UNIFACCAMP
Centro Universitário Campo Limpo Paulista

Projeto Lógico de Banco de Dados NoSQL de Grafos a partir de um modelo conceitual baseado no modelo Entidade-Relacionamento

Victor Martins de Sousa

Agosto / 2018

Dissertação de Mestrado em Ciência da Computação

Projeto Lógico de Banco de Dados NoSQL de Grafos a partir de um modelo conceitual baseado no modelo Entidade-Relacionamento

Esse documento corresponde à Dissertação de Mestrado apresentado à Banca Examinadora no curso de Mestrado em Ciência da Computação do Centro Universitário Campo Limpo Paulista.

Campo Limpo Paulista, 03 de agosto de 2018.

Victor Martins de Sousa

Prof. Dr. Luis Mariano del Val Cura (Orientador)

FICHA CATALOGRÁFICA

Dados Internacionais de Catalogação na Publicação (CIP)
Câmara Brasileira do Livro, São Paulo, Brasil.

S698p

Sousa, Victor Martins

Projeto lógico de banco de dados NoSQL de grafos a partir de um modelo conceitual baseado no modelo entidade-relacionamento / Victor Martins Sousa. Campo Limpo Paulista, SP: UNIFACCAMP, 2018.

Orientador: Prof°. Dr. Luis Mariano del Val Cura

Dissertação (Programa de Mestrado Profissional em Ciência da Computação) – Centro Universitário Campo Limpo Paulista – UNIFACCAMP.

1. Banco de dados. 2. Banco de dados de grafos. 3. Projeto de banco de dados. 4. Modelo entidade-relacionamento. I. Del Val Cura, Luis Mariano. II. Centro Universitário Campo Limpo Paulista. III. Título.

CDD-005.75

DEDICATÓRIA

Dedico este trabalho à minha esposa Annayra, pois esteve presente em todos os momentos em que mais precisei e também, sempre foi a minha inspiração para chegar nesta etapa final. À minha mãe Nilza, que mesmo longe sempre mostrou seu amor por mim. Também dedico ao meu pai Gerson e meu irmão Ygor, por sempre estarem ao meu lado.

AGRADECIMENTOS

Agradeço a Deus por me dar saúde e paz nos momentos de dificuldades da minha vida acadêmica e profissional.

Agradeço a minha esposa Annayra, onde deu toda a força que eu precisava, e sempre ajudou direta e indiretamente em toda minha caminhada durante o mestrado.

Agradeço ao meu professor e orientador Luis Mariano, por ter me aceitado como orientando e também, pela dedicação e paciência que teve comigo em todo estudo dirigido e em todo período de orientação.

Agradeço aos colegas do Mestrado, Anderson Oliveira, Bruno Amaral, Vagner Scamati e em especial, ao Bruno Ponsoni, onde estivemos uma parceria em todas as disciplinas, viagens e demais atividades do mestrado.

Agradeço ao IFSP e ao IFC, instituições que apoiaram a minha busca pelo mestrado e forneceram flexibilidade para concluir meus objetivos. Em especial, ao meu amigo Matheus Braga que sanou todas dúvidas possíveis referentes a linguagem Java.

Agradeço a todos colaboradores da UNIFACCAMP, e em especial, aos docentes que tive a oportunidade de conhecer, onde todos deixaram um pouco de inspiração para minha vida acadêmica.

Resumo

Modelos de bancos de dados NoSQL e seus sistemas de gerenciamento tem surgido como soluções para as aplicações Big Data. Uma característica comum a estes modelos e suas aplicações é a flexibilidade para a criação, evolução e representação heterogênea do esquema de dados das entidades de uma mesma classe. Por esta razão, a modelagem de dados nas aplicações NoSQL tem se baseado, fundamentalmente, em recomendações de boas práticas muitas vezes específicas para cada gerenciador. Neste trabalho apresentamos uma proposta de modelagem conceitual e lógica para aplicações baseadas no modelo de dados NoSQL Orientado a Grafos. Para a modelagem conceitual propõe-se um modelo conceitual de dados baseado no modelo ER que permite flexibilidade na definição do esquema de dados. Para a modelagem lógica é definido um modelo lógico de grafos, que considera restrições de integridades mapeadas a partir do projeto conceitual. Como parte do trabalho definem-se regras de mapeamento entre os modelos que descrevem as restrições definidas na modelagem do problema e uma aplicação para ilustrar estas regras. Adicionalmente são propostos comandos para a linguagem Cypher, para que esta suporte todas as restrições de integridade do modelo lógico de grafos deste trabalho. A proposta é validada com um estudo de caso de um banco de dados para aplicação de streaming musical. Uma aplicação foi desenvolvida para automatização do projeto conceitual, mapeamento conceitual-lógico e geração de comandos Cypher.

Palavras-chave: *Banco de Dados NoSQL, Banco de Dados de Grafos, Projeto de Banco de Dados, Modelo Entidade-Relacionamento.*

Abstract

NoSQL database models and data management systems emerged as solutions of the new Big Data applications. A common feature of those models and their applications is the flexibility for creation, evolution and heterogeneous representation of the data schema of entities of the same class. Traditional data and conceptual data modeling based on the entity relationship and relational models, in general, promote data schemas that are rigid and common to all entities of the same class. For this reason, data modeling in NoSQL applications has been based, fundamentally, on recommendations of best practices often specific to each data store system. In this works we present a proposal of conceptual and logical modeling applications based on NoSQL graph data model. For conceptual modeling it is proposed an extension to the entity-relationship model (EER) that allows flexibility and heterogeneous representations in the schema descriptions. For the logical modeling a graph database is defined, a set vertex and constraints related to conceptual modelling cardinalities. A set of algorithms was proposed to mapping conceptual and logical modeling. Also, we propose rules as a set of integrity constraint for the Cypher lamguage. The proposal is validated with a case study of a database for musical streaming application. An application was developed for conceptual design automation, conceptual-logical mapping, and Cypher command generation.

Keywords: *NoSQL Databases, Graph Databases, Database Modeling, Entity-Relationship Model.*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BD	Banco de Dados
BDG	Banco de Dados de Grafos
ER	Entidade-Relacionamento
EER	Entidade-Relacionamento Estendido
NoSQL	*Not only* SQL
NoAM	NoSQL *Abstract Model*
SGBD	Sistema de Gerenciamento de Banco de Dados
SQL	Structured Query Language
UML	Unified Modeling Language
XML	Extended Markup Language

SUMÁRIO

1 INTRODUÇÃO

As necessidades de armazenamento e processamento de dados na WEB são cada vez maiores e diversas. Fatores que contribuem consideravelmente para estas necessidades são: aplicações *Big Data*, redes sociais, o uso de *Cloud Computing* em diferentes aplicações, entre outros. Em particular, as aplicações *Big Data* têm sido caracterizadas por possuir um grande volume de informação a ser processado, utilizando uma taxa de velocidade muito alta e com uma grande diversidade de formatos e estruturas. Em virtude disso, novas tecnologias têm sido introduzidas para o armazenamento de dados de forma distribuída em múltiplos servidores. A escalabilidade no crescimento do número de servidores de um banco de dados, segundo a demanda, tornou-se uma necessidade das aplicações, adicionando a isto outros fatores como questões de redundância, segurança, entre outros. De modo geral, os sistemas de Banco de Dados (BD) Relacionais (também chamados de “Tradicionais”) não têm oferecido um bom desempenho utilizando o rígido modelo de armazenamento, através de tabelas (Stonebraker, 2012). Como uma solução surgiram os chamados Sistemas de Bancos de Dados NoSQL (*Not only* SQL), que oferecem uma gerência simplificada para grandes volumes de dados baseados em modelos de dados mais simples e suportando também a escalabilidade horizontal, que normalmente é a mais utilizada em *cloud computing*. Os modelos de BDs NoSQL têm sido classificados em quatro tipos (Bugiotti *et al.*, 2014), (Souza *et al*, 2014). Os modelos de BDs NoSQL são:

- **Chave-Valor:** Neste modelo o banco de dados é organizado como uma grande tabela *hash*, com uma estrutura baseada em um conjunto de chaves, sendo que cada chave referencia um valor. A estrutura do modelo é considerado simples e de fácil implementação (Tiwari, 2011), (Souza *et al.*, 2014).
- **Orientado a Colunas:** Neste modelo, também chamado de Colunar, o banco de dados é organizado por colunas. Cada uma destas colunas possui três propriedades: nome, valor e informação de tempo. Os dados do mesmo tipo podem ser agrupados formando famílias de colunas (Chebotko, *et al.* 2015), Poffo *et al.*, 2016), (Sadalage and Fowler, 2013).
- **Orientado a Documentos:** Neste modelo, um banco de dados consiste de uma coleção de documentos, onde cada documento armazena campos e valores, que

também podem ser considerados conjunto de pares de chave-valor. Cada valor pode ser do tipo texto simples ou então, um documento JSON, XML, dentre outros (Lima *et al.*, 2015), (Kaur and Rani, 2013).

- **Orientado a Grafos**: Neste modelo os dados são armazenados em vértices de um grafo. E existindo algum tipo de relação entre os dados, os relacionamentos entre os vértices são representados como arestas. Quando se tratam de grafos da categoria NoSQL, normalmente considera-se um de grafo de propriedades, que permite incorporar inúmeras informações em vértices e também nas arestas (Ghrab, *et al.*, 2013), (Pokorný, 2016).

Um projeto de Banco de Dados deve oferecer consistência, escalabilidade e eficiência para obter o melhor desempenho em sua utilização. Um projeto possui três etapas: I. projeto conceitual, onde os dados do problema são modelados em uma forma abstrata utilizando um modelo conceitual de dados que pode ser compreendido pelo usuário final; II. projeto lógico, onde os dados são modelados em um modelo lógico de dados especifico como, por exemplo, o modelo relacional; e por fim, III a implementação do Banco de Dados em um sistema de gerenciamento específico (Elmasri and Navathe, 2011).

A etapa de projeto conceitual tem sido desenvolvida utilizando o modelo Entidade-Relacionamento (ER) ou ainda, com o surgimento das metodologias orientadas a objetos, os projetos têm sido modelados na linguagem Unified Modeling Language (UML)(Silberschatz et al., 2012). Existem metodologias bem estabelecidas para o mapeamento destes projetos conceituais em projetos lógicos para os modelos relacional, orientado a objetos e semiestruturado baseado em documentos XML (*H*euser, 2008; Necaský, 2006). De modo geral, em todos estes modelos existe uma definição do esquema dos dados, isto é, da estrutura e atributos que cada entidade possui.

Poucos trabalhos abordam projeto de bancos de dados quando se deseja a utilização de modelos de dados NoSQL. Em particular, Bugiotti *et al.* (2014) afirma que não existem trabalhos que abordem os modelos lógicos de BDs NoSQL. Assim, não existe um modelo formal considerado na bibliografia para cada um dos tipos de BDs NoSQL.

Uma característica comum aos modelos de dados NoSQL é a flexibilidade na definição e evolução do esquema dos bancos de dados. De modo geral, um banco de

dados NoSQL pode ser criado sem um esquema de dados predefinido. Adicionalmente, cada entidade de um mesmo tipo ou classe pode ter esquemas diferentes e ainda estes esquemas pode evoluir dinamicamente. Estas características dos modelos de dados NoSQL poderiam explicar o uso limitado das abordagens tradicionais de projetos de bancos de dados que consideram, de modo geral, esquemas de dados rígidos e predefinidos (Sadalage and Fowler, 2013).

Em contrapartida, Okman *et al.* (2011) afirmam que alguns requisitos podem ser necessários em soluções NoSQL, como a consistência dos dados, porém, a verificação de restrições de integridade ainda não está avançada. Izquierdo *et al.* (2013) salientam que em situações de integração de dados é importante conhecer, pelo menos parcialmente, os elementos do banco de dados, ou seja, uma noção do esquema mínimo do BD. Daniel *et al.* (2016) afirmam que um esquema explícito, pode contribuir consideravelmente na integração sistemática de aplicações NoSQL e também na evolução e na manipulação de seus respectivos dados.

Considerando as metodologias consolidadas em projetos de BD de tradicionais, salientadas por Elmasri and Navathe (2011), reconhece-se que estas metodologias podem ser experimentadas em soluções NoSQL. Porém, Bugiotti *et al.* (2014) afirmam que projetos de BDs NoSQL têm utilizado, fundamentalmente, recomendações de boas práticas, muitas vezes associadas a cada sistema específico. Os trabalhos Banerjee *et al.* (2015), Bugiotti *et al.* (2014), Feng *et al.* (2015), Kaur & Rani (2013) e Daniel *et al.* (2016) abordam metodologias para projeto lógico de BDs NoSQL a partir de esquemas conceituais, baseados em algum modelo conceitual utilizado para projetos de banco de dados tradicionais ou então, propostas de modelos conceituais. Recentemente, alguns trabalhos têm introduzido metodologias para o mapeamento de projetos conceituais baseados no modelo ER para projetos lógicos baseados em modelos de dados NoSQL. Poffo *et al.* (2016) e Lima *et al.* (2015) apresentam projetos lógicos de bancos de dados NoSQL orientado a documentos e orientado a colunas, respectivamente. As duas propostas utilizam esquemas conceituais do modelo Entidade Relacionamento Estendido (EER). Pokorný (2016), por sua vez, considera um modelo ER binário para modelar projetos mapeados em bancos de dados orientados a grafos.

