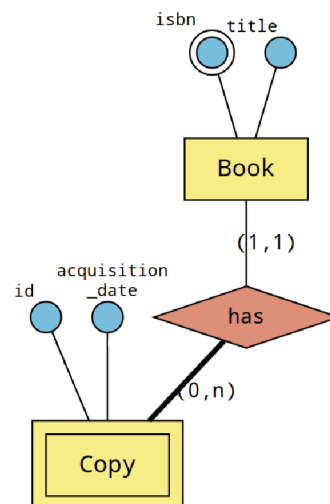
Uma entidade fraca, ou seja, uma entidade cuja existência depende da existência de outra entidade (uma entidade forte), pode ser descrita na linguagem proposta com o prefixo “w” antes do seu nome na definição de um relacionamento. A Figura 36 contém um exemplo de declaração de uma modelagem EER contendo um entidade fraca ao lado da sua representação gráfica.

Figura 36 – Declaração de uma modelagem EER contendo uma entidade fraca

```

1 ent Book {
2   isbn *
3   title
4 }
5
6 ent Copy {
7   id
8   acquisition_date
9 }
10
11 rel has {
12   Book (1,1)
13   w Copy (0,n)
14 }

```



Fonte: Elaboração própria

A forma como o trabalho aqui proposto representa as entidades fracas em um LPG é exatamente igual a forma como entidades comuns foram representadas: através de nós. A Figura 37 mostra a inserção de dados no Neo4j que respeitam a modelagem proposta na Figura 36.

Figura 37 – Inserção de uma entidade fraca

```
1 CREATE (b:Book {isbn: "978-1491508183", title: "Metro 2033"})
2 CREATE (c:Copy {id: 1, acquisition_date: "10/02/2019"})
3 CREATE (b)-[:has]->(c)
```



Fonte: Elaboração própria

O detalhe importante está em como é garantido o comportamento de que a entidade fraca exista apenas enquanto a entidade forte existir. Na modelagem da Figura 36 tem-se a entidade forte sendo “Book” e a entidade fraca sendo “Copy”. Dessa maneira, para reforçar o comportamento da entidade fraca é gerado o *trigger* contido no Trecho de Código 20.

Trecho de Código 20 – Triggers reforçando comportamento da entidade fraca

```
1 MATCH (n:Copy)
2 WHERE NOT (:Book)-[:has]-(n)
3 DETACH DELETE n
```

Esse *trigger* remove qualquer nó com a etiqueta “Copy” que não esteja conectado a um nó com etiqueta “Book” através de um relacionamento com etiqueta “has”. Dessa forma, caso o “Book” associado a uma “Copy” seja removido, a “Copy” também é removida.

4.4.8 Especializações e Generalizações

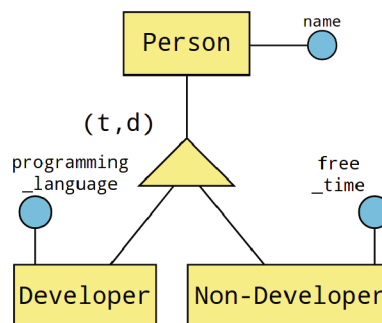
A declaração de uma especialização deve incluir a entidade mãe seguida das características da especialização (total ou parcial, compartilhada ou disjunta). Nas linhas seguintes as entidades filhas devem ser listadas. A Figura 38 mostra um exemplo de declaração de uma modelagem EER contendo uma especialização ao lado da sua representação gráfica.

Figura 38 – Declaração de uma especialização

```

1 ent Person {
2   name
3 }
4
5 ent Developer {
6   programming_language
7 }
8
9 ent Non-Developer {
10  free_time
11 }
12
13 spe {
14   Person (t,d)
15   Developer
16   Non-Developer
17 }

```



Fonte: Elaboração própria

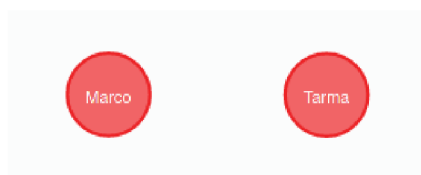
O trabalho aqui proposto representa em um LPG uma instância de uma especialização como um nó que possui tanto a etiqueta da sua entidade mãe quanto a sua própria, tendo assim que satisfazer as restrições das duas. Dessa forma, é possível representar no Neo4j uma instância de cada entidade especializada conforme a Figura 39. Note que a etiqueta “Non-Developer”, por conter caracteres especiais está envolta em caracteres de escape do tipo crase.

Figura 39 – Inserção de dois nós representando entidades especializadas

```

1 CREATE (p1:Person:Developer {name: "Marco", programming_language: "C#"})
2 CREATE (p2:Person:'Non-Developer' {name: "Tarma", free_time: "4 hours/
   day"})

```



Fonte: Elaboração própria

Entretanto, para garantir que a estrutura da especialização seja mantida no BD, são necessários os *triggers* definidos no Trecho de Código 21.

Trecho de Código 21 – Triggers reforçando especializações (t,d)

```

1 MATCH (n) WHERE

```

```
2      ("Developer" IN LABELS(n) OR
3      "Non-Developer" IN LABELS(n)) AND
4      NOT "Person" IN LABELS(n)
5 DETACH DELETE n
6
7 MATCH (n) WHERE
8      ("Developer" IN LABELS(n) AND "Non-Developer" IN LABELS(n))
9 DETACH DELETE n
10
11 MATCH (n:Person) WHERE
12      NOT "Developer" IN LABELS(n) AND
13      NOT "Non-Developer" IN LABELS(n)
14 DETACH DELETE n
```

No primeiro *trigger* garante-se que para um nó ter a etiqueta “Developer” ou a etiqueta “Non-Developer” ele deve também ter a etiqueta “Person”. Dessa forma, as regras de formato da entidade mãe também são aplicadas às instâncias das especializações.

No segundo *trigger*, pelo fato da especialização ser do tipo “d”, é preciso garantir a propriedade *disjointness*, de forma que um nó nunca possui simultaneamente as etiquetas “Developer” e “Non-Developer”. Se a especialização for do tipo “o” (*overlap* / conjunta) esse *trigger* não é necessário.

No terceiro *trigger*, pelo fato da especialização ser do tipo “t”, é preciso garantir a propriedade *completeness*, de forma que não existe um nó que contenha a etiqueta “Person” sem conter simultaneamente a etiqueta “Developer” ou a etiqueta “Non-Developer”. Se a especialização for do tipo “p” (*partial* / parcial) esse *trigger* não é necessário.

4.4.9 Auto-Relacionamento

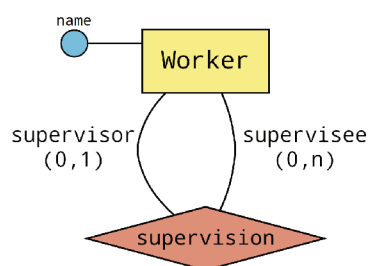
Auto-relacionamentos são relacionamentos que associam instâncias da mesma entidade. Para declará-los, é necessário definir o papel que cada lado do relacionamento representa. Na notação EER em texto, coloca-se o papel logo depois da cardinalidade na declaração do relacionamento. A Figura 40 contém um exemplo de declaração de uma modelagem EER contendo um auto-relacionamento ao lado da sua representação gráfica.

Figura 40 – Declaração de um auto-relacionamento

```

1 ent Worker {
2   name
3 }
4
5 rel supervision {
6   Worker (0,1) supervisor
7   Worker (0,n) supervisee
8 }

```



Fonte: Elaboração própria

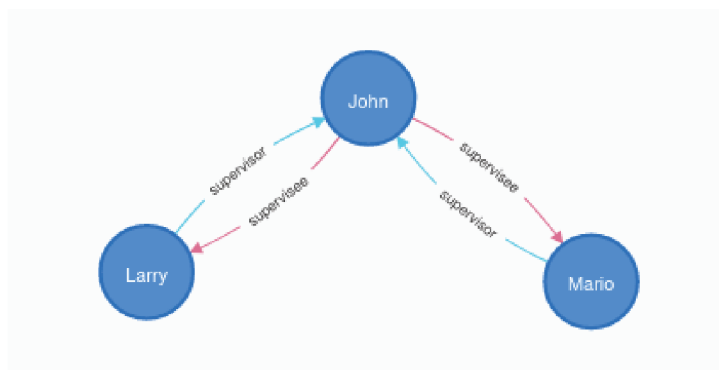
O trabalho aqui proposto representa, em um LPG, um auto-relacionamento como a criação de um relacionamento para cada papel. A Figura 41 contém um exemplo de inserção de um auto-relacionamento no Neo4j.

Figura 41 – Inserção de nós demonstrando um auto-relacionamento

```

1 CREATE (john:Worker {name: "John"})
2 CREATE (mario:Worker {name: "Mario"})
3 CREATE (larry:Worker {name: "Larry"})
4 CREATE (john)-[:supervisee]->(mario)
5 CREATE (john)-[:supervisee]->(larry)
6 CREATE (mario)-[:supervisor]->(john)
7 CREATE (larry)-[:supervisor]->(john)

```



Fonte: Elaboração própria

Para garantirmos essa estrutura, são gerados *triggers* de formato para cada um dos relacionamentos associados aos papéis. O Trecho de Código 22 mostra os *triggers* gerados para a modelagem da Figura 40. Note que essa é a única situação de mapeamento em que são necessários dois relacionamentos no BD para representar um único relacionamento na modelagem EER, nas demais situações os *triggers* buscam por relacionamentos em ambas as direções, permitindo que o administrador do BD realize a inserção do relacionamento

em qualquer um das duas, o que não é possível nesse caso onde cada direção requer um tipo próprio (nesse exemplo: “supervisor” e “supervisee”).

Trecho de Código 22 – Triggers para auto-relacionamentos

```

1 MATCH (n)-[r:supervisor]-() WHERE NOT "Worker" IN LABELS(n) DELETE r;
2
3 MATCH (n)-[r:supervisee]-() WHERE NOT "Worker" IN LABELS(n) DELETE r;
4
5 MATCH (n:Worker)-[r:supervisor]->(n:Worker)
6   WITH n, COLLECT(r) AS rs
7   WHERE SIZE(rs) > 1
8   FOREACH (r IN rs[1..] | DELETE r);

```

Os dois primeiros *triggers* garantem que apenas entidades com a etiqueta “Worker” irão se relacionar através de relacionamentos do tipo “supervisor” e “supervisee”. O terceiro *trigger* reforça a cardinalidade máxima 1 definida na modelagem de forma que um “Worker” terá no máximo um “supervisor”. Note o símbolo “->” denotando a direção do relacionamento, que permite que o mesmo nó que tem um “supervisor” seja também “supervisor” de outro nó.

4.4.10 União

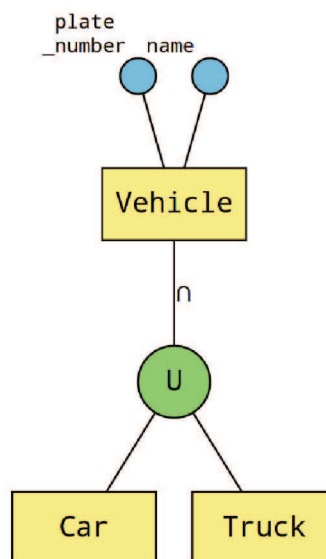
A declaração de uma união deve incluir o seu nome, o nome das entidades a serem agrupadas e os atributos que serão comuns para todas elas. A Figura 42 mostra um exemplo de declaração de uma modelagem EER contendo uma união ao lado da sua representação gráfica.

Figura 42 – Declaração de uma união

```

1 ent Car {}
2 ent Truck {}
3
4 union Vehicle {
5   Car
6   Truck
7   plate_number
8   name
9 }

```



Fonte: Elaboração própria

O trabalho aqui proposto representa, em um LPG, uma união como uma etiqueta para os nós que representam as entidades que fazem parte da união, conforme exemplificado na Figura 43. Desta forma, o nó *Car* possui, além da sua etiqueta principal, a etiqueta da entidade *Vehicle*, que representa a união.

Figura 43 – Inserção de nós representando instâncias das entidades “Car” e “Truck”, respectivamente

```
1 CREATE (c1:Car:Vehicle {name: "Corsa", plate_number: 1234})
2 CREATE (c2:Truck:Vehicle {name: "Scania", plate_number: 5678})
```



Fonte: Elaboração própria

Ao atribuir a etiqueta da união a um nó juntamente com a sua etiqueta principal é possível encontrar todos os nós participantes dessa união (mesmo que eles tenham etiquetas principais diferentes) e também atribuir características em comum para eles. Entretanto, para garantir que a estrutura da união seja mantida no BD, são necessários os *triggers* definidos no Trecho de Código 23.

Trecho de Código 23 – Triggers para garantir o formato da união

```
1 MATCH (n:Vehicle) WHERE NOT "Truck" IN LABELS(n) AND NOT "Car" IN LABELS
   (n) DETACH DELETE n;
2
3 MATCH (n) WHERE NOT n:Vehicle AND (n:Truck OR n:Car) DETACH DELETE n;
```

O primeiro *trigger* remove qualquer nó com a etiqueta “Vehicle” (referente à união) que não contenha a etiqueta “Truck” ou “Car” (entidades agrupadas na união). O segundo *trigger* remove qualquer nó que contenha a etiqueta “Truck” ou “Car” mas não contenha a etiqueta “Vehicle”. No entanto, para casos em que a união não possui nenhum atributo próprio, a aplicação gerará no lugar do segundo *trigger* um *trigger* não-destrutivo, disponível no Trecho de Código 24.