Este trabalho parte da premissa de que toda aplicação de bancos de dados possui um esquema mínimo que reflete aspectos da sua semântica e um conjunto básico de restrições de integridade. Além dos trechos citados anteriormente acerca da importância

de um esquema mínimo ou dos benefícios de um esquema de banco de dados. Schwartz (2015) afirma que “não existe um banco de dados sem esquema”. No entanto, pode-se considerar que as novas aplicações possuem requisitos que exigem flexibilidade na definição do esquema das entidades, que o modelo ER não permite adequadamente.

Esta dissertação apresenta uma proposta de modelagem lógica para Bancos de Dados NoSQL de Grafos. O modelo conceitual é baseado no modelo Entidade-Relacionamento, denominado *Extended Binary Entity-Relationship* (EB-ER), que inclui alguns recursos orientados às aplicações baseadas em grafos. O modelo lógico de grafos proposto é baseado em trabalhos que ofertam modelagem para BDGs, que são apresentados no Capítulo 4, com adição de novos recursos e principalmente, se atentando às restrições de integridade. O projeto lógico é realizado com algoritmos de alto nível que promovem mapeamento nível de esquema conceitual EB-ER em esquema lógico de grafos. Na tentativa de explicitar o esquema em nível de banco de dados, são apresentados comandos de uma linguagem de um sistema de BD de grafos (Neo4j[1]), onde estes comandos suportam parte das restrições de integridade ofertadas no nível lógico e, adicionalmente, são propostos comandos para suportar as restrições ainda não suportadas pelo Neo4j. Adicionalmente, apresenta-se uma aplicação que mapeia um esquema EB-ER para esquema lógico de grafos, seguindo os conceitos apresentados nesta dissertação. A aplicação também possibilita a criação de um conjunto de comandos Neo4J, juntamente aos comandos propostos,

1.1 Objetivos

O objetivo geral deste trabalho é

- Oferecer projeto lógico para banco de dados NoSQL de grafos a partir de um modelo conceitual baseado no Entidade-Relacionamento.

Enquanto os objetivos específicos são:

- Definir um modelo lógico de grafos para representação de banco de dados NoSQL deste tipo;
- Projetar o modelo lógico a partir de um modelo conceitual baseado no modelo entidade-relacionamento;

1 https://neo4j.com/

- Apresentar algoritmos de alto nível para mapeamento entre os níveis conceitual e lógico;
- Oferecer suporte a restrições de integridade para flexibilização na oferta de consistência de um banco de dados de grafos;
- Apresentar propostas de comandos para um sistema de banco de dados de grafos para suportar todas as restrições de integridades apresentadas no esquema lógico;
- Desenvolver aplicação para automatizar o mapeamento de um esquema conceitual para um esquema lógico de grafos.

1.2 Método de pesquisa

Este trabalho teve o método de pesquisa inspirado nos trabalhos de Poffo *et al.* (2016) e Lima *et al.* (2015), que realizam projeto lógico de um modelo NoSQL a partir de esquemas conceituais de um modelo utilizado para projetos de BD tradicionais. A escolha pelo modelo orientado a grafos foi definida a partir da publicação de Pokorný (2016), que se verificou que a oferta de restrições de integridade poderia ser aperfeiçoada e também, o mapeamento entre os níveis conceitual e lógico não foi tão detalhado quanto nos trabalhos de Poffo *et al.* (2016) e Lima *et al.* (2015).

Foi realizada uma revisão da bibliografia acerca de projetos de banco de dados NoSQL. A revisão foi realizada através de engenhos de busca como IEEE Xplorer, ACM Digital Library, CiteSeerX e Google Scholar. As palavras-chave utilizadas foram: ("*logical design*", "NoSQL *database design*", “*graph database model*”, “*graph database schema*” e "NoSQL *schema design*"). Livros conceituados na área de sistemas e projetos de banco de dados foram de grande contribuição. Nesta etapa da pesquisa, diversas variações foram encontradas como por exemplo, a utilização de esquemas conceituais consolidados ou então, a proposta de novos modelos conceituais.

1.3 Contribuições esperadas

Com esse trabalho espera-se contribuir na área de projeto de Banco de Dados NoSQL, e especificamente nos assuntos elencados abaixo:

- Proposta de novos recursos para modelos conceituais de dados, como a oferta de atributos alternativos e a utilização de um papel dominante em relacionamentos.
- Apresentação de um modelo lógico de grafos que suporte todas restrições de integridade oriundas de um esquema conceitual de dados.
- Desenvolvimento de uma ferramenta para realizar o mapeamento de um esquema conceitual para um esquema lógico de grafos. Poffo *et al.* (2016) e Lima *et al.* (2015) sugerem como trabalhos futuros, o desenvolvimento de uma ferramenta para automatizar o projeto lógico de suas respectivas propostas.
- Reflexão a respeito da importância de um esquema mínimo em uma solução NoSQL (neste limitando-se ao modelo de grafos). O esquema mínimo não limita a existência de outros elementos em uma aplicação/base de dados. O intuito é a definição de necessidades específicas do projetista ou de determinada aplicação/integração.

1.4 Organização do trabalho

O trabalho está organizado da seguinte maneira:

- No Capítulo 1 foram introduzidos diversos conceitos a respeito do tema desta pesquisa.
- O Capítulo 2 apresenta um referencial teórico acerca de Projeto Conceitual e Lógico de Banco de Dados. Na contextualização são abordados projetos e modelo de dados para BDs tradicionais, do tipo relacional. Também são abordados os Bancos de Dados desenvolvidos para aplicações atuais, chamados de NoSQL.
- O Capítulo 3 apresenta conceitos de grafos e de sistemas de bancos de dados de grafos. A primeira parte apresenta definições e teorias de Grafos, que servem como base para criação dos sistemas, e seus modelos de dados.
- O Capítulo 4 apresenta uma revisão e comparação de trabalhos que propõem modelos lógicos de grafos e seus mapeamentos a partir de um modelo conceitual. Nesta seção são abordados três trabalhos, primeiramente são apresentados seus respectivos modelos lógicos e uma breve comparação entre os modelos. Em

seguida, são apresentados os projetos lógicos e também, uma comparação entre os mesmos.

- No Capítulo 5 é apresentada a proposta da dissertação, a realização de projeto lógico de banco de dados NoSQL de Grafos a partir de um modelo baseado no ER. O modelo conceitual, denominado de EB-ER, possui adição de novos recursos. O modelo lógico de grafos é baseado nos modelos lógicos abordados no Capítulo 4, com adição de restrições de integridade. Para o mapeamento entre os níveis, são propostos algoritmos de alto nível. Também são apresentados comandos de uma linguagem de um sistema de BD de grafos (Neo4j), onde estes comandos suportam parte das restrições de integridade ofertadas no nível lógico de grafos deste trabalho e, adicionalmente, são propostos comandos para o sistema Neo4j para as restrições não suportadas.
- O Capítulo 6 apresenta a aplicação desenvolvida. O intuito da aplicação é de automatizar o mapeamento de um esquema conceitual EB-ER para um esquema lógico de grafos. A aplicação também oferta a geração de comandos e propostas de comandos do sistema Neo4j a partir das restrições do nível lógico do banco de dados.
- O Capítulo 7 apresenta um estudo de caso que aborda uma plataforma de *streaming* musical. O cenário é representando no modelo conceitual EB-ER, e posteriormente, é realizado o projeto lógico utilizando as definições e regras das propostas na presente dissertação. Finaliza-se apresentando um conjunto de comandos para o sistema de banco de dados de grafos.
- O Capítulo 8 apresenta as considerações finais desta dissertação, junto às contribuições identificadas e trabalhos futuros para o tema desta pesquisa.

2 PROJETO CONCEITUAL E LÓGICO EM BANCO DE DADOS

Um banco de dados pode ser criado diretamente no Sistema Gerenciador de Banco de Dados (SGBD), quando se tem conhecimento de todas as necessidades da aplicação ou da organização. Normalmente esse método não é indicado quando se trata de uma aplicação ou um cenário organizacional que possui informações de média ou alta complexidade e também quando possui um grande volume de informações. O cenário anterior pode ocasionar o armazenamento de informações redundantes em um banco de dados, especialmente quando acontecem alterações nos dados. Esses fatores podem promover inconsistências, causando futuros transtornos e até prejuízos financeiros.

Um banco de dados considerado seguro e consistente pode ser construído utilizando um projeto de banco de dados bem elaborado. O projeto de um BD é necessário para alcançar eficiência, escalabilidade e outras características que permitem obter o melhor desempenho na sua utilização (Elmasri and Navathe, 2011).

Neste capítulo serão tratados os principais conceitos relacionados com projetos de banco de dados e a utilização de modelos de dados em diversos tipos de banco de dados. Na sequência é destacada a importância do ciclo de vida de um projeto de banco de dados.

2.1 Ciclo de vida de um projeto de banco de dados.

O projeto de um BD deve atender as necessidades da organização e requer atenção em diversos aspectos que influenciam nas escolhas de projetos nos níveis conceitual, lógico e físico, níveis que serão detalhados neste capítulo. O projetista de banco de dados interage com os usuários da aplicação que utilizarão o banco de dados, sendo necessária uma maneira de representar os dados e suas relações que possa ser entendida pelos usuários. Esta forma de representação deve poder ser mapeada nos modelos de dados e linguagens de definição de bancos de dados (Silberschatz *et al*, 2012).

A fase inicial do ciclo de um projeto baseia-se na identificação das necessidades da aplicação que utilizará o banco de dados. O resultado dessa fase deve representar todos os requisitos e restrições da aplicação ou organização. Após levantamento dos

requisitos, será iniciado o projeto de banco de dados que se divide em três fases e será detalhado a seguir.

2.2 Projeto conceitual, lógico e físico

A partir do levantamento de informações da aplicação que poderão ser armazenadas pelo BD, aplicam-se essas informações para a elaboração de um modelo de dados conceitual. O modelo conceitual é um modelo de dados abstrato que representa um banco de dados independente do modelo de dados do SGBD que será utilizado. Modelos conceituais possuem maior clareza para os usuários que não detém um conhecimento considerável na área de Tecnologia da Informação (Heuser, 2008).

A segunda fase do ciclo de vida, chamado de projeto lógico consiste no mapeamento do modelo conceitual para o modelo de dados em que será implementado o banco de dados.

O modelo lógico de dados, decorrente do projeto lógico, possui as características da estrutura do banco de dados físico, ou do tipo de SGBD que será utilizado. Nota-se que diferente do modelo conceitual, o modelo lógico possui uma dependência da implementação que será realizada, ou então, o modelo lógico define como será o banco de dados (Heuser, 2008).

Na terceira e última fase do projeto, denominada de projeto físico, o banco de dados é construído fisicamente, a partir das estruturas e restrições do modelo lógico. Nesta fase, os recursos físicos são especificados, definindo a organização e a estrutura interna do banco de dados (Silberschatz *et al*, 2012).

2.3 Projeto conceitual em bancos de dados tradicionais com o Modelo-Entidade Relacionamento.

Um projeto de bancos de dados relacionais utiliza o modelo Entidade-Relacionamento (ER) para sua modelagem conceitual (Heuser, 2008). Este modelo conceitual de dados foi proposto por Chen (1976) e atualmente é muito utilizado por profissionais e acadêmicos da área de banco de dados. A seguir serão detalhados os principais conceitos do ER.

2.3.1 Entidade

Um conjunto de elementos com a mesma característica é representado por uma entidade. No ER, a entidade, agrupa informações dos elementos. Uma entidade é representada graficamente por retângulo (Angelotti, 2010).

2.3.2 Relacionamento

O relacionamento é utilizado para representar uma relação entre entidades. Na representação gráfica de um relacionamento utiliza-se um losango (Angelotti, 2010).

Segundo Elmasri and Navathe (2011) cada entidade possui um papel específico em um relacionamento. O próprio nome do papel apresenta o que o mesmo executa no relacionamento. De modo geral, os papéis não são muito utilizados em relacionamentos em que as entidades participantes são distintas, uma vez que cada nome de tipo entidade participante pode ser usado como o nome do papel.

Um relacionamento recursivo representa associações entre elementos de uma mesma entidade (Angelotti, 2010).

2.3.3 Entidade Fraca

Uma entidade é denominada de entidade fraca quando um elemento da mesma só existirá se houver um relacionamento com um elemento de uma outra entidade (Silberschatz *et al*, 2012). Um exemplo é o cadastro de funcionários de uma organização, para os quais também são armazenados os dados de seus respectivos cônjuges, se houver. É evidente que só existirá um cônjuge se houver um funcionário para se relacionar, então, pode-se considerar que cônjuge é uma entidade fraca.

2.3.4 Atributos

Os atributos representam uma ou mais propriedades definidas para uma entidade, sendo que cada elemento da entidade possuirá os atributos definidos para essa entidade. Além das entidades, os relacionamentos podem receber atributos, também chamados de atributos descritivos, que nesse caso são propriedades particulares do relacionamento em si (Silberschatz *et al*, 2012). Um atributo é representado graficamente por um círculo

ligado a entidade, ou relacionamento, e sua respectiva identificação. A seguir, os tipos de atributos:

- **Atributo Simples:** É um atributo indivisível, que não pode ser decomposto (Angelotti, 2010). Exemplo “CPF”.
- **Atributo Composto:** Este tipo de atributo permite a decomposição em outros atributos simples (Angelotti, 2010). Um exemplo de o atributo composto é “Endereço”, onde pode ser decomposto em atributos simples “Rua”, “Número”, “Complemento” e “Bairro”.
- **Atributo multivalorado:** Consiste em um atributo multivalorado, um atributo que permite o armazenamento de mais de um valor no mesmo campo (Heuser, *2008*; Angelotti, 2010). Um exemplo de atributo multivalorado é “Email”, onde um elemento pode possuir vários endereços de e-mail, a identificação é realizada através da cardinalidade, (0,N), a posição “0” define a ocorrência mínima, já a posição “N” a ocorrência máxima. A cardinalidade de atributos é derivada da cardinalidade de relacionamentos, e normalmente, quando se tratam de atributos simples (também chamados de monovalorados), a representação no digrama conceitual é omitida (Heuser, 2008). Embora este atributo possua uma característica interessante, Heuser (2008) e Angelotti (2010) não indicam a utilização do mesmo, pois no nível lógico não possui implementação direta.

2.3.5 Restrições de integridade

Para obter um banco de dados íntegro, onde seus respectivos dados apresentem realmente o que é representado pelos modelos conceituais pelo BD e principalmente, oferecendo consistência entre si, faz-se necessária a utilização de restrições de integridade. Essas são consideradas regras que podem oferecer consistência aos dados de um BD (Heuser, 2008). Para Silberchatz *et al.* (2012), “um esquema ER de uma empresa pode definir certas restrições às quais o conteúdo de um banco de dados precisa se conformar”. Na sequência, são apresentadas algumas restrições para o modelo ER.

2.3.5.1 Cardinalidade

A cardinalidade é utilizada para representar a quantidade de associações de um elemento de uma entidade com outros elementos de outra entidade, através de um relacionamento. As associações entre os elementos podem ser 0, 1 ou vários (representado por N). A representação de uma cardinalidade é realizada através da representação do número mínimo e o máximo de associações, ou então, apenas o número máximo de associações para cada entidade em um relacionamento. Predominantemente utiliza-se apenas a associação máxima. Então, a máxima sempre apresentará 1 ou N, separados por um “x” (Angelotti, 2010). A representação máxima proporciona três tipos de cardinalidade.

Tabela 1: Tipos de cardinalidade

Tipo de cardinalidade	Características
1 x 1	Um elemento de uma entidade é capaz de se relacionar com apenas um elemento da outra entidade, e o contrário também.
1 x N	Um elemento de uma entidade suporta o relacionamento com vários elementos da outra entidade, já o contrário não é possível, um elemento da outra entidade pode se relacionar apenas uma vez.
N x N	Um elemento de uma entidade suporta o relacionamento com vários elementos da outra entidade, e o contrário também é possível.

Também pode-se considerar o tipo de cardinalidade N x 1, que consiste o oposto de 1 x N, ou seja, apenas invertendo os lados (Silberschatz *et al.*, 2012).

2.3.5.1 Chave primária

Um atributo ou um conjunto de atributos de uma entidade pode ser definido como chave primária da mesma. Esta restrição oferecerá o recurso de identificar

unicamente um elemento de uma entidade. Para isto, o valor de uma chave primária não poderá repetir e também não poderá ser nulo (Angelotti, 2010).

2.3.5.2 Participação

A participação de uma entidade em um relacionamento pode ser de dois tipos: parcial ou total. O primeiro tipo determina que apenas alguns elementos de uma entidade participam de um relacionamento. Já o segundo tipo define que todos os elementos de uma entidade devem participar de um relacionamento (Silberschatz *et al.*, 2012).

2.3.6 Diagrama Entidade-Relacionamento

Um diagrama ER é a representação gráfica de um esquema ER (Angelotti, 2010). Na Figura 1 é apresentado um diagrama ER para ilustrar os componentes do modelo conceitual ER apresentados anteriormente.

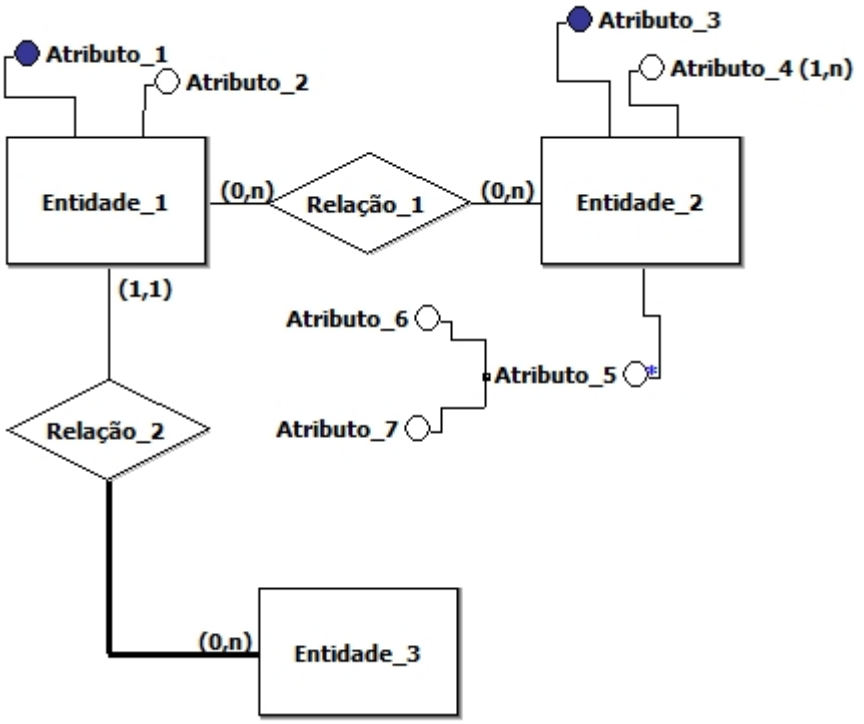

Figura 1: Diagrama Entidade-Relacionamento com elementos ER

Cada entidade é representada por um retângulo. Já um relacionamento é representado por um losango, ligando as entidades envolvidas na associação (Exemplos:

“Relação_1” e “Relação_2”). A “Entidade_3” é do tipo fraca, diferenciando graficamente através de uma linha mais densa ligando esta entidade a um relacionamento. “Entidade_1” e “Entidade_2” são entidades (ou entidades forte). “Atributo_1” e “Atributo_3” são atributos definido com a restrição de chave primária de suas respectivas entidades. “Atributo_2” é um atributo simples, onde sua cardinalidade máxima é 1 (não apresentada). Diferente deste, “Atributo_4” possui cardinalidade máxima “N” (muitos), em que caracteriza um atributo multivalorado. “Atributo_5” é um atributo do tipo composto, onde o mesmo possui dois atributos simples (ou atributos elementos): “Atributo_6” e “Atributo_7”. O relacionamento “Relação_2” possui cardinalidade mínima “1” e máxima “1” para “Entidade_1” e cardinalidade mínima “0” e máxima “M” para “Entidade_3”. O relacionamento “Relação_1” possui cardinalidade mínima “0” e máxima “N” para “Entidade_1” e cardinalidade mínima “0” e máxima “N” para “Entidade_2”.

2.3.7 Entidade-Relacionamento Estendido

A complexidade de diversas aplicações fizeram surgir novas propostas para que o modelo ER. Alguns conceitos foram incorporados no ER, assim surgindo uma extensão chamada de Entidade-Relacionamento Estendido ou *Extended Entity-Relationship* (EER) (Elmasri and Navathe, 2011), (Silberschatz *et al*, 2012).

- **Especialização:** Conforme Angelotti (2010), “Especialização consiste na subdivisão de uma entidade mais genérica em um conjunto de entidades especializadas”. A utilização deste conceito permite especificar ainda mais o esquema do banco de dados, pois uma entidade pode conter subgrupos de entidades, em que esses subgrupos possuam atributos e/ou relacionamentos específicos. Um exemplo de especialização pode ser considerado em: uma entidade “Pessoa”, como entidade genérica, e as entidades “Aluno” e “Professor”, como entidades especializadas.
- **Generalização:** Pode-se considerar que a generalização possui o processo inverso da especialização, pois as diferenças entre diversas entidades são suprimidas, e as características comuns são generalizadas em uma superclasse (Elmasri and Navathe, 2011). Um exemplo de generalização pode ser duas

entidades “Carro” e “Caminhão”, onde uma entidade genérica poder definida como “Veículo”;

- **Agregação:** Uma agregação é utilizada quando é necessário realizar associação de elementos de um relacionamento com elementos de outro relacionamento (Elmasri and Navathe, 2011). Um exemplo é um relacionamento “Compra” que associa as entidades “Cliente” e “Produto”. Outra entidade “Prestação” é utilizada na aplicação, porém esta entidade não possui relação a um cliente ou a um produto, e sim a quando uma existe uma compra. Neste caso o relacionamento “Compra” é um exemplo de agregação.

2.4 Projeto lógico em bancos de dados tradicionais.

Nesta seção serão abordados os recursos utilizados para o projeto lógico de bancos de dados tradicionais, do tipo relacional, a partir do modelo ER.

O modelo relacional consiste em um conjunto de tabelas. Cada tabela armazena informações de um tipo de elemento/objeto. Cada elemento de uma tabela é representado por uma linha. Cada coluna de uma tabela representa uma propriedade específica daquele tipo de elemento. Cada coluna pode exigir um valor (tipo obrigatória) ou permitir a ausência de valor (tipo opcional). Cada tabela pode possuir uma chave primária, composta por uma ou mais colunas, que identifica unicamente cada linha da tabela. As relações entre as tabelas são realizadas através de chaves estrangeiras, onde uma coluna de uma tabela referencia uma outra tabela, normalmente, pela chave primária da tabela referenciada, ou então, uma chave única (coluna onde não é permitida a repetição de valores) (Silberschatz *et al.*, 2012).

Um esquema conceitual é apresentado através de um diagrama ER com elementos utilizando identificações aleatórias. Este diagrama é apresentado na Figura 2, onde utiliza diversos componentes para possibilitar uma visão geral.

Neste diagrama, são abordados: entidades, atributos simples representados por círculos, e chaves primárias (também chamados de atributos identificadores) representadas por círculos preenchidos com uma cor mais escura, relacionamentos binários (possuem duas entidades), relacionamentos ternários (possuem três entidades) e cardinalidade dos relacionamentos.

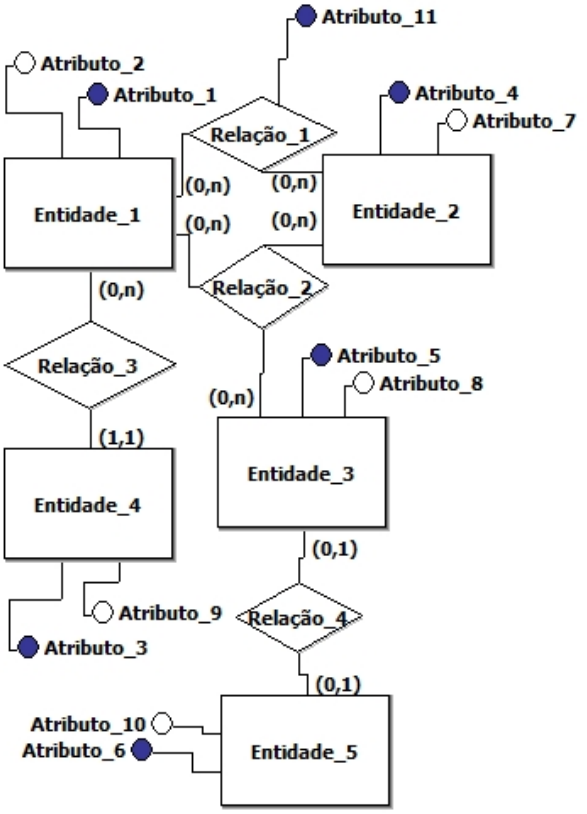

Figura 2: Diagrama Entidade-Relacionamento

O diagrama ER da Figura 2 foi convertido para diagrama relacional, utilizando o modelo lógico, apresentado na Figura 3.