Trecho de Código 24 – Trigger não-destrutivo para garantir o formato da união

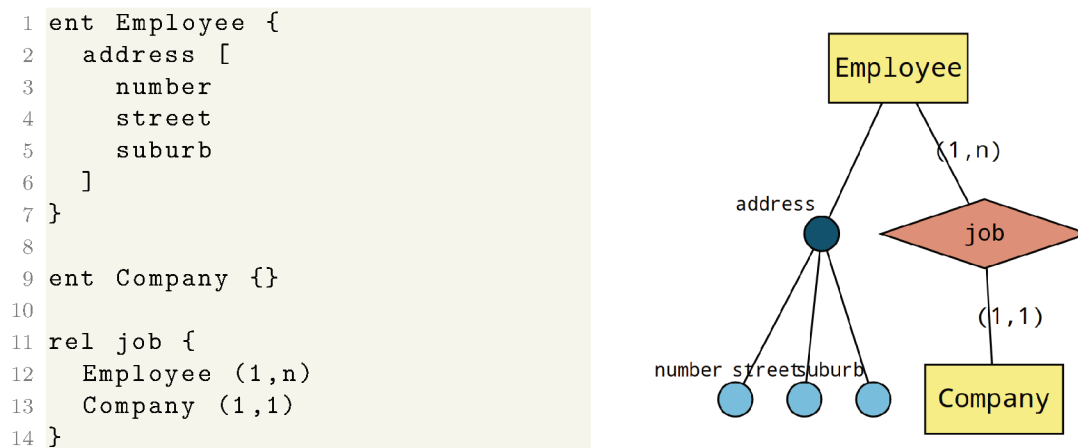
```
1 MATCH (n) WHERE NOT n:Vehicle AND (n:Truck OR n:Car) SET n:Vehicle;
```

Dado esse *trigger*, qualquer nó que contenha a etiqueta “Truck” ou “Car” mas não “Vehicle” terá a etiqueta “Vehicle” adicionada a si. Isso só é possível pois nesse caso o BD possui todas as informações necessárias para manter a estrutura dos dados consistentes com a modelagem sem a remoção de nenhum dado.

4.4.11 Strict mode

O *strict mode* é uma funcionalidade opcional da aplicação (ativa por padrão) que impede que nós com etiquetas e relacionamentos com tipos que não tenham sido especificados na modelagem EER sejam inseridos no BD. Considere a modelagem presente na Figura 44.

Figura 44 – Modelagem EER contendo atributos compostos e relacionamentos



Fonte: Elaboração própria

Para garantir que o formato dos dados seja mantido, o *strict mode* gera os seguintes *triggers* mostrados no Trecho de Código 25. Note que a etiqueta “Employee_address” e o tipo “has_employee_address” são referentes ao atributo composto “address”. Mais informações sobre o mapeamento de atributos compostos encontram-se na Seção 4.4.5.

Trecho de Código 25 – Exemplo de triggers gerados pelo *strict mode*

```

1 MATCH (n) WHERE
2   NOT "Employee" IN LABELS(n) AND
3   NOT "Company" IN LABELS(n) AND
4   NOT "Employee_address" IN LABELS(n)
5 DETACH DELETE n
6
7 MATCH ()-[r]-() WHERE
8   TYPE(r) <> "job" AND
9   TYPE(r) <> "has_employee_address"
10 DETACH DELETE r

```

Dados os *triggers* no Trecho de Código 25, qualquer nó que não contenha uma etiqueta especificada na modelagem EER e qualquer relacionamento cujo tipo não esteja no modelagem EER são removidos do BD. Através dessa funcionalidade, garante-se que qualquer dado inserido no BD respeita a modelagem EER.

5 ESTUDO DE CASO: SISTEMA DE INFORMAÇÃO PARA UMA BIBLIOTECA

A fim de validar o funcionamento da ferramenta proposta, essa seção traz uma modelagem EER para o domínio de uma biblioteca.

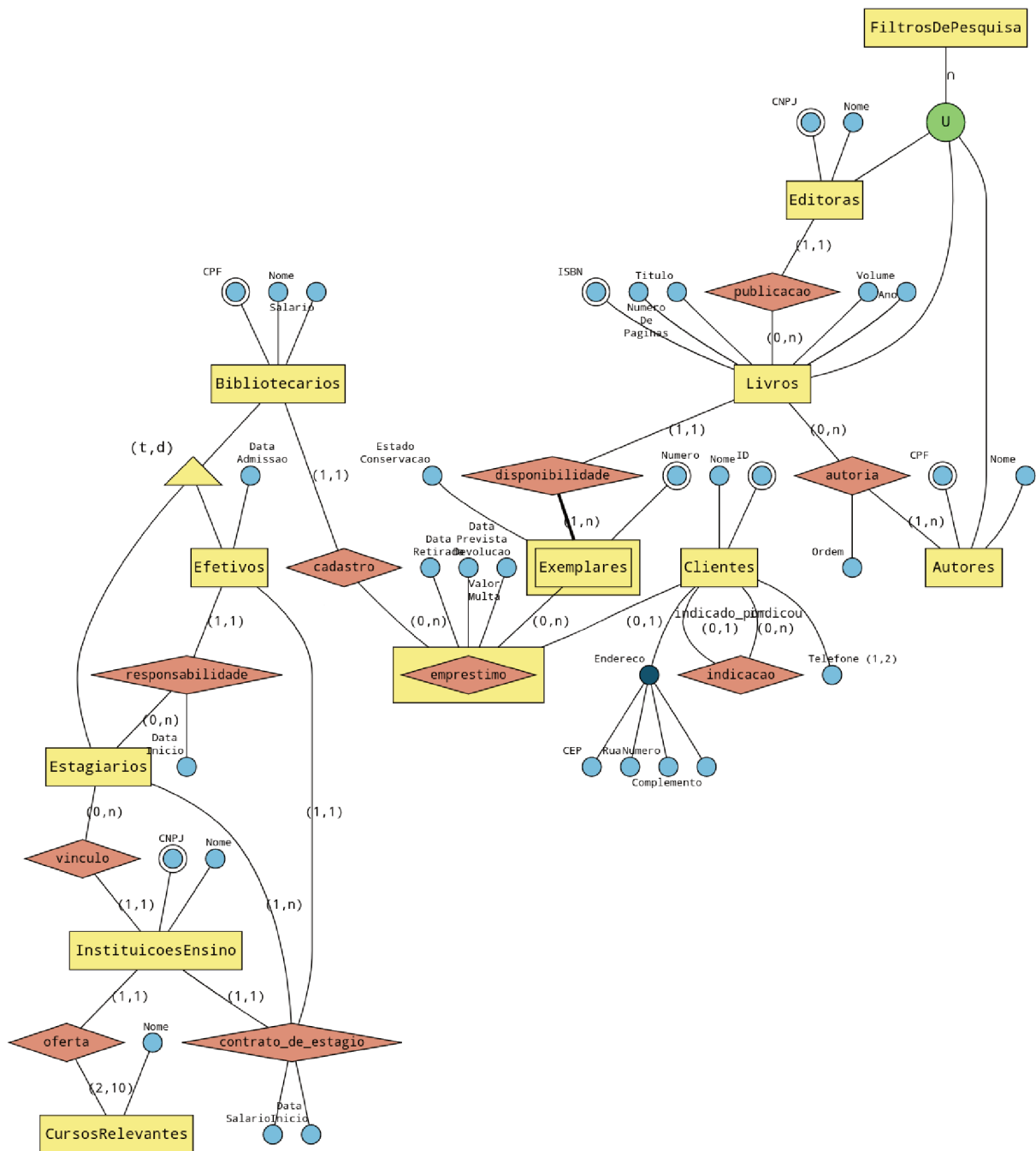
Uma biblioteca mantém um conjunto de livros, esses livros estão disponíveis através de exemplares. A biblioteca mantém informações sobre os livros, seus autores e editoras a fim de permitir a sua utilização em filtros de pesquisa no seu acervo.

Para realização de um empréstimo, é necessário que o cliente escolha alguns exemplares de livros disponíveis e comunique a um bibliotecário, que irá realizar o cadastro desse empréstimo. O bibliotecário pode ser um efetivo ou estagiário. Cada estagiário tem um efetivo responsável por ensiná-lo a utilizar o sistema da biblioteca. Para a contratação de um estagiário, um contrato de estágio se faz necessário. O contrato vincula um estagiário, sua instituição de ensino e também um efetivo da biblioteca. A biblioteca só contrata estagiários de instituições de ensino que ofertem pelo menos dois cursos relevantes para a área de biblioteconomia.

Quando um cliente realiza o seu cadastro na biblioteca, ele avisa se foi indicado por algum outro cliente, para que a biblioteca possa manter um controle das indicações. A biblioteca guarda até dois números de telefone de um cliente, para poder contatá-lo em caso de atrasos na devolução, e também seu endereço, incluindo o CEP, a rua, o número e o complemento.

Essa modelagem utiliza todos os recursos da notação EER abordados nas seções anteriores, sendo eles: entidades, relacionamentos, entidades fracas, auto-relacionamentos, relacionamentos entre três ou mais entidades, entidades associativas, atributos compostos, atributos multivalorados, especializações e uniões. A Figura 45 mostra a representação gráfica dessa modelagem EER gerada pela ferramenta tendo como entrada a modelagem na notação EER em texto presente no Apêndice A.

Figura 45 – Representação gráfica da modelagem EER para o estudo de caso



Fonte: Elaboração própria

O esquema para o BD Neo4j gerado pela ferramenta a partir dessa modelagem está disponível no Apêndice B.

As linhas 1-36 contêm as inserções dos *triggers* referentes ao *strict mode*, que controlam as etiquetas (para nós) e tipos (para relacionamentos) permitidos no BD, a fim de remover qualquer nó ou relacionamento não previsto pela modelagem. Note que além das etiquetas e tipos explicitamente definidos na modelagem, estão presentes também as

etiquetas e tipos necessários para dar suporte às funcionalidades dos modelos ER e EER cujos mapeamentos para LPG não são triviais como: “has_clientes_endereco” (para suportar o atributo composto “Endereco” de “Clientes”), “associated_to_emprestimo” (para suportar a entidade associativa “emprestimo”) e “connected_to_contrato_de_estagio” (para suportar o relacionamento n-ário “contrato_de_estagio”).

As linhas 38-77 contêm as inserções das *constraints* que controlam quais propriedades cada nó ou relacionamento deve conter com base na sua etiqueta ou tipo. Ainda, para as propriedades cujo atributo foi modelado como único, é adicionada a *constraint* de unicidade daquele valor entre os nós ou relacionamentos que compartilham da mesma etiqueta.

As linhas 79-86 contêm as inserções dos *triggers* que controlam a cardinalidade dos atributos multivalorados. Já as linhas 88-109 contêm as inserções dos *triggers* que controlam o comportamento das especializações. O primeiro *trigger* remove quaisquer nós que contenham a etiqueta de uma das entidades especializadas (“Estagiarios” e “Efetivos”) sem conter também a da entidade genérica “Bibliotecarios”. Note que não é possível apenas inserir a etiqueta da entidade genérica nesses nós porque “Bibliotecarios” contém *constraints* que requerem propriedades que esses nós podem não ter. Como a especialização nesse caso é disjunta, o segundo *trigger* remove qualquer nó que contenha duas ou mais etiquetas das entidades especializadas. Como a especialização é total, o terceiro *trigger* remove qualquer nó que contenha a etiqueta da entidade generalizada sem conter também a de uma das entidades especializadas.

As linhas 111-126 contêm as inserções dos *triggers* que controlam o comportamento das uniões. O primeiro *trigger* remove qualquer nó que contenha a etiqueta da união “FiltroDePesquisa” sem conter também a etiqueta de uma das entidades agrupadas (“Autores”, “Livros” ou “Editoras”). Como a união não possui atributos próprios (logo não possui nenhuma *constraint*), o segundo *trigger* atribui a etiqueta da união para qualquer nó que contenha apenas a etiqueta de uma das entidades agrupadas.

As linhas 128-134 contêm a inserção do *trigger* que controla o comportamento da entidade fraca “Exemplares”. Esse *trigger* remove qualquer nó com a etiqueta “Exemplares” que deixe de estar associado a um nó com a etiqueta “Livros” através de um relacionamento do tipo “disponibilidade”.

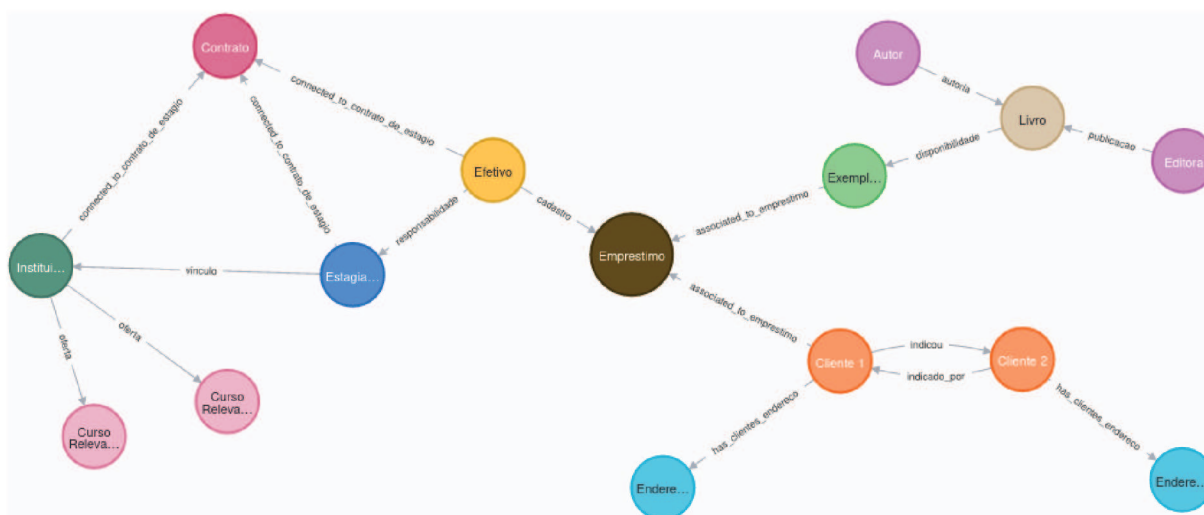
As linhas 136-280 contêm as inserções dos *triggers* que controlam o formato dos relacionamentos no BD. Esses *triggers* garantem que os relacionamentos só acontecem entre as entidades com as etiquetas corretas a fim de respeitar a modelagem.

As linhas 282-407 contêm as inserções dos *triggers* que controlam as cardinalidades dos relacionamentos no BD. Note que, além dos *triggers* para o controle da cardinalidade dos relacionamentos triviais (aqueles entre duas entidades e explicitamente modelados), estão inclusos os *triggers* que controlam as cardinalidades para os atributos compostos, relacionamentos n-ários e as entidades associativas. Todos esses relacionamentos são mo-

delados como nós e, portanto, requerem a criação de novos relacionamentos que precisam ter a cardinalidade controlada. Nas linhas 402-407, por exemplo, está definido o *trigger* que impede que mais de um “Cliente” se conecte ao mesmo “emprestimo”. Esse *trigger* é definido sobre o relacionamento “associated_to_emprestimo”, que não está presente na modelagem original, mas se faz necessário pelo fato do “emprestimo” ser uma entidade associativa e precisar de um novo relacionamento para poder se relacionar com outros nós (conforme mapeamento proposto na seção 4.4.4).