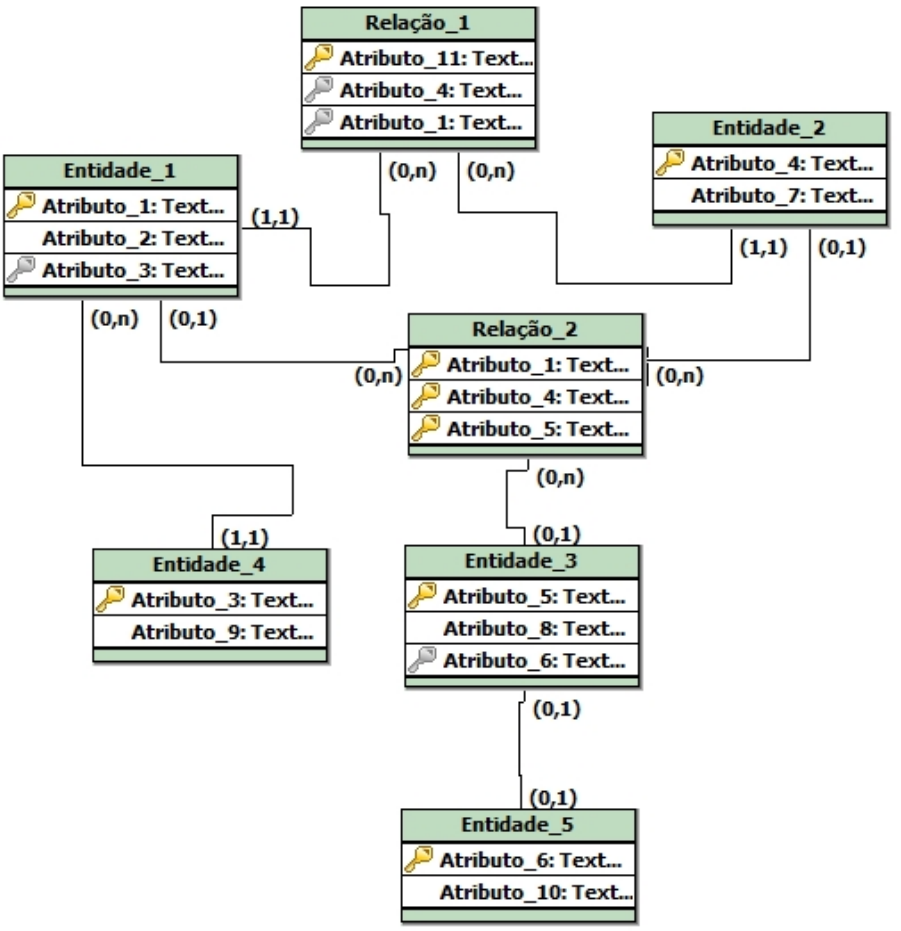

Figura 3: Diagrama Relacional gerado a partir do DER da Figura 2

A seguir, é apresentada uma breve explicação do mapeamento utilizando conceitos de Heuser (2008), detalhando pontos considerados fundamentais:

- **Entidade:** Geralmente uma entidade se tornará uma tabela no modelo lógico relacional, com exceção de alguns casos, onde sugere-se realizar uma união de duas entidades para formar uma única tabela. Esse caso acontecerá em um relacionamento específico de cardinalidade 1 x 1, que será explicado posteriormente.
- **Relacionamento 1 x 1:** Ambas entidades se tornam tabelas, se as participações em ambas das partes forem opcionais (o mínimo for 0, exemplo (0,1)). Um exemplo são as “Entidade 3” e “Entidade 5” na Figura 2. Os outros dois possíveis casos serão: considerando a cardinalidade mínima definida ‘1’, as duas entidades formam uma única tabela, herdando os atributos das duas entidades e no último caso, em um relacionamento com uma associação é obrigatória e a

com a outra opcional, as duas entidades se tornam tabelas, porém é adicionado uma chave estrangeira na tabela que participava como opcional, referenciando a chave primária da tabela que participava como obrigatória.

- **Relacionamento 1 x N:** Ambas entidades se tornam tabelas, porém a entidade no lado N recebe uma chave estrangeira, e esta referencia a chave primária do lado 1. Esse relacionamento é mostrado na Figura 2, “Entidade 1” e “Entidade 4” e depois seu resultado é ilustrado na Figura 3.
- **Relacionamento N x N:** Ambas entidades e o relacionamento se tornam tabelas. A tabela criada a partir do relacionamento recebe duas chaves estrangeiras, uma de cada tabela, que inicialmente eram entidades. Um exemplo que pode ser considerado é apresentado na Figura 2, onde “Relação_1” relaciona as entidades “Entidade 1” e “Entidade 2”.
- **Relacionamento ternário:** Nesse caso, as três entidades e o relacionamento se tornam tabelas. A tabela resultante do relacionamento recebe três chaves estrangeiras, cada uma delas referenciando a chave primária correspondente a tabela que no modelo conceitual era entidade.
- **Relacionamento n-ário:** O conceito utilizado no relacionamento ternário também é utilizado em relacionamentos com quatro entidades ou mais. Sendo assim, todas entidades e o relacionamento tornam-se tabelas.
- **Entidade fraca:** Para esta, é criada uma tabela com todos os atributos da entidade fraca e uma chave estrangeira com qual se relaciona com a tabela gerada da entidade com que se relacionava no modelo conceitual. A chave estrangeira fará parte da chave primária da tabela criada pela entidade fraca.
- **Atributo (atributo simples):** Os atributos são definidos como atributos nas tabelas, que também são chamados de colunas. Para atributos descritivos, definidos em relacionamentos, são gerados em uma tabela, gerada pelo relacionamento, ou então, em uma das tabelas geradas pelas entidades envolvidas.
- **Atributo multivalorado:** Para cada atributo multivalorado cria-se uma nova tabela, onde terá o atributo simples correspondente e uma chave primária que

será uma chave estrangeira referenciando a tabela criada através da entidade que possuía o atributo multivalorado.

- **Atributo composto:** O atributo composto definido em uma entidade é decomposto em atributos simples na tabela mapeamento.
- **Especialização/generalização:** Existem três opções, sendo a primeira criar uma tabela para cada entidade, onde o atributo identificador da tabela criada através da entidade generalizada torna-se chave primária e estrangeira nas tabelas criadas através das entidades especializadas. Outra opção, é criar uma tabela para cada entidade especializada, onde cada tabela receberá também os atributos da entidade generalizada. Uma terceira opção, que diminui a quantidade de tabelas geradas e dados redundantes, é a criação de uma tabela única com todos atributos, da entidade genérica e das entidades especializadas, com a adição de um atributo tipo (para identificar a qual especialização o registro pertence). Por outro lado, esta última opção aumenta, consideravelmente, a possibilidade de existirem campos nulos.

2.5 Restrições de Integridade em Banco de Dados Relacionais

Em Heuser (2008) são pontuadas as principais restrições de integridade em BDs Relacionais:

- **domínio:** integridade que restringe um valor de um campo a um tipo de dado (ex: inteiro, real, binário, entre outras);
- **vazio:** integridade que determina a obrigatoriedade ou não de um valor em um campo de uma coluna. Essa integridade define uma coluna obrigatória ou opcional;
- **chave:** integridade que determina que um ou mais valores identificam um registro em uma tabela, também chamada de chave primária;
- **referencial:** integridade que atua em uma chave estrangeira, na obrigatoriedade da existência de uma chave primária na coluna da tabela referenciada.

Na seção 3.3 são abordadas as restrições de integridade definidas para um Banco de Dados Orientados a Grafos, categoria de banco de dados NoSQL utilizada nesta dissertação.

2.6 Modelagem em banco de dados XML

A Extended Markup Language (XML) é uma linguagem que possibilita a formatação e organização de dados em um documento com definições estruturadas, ou semiestruturadas. Essas definições resultam em arquivos/documentos; compatíveis para utilização na Internet e protocolos, legíveis pelas pessoas e que possuem independência das plataformas utilizadas na troca de informações (Almeida, 2012). Necaský (2006), define requisitos de um modelo conceitual para XML. Dentre estes, alguns podem ser considerados para bancos de dados NoSQL, estes requisitos são:

- **Representações formais**: Estas possibilitam a construção de operações sobre a estrutura do modelo e seus respectivos dados. Também proporcionam que um modelo possa ser comparado com outros modelos conceituais;
- **Notação gráfica**: Este recurso oferta uma maneira mais intuitiva na modelagem;
- **Mapeamento para o nível lógico**: Considerar algoritmos, ou regras, para a modelagem lógica e suportar as restrições de integridades definidas em esquemas conceituais;
- **Cardinalidade para todos os participantes**: deve-se especificar as restrições de cardinalidades nas participações entre elementos das entidades envolvidas em um relacionamento;
- **Atributos em relacionamentos**: Recurso encontrado em diversos modelos conceituais;
- **Dados centrados em documentos**: Permitir que relacionamentos e atributos sejam representados dentro de um documento. Essa característica pode ser utilizada durante o mapeamento, assim realizando junções de entidades quando necessárias ou mais adequadas conforme o projeto lógico considerado.
- **Reutilização de conteúdo**: Baseado em definições de heranças ou “Is_a” utilizados no modelo EER.

Outros recursos que podem ser considerados interessantes para um modelo conceitual são descritos em outros trabalhos, que serão citados a seguir. Em Badia (2002), os conceitos de atributos opcionais e obrigatórios são apresentados para um ER estendido. Cada atributo é marcado como opcional ou obrigatório.

Um recurso definido por Dobbie *et al.* (2000), na introdução do Modelo ORA-SS (*Object Relationship-Attribute Model for Semistructured Data*), é a disjunção entre dois ou mais atributos. Assim, é permitido modelar uma estrutura irregular para atributos.

2.7 Modelagem em Banco de Dados NoSQL: trabalhos correlatos

Utilizando representações conceituais, o modelo de dados pode representar entidades do mundo real e as relações entre as mesmas. Um modelo de dados consiste em três componentes: tipos de estrutura de dados; operadores ou regras de inferência e regras de integridade (Angles, 2012).

O trabalho de Bugiotti *et al.* (2014) apresenta uma proposta para projeto de BDs NoSQL. Na modelagem lógica, utiliza-se um modelo de dados abstrato, chamado de Model Abstract NoSQL (NoAM). O NoAM fornece uma representação intermediária para os BDs NoSQL, independente do sistema, com a utilização das semelhanças e abstração das diferenças em cada modelo NoSQL. O projeto completo de um BD NoSQL é separado e ordenado nas seguintes atividades: modelagem conceitual, identificar entidades e relacionamentos; projeto agregado, agrupar entidades relacionadas em agregados; particionamento agregado, dividir agregados em elementos menores; projeto do BD NoSQL em alto nível, mapear uma representação intermediária através da utilização do NoAM; e por fim, a implementação. Nesta última fase o trabalho realiza exemplos de implementações em BDs NoSQL Orientado a Chave – Valor, Orientado a Documentos e Orientado a Colunas.

Os trabalhos de Poffo *et al.* (2016) e Lima *et al.* (2015) propõem trabalhos de modelagem lógica de bancos de dados NoSQL a partir de esquemas Entidade e Relacionamento Estendido ou (EER). Poffo *et al.* (2016), propõe um projeto lógico para BDs NoSQL Orientado a Colunas, fornecendo notações lógicas e diagramáticas para o modelo lógico. No trabalho, é fornecido um conjunto de algoritmos para realizar o processo de mapeamento do modelo conceitual para o modelo lógico. O trabalho apresenta comparações de sua metodologia mostrando melhores resultados quando comparado à proposta de Bugiotti *et al.* (2014). Já Lima *et al.* (2015) propõe um projeto lógico para BD NoSQL Orientados a Documentos, em que se considera a carga de trabalho esperada na utilização da aplicação e apresentando um modelo lógico de Documento NoSQL. O trabalho apresenta um conjunto de algoritmos para realizar o

mapeamento de EER para o modelo lógico. O trabalho mostra um melhor desempenho da proposta de mapeamento quando comparado a um projeto convencional, em que não se considera a carga de trabalho esperada.

Banerjee *et al.* (2015) apresenta um projeto de BD NoSQL Orientado a Documentos. O modelo lógico considerado é representado com a utilização do JSON (*Java Script Object Notation*). Já o modelo conceitual é realizado através da proposta, realizada pelo autor, de um modelo chamado de GOOSSDM (Graph Object Oriented Semi-Structured Data Model).