Para facilitar a visualização da estrutura definida pelo esquema gerado, foram inseridos valores de teste em uma instância do Neo4j que já contém o esquema gerado pela ferramenta, conforme o código disponível no Apêndice C. A visualização desse LPG está disponível na Figura 46. Note que a propriedade “Nome” (para os nós que a possuem) foi utilizada para representar o nó na visualização, a fim de facilitar a sua leitura.

Figura 46 – LPG respeitando o esquema definido pela modelagem EER para o estudo de caso



Fonte: Elaboração própria

6 ANÁLISE DE DESEMPENHO

Esta seção demonstra os resultados de uma análise do tempo de processamento das etapas *ER to JSON* (construção da representação intermediária) e *JSON to Cypher* (construção do esquema do BD) para modelagens EER com tamanhos variados. O objetivo dessa análise é verificar a complexidade dos algoritmos desenvolvidos na ferramenta.

A fim de facilitar a realização dos testes, foi implementado um gerador de modelagens EER sintéticas com tamanho arbitrário de entidades e relacionamentos. Em todas as modelagens geradas foi estabelecido um número fixo de três atributos em cada entidade e em cada relacionamento. As cardinalidades utilizadas nos relacionamentos são 1/3 (1,1), 1/3 (1,n) e 1/3 (n,m). Os seguintes cenários de teste foram gerados:

- Modelagem com 50 entidades e 50 relacionamentos;
- Modelagem com 100 entidades e 100 relacionamentos;
- Modelagem com 200 entidades e 200 relacionamentos;
- Modelagem com 400 entidades e 400 relacionamentos.

Os testes foram realizados em um computador com um processador Intel i7 3770 3.9 GHz e 16 GB de RAM 1600 MHz, uma configuração muito superior à necessária para execução da ferramenta. Para realização dos testes, o servidor da aplicação foi executado localmente e requisições sucessivas foram submetidas aos *endpoints* que realizam cada etapa de processamento, resultando em 20 execuções de cada algoritmo para cada cenário de teste. As Figuras 47 e 48 apresentam, respectivamente, os tempos de resposta (em milissegundos) obtidos nas etapas *ER to JSON* e *JSON to Cypher* para cada cenário.

Figura 47 – Tempo de processamento da etapa ER to JSON nos diferentes cenários

E + R	Tempo (ms)									
50 + 50	4,57	4,13	4,34	3,11	3,24	8,12	4,35	5,32	3,49	5,38
	6,42	5,5	4,75	5,79	3,59	3,73	5,89	5,11	6,18	4,23
100 + 100	5,13	11,6	3,22	4,47	3,7	3,85	8,88	9,84	2,62	3,31
	2,31	3,76	2,95	10,3	2,4	13,7	2,46	9,4	8,57	5,23
200 + 200	13,1	18,2	15,6	11,4	16	10,7	14,4	14,7	7,47	18,4
	16,7	10,9	7,42	17,9	14,6	11,7	7,53	16,5	11,3	10,7
400 + 400	19,7	15,7	26,1	26	21,7	22	22,3	20,6	22,1	21,1
	22,9	15,3	22,5	22,1	29,1	22,5	23,5	19,7	23,8	22,4

Fonte: Elaboração própria

Figura 48 – Tempo de processamento da etapa JSON to Cypher nos diferentes cenários

E + R	Tempo (ms)									
	8,9	8,56	8,04	5,71	8,09	8,6	10,8	6,94	7,03	5,42
50 + 50	8,12	8,26	8,91	8,55	8,53	5,62	5,26	7,28	6,44	9,86
	6,52	12,4	6,28	4,92	5,01	4,78	8,16	4,95	4,66	7,31
100 + 100	7,36	8,48	12,8	8,7	8,99	10,7	11,8	4,31	9,79	10,5
	36	15,5	27,6	41,7	47,5	42,4	38,2	33,4	35,1	38,3
200 + 200	38,2	48,5	38,4	30,5	30,8	38,3	42	39,2	39,8	27
	48	69,8	50,9	59,5	55,6	61,4	53,8	58,9	50	56,4
400 + 400	54,2	39,1	46	44,1	56,7	57,2	56,5	61,7	51,3	47,3

Fonte: Elaboração própria

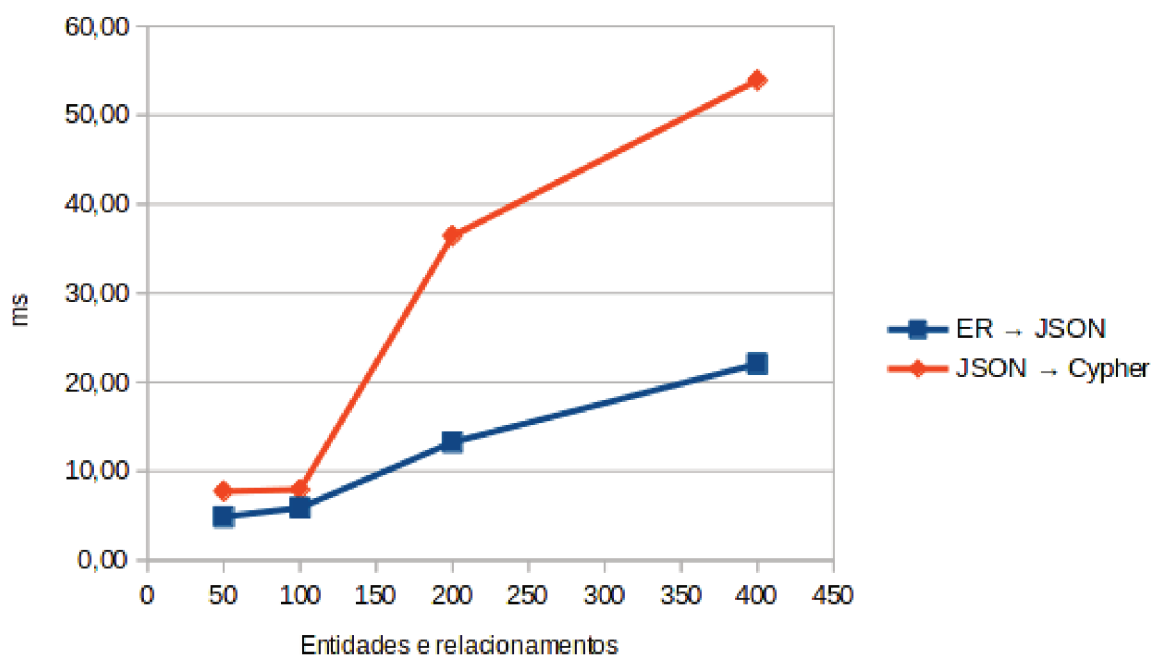
As Figuras 49 e 50 mostram, respectivamente, a média aritmética dos tempos de resposta observados em cada cenário e o gráfico referente a essas medições.

Figura 49 – Tempo médio de processamento de cada etapa por cenário

ER → JSON		JSON → Cypher	
N + M	Tempo (ms)	N+M	Tempo (ms)
50	4,86	50	7,75
100	5,89	100	7,92
200	13,26	200	36,42
400	22,06	400	53,92

Fonte: Elaboração própria

Figura 50 – Análise do tempo de processamento para múltiplos cenários de teste



Fonte: Elaboração própria

Como é possível observar, para modelagens com até 100 entidades e 100 relacionamentos o crescimento do tempo de processamento para ambas as etapas se deu de forma lenta. Entretanto, ao passar desse ponto, o tempo de processamento subiu rapidamente. Considerando n como a soma do número de entidades e relacionamentos ($E + R$) presentes em uma modelagem de entrada, no que se refere à análise de complexidade temporal o comportamento observado na etapa *ER to JSON* se assemelha ao de uma função $O(n)$ e o comportamento para a etapa *JSON to Cypher* ao de uma função $O(n \log n)$.

Esses resultados indicam que a etapa *ER to JSON* apresenta uma boa complexidade de tempo. Já a etapa *JSON to Cypher* pode sofrer aprimoramentos nos seus algoritmos a fim de alcançar uma complexidade de tempo mais próxima à etapa *ER to JSON*.

7 CONCLUSÃO

Este trabalho de conclusão de curso teve como objetivo principal o desenvolvimento de uma ferramenta que transforma modelagens ER e EER em esquemas, na linguagem Cypher, para o SGBD de grafo Neo4j. Os esquemas fazem uso da funcionalidade de *constraints*, fornecida pelo SGBD, e também da interface de *triggers*, proveniente do plugin APOC, para garantir que os dados inseridos no BD respeitem a modelagem conceitual.

A principal motivação para a realização desse trabalho é facilitar o projeto de BDs de grafos, permitindo ainda que conhecimentos prévios na utilização dos modelos ER e EER para modelagem conceitual de BDs relacionais possam ser reaproveitados pelos profissionais da área na criação de BDs de grafos.

As principais contribuições desse trabalho são:

- A notação EER em texto, capaz de representar de forma textual todos os conceitos presentes no modelo EER;
- Regras para o mapeamento de todos os conceitos do modelo EER para LPG;
- A implementação, através de *triggers* e *constraints*, das restrições de integridade no Neo4j referentes aos conceitos do modelo EER;
- A ferramenta *Dango*, que recebe modelagens EER em texto como entrada, gera a visualização gráfica, a representação intermediária JSON e por fim o esquema para o BD Neo4j.

O estudo de caso presente no Capítulo 5 demonstra que é possível utilizar a notação EER em texto para representar modelagens EER não-triviais e a ferramenta *Dango* para transformar tal modelagem em um esquema válido e funcional para o BD Neo4j. Dessa forma, o objetivo geral do trabalho pode ser considerado atingido.

Como atividades futuras relacionadas a este trabalho considera-se:

- A análise do plano de execução dos comandos *Cypher* gerados pela ferramenta a fim de avaliar a performance dos *triggers* gerados e propor otimizações para o controle do esquema do BD;
- Uma avaliação de usabilidade da ferramenta em turmas de disciplinas avançadas de BD na graduação e no Programa de Pós-Graduação em Ciência da Computação (PPGCC).
- Tentar obter o código fonte dos trabalhos relacionados a fim de realizar uma análise comparativa em termos de desempenho e qualidade dos esquemas gerados para os BDs de grafo.

- Realizar a integração dos algoritmos implementados com a ferramenta para projeto de BDs *brModeloWeb*, a fim de estender a sua funcionalidade para suportar o projeto de BDs de grafo.

Por fim, cabe observar que o código da aplicação desenvolvida está disponível na íntegra em um repositório público do GitHub¹.

¹ <https://github.com/telmotrooper/dango>

REFERÊNCIAS

- AKOKA, Jacky; COMYN-WATTIAU, Isabelle; PRAT, Nicolas. A four V's design approach of NoSQL graph databases. *In: SPRINGER. INTERNATIONAL Conference on Conceptual Modeling. [S.l.: s.n.], 2017. P. 58–68.*
- BACHMAN, Michal. **Modelling Data in Neo4j: Bidirectional Relationships**. 2013. Disponível em: <https://graphaware.com/neo4j/2013/10/11/neo4j-bidirectional-relationships.html>. Acesso em: 26 mar. 2020.
- CATTELL, Rick. Scalable SQL and NoSQL data stores. **Acm Sigmod Record**, ACM New York, NY, USA, v. 39, n. 4, p. 12–27, 2011. Disponível em: <http://www.cattell.net/datastores/Datastores.pdf>. Acesso em: 23 mar. 2020.
- CHEN, Peter Pin-Shan. The entity-relationship model—toward a unified view of data. **ACM transactions on database systems (TODS)**, Acm New York, NY, USA, v. 1, n. 1, p. 9–36, 1976. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.526.369&rep=rep1&type=pdf>. Acesso em: 26 mar. 2020.
- CODD, Edgar Frank. A relational model of data for large shared data banks. **Communications of the ACM**, ACM New York, NY, USA, v. 26, n. 1, p. 64–69, 1983.
- DANIEL, Gwendal; SUNYÉ, Gerson; CABOT, Jordi. UMLtoGraphDB: mapping conceptual schemas to graph databases. *In: SPRINGER. INTERNATIONAL Conference on Conceptual Modeling. [S.l.: s.n.], 2016. P. 430–444.* Disponível em: <https://hal.archives-ouvertes.fr/hal-01344015/document>. Acesso em: 24 mar. 2020.
- DE VIRGILIO, Roberto; MACCIONI, Antonio; TORLONE, Riccardo. Model-driven design of graph databases. *In: SPRINGER. INTERNATIONAL Conference on Conceptual Modeling. [S.l.: s.n.], 2014. P. 172–185.* Disponível em: https://www.researchgate.net/publication/312721340_Model-Driven_Design_of_Graph_Databases. Acesso em: 26 mar. 2020.
- EASTERBROOK, Steve. **Lecture 12: Entity Relationship Modelling**. 2005. Disponível em: <http://www.cs.toronto.edu/~sme/CSC340F/slides/12-relationships.pdf>. Acesso em: 26 mar. 2020.
- ELMASRI, Ramez; NAVATHE, Shamkant B. **Fundamentals of Database Systems**. 7. ed. [S.l.]: Pearson Higher Education, 2015.
- DB-ENGINES. **DBMS popularity broken down by database model**. 2020. Disponível em: https://db-engines.com/en/ranking_categories. Acesso em: 26 mar. 2020.

GALLANT, Andrew. **ERD: An Entity-Relationship diagram generator written in Haskell**. 2020. Disponível em: <https://github.com/BurntSushi/erd>. Acesso em: 1 abr. 2020.

HOCK-CHUAN, Chua. **MySQL Sample Databases**. 2012. Disponível em: <https://www.ntu.edu.sg/home/ehchua/programming/sql/sampledatabases.html>. Acesso em: 26 mar. 2020.

HOLISTICS. **dbdiagram.io - Database Relationship Diagrams Design Tool**. 2020. Disponível em: <https://dbdiagram.io/>. Acesso em: 26 mar. 2020.

JING HAN *et al.* Survey on NoSQL database. *In: IEEE. 2011 6th International Conference on Pervasive Computing and Applications*. [S.l.: s.n.], 2011. P. 363–366. Disponível em: <https://ieeexplore.ieee.org/document/6106531>. Acesso em: 26 mar. 2020.

LV, Teng; YAN, Ping; HE, Weimin. Survey on JSON data modelling. *In: JOURNAL of Physics: Conference Series (JPCS)*. [S.l.: s.n.], 2018. P. 683–687.

NEO4J, Inc. **The Neo4j Developer Manual v3.4**. [S.l.], 2019. Disponível em: <https://neo4j.com/docs/developer-manual/current/index.html>. Acesso em: 28 mar. 2020.

PHAN, Phuong Duy. **DBML - Database Markup Language**. 2020. Disponível em: <https://www.dbml.org>. Acesso em: 1 abr. 2020.

POKORNY, Jaroslav. Modelling of graph databases. **Journal of Advanced Engineering and Computation**, v. 1, n. 1, p. 04–17, 2017. Disponível em: <http://jaec.vn/index.php/JAEC/article/view/44/17>. Acesso em: 26 mar. 2020.

PRATT, Philip J.; ADAMSKI, Joseph J. **Concepts of Database Management**. 7. ed. [S.l.]: Cengage Learning, 2011.

ROBINSON, Ian; WEBBER, Jim; EIFREM, Emil. **Graph Databases: New Opportunities for Connected Data**. 2. ed. Sebastopol, CA, USA: O'Reilly Media, Inc., 2015. Disponível em: <https://neo4j.com/graph-databases-book/>. Acesso em: 23 mar. 2020.

RODRIGUEZ, Marko A.; NEUBAUER, Peter. Constructions from dots and lines. **Bulletin of the American Society for Information Science and Technology**, Wiley Online Library, v. 36, n. 6, p. 35–41, 2010. Disponível em: <https://arxiv.org/pdf/1006.2361.pdf>. Acesso em: 23 mar. 2020.

ROE, Charles. **The Question of Database Transaction Processing: An ACID, BASE, NoSQL Primer**. [S.l.], 2013. Disponível em: <https://>

[//elearning.unimib.it/pluginfile.php/267343/mod_resource/content/1/ACID-BASE-NoSQL.pdf](http://elearning.unimib.it/pluginfile.php/267343/mod_resource/content/1/ACID-BASE-NoSQL.pdf). Acesso em: 26 mar. 2020.

SCHMITZ, Roman. **Erwiz: A tool for text-based entity-relationship diagrams**. 2011. Disponível em: <https://github.com/slopjong/Erwiz>. Acesso em: 7 nov. 2020.

SOUSA, Victor Martins de. **Projeto Lógico de Banco de Dados NoSQL de Grafos a partir de um modelo conceitual baseado no modelo Entidade-Relacionamento**. 2018. Disponível em: <http://www.cc.faccamp.br/Dissertacoes/VictorMartinsSousa.pdf>. Acesso em: 26 jun. 2020.

SOUZA NETO, Milton Bittencourt de. **brModeloWeb: Ferramenta Web para Ensino e Modelagem de Banco de Dados**. 2016. Disponível em: <https://repositorio.ufsc.br/xmlui/handle/123456789/171508>. Acesso em: 26 mar. 2020.

VÁGNER, Anikó. Store and Visualize EER in Neo4j. *In: PROCEEDINGS of the 2nd International Symposium on Computer Science and Intelligent Control*. Stockholm, Sweden: Association for Computing Machinery, 2018. (ISCSIC '18). DOI: 10.1145/3284557.3284694. Disponível em: <https://dl.acm.org/doi/10.1145/3284557.3284694>. Acesso em: 26 mar. 2020.

WEST, Douglas B. **Introduction to Graph Theory**. 2. ed. [S.l.]: Pearson Education (India), 2002. Disponível em: <http://docshare01.docshare.tips/files/26167/261678089.pdf>. Acesso em: 24 mar. 2020.

Apêndices

APÊNDICE A – MODELAGEM EER REFERENTE AO ESTUDO DE CASO (NA NOTAÇÃO EER EM TEXTO)

```
1 ent Bibliotecarios {
2   CPF *
3   Nome
4   Salario
5 }
6
7 ent Estagiarios {}
8
9 ent Efetivos {
10  DataAdmissao
11 }
12
13 ent InstituicoesEnsino {
14  CNPJ *
15  Nome
16 }
17
18 ent Exemplares {
19  Numero *
20  EstadoConservacao
21 }
22
23 ent Clientes {
24  ID *
25  Nome
26  Telefone <1,2>
27  Endereco [
28    CEP
29    Rua
30    Numero
31    Complemento
32  ]
33 }
34
35 ent Livros {
36  ISBN *
37  Titulo
38  Volume
39  Ano
40  NumeroDePaginas
41 }
42
43 ent Autores {
44  CPF *
```

```
45     Nome
46 }
47
48 ent Editoras {
49     CNPJ *
50     Nome
51 }
52
53 ent CursosRelevantes {
54     Nome
55 }
56
57 spe {
58     Bibliotecarios (t,d)
59     Estagiarios
60     Efetivos
61 }
62
63 rel oferta {
64     InstituicoesEnsino (1,1)
65     CursosRelevantes (2,10)
66 }
67
68 rel responsabilidade {
69     Efetivos (1,1)
70     Estagiarios (0,n)
71     DataInicio
72 }
73
74 rel vinculo {
75     Estagiarios (0,n)
76     InstituicoesEnsino (1,1)
77 }
78
79 rel cadastro {
80     Bibliotecarios (1,1)
81     Emprestimo (0,n)
82 }
83
84 rel disponibilidade {
85     Livros (1,1)
86     w Exemplares (1,n)
87 }
88
89 rel publicacao {
90     Editoras (1,1)
91     Livros (0,n)
```

```
92 }
93
94 rel autoria {
95     Livros (0,n)
96     Autores (1,n)
97     Ordem
98 }
99
100 aent Emprestimo {
101     Exemplares (0,n)
102     Clientes (0,1)
103     DataRetirada
104     DataPrevistaDevolucao
105     ValorMulta
106 }
107
108 rel indicacao {
109     Clientes (0,1) indicado_por
110     Clientes (0,n) indicou
111 }
112
113 union FiltrosDePesquisa {
114     Editoras
115     Livros
116     Autores
117 }
118
119 rel contrato_de_estagio {
120     InstituicoesEnsino (1,1)
121     Estagiarios (1,n)
122     Efetivos (1,1)
123     DataInicio
124     Salario
125 }
```

APÊNDICE B – ESQUEMA NEO4J REFERENTE AO ESTUDO DE CASO

```

1 /* Strict mode */
2
3 CALL apoc.trigger.add('Strict mode (nodes)', 'CALL apoc.periodic.submit
  ("Strict mode (nodes)", \'
4   MATCH (n) WHERE
5     NOT "Bibliotecarios" IN LABELS(n) AND
6     NOT "Estagiarios" IN LABELS(n) AND
7     NOT "Efetivos" IN LABELS(n) AND
8     NOT "InstituicoesEnsino" IN LABELS(n) AND
9     NOT "Exemplares" IN LABELS(n) AND
10    NOT "Clientes" IN LABELS(n) AND
11    NOT "Livros" IN LABELS(n) AND
12    NOT "Autores" IN LABELS(n) AND
13    NOT "Editoras" IN LABELS(n) AND
14    NOT "CursosRelevantes" IN LABELS(n) AND
15    NOT "Clientes_Endereco" IN LABELS(n) AND
16    NOT "Emprestimo" IN LABELS(n) AND
17    NOT "contrato_de_estagio" IN LABELS(n)
18  DETACH DELETE n
19  \')', {phase: 'after'});
20
21 CALL apoc.trigger.add('Strict mode (relationships)', 'CALL apoc.periodic
  .submit("Strict mode (relationships)", \'
22  MATCH ()-[r]-() WHERE
23    TYPE(r) <> "oferta" AND
24    TYPE(r) <> "responsabilidade" AND
25    TYPE(r) <> "vinculo" AND
26    TYPE(r) <> "cadastro" AND
27    TYPE(r) <> "disponibilidade" AND
28    TYPE(r) <> "publicacao" AND
29    TYPE(r) <> "autoria" AND
30    TYPE(r) <> "has_clientes_endereco" AND
31    TYPE(r) <> "associated_to_emprestimo" AND
32    TYPE(r) <> "connected_to_contrato_de_estagio" AND
33    TYPE(r) <> "indicado_por" AND
34    TYPE(r) <> "indicou"
35  DETACH DELETE r
36  \')', {phase: 'after'});
37
38 /* Constraints */
39
40 CREATE CONSTRAINT ON (b:Bibliotecarios) ASSERT exists(b.CPF);
41 CREATE CONSTRAINT ON (b:Bibliotecarios) ASSERT exists(b.Nome);
42 CREATE CONSTRAINT ON (b:Bibliotecarios) ASSERT exists(b.Salario);
43 CREATE CONSTRAINT ON (b:Bibliotecarios) ASSERT (b.CPF) IS UNIQUE;