Em Kaur and Rani (2013) apresentam-se a modelagem e a sintaxe de consulta de dados para BD NoSQL Orientados a Grafos e Orientados a Documentos. A modelagem é realizada de maneira única para cada tipo de BD: Diagrama de Classe é utilizado em Orientados a Documentos e uma ferramenta específica é utilizada em Orientado a Grafos. As sintaxes de consulta são apresentadas posteriormente, entre os dois modelos NoSQL, e também para BD Relacional.

Feng *et al.* (2015) propõem um projeto de BDs NoSQL Orientado a Colunas a partir de diagramas de classe UML. No processo de modelagem utiliza-se um metamodelo para o nível conceitual e outro para o modelo Orientado a Colunas. A proposta possui um sistema para geração de anotações, que trazem informações extras sobre o esquema gerado.

Daniel *et al.* (2016) propõe uma estrutura UMLtoGraphDB, onde são mapeados esquemas conceituais UML em banco de dados de grafos. O trabalho pontua duas contribuições: a) geração de consultas para verificação de restrições *Object Constraint Language* (OCL) definidas através do esquema; b) proposta de um metamodelo GraphDB, onde o mesmo possui uma representação para facilitar a integração com vários tipos de BDGs.

Os BDs NoSQL ofertam uma infraestrutura com flexibilidade e "schemaless", onde a estrutura do esquema é implicitamente definida através dos dados armazenados. Porém em algumas situações que necessitam realizar algum tipo de integração de dados, é importante conhecer, ou pelo menos, conhecer parcialmente, os elementos conceituais do banco de dados (Daniel *et al.*, 2016), (Izquierdo *et al.*, 2013). A ausência de um esquema explícito de banco de dados torna-se complexa a integração sistemática de dados de aplicações *Big Data* que utilizam BDs NoSQL, e também dificulta o

processamento, consulta, evolução e outros tipos de manipulação de dados (Wischenbart *et al.*, 2012).

Além dos benefícios das soluções NoSQL, alguns requisitos podem ser necessários, como a consistência dos dados. Pois não existem mecanismos avançados que ofereçam verificação de restrições de integridade (Daniel *et al.*, 2016),(Okman *et al.*, 2011).

Nesta seção foram abordados alguns trabalhos de projeto lógico para BDs NoSQL. Os trabalhos apresentam um modelo lógico de dados e na maioria das situações, consideram a modelagem lógica a partir de um modelo conceitual de dados, como os modelos ER, EER ou UML.

2.8 Conclusões do capítulo

Neste capítulo foram apresentados diversos conceitos em projetos de banco de dados e modelos de dados. Abordou-se a modelagem conceitual e lógica para BDs tradicionais, onde amplamente se utiliza o modelo ER para o modelo conceitual de dados e, obviamente, o modelo relacional como modelo lógico de dados. Também foram apresentadas algumas propostas para modelagem de BDs NoSQL. Foi destacada a importância das restrições de integridade para bancos de dados. Adicionalmente foi apresentado um trabalho onde são listados requisitos para modelagem em XML, considerados relevantes para o tema da presente pesquisa.

3 MODELAGEM EM BANCOS DE DADOS ORIENTADOS A GRAFOS

Os Bancos de Dados Orientados a Grafos, ou Bancos de Dados de Grafos (BDG), possuem base formal baseada na definição matemática de grafo. Os modelos de grafos utilizados pelos sistemas de BDG podem variar conforme o tipo de grafo considerado (Pokorný, 2016). Na seção a seguir serão abordadas as principais definições de grafos.

3.1 Definições básicas de Grafos

Um grafo **G (V, E)** consiste num conjunto finito não vazio de vértices **V**, também chamados de vértices, e um conjunto **E** de arestas, sendo que **E ⊆ V x V** define uma relação entre os vértices. O **tamanho** de um grafo, representado como **|E|**, é definido pela quantidade de arestas. A **ordem** de um grafo, representado como **|V|**, é definida pela quantidade de vértices. O número de arestas relacionadas com um vértice é chamado de **grau** do vértice (Erven, 2015). Arestas que não distinguem uma direção no relacionamento entre dois vértices são denominadas de simétricas (ou bidirecionais).

Um grafo pode ser representado graficamente com círculos ou pontos, para representar os vértices, e linhas para representação de arestas. Um exemplo de grafo é ilustrado de maneira gráfica na Figura 4.

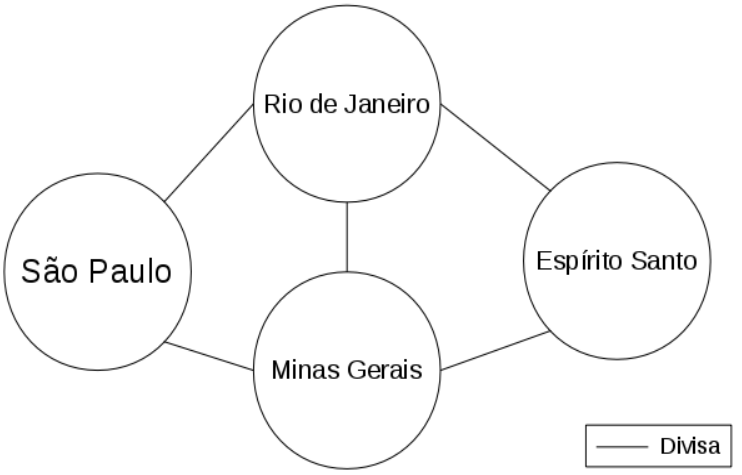

Figura 4: Exemplo de Grafo

Na Figura 4, os vértices representam estados brasileiros, que são ligadas por arestas que representam a existência de uma divisa entre dois estados, ou relação entre os vértices.

Um grafo que diferencia o sentido da relação entre dois vértices pode ser chamado dígrafo, ou também de grafo direcionado (Erven, 2015). Neste grafo, as arestas apresentam uma direção, isto é, distinguem o vértice de início e o vértice de fim de cada aresta.

Um exemplo de grafo direcionado é ilustrado na Figura 5, onde a aresta identificada como “Leciona” tem como vértice início “Professor X” e como vértice fim “Teoria dos Grafos”, representando que o “Professor X” leciona a disciplina “Teoria dos Grafos”.

Figura 5: Exemplo de grafo direcionado

Outro tipo de grafo é o denominado de multigrafo. O mesmo permite a existência de mais de uma aresta para um mesmo par de vértices (Chartrand, 2006). Outro exemplo de grafo é ilustrado na Figura 6, nesse caso um multigrafo direcionado. Nela, o vértice “Professor Y” possui duas relações com o vértice “Banco de Dados”. Através das relações é possível verificar que em dois períodos diferentes, 2013 e 2014, o professor lecionou a mesma disciplina.

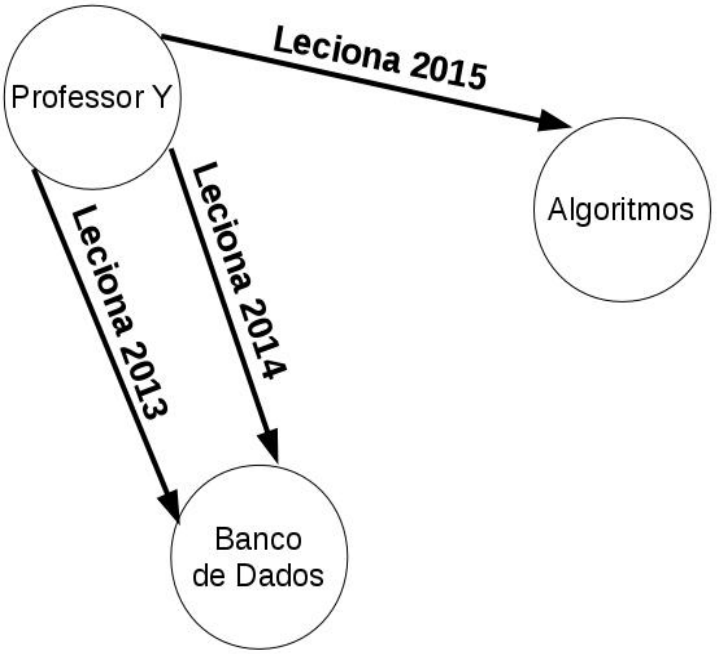

Figura 6: Exemplo de grafo multigrafo

Um grafo de propriedades é um multigrafo, dirigido e rotulado. Sua característica de multigrafo, mencionada anteriormente, possibilita mais de uma aresta para o mesmo par de vértices. A aresta direcionada identifica o sentido de uma aresta, ou seja, existe um vértice de início e outro de fim (comentado anteriormente). A definição de grafo rotulado permite que sejam atribuídos rótulos a vértices e arestas, para que seja possível identificar elementos comuns no grafo. Vértices e arestas podem ter propriedades que permitem lhe incorporar informações. Cada um destes elementos pode possuir pares de chave-valor (Ghrab *et al.*, 2016). A Figura 7 apresenta um exemplo de grafo de propriedades.

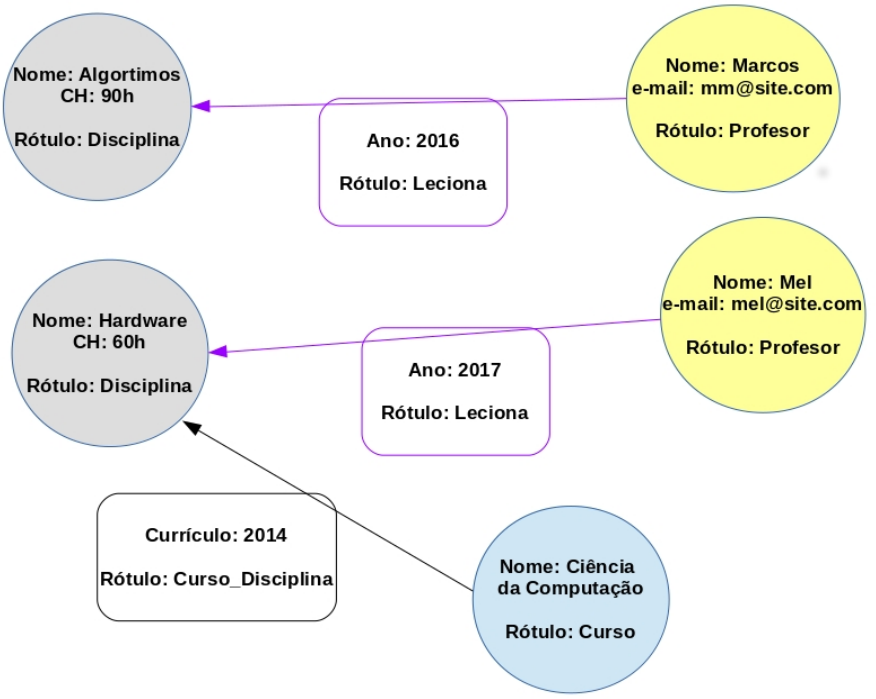

Figura 7: Grafo de propriedades

O grafo apresentou cinco vértices, agrupados em três rótulos. Três arestas direcionadas foram apresentadas, agrupadas em dois rótulos. Todos elementos possuem propriedades.

3.2 Sistemas de Banco de Dados de Grafos

A maioria dos modelos de dados NoSQL e seus sistemas de gerenciamento foram introduzidos nos últimos anos e suas tecnologias são recentes. No caso dos Bancos de Dados de Grafos (BDGs), eles surgiram a partir dos anos 80 (Wood, 2012), mas foram introduzidos novamente como parte do movimento NoSQL por ser uma excelente alternativa para modelar diversas aplicações atuais (Penteado *et al.*, 2014).

Atualmente existem diversos sistemas de BDGs de diferentes desenvolvedores. Conforme DB-engines[2], os principais exemplos de sistemas são: Neo4j[3], Microsoft Azure Cosmos DB[4], OrientDB[5] e Titan[6].

---

2 https://db-engines.com/en/article/Graph+DBMS

3 https://neo4j.com/

Uma característica atual dos BDG é a ausência de um único modelo de dados padrão como existe no modelo relacional. Desta maneira, diferentes sistemas de BDG utilizam diferentes definições do modelo de dados de grafo.

Todos os sistemas de BDGs implementam os principais conceitos de grafos, porém existem algumas características que diferenciam os sistemas BDGs entre si (Erven, 2015).

Segundo Angles (2012) *apud* Erven (2015) a estrutura de um banco de dados de grafos é definida a partir dos tipos de grafos, vértices e arestas considerados. Os tipos de grafos que têm sido utilizados são descritos a seguir:

- **Grafos Simples:** Nesta abordagem , o grafo é definido por vértices e arestas que apenas armazenam uma informação ou um rótulo;
- **Hipergrafo:** Nesta abordagem um grafo suporta arestas que relacionam dois ou mais vértices;
- **Grafo com atributos:** Quando comparada com o grafo simples esta estrutura possui recursos adicionais, que permitem o armazenamento de atributos em um vértice ou uma aresta;
- **Grafo aninhado e hipervértices:** Nesta abordagem um vértice pode também ser um grafo, que pode se relacionar com uma outro vértice.