```

```
44 CREATE CONSTRAINT ON (e:Efetivos) ASSERT exists(e.DataAdmissao);
45 CREATE CONSTRAINT ON (i:InstituicoesEnsino) ASSERT exists(i.CNPJ);
46 CREATE CONSTRAINT ON (i:InstituicoesEnsino) ASSERT exists(i.Nome);
47 CREATE CONSTRAINT ON (i:InstituicoesEnsino) ASSERT (i.CNPJ) IS UNIQUE;
48 CREATE CONSTRAINT ON (e:Exemplares) ASSERT exists(e.Numero);
49 CREATE CONSTRAINT ON (e:Exemplares) ASSERT exists(e.EstadoConservacao);
50 CREATE CONSTRAINT ON (e:Exemplares) ASSERT (e.Numero) IS UNIQUE;
51 CREATE CONSTRAINT ON (c:Clientes) ASSERT exists(c.ID);
52 CREATE CONSTRAINT ON (c:Clientes) ASSERT exists(c.Nome);
53 CREATE CONSTRAINT ON (c:Clientes) ASSERT (c.ID) IS UNIQUE;
54 CREATE CONSTRAINT ON (c:Clientes_Endereco) ASSERT exists(c.CEP);
55 CREATE CONSTRAINT ON (c:Clientes_Endereco) ASSERT exists(c.Rua);
56 CREATE CONSTRAINT ON (c:Clientes_Endereco) ASSERT exists(c.Numero);
57 CREATE CONSTRAINT ON (c:Clientes_Endereco) ASSERT exists(c.Complemento);
58 CREATE CONSTRAINT ON (l:Livros) ASSERT exists(l.ISBN);
59 CREATE CONSTRAINT ON (l:Livros) ASSERT exists(l.Titulo);
60 CREATE CONSTRAINT ON (l:Livros) ASSERT exists(l.Volume);
61 CREATE CONSTRAINT ON (l:Livros) ASSERT exists(l.Ano);
62 CREATE CONSTRAINT ON (l:Livros) ASSERT exists(l.NumeroDePaginas);
63 CREATE CONSTRAINT ON (l:Livros) ASSERT (l.ISBN) IS UNIQUE;
64 CREATE CONSTRAINT ON (a:Autores) ASSERT exists(a.CPF);
65 CREATE CONSTRAINT ON (a:Autores) ASSERT exists(a.Nome);
66 CREATE CONSTRAINT ON (a:Autores) ASSERT (a.CPF) IS UNIQUE;
67 CREATE CONSTRAINT ON (e:Editoras) ASSERT exists(e.CNPJ);
68 CREATE CONSTRAINT ON (e:Editoras) ASSERT exists(e.Nome);
69 CREATE CONSTRAINT ON (e:Editoras) ASSERT (e.CNPJ) IS UNIQUE;
70 CREATE CONSTRAINT ON (c:CursosRelevantes) ASSERT exists(c.Nome);
71 CREATE CONSTRAINT ON ()-[r:responsabilidade]-() ASSERT exists(r.
    DataInicio);
72 CREATE CONSTRAINT ON ()-[a:autoria]-() ASSERT exists(a.Ordem);
73 CREATE CONSTRAINT ON (c:contrato_de_estagio) ASSERT exists(c.DataInicio)
    ;
74 CREATE CONSTRAINT ON (c:contrato_de_estagio) ASSERT exists(c.Salario);
75 CREATE CONSTRAINT ON (e:Emprestimo) ASSERT exists(e.DataRetirada);
76 CREATE CONSTRAINT ON (e:Emprestimo) ASSERT exists(e.
    DataPrevistaDevolucao);
77 CREATE CONSTRAINT ON (e:Emprestimo) ASSERT exists(e.ValorMulta);
78
79 /* Multivalued attributes */
80
81 CALL apoc.trigger.add('Clientes telefone multivalued', 'CALL apoc.
    periodic.submit("Clientes telefone multivalued", \'
82     MATCH (n:Clientes) WHERE
83         size(n.Telefone) < 1 OR
84         size(n.Telefone) > 2
85     DETACH DELETE n
86 \')', {phase: 'after'});
```

```
87
88 /* Specializations */
89
90 CALL apoc.trigger.add('Bibliotecarios children', 'CALL apoc.periodic.
    submit("Bibliotecarios children", \'
91     MATCH (n) WHERE
92         ("Estagiarios" IN LABELS(n) OR
93         "Efetivos" IN LABELS(n)) AND
94         NOT "Bibliotecarios" IN LABELS(n)
95     DETACH DELETE n
96 \')', {phase: 'after'});
97
98 CALL apoc.trigger.add('Bibliotecarios disjointness', 'CALL apoc.periodic
    .submit("Bibliotecarios disjointness", \'
99     MATCH (n) WHERE
100         ("Estagiarios" IN LABELS(n) AND "Efetivos" IN LABELS(n))
101     DETACH DELETE n
102 \')', {phase: 'after'});
103
104 CALL apoc.trigger.add('Bibliotecarios completeness', 'CALL apoc.periodic
    .submit("Bibliotecarios completeness", \'
105     MATCH (n:Bibliotecarios) WHERE
106         NOT "Estagiarios" IN LABELS(n) AND
107         NOT "Efetivos" IN LABELS(n)
108     DETACH DELETE n
109 \')', {phase: 'after'});
110
111 /* Unions */
112
113 CALL apoc.trigger.add('Union filtrosdepesquisa for parent', 'CALL apoc.
    periodic.submit("Union filtrosdepesquisa for parent", \'
114     MATCH (n:FiltrosDePesquisa) WHERE
115         NOT "Autores" IN LABELS(n) AND
116         NOT "Livros" IN LABELS(n) AND
117         NOT "Editoras" IN LABELS(n)
118     DETACH DELETE n
119 \')', {phase: 'after'});
120
121 CALL apoc.trigger.add('Union filtrosdepesquisa for children', 'CALL apoc
    .periodic.submit("Union filtrosdepesquisa for children", \'
122     MATCH (n) WHERE
123         NOT n:FiltrosDePesquisa AND
124         (n:Autores OR n:Livros OR n:Editoras)
125     SET n:FiltrosDePesquisa
126 \')', {phase: 'after'});
127
128 /* Weak entities */
```

```
129
130 CALL apoc.trigger.add('Weak entity exemplares in disponibilidade', 'CALL
      apoc.periodic.submit("Weak entity exemplares in disponibilidade", \'
131     MATCH (n:Exemplares)
132           WHERE NOT (:Livros)-[:disponibilidade]-(n)
133     DETACH DELETE n
134 \')', {phase: 'after'});
135
136 /* Relationships (format) */
137
138 CALL apoc.trigger.add('Has_clientes_endereco clientes_endereco clientes
      ', 'CALL apoc.periodic.submit("Has_clientes_endereco
      clientes_endereco clientes", \'
139     MATCH (n)-[r:has_clientes_endereco]-(:Clientes_Endereco) WHERE NOT "
      Clientes" IN LABELS(n) DELETE r
140 \')', {phase: 'after'});
141
142 CALL apoc.trigger.add('Has_clientes_endereco clientes_endereco', 'CALL
      apoc.periodic.submit("Has_clientes_endereco clientes_endereco", \'
143     MATCH (n)-[r:has_clientes_endereco]-(:Clientes) WHERE NOT "
      Clientes_Endereco" IN LABELS(n) DELETE r
144 \')', {phase: 'after'});
145
146 CALL apoc.trigger.add('Clientes has_clientes_endereco clientes_endereco
      ', 'CALL apoc.periodic.submit("Clientes has_clientes_endereco
      clientes_endereco", \'
147     MATCH (n)-[r:has_clientes_endereco]-() WHERE NOT "Clientes" IN
      LABELS(n) AND NOT "Clientes_Endereco" IN LABELS(n) DELETE r
148 \')', {phase: 'after'});
149
150 CALL apoc.trigger.add('Oferta cursosrelevantes instituicoesensino', '
      CALL apoc.periodic.submit("Oferta cursosrelevantes instituicoesensino
      ", \'
151     MATCH (n)-[r:oferta]-(:CursosRelevantes) WHERE NOT "
      InstituicoesEnsino" IN LABELS(n) DELETE r
152 \')', {phase: 'after'});
153
154 CALL apoc.trigger.add('Oferta cursosrelevantes', 'CALL apoc.periodic.
      submit("Oferta cursosrelevantes", \'
155     MATCH (n)-[r:oferta]-(:InstituicoesEnsino) WHERE NOT "
      CursosRelevantes" IN LABELS(n) DELETE r
156 \')', {phase: 'after'});
157
158 CALL apoc.trigger.add('Instituicoesensino oferta cursosrelevantes', '
      CALL apoc.periodic.submit("Instituicoesensino oferta cursosrelevantes
      ", \'
159     MATCH (n)-[r:oferta]-() WHERE NOT "InstituicoesEnsino" IN LABELS(n)
```

```
    AND NOT "CursosRelevantes" IN LABELS(n) DELETE r
160 \')', {phase: 'after'});
161
162 CALL apoc.trigger.add('Responsabilidade estagiarios efetivos', 'CALL
    apoc.periodic.submit("Responsabilidade estagiarios efetivos", \'
163     MATCH (n)-[r:responsabilidade]-(:Estagiarios) WHERE NOT "Efetivos"
    IN LABELS(n) DELETE r
164 \')', {phase: 'after'});
165
166 CALL apoc.trigger.add('Responsabilidade estagiarios', 'CALL apoc.
    periodic.submit("Responsabilidade estagiarios", \'
167     MATCH (n)-[r:responsabilidade]-(:Efetivos) WHERE NOT "Estagiarios"
    IN LABELS(n) DELETE r
168 \')', {phase: 'after'});
169
170 CALL apoc.trigger.add('Efetivos responsabilidade estagiarios', 'CALL
    apoc.periodic.submit("Efetivos responsabilidade estagiarios", \'
171     MATCH (n)-[r:responsabilidade]-() WHERE NOT "Efetivos" IN LABELS(n)
    AND NOT "Estagiarios" IN LABELS(n) DELETE r
172 \')', {phase: 'after'});
173
174 CALL apoc.trigger.add('Vinculo instituicoesensino estagiarios', 'CALL
    apoc.periodic.submit("Vinculo instituicoesensino estagiarios", \'
175     MATCH (n)-[r:vinculo]-(:InstituicoesEnsino) WHERE NOT "Estagiarios"
    IN LABELS(n) DELETE r
176 \')', {phase: 'after'});
177
178 CALL apoc.trigger.add('Vinculo instituicoesensino', 'CALL apoc.periodic.
    submit("Vinculo instituicoesensino", \'
179     MATCH (n)-[r:vinculo]-(:Estagiarios) WHERE NOT "InstituicoesEnsino"
    IN LABELS(n) DELETE r
180 \')', {phase: 'after'});
181
182 CALL apoc.trigger.add('Estagiarios vinculo instituicoesensino', 'CALL
    apoc.periodic.submit("Estagiarios vinculo instituicoesensino", \'
183     MATCH (n)-[r:vinculo]-() WHERE NOT "Estagiarios" IN LABELS(n) AND
    NOT "InstituicoesEnsino" IN LABELS(n) DELETE r
184 \')', {phase: 'after'});
185
186 CALL apoc.trigger.add('Cadastro emprestimo bibliotecarios', 'CALL apoc.
    periodic.submit("Cadastro emprestimo bibliotecarios", \'
187     MATCH (n)-[r:cadastro]-(:Emprestimo) WHERE NOT "Bibliotecarios" IN
    LABELS(n) DELETE r
188 \')', {phase: 'after'});
189
190 CALL apoc.trigger.add('Cadastro emprestimo', 'CALL apoc.periodic.submit
    ("Cadastro emprestimo", \'
```



```
191     MATCH (n)-[r:cadastro]-(:Bibliotecarios) WHERE NOT "Emprestimo" IN
192     LABELS(n) DELETE r
193 \')', {phase: 'after'});
194 CALL apoc.trigger.add('Bibliotecarios cadastro emprestimo', 'CALL apoc.
195     periodic.submit("Bibliotecarios cadastro emprestimo", \'
196     MATCH (n)-[r:cadastro]-() WHERE NOT "Bibliotecarios" IN LABELS(n)
197     AND NOT "Emprestimo" IN LABELS(n) DELETE r
198 \')', {phase: 'after'});
199 CALL apoc.trigger.add('Disponibilidade exemplares livros', 'CALL apoc.
200     periodic.submit("Disponibilidade exemplares livros", \'
201     MATCH (n)-[r:disponibilidade]-(:Exemplares) WHERE NOT "Livros" IN
202     LABELS(n) DELETE r
203 \')', {phase: 'after'});
204 CALL apoc.trigger.add('Disponibilidade exemplares', 'CALL apoc.periodic.
205     submit("Disponibilidade exemplares", \'
206     MATCH (n)-[r:disponibilidade]-(:Livros) WHERE NOT "Exemplares" IN
207     LABELS(n) DELETE r
208 \')', {phase: 'after'});
209 CALL apoc.trigger.add('Livros disponibilidade exemplares', 'CALL apoc.
210     periodic.submit("Livros disponibilidade exemplares", \'
211     MATCH (n)-[r:disponibilidade]-() WHERE NOT "Livros" IN LABELS(n) AND
212     NOT "Exemplares" IN LABELS(n) DELETE r
213 \')', {phase: 'after'});
214 CALL apoc.trigger.add('Publicacao livros editoras', 'CALL apoc.periodic.
215     submit("Publicacao livros editoras", \'
216     MATCH (n)-[r:publicacao]-(:Livros) WHERE NOT "Editoras" IN LABELS(n)
217     DELETE r
218 \')', {phase: 'after'});
219 CALL apoc.trigger.add('Publicacao livros', 'CALL apoc.periodic.submit("
220     Publicacao livros", \'
221     MATCH (n)-[r:publicacao]-(:Editoras) WHERE NOT "Livros" IN LABELS(n)
222     DELETE r
223 \')', {phase: 'after'});
224 CALL apoc.trigger.add('Editoras publicacao livros', 'CALL apoc.periodic.
225     submit("Editoras publicacao livros", \'
226     MATCH (n)-[r:publicacao]-() WHERE NOT "Editoras" IN LABELS(n) AND
227     NOT "Livros" IN LABELS(n) DELETE r
228 \')', {phase: 'after'});
229 CALL apoc.trigger.add('Autoria autores livros', 'CALL apoc.periodic.
```

```
submit("Autoria autores livros", \  
223   MATCH (n)-[r:autoria]-(:Autores) WHERE NOT "Livros" IN LABELS(n)  
DELETE r  
224 \')', {phase: 'after'});  
225  
226 CALL apoc.trigger.add('Autoria autores', 'CALL apoc.periodic.submit("  
Autoria autores", \  
227   MATCH (n)-[r:autoria]-(:Livros) WHERE NOT "Autores" IN LABELS(n)  
DELETE r  
228 \')', {phase: 'after'});  
229  
230 CALL apoc.trigger.add('Livros autoria autores', 'CALL apoc.periodic.  
submit("Livros autoria autores", \  
231   MATCH (n)-[r:autoria]-() WHERE NOT "Livros" IN LABELS(n) AND NOT "  
Autores" IN LABELS(n) DELETE r  
232 \')', {phase: 'after'});  
233  
234 CALL apoc.trigger.add('Clientes indicado_por clientes', 'CALL apoc.  
periodic.submit("Clientes indicado_por clientes", \  
235   MATCH (n)-[r:indicado_por]-() WHERE NOT "Clientes" IN LABELS(n)  
DELETE r  
236 \')', {phase: 'after'});  
237  
238 CALL apoc.trigger.add('Clientes indicou clientes', 'CALL apoc.periodic.  
submit("Clientes indicou clientes", \  
239   MATCH (n)-[r:indicou]-() WHERE NOT "Clientes" IN LABELS(n) DELETE r  
240 \')', {phase: 'after'});  
241  
242 CALL apoc.trigger.add('Connected_to_contrato_de_estagio  
instituicoesensino contrato_de_estagio', 'CALL apoc.periodic.submit("  
Connected_to_contrato_de_estagio instituicoesensino  
contrato_de_estagio", \  
243   MATCH (n)-[r:connected_to_contrato_de_estagio]-(:InstituicoesEnsino)  
WHERE NOT "contrato_de_estagio" IN LABELS(n) DELETE r  
244 \')', {phase: 'after'});  
245  
246 CALL apoc.trigger.add('Connected_to_contrato_de_estagio estagiarios  
contrato_de_estagio', 'CALL apoc.periodic.submit("  
Connected_to_contrato_de_estagio estagiarios contrato_de_estagio", \  
247   MATCH (n)-[r:connected_to_contrato_de_estagio]-(:Estagiarios) WHERE  
NOT "contrato_de_estagio" IN LABELS(n) DELETE r  
248 \')', {phase: 'after'});  
249  
250 CALL apoc.trigger.add('Connected_to_contrato_de_estagio efetivos  
contrato_de_estagio', 'CALL apoc.periodic.submit("  
Connected_to_contrato_de_estagio efetivos contrato_de_estagio", \  
251   MATCH (n)-[r:connected_to_contrato_de_estagio]-(:Efetivos) WHERE NOT
```

```

    "contrato_de_estagio" IN LABELS(n) DELETE r
252 \')', {phase: 'after'});
253
254 CALL apoc.trigger.add('Contrato_de_estagio associations', 'CALL apoc.
    periodic.submit("Contrato_de_estagio associations", \'
255     MATCH (n)-[r:connected_to_contrato_de_estagio]-(:contrato_de_estagio
    ) WHERE
256         NOT n:'InstituicoesEnsino' AND
257         NOT n:'Estagiarios' AND
258         NOT n:'Efetivos'
259     DETACH DELETE r
260 \')', {phase: 'after'});
261
262 CALL apoc.trigger.add('Associated_to_emprestimo exemplares emprestimo',
    'CALL apoc.periodic.submit("Associated_to_emprestimo exemplares
    emprestimo", \'
263     MATCH (n)-[r:associated_to_emprestimo]-(:Exemplares) WHERE NOT "
    Emprestimo" IN LABELS(n) DELETE r
264 \')', {phase: 'after'});
265
266 CALL apoc.trigger.add('Associated_to_emprestimo clientes emprestimo', '
    CALL apoc.periodic.submit("Associated_to_emprestimo clientes
    emprestimo", \'
267     MATCH (n)-[r:associated_to_emprestimo]-(:Clientes) WHERE NOT "
    Emprestimo" IN LABELS(n) DELETE r
268 \')', {phase: 'after'});
269
270 CALL apoc.trigger.add('Associated_to_emprestimo efetivos emprestimo', '
    CALL apoc.periodic.submit("Associated_to_emprestimo efetivos
    emprestimo", \'
271     MATCH (n)-[r:associated_to_emprestimo]-(:Efetivos) WHERE NOT "
    Emprestimo" IN LABELS(n) DELETE r
272 \')', {phase: 'after'});
273
274 CALL apoc.trigger.add('Emprestimo associations', 'CALL apoc.periodic.
    submit("Emprestimo associations", \'
275     MATCH (n)-[r:associated_to_emprestimo]-(:emprestimo) WHERE
276         NOT n:'Exemplares' AND
277         NOT n:'Clientes' AND
278         NOT n:'Efetivos'
279     DETACH DELETE r
280 \')', {phase: 'after'});
281
282 /* Relationships (cardinalities) */
283
284 CALL apoc.trigger.add('Clientes_endereco has_clientes_endereco less than
    1 clientes', 'CALL apoc.periodic.submit("Clientes_endereco