De modo geral existem dois tipos de vértices nas abordagens:

- **Vértice rotulado:** O vértice armazena apenas um rótulo ou uma informação específica como um identificador;
- **Vértice com atributos:** O vértice armazena diversas propriedades.

Um grafo pode suportar vértices com as duas caracteristicas, ou seja, um vértice pode possuir rótulo e atributo(s).

Os tipos de arestas são semelhantes aos vértices:

---

4 https://azure.microsoft.com/pt-br/services/cosmos-db/

5 https://orientdb.com/

6 http://*titan.thinkaurelius.com*

- **Aresta rotulada:** A aresta armazena unicamente um rótulo, que representa uma informação ou é utilizado para a identificação do relacionamento entre os vértices associados à aresta.
- **Aresta com atributos:** A aresta pode armazenar vários atributos com informações do relacionamento entre os vértices.

Um grafo pode suportar arestas com as duas caracteristicas, ou seja, uma aresta pode possuir um rótulo e atributo(s).

Atualmente, não existe uma definição de modelo lógico de Banco de Dados de Grafos amplamente utilizada, porém são encontradas diversas definições apresentadas na bibliografia.

3.3 Restrições de integridade em Sistemas de Banco de Dados de Grafos

As restrições de integridade são regras que podem definir a consistência para um banco de dados. Pokorný (2016) cita três tipos de restrições:

- restrições inerentes em BDs de Grafos (a) e Relacionais (b): a. Um rótulo de vértice é único e arestas são compostas por rótulos e possui os vértices que cada uma relaciona; b. Nenhuma linha possuirá um valor nulo em uma chave primária e não devem existir chaves estrangeiras sem a existência de suas referências;
- Uma restrição explícita é definida através de uma linguagem de definição de dados;
- uma restrição implícita é uma "consequência lógica entre restrições inerentes e explícitas".

Angles (2012) apresenta uma comparação de diferentes modelos de BDGs. Nessa comparação afirma-se que ainda não existem muitas pesquisas em restrições de integridade para BDG. O autor ainda lista algumas restrições:

- verificação de tipo, para testar a consistência de uma instância em relação as definições do esquema;
- identificação de vértice ou aresta, verificar se uma instância pode ser identificada através de um valor;
- integridade referencial, certificar que uma referência é realizada em uma entidade/instância existente;

- verificação de cardinalidade, onde verifica a singularidade de relacionamentos ou propriedades;
- dependência funcional, onde um elemento do grafo pode determinar o valor de outro;
- restrições de padrão de grafo, verificação de uma restrição estrutural, como por exemplo, uma restrição de caminho.

Recentemente alguns trabalhos têm abordado restrições de integridade para BDGs. Em Rabuzin *et al.* (2017) são apresentadas as principais linguagens de consulta e definição de dados para BDGs, e é proposta uma extensão para adição de uma restrição de dados únicos para a linguagem Cypher[7] do sistema de BDG Neo4j. Sestak *et al.* (2016) salientam que a linguagem Cypher apresenta algumas desvantagens nas declarações de restrições, como exemplo, coloca a impossibilidade de definir várias restrições simultaneamente em um único comando. Os autores ainda propõem a definição de um tipo de valor para um atributo, uma vez que o Cypher atualmente não oferta. Pokorný *et al.* (2017) se concentra na discussão dos recursos do Neo4j para a definição de restrições de integridade e esquema.

3.3.1 Esquemas

Além das restrições de integridade, o conceito de esquema é bastante discutido quando se fala em consistência de banco de dados. Angles (2012) afirma que o não suporte a um esquema de banco de dados, não pode ser uma justificativa para a "inexistência de restrições de integridades" pois a consistência de um BD é tão importante ou até mais que um esquema flexível. Um esquema evolutivo pode suportar estruturas flexíveis, como, por exemplo, um atributo ou relacionamento opcional, que permitiria uma escolha presente ou futura sobre a existência desse atributo. Pokorný (2016) apresenta um esquema conceitual de banco de dados de grafos e também destaca sua importância. O autor ainda revisa restrições em sistemas de BDGs, onde menciona que o sistema OrientDB[8] fornece três funções: esquema completo, sem esquema e esquema híbrido (oferece uma mescla entre os dois primeiros).

7 https://neo4j.com/developer/cypher-query-language

8 www.orientdb.com/orientdb/

Schwartz (2015) argumenta que a ausência de um esquema em um BD é uma falácia. O autor argumenta que o esquema sempre está no código. Poffo *et al.* (2016) afirmam um esquema que é apresentado pelo BD ou então, a consistência dos dados fica a cargo da aplicação.

3.4 Conclusões do capítulo

O capítulo apresentou as principais definições de teoria de grafos, onde foram descritos diversos tipos de grafos. Estas definições são motivadoras para o surgimento dos sistemas de banco de dados de grafos. Ainda nesse capítulo, foram apresentados diversos tipos de modelos de dados de grafos, a partir destes, baseiam-se diferentes tipos de sistemas de banco de dados de grafos. O capítulo finalizou com a abordagem de restrições de integridade em sistemas BDGs e utilização de esquema em banco de dados, onde a importância destes conceitos é destacada em diversos trabalhos.

4 MODELOS LÓGICOS E PROJETOS LÓGICOS DE BANCO DE DADOS ORIENTADOS A GRAFOS

Neste capítulo serão introduzidos alguns trabalhos que propõem uma representação lógica de um modelo lógicos de dados de grafos. Também serão apresentados os respectivos modelos conceituais e mapeamentos utilizados.

Um modelo de dados pode garantir a representação completa e precisa de todos os objetos de um Banco de Dados. Kaur and Rani (2013, p. 3) afirmam que “Um bom modelo de dados constrói uma ponte entre os componentes do projeto de banco de dados e os dados do mundo real”.

A exposição dos trabalhos será consideravelmente detalhada, ao ponto de expor definições, representações gráficas e algumas características específicas de cada proposta. Decidiu-se apresentar os trabalhos de maneira separada, para melhor organização.

O capítulo finaliza-se com uma comparação entre as propostas, em que os “quesitos” escolhidos são específicos para contribuição no presente trabalho.

A seguir serão apresentados formalmente três modelos lógicos de grafos apresentados na literatura.

4.1 Banco de Dados Pokorný (2016)

Um BD de Grafos **G = (V, E, N, Ɛ, φ, A, Att)** é um grafo dirigido, rotulado, multigrafo e com atributos.

- **V** é um conjunto finito de vértices com rótulos desenhados de um alfabeto infinito **N**;
- **E** é um conjunto de arestas;
- **φ** é uma função incidente do mapeamento **E** para **V x V**;
- Os rótulos de arestas são retirados do conjunto finito de símbolos **Ɛ**, e **λ** é e rotulagem de aresta mapeamento de função **E** para **Ɛ**;
- **A** é um conjunto de atributos representados por pares **(A_i, $valor_i$)**;

- **Att** é um mapeamento atribuindo a cado vértice / aresta de um subconjunto (possivelmente vazio) de atributos de **A**.

Um exemplo de projeto lógico utilizando este modelo de BDG é apresentado utilizando um ambiente acadêmico. A seguir, um exemplo utilizado no trabalho: são utilizadas as entidades **Língua**, **Professor** e **Cidade**, bem como os relacionamentos entre as entidades **Ensina** e **Nascido_Em**. O primeiro relacionamento satisfaz as entidades **Professor** e **Língua** e enquanto o segundo relaciona **Professor** e **Cidade**.

Uma ilustração das instâncias do BDG para o ambiente acadêmico é apresentada na Figura 8. O trabalho salienta que, para melhor clareza das instâncias, os possíveis atributos são omitidos.

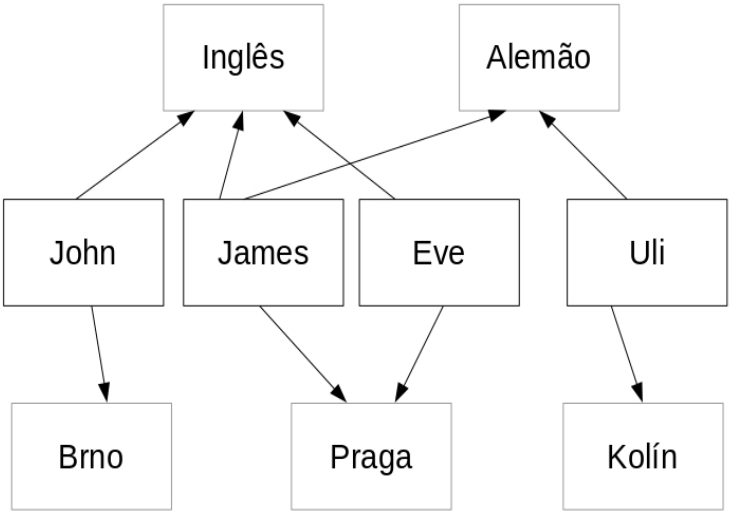

Figura 8: Banco de Dados de Grafos (Pokorný, 2016)

Para representar as instâncias através de um esquema, é utilizado o rótulo de cada tipo de entidade, e também utiliza-se o rótulo para ilustrar os relacionamentos. O esquema de banco de dados de grafos é ilustrado na Figura 9.

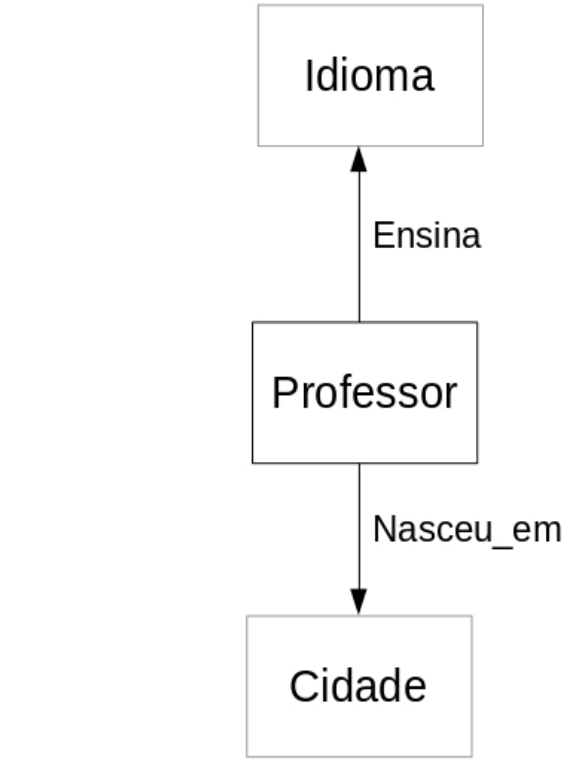

Figura 9: Esquema de Banco de Dados de Grafos (Pokorný, 2016)

4.1.1 Restrições de Integridade

Um professor pode ensinar mais de uma língua e também que um professor nasceu em apenas uma cidade. Esses dois exemplos de situações podem ser definidas utilizando restrições de integridade. Porém, o trabalho relata que as restrições de integridades devem ser definidas no nível conceitual do BD.

4.1.1.1 Abordagens formais para restrições de integridade

Uma Dependência Funcional (DF) é uma restrição de integridade natural e muito útil, porém em BDGs propõe-se uma abordagem exclusiva. Na maioria das vezes, uma aresta orientada não indica uma DF. Um exemplo de DF é ilustrado na Figura 10.

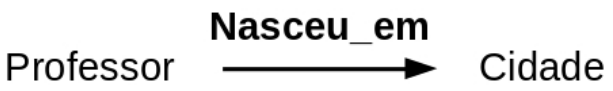

Figura 10: Exemplo de dependência funcional (Pokorný, 2016)

Um exemplo dependência no domínio acadêmico é ilustrado na Figura 11.

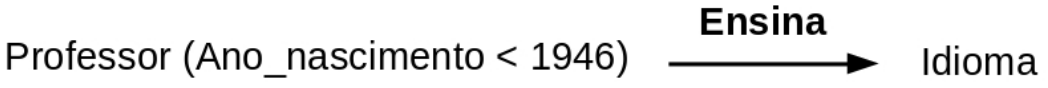

Figura 11: Exemplo de dependência funcional no estudo de caso abordado no modelo (Pokorný, 2016)

Nesse caso, os professores de mais de 70 anos ensinam apenas uma língua. O trabalho destaca a nomenclatura de dependências funcionais condicionais, que podem ser apresentadas por expressões. Um exemplo de DF representada formalmente é ilustrada na Figura 12.

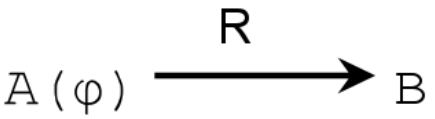

Figura 12: Dependência funcional representada formalmente (Pokorný, 2016)

No trabalho, uma cardinalidade pode ser representada como: **(E1: (a, b), E2: (c, d))**, onde **a,c ∈ {0,1}**, **b,d ∈ {1, N}**, e N onde significa que qualquer número maior que 1.

O trabalho ainda salienta que as Restrições de Integridade (RIs) podem não ser satisfeitas por BDGs, e que na prática, a base de dados pode ser inconsistente. Uma RI para um vértice e seus vizinhos é relativamente simples, porém uma restrição estrutural no BDG pode ser complexa.

4.2 Banco de Dados Virgilio *et al. (*2014)

No trabalho de Virgilio *et al. (*2014) um BDG é definido como um multigrafo dirigido **G = (N, E)**, onde

- Cada vértice **n ∈ N**
- Cada aresta **e ∈ E**

- Cada **n** ou **e** estão associados com um conjunto de par (chave, valor), chamados de propriedades

Um exemplo de BDG considerando a definição anterior é apresentado na Figura 13, que utiliza um cenário para um BD de blogs. Neste domínio, os usuários podem atuar como administradores ou seguidores dos blogs.

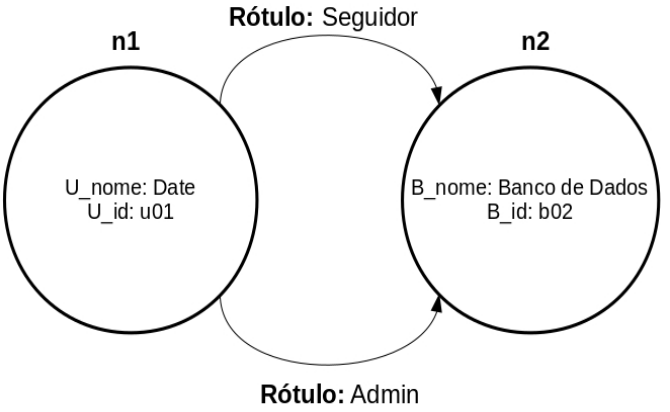

Figura 13: Exemplo de grafo de propriedades (Virgilio *et al.*, 2014)

No grafo da figura são apresentados dois vértices: *n1* é um usuário e *n2* trata-se de um blog. Os dois vértices possuem duas arestas, representando dois relacionamentos entre o usuário *n1* e o *n2*. Analisando os rótulos das arestas, nota-se que o usuário é seguidor e administrador do blog. Outro recurso utilizado no exemplo é o conceito de atributos.

4.3 Banco de Dados Ghrab *et al. (*2016)

O modelo GRAD estende o conceito de grafos de propriedades "atribuindo semântica a eles" (GHRAB *et al.*, 2016). A definição do grafo no modelo GRAD é apresentada a seguir: $\mathbf{G = (V, E, L_v, F_v, L_e, F_e, \Lambda_v, H_v, h_l, \Lambda_e, H_e)}$, em que:

- $\mathbf{V = (V_e \cup V_a \cup V_l)}$ é um conjunto de vértices, onde **Ve** é um conjunto de vértices de entidade, $\mathbf{V_a}$ um conjunto de vértices de atributos e $\mathbf{V_l}$ um conjunto de vértices literais.

- **$E = (E_e \cup E_a \cup E_l)$** é um conjunto de arestas, onde **E_e** um conjunto de arestas de entidade, **E_a** um conjunto de arestas de atributos e **E_l** um conjunto de arestas literais.
- **L_v** é um conjunto de rótulos em vértices entidade e vértices atributo. **$F_v: (V_e \cup V_a) \rightarrow L_v$** função que atribui rótulo a cado vértice entidade ou atributo.
- **L_e** é um conjunto de rótulos em arestas de entidade. **$F_e: E_e \rightarrow L_e$** função que atribui rótulo a cada aresta de entidade.
- **Λ_v** é um conjunto de identificadores de vértice de entidade. Cada identificador é representado por um vetor de identificadores ($a^1, a^2, ..., a^k$).
- **H_v** é o conjunto de funções que retorna o valor x de um vértice de seu i-ésimo atributo.
- **h_l** é a função retorna os valores armazenados nos vértices literais.
- **Λ_e** é o conjunto de atributos das arestas, que são representados por pares de chave/valor.
- **H_e** é o conjunto de funções que retorna o valor x de uma aresta de seu i-ésimo atributo.

No trabalho considera-se um conceito de classe, que representa um conjunto de “coisas” de um tipo específico que compartilham uma estrutura comum e relacionamentos. Caracterizam-se por predicados e limitam apenas ao tipo unário. Uma classe é um conjunto de elementos de grafo que satisfazem um predicado unário aplicado nos rótulos de vértices de entidade.

Outro conceito é o de objeto, que possui identificadores únicos, atributos e relações. Cada objeto pertence a uma classe apenas, e objetos que têm uma mesma característica são agrupados em uma mesma classe.

No modelo GRAD, o núcleo de um objeto é representado por um vértice de entidade, que contém um rótulo e os atributos identificadores. Já os vértices de atributos são anexados aos vértices entidades e denotam atributos não identificadores. Os vértices literais apresentam valor real do seu vértice atributo.

- **vértice entidade**: denota o tipo da entidade e a classe à qual ele pertence e possui um ID. Representa a parte central de um elemento do mundo real. O ID mapeia o conceito de chave primária no modelo relacional. Um exemplo seria o par <FILME, {3884, Star Trek}> da classe FILME.

- **vértice atributo**: contém apenas o rótulo que descreve seu nome. Um exemplo (FILME) = {Pontuação, Título, WebID}.
- **vértice literal:** Trata-se de uma representação de um valor real de atributo. Um exemplo é utilizar o vértice literal para representar o título em diferentes idiomas.

Os relacionamentos considerados são apenas os binários. A seguir os tipos de arestas, que representam os possíveis relacionamentos no GRAD:

- **Aresta de entidade**: entre par de vértices de entidade e_i denota $\in$ {Associação, Generalização, Agregação, Composição}.
- $\mathbf{E_{as}}$ aresta de associação: mais comum de relações, onde dois vértices de entidade estão associados um ao outro.
- $\mathbf{E_{g}}$ aresta de generalização: relaciona um vértice de entidade de subclasse com seu vértice de entidade de superclasse. Geralmente é referido como um relacionamento "Is_a".
- $\mathbf{E_{ag}}$ aresta de agregação: relação de todo / parte entre dois vértices de entidade. Em UML reflete a fraca dependência entre as duas entidades. Ex: Este tipo de arestas descreve uma relação hierárquica entre entidades de dados, como a relação entre cidades e países.
- $\mathbf{E_{c}}$ aresta de composição: forma de agregação mais forte que reflete que a existência da parte depende da existência do vértice de entidade composto (inteiro). Esta relação mapeia a composição (agregação composta) em UML.

Cada vértice de entidade pode ter muitas arestas de associação, mas no máximo uma aresta de generalização de saída. Utiliza-se a propriedade muitos-para-um desses relacionamentos (isto é, generalização, agregação e composição) para introduzir a noção de relação pai-filho.

Aresta de atributo: representa uma relação de composição entre o vértice de atributo e o vértice entidade pai. “As arestas de atributo controlam os atributos de alteração extraídos como novos vértices”

Aresta literal: relação de composição entre um vértice literal (parte) e seu vértice de atributo pai (composto). “Ex: indicar o título do filme em um determinado idioma, enquanto o próprio título é armazenado no vértice literal relacionado”.

Uma representação dos componentes gráficos do modelo GRAD é ilustrada na Figura 14.

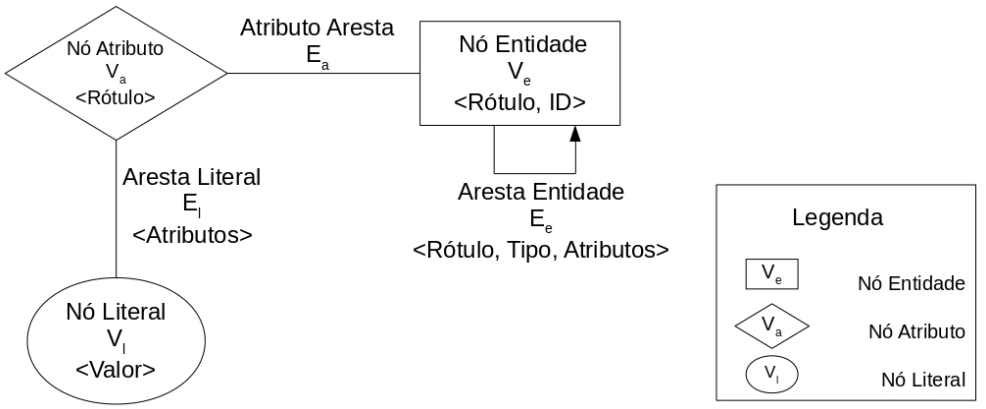

Figura 14: Notação gráfica (Ghrab et al, 2016)

Por fim, o modelo GRAD apresenta definição de hipervértice. Os hipervértices são utilizados para representar um objeto do mundo real, ou então, em GRAD, são utilizados sendo representados por subgrafos. Cada hipervértice agrupa um vértice de entidade, todos os seus vértices de atributo e literal e as arestas entre eles.

4.3.1 Restrições de integridade

Os autores classificaram as restrições de integridade mais relevantes em (Angles, 2012):

- Integridade da entidade de grafo: garante que cada entidade do mundo real é representada por um único hipervértice. Também fornece mecanismos para identificar vértices e arestas através de atributos específicos ou “propriedades estruturais tais como vizinhanças”;
- Restrições semânticas: definidas pelo usuário. 1º tipo: definições em elementos dos grafos e 2º tipo: verificação da cardinalidade entre as classes dos vértices.

4.4 Comparação entre os modelos lógicos de grafos

Na Tabela 2, os três modelos de lógicos de grafos são comparados os quesitos relacionados com o tema da pesquisa deste trabalho.

Tabela 2: Comparativo entre os modelos lógicos de grafos

	(Pokorný, 2016)	(Virgilio et al., 2014)	(Ghrab et al., 2016)
Rótulos de vértices	x	x	x
Rótulos de arestas	x	x	x
Atributos de vértices	x	x	x
Atributos de arestas	x	x	x
Restrições de Integridade	x		x

Rótulos de vértices, rótulos de arestas, atributos de vértices e atributos de arestas são analisados por serem elementos de um grafo de propriedades. Em relação às restrições de integridade, justifica-se sua análise pois diversos autores salientam a importância deste quesito, em que é possível garantir consistência de um banco de dados.

4.5 Mapeamento de modelo conceitual para modelos lógicos orientado a grafos

A partir desta seção são apresentadas as metodologias de mapeamento de nível conceitual para nível lógico de cada um dos modelos lógicos de bancos de dados de grafos apresentados anteriormente.

4.5.1 Mapeamento do modelo lógico de grafos de Pokorný (2016)

Neste trabalho é considerado um modelo ER binário para o esquema conceitual. São utilizados seguintes recursos: entidades do tipo forte e fraca, Relacionamentos, Atributos, chaves de identificação, chaves de identificação parciais, hierarquia “Is_a” e restrições de integridade min-max.

O modelo ER binário do trabalho é definido como **Esquema Conceitual de Grafo** = **<E, R, H, CC>**, onde:

- **E** é um conjunto de tipos de entidades, cada entidade é dada pelo seu nome $\mathbf{E_i}$ e um conjunto de atributos $\mathbf{A_{Ei}}$;
- Cada $\mathbf{E_i}$ possui uma chave de identificação $\mathbf{K_{Ei}}$ que é formada por um ou mais atributos de $\mathbf{A_{Ei}}$;
- **R** é um conjunto de relacionamento do tipo binário, cada relacionamento **R** é dado por um par $\mathbf{(E_{i1}, E_{i2})}$ e um conjunto de atributos $\mathbf{A_R}$;
- Existem dois nomes de relacionamento para cada relacionamento.
- Se $\mathbf{E_{i1} = E_{i2}}$ para **R**, então esse tipo de relacionamento é chamada recursivo.
- **H** é um conjunto de hierarquia "Is_a" para Entidades;
- **CC** é um conjunto de Restrições de Integridade..
- $\mathbf{E_W}$ é o conjunto de entidades fracas. $\mathbf{E_W \subset E}$, e pode ser possivelmente vazio. Para cada $\mathbf{E_W}$ existe pelo menos uma sequência $\mathbf{E_1,...,E_s}$, tal que $\mathbf{E_1 = E}$, $\mathbf{E_{i-1}}$ é uma identificação dependente de $\mathbf{E_i, i = 2, s-1}$ e $\mathbf{E_s}$ é uma entidade forte. Chave de identificação de $\mathbf{E_W}$ é a união de todas as chaves de identificação parciais e completas deste sequência.