```

```
has_clientes_endereco less than 1 clientes", \'
285 MATCH (n:Clientes_Endereco) WHERE NOT (:Clientes)-[:
has_clientes_endereco]-(n) DETACH DELETE n
286 \')', {phase: 'after'});
287
288 CALL apoc.trigger.add('Clientes_endereco has_clientes_endereco more than
1 clientes', 'CALL apoc.periodic.submit("Clientes_endereco
has_clientes_endereco more than 1 clientes", \'
289 MATCH (n:Clientes_Endereco)-[r:has_clientes_endereco]-(:Clientes)
290 WITH n, COLLECT(r) AS rs
291 WHERE SIZE(rs) > 1
292 FOREACH (r IN rs[1..] | DELETE r)
293 \')', {phase: 'after'});
294
295 CALL apoc.trigger.add('Clientes has_clientes_endereco less than 1
clientes_endereco', 'CALL apoc.periodic.submit("Clientes
has_clientes_endereco less than 1 clientes_endereco", \'
296 MATCH (n:Clientes) WHERE NOT (:Clientes_Endereco)-[:
has_clientes_endereco]-(n) DETACH DELETE n
297 \')', {phase: 'after'});
298
299 CALL apoc.trigger.add('Clientes has_clientes_endereco more than 1
clientes_endereco', 'CALL apoc.periodic.submit("Clientes
has_clientes_endereco more than 1 clientes_endereco", \'
300 MATCH (n:Clientes)-[r:has_clientes_endereco]-(:Clientes_Endereco)
301 WITH n, COLLECT(r) AS rs
302 WHERE SIZE(rs) > 1
303 FOREACH (r IN rs[1..] | DELETE r)
304 \')', {phase: 'after'});
305
306 CALL apoc.trigger.add('Cursosrelevantes oferta more than 1
instituiçoesensino', 'CALL apoc.periodic.submit("Cursosrelevantes
oferta more than 1 instituiçoesensino", \'
307 MATCH (n:CursosRelevantes)-[r:oferta]-(:InstituiçoesEnsino)
308 WITH n, COLLECT(r) AS rs
309 WHERE SIZE(rs) > 1
310 FOREACH (r IN rs[1..] | DELETE r)
311 \')', {phase: 'after'});
312
313 CALL apoc.trigger.add('Instituiçoesensino oferta less than 2
cursosrelevantes', 'CALL apoc.periodic.submit("Instituiçoesensino
oferta less than 2 cursosrelevantes", \'
314 MATCH (n:InstituiçoesEnsino)-[r:oferta]-(:CursosRelevantes)
315 WITH n, COLLECT(r) AS rs
316 WHERE SIZE(rs) < 2
317 FOREACH (r IN rs | DELETE r)
318 \')', {phase: 'after'});
```

```
319
320 CALL apoc.trigger.add('Instituicoesensino oferta more than 10
      cursosrelevantes', 'CALL apoc.periodic.submit("Instituicoesensino
      oferta more than 10 cursosrelevantes", \'
321     MATCH (n:InstituicoesEnsino)-[r:oferta]-(:CursosRelevantes)
322     WITH n, COLLECT(r) AS rs
323     WHERE SIZE(rs) > 10
324     FOREACH (r IN rs[10..] | DELETE r)
325 \')', {phase: 'after'});
326
327 CALL apoc.trigger.add('Estagiarios responsabilidade more than 1 efetivos
      ', 'CALL apoc.periodic.submit("Estagiarios responsabilidade more than
      1 efetivos", \'
328     MATCH (n:Estagiarios)-[r:responsabilidade]-(:Efetivos)
329     WITH n, COLLECT(r) AS rs
330     WHERE SIZE(rs) > 1
331     FOREACH (r IN rs[1..] | DELETE r)
332 \')', {phase: 'after'});
333
334 CALL apoc.trigger.add('Estagiarios vinculo more than 1
      instituicoesensino', 'CALL apoc.periodic.submit("Estagiarios vinculo
      more than 1 instituicoesensino", \'
335     MATCH (n:Estagiarios)-[r:vinculo]-(:InstituicoesEnsino)
336     WITH n, COLLECT(r) AS rs
337     WHERE SIZE(rs) > 1
338     FOREACH (r IN rs[1..] | DELETE r)
339 \')', {phase: 'after'});
340
341 CALL apoc.trigger.add('Emprestimo cadastro more than 1 bibliotecarios',
      'CALL apoc.periodic.submit("Emprestimo cadastro more than 1
      bibliotecarios", \'
342     MATCH (n:Emprestimo)-[r:cadastro]-(:Bibliotecarios)
343     WITH n, COLLECT(r) AS rs
344     WHERE SIZE(rs) > 1
345     FOREACH (r IN rs[1..] | DELETE r)
346 \')', {phase: 'after'});
347
348 CALL apoc.trigger.add('Exemplares disponibilidade more than 1 livros', '
      CALL apoc.periodic.submit("Exemplares disponibilidade more than 1
      livros", \'
349     MATCH (n:Exemplares)-[r:disponibilidade]-(:Livros)
350     WITH n, COLLECT(r) AS rs
351     WHERE SIZE(rs) > 1
352     FOREACH (r IN rs[1..] | DELETE r)
353 \')', {phase: 'after'});
354
355 CALL apoc.trigger.add('Livros publicacao more than 1 editoras', 'CALL
```

```
apoc.periodic.submit("Livros publicacao more than 1 editoras", \'
356 MATCH (n:Livros)-[r:publicacao]-(:Editoras)
357 WITH n, COLLECT(r) AS rs
358 WHERE SIZE(rs) > 1
359 FOREACH (r IN rs[1..] | DELETE r)
360 \')', {phase: 'after'});
361
362 CALL apoc.trigger.add('Clientes indicado_por more than 1 clientes', '
CALL apoc.periodic.submit("Clientes indicado_por more than 1 clientes
", \'
363 MATCH (n:Clientes)-[r:indicado_por]->(:Clientes)
364 WITH n, COLLECT(r) AS rs
365 WHERE SIZE(rs) > 1
366 FOREACH (r IN rs[1..] | DELETE r)
367 \')', {phase: 'after'});
368
369 CALL apoc.trigger.add('Contrato_de_estagio
connected_to_contrato_de_estagio less than 1 instituicoesensino', '
CALL apoc.periodic.submit("Contrato_de_estagio
connected_to_contrato_de_estagio less than 1 instituicoesensino", \'
370 MATCH (n:contrato_de_estagio) WHERE NOT (:InstituicoesEnsino)-[:
connected_to_contrato_de_estagio]-(n) DETACH DELETE n
371 \')', {phase: 'after'});
372
373 CALL apoc.trigger.add('Contrato_de_estagio
connected_to_contrato_de_estagio more than 1 instituicoesensino', '
CALL apoc.periodic.submit("Contrato_de_estagio
connected_to_contrato_de_estagio more than 1 instituicoesensino", \'
374 MATCH (n:contrato_de_estagio)-[r:connected_to_contrato_de_estagio
]-(:InstituicoesEnsino)
375 WITH n, COLLECT(r) AS rs
376 WHERE SIZE(rs) > 1
377 FOREACH (r IN rs[1..] | DELETE r)
378 \')', {phase: 'after'});
379
380 CALL apoc.trigger.add('Contrato_de_estagio
connected_to_contrato_de_estagio less than 1 estagiarios', 'CALL apoc
.periodic.submit("Contrato_de_estagio
connected_to_contrato_de_estagio less than 1 estagiarios", \'
381 MATCH (n:contrato_de_estagio) WHERE NOT (:Estagiarios)-[:
connected_to_contrato_de_estagio]-(n) DETACH DELETE n
382 \')', {phase: 'after'});
383
384 CALL apoc.trigger.add('Contrato_de_estagio
connected_to_contrato_de_estagio more than 1 estagiarios', 'CALL apoc
.periodic.submit("Contrato_de_estagio
connected_to_contrato_de_estagio more than 1 estagiarios", \'
```

```
385     MATCH (n:contrato_de_estagio)-[r:connected_to_contrato_de_estagio
386     ]-(:Estagiarios)
387     WITH n, COLLECT(r) AS rs
388     WHERE SIZE(rs) > 1
389     FOREACH (r IN rs[1..] | DELETE r)
390 \')', {phase: 'after'});
391 CALL apoc.trigger.add('Contrato_de_estagio
392     connected_to_contrato_de_estagio less than 1 efetivos', 'CALL apoc.
393     periodic.submit("Contrato_de_estagio connected_to_contrato_de_estagio
394     less than 1 efetivos", \'
395     MATCH (n:contrato_de_estagio) WHERE NOT (:Efetivos)-[:
396     connected_to_contrato_de_estagio]-(n) DETACH DELETE n
397     \')', {phase: 'after'});
398 CALL apoc.trigger.add('Contrato_de_estagio
399     connected_to_contrato_de_estagio more than 1 efetivos', 'CALL apoc.
400     periodic.submit("Contrato_de_estagio connected_to_contrato_de_estagio
401     more than 1 efetivos", \'
402     MATCH (n:contrato_de_estagio)-[r:connected_to_contrato_de_estagio
403     ]-(:Efetivos)
404     WITH n, COLLECT(r) AS rs
405     WHERE SIZE(rs) > 1
406     FOREACH (r IN rs[1..] | DELETE r)
407     \')', {phase: 'after'});
```

APÊNDICE C – LPG REFERENTE AO ESTUDO DE CASO

```

1 CREATE (a:Autores:FiltrosDePesquisa {CPF: "000.000.000-00", Nome: "Autor
  "})
2 CREATE (e:Editoras:FiltrosDePesquisa {CNPJ: "00.000.000/0000-00", Nome:
  "Editora"})
3
4 CREATE (c1:Clientes {ID: 1, Nome: "Cliente 1", Telefone: ["123456789",
  "987654321"]})
5 CREATE (c2:Clientes {ID: 2, Nome: "Cliente 2", Telefone: ["147258369",
  "789456123"]})
6 CREATE (ce1:Clientes_Endereco {CEP: "00000-00", Rua: "Endereco 1",
  Numero: "1", Complemento: ""})
7 CREATE (ce2:Clientes_Endereco {CEP: "11111-11", Rua: "Endereco 2",
  Numero: "2", Complemento: ""})
8
9 CREATE (c1)-[:has_clientes_endereco]->(ce1)
10 CREATE (c2)-[:has_clientes_endereco]->(ce2)
11
12 CREATE (c1)-[:indicou]->(c2)
13 CREATE (c2)-[:indicado_por]->(c1)
14
15 CREATE (l:Livros:FiltrosDePesquisa {Titulo: "Livro", ISBN:
  "000-0-00-000000-0", Volume: 1, Ano: 2020, NumeroDePaginas: 120})
16 CREATE (a)-[:'autoria' {Ordem: "1"}]->(l)
17 CREATE (e)-[:publicacao]->(l)
18
19 CREATE (ex:Exemplares {Numero: 1, EstadoConservacao: "otimo", Nome: "
  Exemplar"})
20 CREATE (l)-[:disponibilidade]->(ex)
21
22 CREATE (es:Estagiarios:Bibliotecarios {CPF: "111.111.111-11", Nome: "
  Estagiario", Salario: "R$ 600"})
23 CREATE (ef:Efetivos:Bibliotecarios {CPF: "222.222.222-22", Nome: "
  Efetivo", Salario: "R$ 1.600", DataAdmissao: "01/10/2017"})
24 CREATE (ef)-[:responsabilidade {DataInicio: "01/02/2020"}]->(es)
25
26 CREATE (cde:contrato_de_estagio {Salario: "R$ 400", DataInicio:
  "01/02/2020", Nome: "Contrato de Estagio"})
27
28 CREATE (i:InstituicoesEnsino {CNPJ: "11.111.111/1111-11", Nome: "
  Instituicao de Ensino"})
29 CREATE (es)-[:vinculo]->(i)
30
31 CREATE (es)-[:connected_to_contrato_de_estagio]->(cde)
32 CREATE (ef)-[:connected_to_contrato_de_estagio]->(cde)
33 CREATE (i)-[:connected_to_contrato_de_estagio]->(cde)

```



```
34
35 CREATE (cr1:CursosRelevantes {Nome: "Curso Relevante 1"})
36 CREATE (cr2:CursosRelevantes {Nome: "Curso Relevante 2"})
37
38 CREATE (i)-[:oferta]->(cr1)
39 CREATE (i)-[:oferta]->(cr2)
40
41 CREATE (em:Emprestimo {DataRetirada: "09/11/2020", DataPrevistaDevolucao
    : "16/11/2020", ValorMulta: "R$ 0,00", Nome: "Emprestimo"})
42
43 CREATE (ef)-[:cadastro]->(em)
44 CREATE (ex)-[:associated_to_emprestimo]->(em)
45 CREATE (c1)-[:associated_to_emprestimo]->(em)
```

APÊNDICE D – ARTIGO NO FORMATO SBC

Dango: Ferramenta para Projeto de Bancos de Dados NoSQL de Grafos a partir de Diagramas Entidade-Relacionamento Estendido

Telmo Henrique Valverde da Silva¹, Ronaldo dos Santos Mello¹

¹Departamento de Informática e Estatística (INE)
Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brasil

telmo.valverde@grad.ufsc.br, r.mello@ufsc.br

Abstract. *The modern world is composed by highly connected data: electric grids composed by light poles, cables and converters that form supply lines; social networks where friends are connected, like each other's posts and share them. But how could one store this kind of data in order to query it efficiently? It could be stored in a conventional relational database, but that would require a large number of costly JOIN operations. The solution comes through graph databases, which use graph theory to model and store highly connected data. Since relationships are a central concept in the graph model, connections can be traversed in both ways with inexpensive operations. As the usage of the Enhanced Entity-Relationship (EER) model is ubiquitous for database design, this article proposes a tool to map existing EER diagrams into schemas for the Neo4j graph database.*

Resumo. *O mundo moderno é composto por dados altamente interligados: redes elétricas compostas por postes, cabos e transformadores que formam linhas de abastecimento; redes sociais onde amigos se conectam, curtem postagens e as compartilham entre si. Porém como armazenar esses dados a fim de consultá-los de forma eficiente? Eles poderiam ser colocados em Bancos de Dados (BDs) relacionais convencionais, mas isso implicaria em um número grande de custosas operações de junção. A solução então vem por meio dos BDs de grafos, que utilizam da teoria de grafos para modelar e armazenar dados altamente interligados. Como relacionamentos são um conceito central no modelo de dados de grafos, conexões podem ser consultadas em ambos os sentidos com operações eficientes. Como o uso do modelo Entidade-Relacionamento Estendido (EER) é muito comum para o projeto de BDs, este artigo propõe uma ferramenta para mapear modelagens EER existentes em esquemas para o BD de grafo Neo4j.*

1. Introdução

Bancos de Dados (BDs) relacionais são conhecidos pelos seus rígidos controles de integridade dos dados. Esses controles limitam a sua eficiência em situações com um número de dados muito grande (*big data*). Ainda, o seu modelo organizacional dificulta o armazenamento de dados sem esquema rígido e o de dados que têm muitas ligações entre si. Na intenção de contornar esses problemas surgiram os BDs NoSQL, que oferecem alternativas ao modelo relacional para o armazenamento e gerência de dados [Cattell 2011].

BDs de grafos são BDs NoSQL que permitem o armazenamento e acesso de forma eficiente à relacionamentos complexos e dinâmicos em dados altamente conectados [Robinson et al. 2015], tornando-os bastante adequados para dados de aplicações como redes sociais, sistemas de recomendações, sistemas de informação geográfica, entre outros.

Grafos são um conceito proveniente da matemática, mais especificamente da chamada teoria dos grafos [West 2002]. São estruturas compostas de dois conjuntos: o dos vértices (aqui chamados de nós) e o das arestas (aqui chamadas de relacionamentos). Nós podem ser imaginados como uma espécie de objeto ou indivíduo e os seus relacionamentos são conexões que se estabelecem entre eles. A partir dessa definição, muitas situações reais se tornam fáceis de modelar, como por exemplo, um conjunto de pessoas que seguem umas às outras em uma rede social (Figura 1).

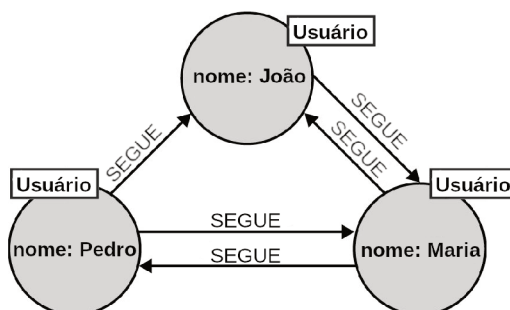


Figura 1. Exemplo de um grafo simples representando uma rede social

O grafo da Figura 1 possui algumas características interessantes: Como qualquer outro grafo, ele possui pelo menos um nó e zero ou mais relacionamentos. Os nós possuem propriedades. Os nós podem ter etiquetas. Os relacionamentos têm nomes e direções. Apesar de não mostrado na Figura 1, os relacionamentos podem ter também propriedades. Essas características fazem com que esse grafo seja um *Labeled Property Graph* (LPG) [Robinson et al. 2015] ou ainda um *Property Graph* [Rodriguez and Neubauer 2010], ou seja, um grafo etiquetado e com propriedades. Esse tipo de grafo é a base para os BDs de grafos, pois apesar de conceitualmente simples, têm um poder de expressividade muito alto.

Apesar da forma intuitiva como grafos podem ser utilizados para representar diversos tipos de situações, a forma de representação conceitual predominante na área de BD ainda é o modelo Entidade-Relacionamento Estendido (EER). Dessa forma, percebe-se a utilidade de uma ferramenta capaz de mapear quaisquer modelagens EER convencionais para um LPG e também construir uma série de restrições de integridade para garantir que os dados inseridos se comportarão conforme as restrições impostas pela modelagem.

1.1. Trabalhos Relacionados

A Tabela 1 contém uma comparação entre os trabalhos relacionados e o trabalho atual no que se refere aos conceitos dos modelos Entidade-Relacionamento (ER) e EER mapeados para BDs de grafo e às restrições de integridade implementadas. Note que as demais contribuições realizadas pelos trabalhos relacionados não foram incluídas na lista, como, por exemplo, a inclusão de estimativas relacionadas à *Big Data* na modelagem

proposta por [Akoka et al. 2017] e a abordagem para otimização do grafo proposta por [De Virgilio et al. 2014].

Tabela 1. Comparação entre os trabalhos relacionados e o trabalho atual

Trabalho	Conceitos do ER	Conceitos do EER	Restrições de integridade	BDs de grafos suportados
(VIRGLIO; MACCIONI; TORLONE, 2014)	Apenas o básico (entidades, relacionamentos e atributos)	X	Não possui	Qualquer um
(AKOKA; COMYN-WATTIAU; PRAT, 2017)	Básico + relacionamentos n-ários	Especializações	Não possui	Qualquer um
(POKORNY, 2017)	Básico + entidades fracas	Especializações	Parcial (algumas restrições são suportadas, a depender dos recursos providos pelo SGBD)	Neo4j, Titan, OrientDB e Stardog
(SOUSA, 2018)	Básico + auto-relacionamentos	X	Parcial, propõe extensões para a linguagem Cypher para reforçar restrições em nível do SGBD	Neo4j
Trabalho atual	Todos	Todos (Especializações e uniões)	Total (em nível do SGBD)	Neo4j

O trabalho de [De Virgilio et al. 2014] realiza o controle das restrições de integridade do dado a nível da aplicação e não do Sistema de Gestão de Banco de Dados (SGBD). Dessa forma, as restrições só são mantidas enquanto as interações com o BD forem realizadas exclusivamente pela ferramenta fornecida por ele. O trabalho de [Akoka et al. 2017] fornece algoritmos para mapear alguns dos conceitos do EER para LPG, entretanto ele não se preocupa em realizar o controle das cardinalidades dos relacionamentos durante a utilização do BD. O trabalho de [Pokorny 2017], por ser muito agnóstico, não fornece restrições de integridade suficientes para que qualquer um dos seus SGBDs alvo respeite todas as restrições da modelagem conceitual realizada. O trabalho de [de Sousa 2018] propõe extensões para a linguagem *Cypher* do *Neo4j* para lidar com as situações que não são suportadas nativamente pelo SGBD. O trabalho, entretanto, não fornece implementações para essas extensões.

Nenhum desses trabalhos fornece uma solução que contemple simultaneamente as funcionalidades de especializações e de uniões do modelo EER e que contenha uma implementação garantindo que o SGBD alvo (nesse caso o *Neo4j*) respeite todas as restrições presentes na modelagem. O trabalho aqui proposto utiliza uma combinação das restrições de integridade de dados disponibilizadas pelo *Neo4j* e da interface de *triggers* fornecidas pelo plugin *Awesome Procedures On Cypher* (APOC) para realizar todo o controle de integridade dos dados a nível do SGBD. Dessa forma, são propostos mapeamentos para todos os conceitos do modelo EER que podem ser executados diretamente em uma instância do *Neo4j* para que ela passe a reforçar as regras da modelagem EER.

2. Ferramenta

A ferramenta aqui proposta é uma aplicação web para projeto de BDs de grafo. O seu enfoque está em permitir que os usuários modelem um BD através de uma modelagem conceitual ER ou EER (normalmente utilizadas para BDs relacionais) e possam gerar a partir dela uma sequência de comandos *Cypher* que construirão o esquema desse BD em uma instância *Neo4j*, incluindo a implementação de restrições de integridade para garantir que os dados se comportarão conforme as restrições impostas pela modelagem. A Figura 2 mostra tela principal da ferramenta.

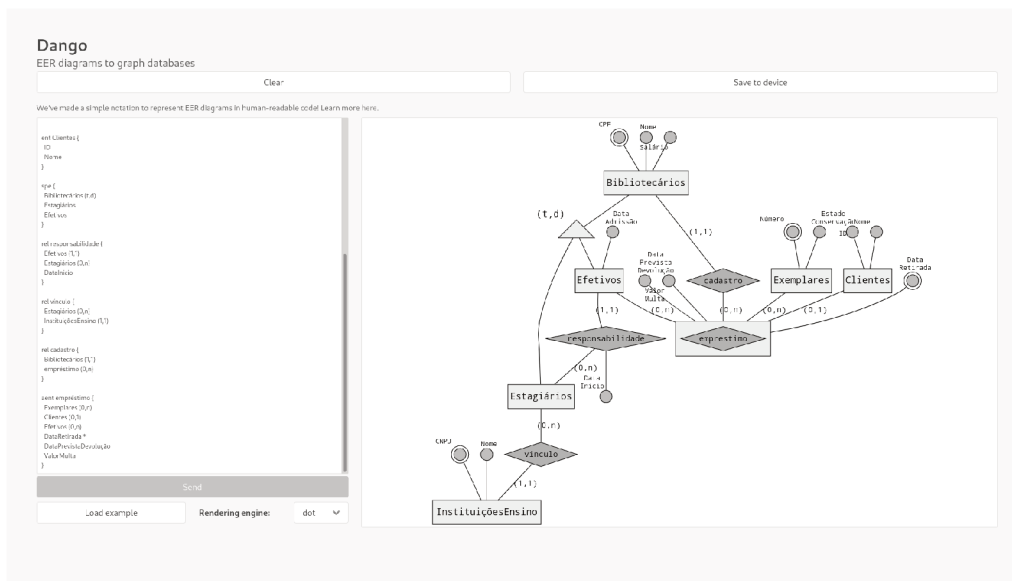


Figura 2. Tela principal da ferramenta

O componente *Modeler* (do lado esquerdo) auxilia na definição de uma modelagem EER em texto. O componente *Visualizer* (do lado direito) gera em tempo real uma visualização gráfica da modelagem EER em texto escrita pelo usuário. O usuário pode clicar no botão “Send” para enviar a modelagem realizada para o módulo *EER to JSON*, que irá retornar uma representação intermediária dessa modelagem no formato JavaScript Object Notation (JSON). Nesse ponto, o usuário terá a opção de chamar o componente *JSON to Cypher* para gerar a sequência de comandos *Cypher* que representará, através de *constraints* e *triggers*, o esquema Neo4j resultante da modelagem EER em texto realizada. Caso queira, poderá também executar os comandos gerados em uma instância local do Neo4j diretamente da interface da ferramenta, utilizando o *Database Driver* do lado do cliente. A interação entre esses módulos pode ser vista na Figura 3, que apresenta a arquitetura da ferramenta.

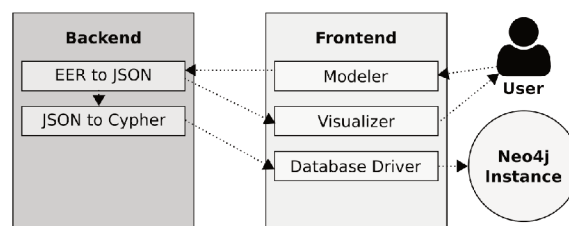


Figura 3. Arquitetura da ferramenta

3. Notação EER em Texto e Regras de Mapeamento

Uma das contribuições desse trabalho é a linguagem “EER em texto”, que permite a representação textual de modelagens EER em uma notação simples, com uma estrutura feita para se assemelhar ao JSON, formato amplamente utilizado em aplicações web na atualidade [Lv et al. 2018]. Essa linguagem foi criada na ausência de uma notação existente capaz de representar todos os conceitos do modelo EER, como definido por [Elmasri and Navathe 2015]. Os blocos de construção básicos da linguagem são

as declarações: “ent” (entidade), “rel” (relacionamento), “aent” (entidade associativa), “spe” (especialização) e “union” (união).

3.1. Entidades e relacionamentos

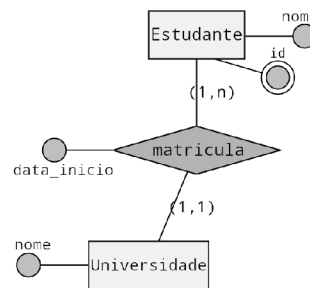
A declaração de uma entidade deve conter o seu nome e seus atributos. Os atributos cujos valores devem ser únicos devem ser seguidos de um “*” (asterisco). A declaração de um relacionamento deve conter o seu nome, o nome das entidades relacionados e suas respectivas cardinalidades, bem como os atributos do relacionamento. A Figura 4 contém um exemplo de declaração de entidades e relacionamentos.

Figura 4. Exemplo de declaração de entidades e relacionamentos

```

1 ent Estudante {
2   nome
3   id *
4 }
5
6 ent Universidade {
7   nome
8 }
9
10 rel matricula {
11   Estudante (1,n)
12   Universidade (1,1)
13   data_inicio
14 }

```



Para garantir que os dados inseridos no BD respeitarão a modelagem realizada, se fazem necessárias as *constraints* e *triggers* presentes no Código 1. Nas linhas 1-5 são definidas as *constraints* que garantem que as propriedades referentes aos atributos da modelagem estarão presentes nos nós e relacionamentos corretos. A linha 3 garante que a propriedade “id” será única entre nós com a etiqueta “Estudante”. Nas linhas 7-9 estão os *triggers* que garantem que o formato do relacionamento será respeitado, de forma que nós com a etiqueta “Estudante” se conectarão a nós com a etiqueta “Universidade” através de relacionamentos do tipo “matricula”. Nas linhas 11-13 é realizado o controle de cardinalidade máxima 1, removendo relacionamentos excedentes caso necessário.

Código 1. Constraints e triggers referentes às entidades e relacionamentos

```

1 CREATE CONSTRAINT ON (e:Estudante) ASSERT exists(e.nome);
2 CREATE CONSTRAINT ON (e:Estudante) ASSERT exists(e.id);
3 CREATE CONSTRAINT ON (e:Estudante) ASSERT (e.id) IS UNIQUE;
4 CREATE CONSTRAINT ON (u:Universidade) ASSERT exists(u.nome);
5 CREATE CONSTRAINT ON ()-[m:matricula]-() ASSERT exists(m.data_inicio);
6
7 MATCH (n)-[r:matricula]-(:Universidade) WHERE NOT "Estudante" IN LABELS(n) DELETE r;
8 MATCH (n)-[r:matricula]-(:Estudante) WHERE NOT "Universidade" IN LABELS(n) DELETE r;
9 MATCH (n)-[r:matricula]-() WHERE NOT "Estudante" IN LABELS(n) AND NOT "Universidade" IN LABELS(n) DELETE r;
10
11 MATCH (n:Estudante)-[r:matricula]-(:Universidade)
12 WITH n, COLLECT(r) AS rs WHERE SIZE(rs) > 1
13 FOREACH (r IN rs[1..] | DELETE r);

```

3.2. Entidades fracas, atributos compostos e atributos multivalorados

Uma entidade fraca, ou seja, uma entidade cuja existência depende da existência de outra entidade (uma entidade forte), pode ser descrita na linguagem proposta com o prefixo “w” antes do seu nome na definição de um relacionamento. Um atributo composto (atributo

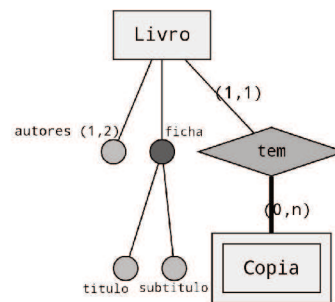
que agrega outros atributos) pode ser definido através da utilização de colchetes na linguagem proposta neste trabalho. Um atributo multivalorado, ou seja, um atributo que contém múltiplos valores, pode ser representado através dos símbolos de menor (“<”) e maior (“>”) na linguagem proposta para representar a sua cardinalidade. A Figura 5 contém um exemplo de modelagem contendo entidades fracas, atributos compostos e atributos multivalorados.

Figura 5. Exemplo de declaração de entidades fracas, atributos compostos e atributos multivalorados

```

1 ent Livro {
2   autores <1,2>
3   ficha [
4     titulo
5     subtitulo
6   ]
7 }
8
9 ent Cópia { }
10
11 rel tem {
12   Livro (1,1)
13   w Cópia (0,n)
14 }

```



Para garantir que os dados inseridos no BD respeitarão a modelagem realizada, se fazem necessárias as *constraints* e *triggers* presentes no Código 2. O *trigger* da linha 1 verifica se a quantidade de valores na propriedade “autores” respeita as cardinalidades (1,2), caso contrário o nó é removido. O *trigger* da linha 2 verifica se existe algum nó com a etiqueta “Cópia” que não esteja conectado a um nó com etiqueta “Livro”, caso houver ele será removido. Os *triggers* para controle do atributo composto foram omitidos, pois funcionam da mesma forma que os *triggers* de relacionamentos quando a cardinalidade é (1,1).

Código 2. Constraints e triggers gerados para entidades fracas e atributos multivalorados

```

1 MATCH (n:Livro) WHERE size(n.autores) < 1 OR size(n.autores) > 2 DETACH DELETE n;
2 MATCH (n:Cópia) WHERE NOT (:Livro)-[:tem]-(n) DETACH DELETE n;

```

3.3. Relacionamentos entre três ou mais entidades

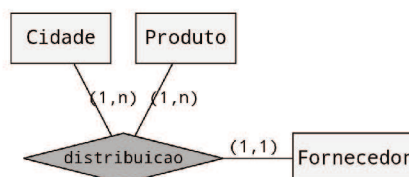
A declaração de um relacionamento entre três ou mais entidades é bastante similar a de um relacionamento simples, bastando adicionar mais entidades na declaração do relacionamento. A Figura 6 contém um exemplo de modelagem contendo um relacionamentos entre três ou mais entidades.

Figura 6. Exemplo de declaração de relacionamento entre três ou mais entidades

```

1 ent Cidade {}
2 ent Fornecedor {}
3 ent Produto {}
4
5 rel distribuicao {
6   Cidade (1,n)
7   Fornecedor (1,1)
8   Produto (1,n)
9 }

```



Para garantir que os dados inseridos no BD respeitarão a modelagem realizada, se fazem necessárias as *constraints* e *triggers* presentes no Código 3. As linhas 1-4 definem os *triggers* que controlam como nós irão se relacionar entre si, como “distribuicao” é um relacionamento entre três ou mais entidades, os nós deverão se conectar a ele através de relacionamentos do tipo “connected_to_distribuicao”.

Código 3. Triggers gerados para relacionamentos n-ários com $n \geq 3$

```

1 MATCH (n)-[r:connected_to_distribuicao]-(:Cidade) WHERE NOT "distribuicao" IN LABELS(n)
  DELETE r;
2 MATCH (n)-[r:connected_to_distribuicao]-(:Fornecedor) WHERE NOT "distribuicao" IN LABELS
  (n) DELETE r;
3 MATCH (n)-[r:connected_to_distribuicao]-(:Produto) WHERE NOT "distribuicao" IN LABELS(n)
  DELETE r;
4 MATCH (n)-[r:connected_to_distribuicao]-(:distribuicao) WHERE NOT n:`Cidade` AND NOT n:`
  Fornecedor` AND NOT n:`Produto` DETACH DELETE r;

```

3.4. Entidades associativas

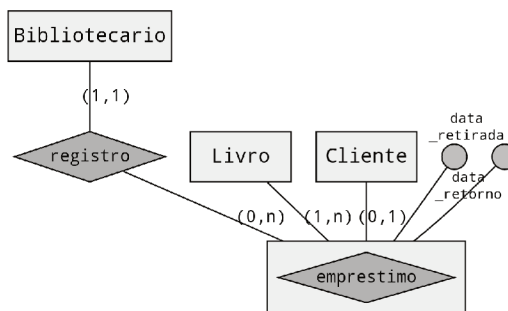
A declaração de uma entidade associativa é muito similar a de um relacionamento: ela deve conter o seu nome, o nome das entidades que a constituem e suas respectivas cardinalidades e seus atributos. Entretanto, a diferença é que, caso alguma das entidades se conecte à entidade associativa através de um relacionamento, deve ser feita uma declaração própria para o relacionamento e o nome da entidade deve ser omitido da lista de entidades da entidade associativa. A Figura 7 contém um exemplo de modelagem contendo uma entidade associativa.

Figura 7. Exemplo de declaração de entidades e relacionamentos

```

1 ent Bibliotecario { }
2 ent Livro { }
3 ent Cliente { }
4
5 aent emprestimo {
6   Livro (1,n)
7   Cliente (0,1)
8   data_retirada
9   data_retorno
10 }
11
12 rel registro {
13   Bibliotecario (1,1)
14   emprestimo (0,n)
15 }

```



Para garantir que os dados inseridos no BD respeitarão a modelagem realizada, se fazem necessárias as *constraints* e *triggers* presentes no Código 4. As linhas 1-3 definem os *triggers* que controlam como os nós irão se relacionar entre si, como “emprestimo” é uma entidade associativa, os nós participantes dessa associação deverão se conectar a ela através de relacionamentos do tipo “associated_to_emprestimo”.

Código 4. Triggers gerados para entidades associativas

```

1 MATCH (n)-[r:associated_to_emprestimo]-(:Livro) WHERE NOT "emprestimo" IN LABELS(n)
  DELETE r;
2 MATCH (n)-[r:associated_to_emprestimo]-(:Cliente) WHERE NOT "emprestimo" IN LABELS(n)
  DELETE r;
3 MATCH (n)-[r:associated_to_emprestimo]-(:emprestimo) WHERE NOT n:`Livro` AND NOT n:`
  Cliente` DETACH DELETE r;

```

3.5. Especializações e generalizações

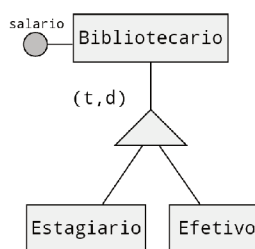
A declaração de uma especialização deve incluir a entidade mãe seguida das características da especialização (total ou parcial, compartilhada ou disjunta). Nas linhas seguintes as entidades filhas devem ser listadas. A Figura 8 contém um exemplo de modelagem contendo especializações e generalizações.

Figura 8. Exemplo de modelagem contendo especializações e generalizações

```

1 ent Bibliotecario {
2   salario
3 }
4
5 ent Estagiario {}
6
7 ent Efetivo {}
8
9 spe {
10  Bibliotecario (t,d)
11  Estagiario
12  Efetivo
13 }

```



Para garantir que os dados inseridos no BD respeitarão a modelagem realizada, se fazem necessárias as *constraints* e *triggers* presentes no Código 5. O *trigger* da linha 1 garante que não haverão nós com etiquetas da entidade especializada sem ter também a etiqueta da entidade generalizada. O *trigger* da linha 2 garante a propriedade de *disjointness*, de forma que um nó não pode carregar mais de uma etiqueta de entidade especializada. O *trigger* da linha 3 garante a propriedade de *completeness*, de forma que não serão permitidos nós que contenham apenas a etiqueta da entidade generalizada sem conter também uma etiqueta de entidade especializada.

Código 5. Triggers gerados para especializações e generalizações

```

1 MATCH (n) WHERE ("Estagiario" IN LABELS(n) OR "Efetivo" IN LABELS(n)) AND NOT "
  Bibliotecario" IN LABELS(n) DETACH DELETE n;
2 MATCH (n) WHERE ("Estagiario" IN LABELS(n) AND "Efetivo" IN LABELS(n)) DETACH DELETE n;
3 MATCH (n:Bibliotecario) WHERE NOT "Estagiario" IN LABELS(n) AND NOT "Efetivo" IN LABELS(
  n) DETACH DELETE n;

```

3.6. Auto-relacionamentos

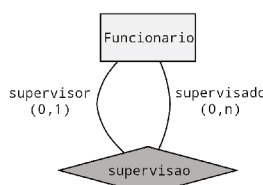
Auto-relacionamentos são relacionamentos que associam instâncias da mesma entidade. Para declará-los, é necessário definir o papel que cada lado do relacionamento representa. Na notação EER em texto, coloca-se o papel logo depois da cardinalidade na declaração do relacionamento. A Figura 9 contém um exemplo de modelagem contendo um auto-relacionamento.

Figura 9. Exemplo de uma modelagem contendo um auto-relacionamento

```

1 ent Funcionario {}
2
3 rel supervisao {
4   Funcionario (0,1) supervisor
5   Funcionario (0,n) supervisado
6 }

```



Para garantir que os dados inseridos no BD respeitarão a modelagem realizada, se fazem necessárias as *constraints* e *triggers* presentes no Código 6. Os *triggers* definidos nas linhas 1-2 controlam o formato dos relacionamentos baseado no papel referente ao respectivo lado do relacionamento.

Código 6. Triggers gerados para auto-relacionamentos

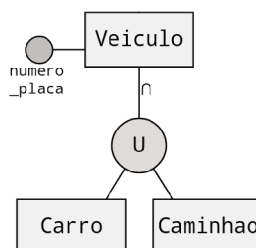
```
1 MATCH (n)-[r:supervisor]-() WHERE NOT "Funcionario" IN LABELS(n) DELETE r;
2 MATCH (n)-[r:superviado]-() WHERE NOT "Funcionario" IN LABELS(n) DELETE r;
```

3.7. Uniões

A declaração de uma união deve incluir o seu nome, o nome das entidades a serem agrupadas e os atributos que serão comuns para todas elas. A Figura 10 contém um exemplo de modelagem contendo uma união.

Figura 10. Exemplo de modelagem contendo uma união

```
1 ent Carro {}
2 ent Caminhao {}
3
4 union Veiculo {
5   Carro
6   Caminhao
7   numero_placa
8 }
```



Para garantir que os dados inseridos no BD respeitarão a modelagem realizada, se fazem necessárias as *constraints* e *triggers* presentes no Código 7. O *trigger* da linha 1 garante que não haverão nós que carreguem apenas a etiqueta da união e o *trigger* da linha 2 garante que todo nó com etiqueta participante da união também terá a etiqueta da união.

Código 7. Triggers gerados para uniões

```
1 MATCH (n:Veiculo) WHERE NOT "Caminhao" IN LABELS(n) AND NOT "Carro" IN LABELS(n) DETACH
  DELETE n;
2 MATCH (n) WHERE NOT n:Veiculo AND (n:Caminhao OR n:Carro) DETACH DELETE n;
```

3.8. Strict mode

O *strict mode* é uma funcionalidade opcional da aplicação (ativa por padrão) que impede que nós com etiquetas e relacionamentos com tipos que não tenham sido especificados na modelagem EER sejam inseridos no BD. Dada a modelagem presente na Figura 4, os *triggers* gerados pelo *strict mode* seriam os presentes no Código 8.

Código 8. Triggers gerados pelo strict mode

```
1 MATCH (n) WHERE NOT "Estudante" IN LABELS(n) AND NOT "Universidade" IN LABELS(n) DETACH
  DELETE n;
2 MATCH ()-[r]-() WHERE TYPE(r) <> "matricula" DETACH DELETE r;
```

4. Conclusão

Este trabalho teve como objetivo principal o desenvolvimento de uma ferramenta que transforma modelagens ER e EER em esquemas, na linguagem Cypher, para o SGBD de grafo Neo4j. Os esquemas fazem uso da funcionalidade de *constraints*, fornecida pelo SGBD, e também da interface de *triggers*, proveniente do plugin APOC, para garantir que os dados inseridos no BD respeitem a modelagem conceitual. O código da aplicação desenvolvida está disponível na íntegra em um repositório público do GitHub¹.

As principais contribuições deste trabalho são: A notação EER em texto, capaz de representar de forma textual todos os conceitos presentes no modelo EER. As regras para o mapeamento de todos os conceitos do modelo EER para LPG. A implementação, através de *triggers* e *constraints*, das restrições de integridade no Neo4j referentes aos conceitos do modelo EER. A ferramenta *Dango*, que recebe modelagens EER em texto como entrada, gera a visualização gráfica, a representação intermediária JSON e por fim o esquema para o BD Neo4j.

Como atividades futuras relacionadas a este trabalho considera-se: A análise do plano de execução dos comandos *Cypher* gerados pela ferramenta a fim de avaliar a performance dos *triggers* gerados e propor otimizações para o controle do esquema do BD. Tentar obter o código fonte dos trabalhos relacionados a fim de realizar uma análise comparativa em termos de desempenho e qualidade dos esquemas gerados para os BDs de grafo.

Referências

- Akoka, J., Comyn-Wattiau, I., and Prat, N. (2017). A four v's design approach of nosql graph databases. In *International Conference on Conceptual Modeling*, pages 58–68. Springer.
- Cattell, R. (2011). Scalable sql and nosql data stores. *Acm Sigmod Record*, 39(4):12–27.
- de Sousa, V. M. (2018). Projeto lógico de banco de dados nosql de grafos a partir de um modelo conceitual baseado no modelo entidade-relacionamento.
- De Virgilio, R., Maccioni, A., and Torlone, R. (2014). Model-driven design of graph databases. In *International Conference on Conceptual Modeling*, pages 172–185. Springer.
- Elmasri, R. and Navathe, S. B. (2015). *Fundamentals of Database Systems*. Pearson Higher Education, 7 edition.
- Lv, T., Yan, P., and He, W. (2018). Survey on json data modelling. In *Journal of Physics: Conference Series (JPCS)*, volume 1069, pages 683–687.
- Pokorny, J. (2017). Modelling of graph databases. *Journal of Advanced Engineering and Computation*, 1(1):04–17.
- Robinson, I., Webber, J., and Eifrem, E. (2015). *Graph Databases: New Opportunities for Connected Data*. O'Reilly Media, Inc., Sebastopol, CA, USA, 2 edition.
- Rodriguez, M. A. and Neubauer, P. (2010). Constructions from dots and lines. *Bulletin of the American Society for Information Science and Technology*, 36(6):35–41.
- West, D. B. (2002). *Introduction to Graph Theory*. Pearson Education (India), 2 edition.

¹<https://github.com/telmotooper/dango>