  Em cada hierarquia "Is_a" $H_E \in \mathbf{H}$, $H_E \subseteq \mathbf{E}$ x **E**,
- Entidade **E** é a origem de $\mathbf{H_E}$ com chave de identificação $\mathbf{K_E}$,
- O grafo associado a $\mathbf{H_E}$ é uma árvore com a raiz **E**,
- Não há hierarquia **'HE'** $\in$ **H** tais que a árvore associada a $\mathbf{H_E}$ é uma sub-árvore da árvore, que está associada à hierarquia $\mathbf{H_E}$, exceto o caso, quando $\mathbf{H_E}$ só tem uma raiz.

Para cada tipo de relação R $\in$ **R** existem duas restrições de integridade "min-max" no **CC** e vice-versa, para cada relação min-max de IC do **CC,** há pelo menos um **R** em **R** tendo este IC como min-max IC.

Conceitualmente, outros tipos genéricos de relacionamento, por exemplo, relacionamentos "é-parte-de", poderiam ser considerados no modelo ER binário. Eles também podem ser descritos simplesmente com construções conceituais de gráfico. A seguir são apresentadas as regras de mapeamentos que fornecem o mapeamento do esquema conceitual de grafos, apresentados anteriormente, para o modelo lógico de grafos.

4.5.1.1 Regras de Mapeamento em Pokorný (2016)

No trabalho, são apresentadas regras para realizar o mapeamento entre o esquema conceitual de grafos (apresentado em 4.6) para o banco de dados de grafos (apresentado em 4.1). A seguir, as regras de mapeamento.

I. **Entidade do tipo forte.** Para cada entidade do tipo forte, ou entidade, E ∈ **E**, cria-se um tipo de vértice D_E, onde são incluídos todos os atributos de um A_E e a chave de identificação K_E.
II. **Entidade fraca.** E_1, …, E_s é uma sequência de entidades a partir E, onde E_s é uma entidade do tipo forte, e E_i, i=1, ..., s-1, são entidades do tipo fraca com chaves parciais PK_i. Para cada entidade do tipo fraca E_i, cria-se um vértice D_{Ei}, que inclui todos os atributos de uma chave E_i e identificação K_{i+1} a partir de D_{Ei+1}.
III. **Relacionamento**. Para cada relacionamento R_i ∈ **R**, criar uma aresta D_R, que inclui todos os atributos de A_{Ri}. No trabalho, salienta-se que o rótulo e a direção da aresta devem ser especificados.
IV. **Restrições de integridade.** As restrições inerentes, como a cardinalidade, tornam-se restrições explícitas em D. Restrições explícitas de **CC** também se tornam restrições explícitas em D.
V. **Hierarquia "Is_a"**. As hierarquias são transformadas em arestas com rótulo Is_a no nível de banco de dados de grafos. Recomenda-se propagar a chave de identificação K_E da **hierarquia "Is_a"** em todos os vértices desta hierarquia em D.

4.5.1.2 Exemplo de Mapeamento em Pokorný (2016)

O trabalho utiliza um cenário acadêmico para ilustrar sua proposta de projeto. Nesta proposta, os professores ensinam idiomas e para eles são armazenados: cidade e rua onde se reside e o tipo (pessoa).

O nível conceitual é representado graficamente na Figura 15.

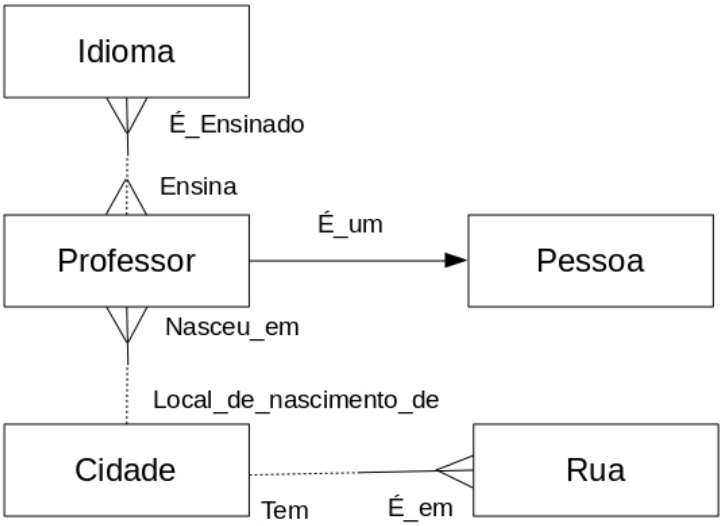

Figura 15: Esquema conceitual de grafos (Pokorný, 2016)

O esquema conceitual de grafos da figura anterior apresenta cinco entidades e quatro relacionamentos. É possível verificar as cardinalidades dos relacionamentos: Idioma x Professor: N x N; Professor x Cidade: 1 x N; Cidade x Rua: 1 x N. Uma hierarquia “Is_a” é utilizada onde todo elemento associado a “Professor” é do tipo “Pessoa”. Além da cardinalidade apresentada nos relacionamentos, é possível verificar o min-max onde é possível verificar as partes tracejadas.

O esquema de banco de dados de grafos resultante a partir do esquema conceitual de grafos, da Figura 15, é apresentado na Figura 16.

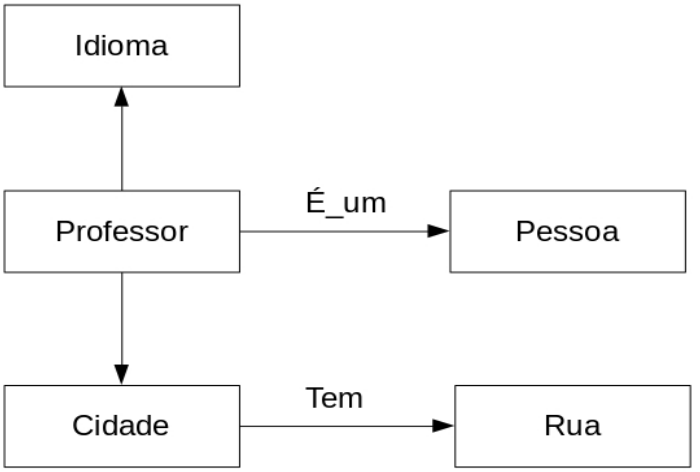

Figura 16: Esquema de banco de dados de grafos (Pokorný, 2016)

No esquema do banco de dados de grafos, cada entidade recebe o rótulo comum entre as entidades e cada relacionamento recebe uma aresta direcionada. No trabalho, não é apresentada uma implementação do projeto em um sistema de banco de dados de grafos.

4.5.2 Mapeamento do modelo lógico de Virgilio *et al. (*2014)

Inicialmente é considerado um modelo conceitual utilizando o modelo ER. Um dos objetivos desse trabalho é reduzir a quantidade de objetos com a realização da agregação de entidades. No trabalho é afirmado que a intervenção humana no mapeamento do modelo conceitual para o modelo lógico é reduzida, buscando um mapeamento automático.

O objetivo principal no projeto de um BD de Grafos é a minimização das operações de acesso a dados necessários em percursos de grafo no momento da consulta. Como opções sugere-se: (i) pela adição de arestas entre vértices ou (ii) pela fusão de diferentes vértices.

No trabalho, são considerados elementos básicos do modelo ER como: entidade, relacionamentos, atributos e cardinalidade. Como exemplo, é utilizado um diagrama de uma aplicação para blog, que é ilustrado na Figura 17.

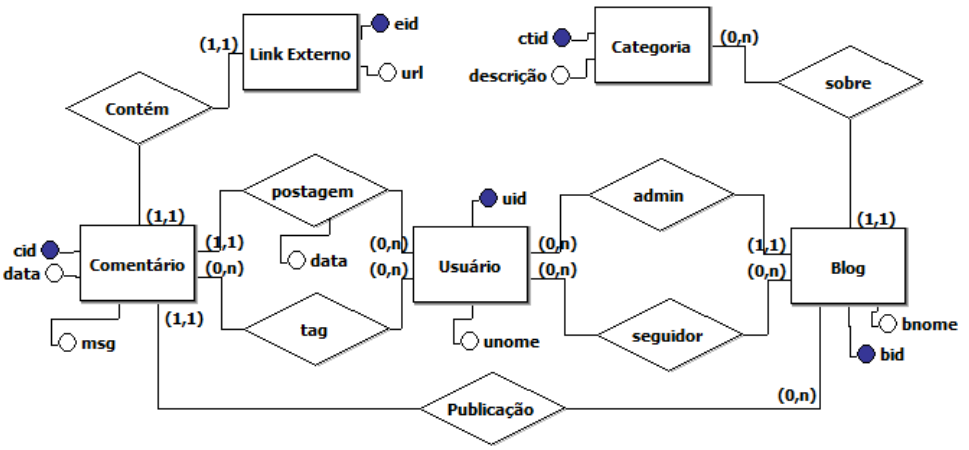

Figura 17: Diagrama ER (Virgilio *et al.*, 2014)

O projeto é iniciado a partir de um diagrama ER e é separado em 3 fases:

- 1ª fase: geração de um diagrama ER Orientado (O-ER);
- 2ª fase: partição dos elementos (entidades e relacionamentos) do diagrama obtido;
- 3ª fase: definição de um modelo sobre a partição resultante.

Na primeira fase, o diagrama é transformado em um grafo direcionado, rotulado e “pesado”. O diagrama O-ER recebe um peso **w** em cada extremidade, através de 3 regras:

A) Uma relação um-para-um torna-se uma aresta dupla dirigida **e** tal **w (e) = 0**;

B) Uma relação um-para-muitos se torna uma aresta de direção única **e** tal que **w (e) = 1** indo da entidade com multiplicidade inferior à multiplicidade superior da entidade;

C) Uma relação de muitos para muitos se torna uma aresta de dupla direção **e w (e) = 2**.

Nessa fase, as entidades e relacionamentos, ambas com suas propriedades, são mantidas. Um diagrama O-ER aplicado no modelo ER da Figura 17 é apresentado na Figura 18.

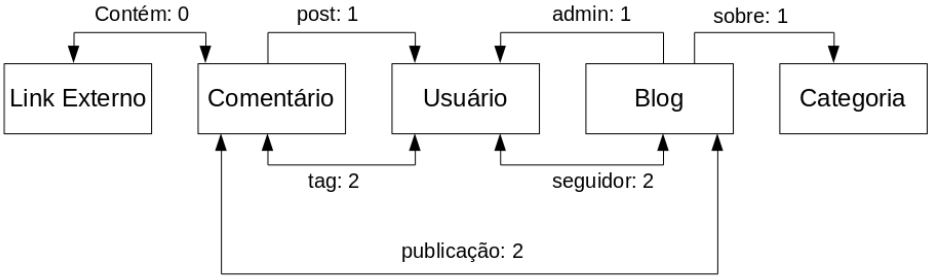

Figura 18: Diagrama O-ER (Virgilio *et al.*, 2014)

Destaca-se a preocupação com a cardinalidade N:M nos relacionamentos. Virgílio *et al*. (2014, p. 176) "analisando cuidadosamente as relações muitos-para-muitos do diagrama ER, que pode introduzir muitas conexões entre vértices e, portanto, propriedades conflitantes em sua agregação."

A segunda fase consiste no particionamento do diagrama, que objetiva agrupar entidades em que suas respectivas instâncias façam parte de possíveis consultas. Para que seja possível diminuir a quantidade de acesso ao BD. Nessa fase, consideram o Grafo **(N, E, w)** onde **N** é o conjunto de vértices (entidades), **E** é o conjunto de arestas e **w** a função de ponderação. Em seguida, considere as seguintes funções de peso para vértices de um diagrama O-ER na Figura 19.

$$w^{+}(n) = \sum_{e \in out(n)} w(e)$$

$$w^{-}(n) = \sum_{e \in in(n)} w(e)$$

Figura 19: Funções para peso de vértice (Virgilio *et al.*, 2014)

As funções **w-(n)** e **w+(n)** calculam a soma dos pesos das arestas de entrada e saída de um vértice, respectivamente. O particionamento segue algumas regras para realizar o agrupamento entre entidades, ou vértices:

- Regra 1: se um vértice **n** é desconectado, então ele forma um grupo por si só;
- Regra 2: se um vértice **n** tem **w-(n) > 1** e **w + (n) >= 1** então **n** forma um grupo por si só. Intuitivamente, neste caso o vértice **n** representa uma entidade envolvida com alta multiplicidade em relações de muitos para muitos. Portanto, não se agrega **n** com outros vértices com peso semelhante.
- Regra 3: se um vértice **n** tem **w - (n) <= 1** e **w + (n) <= 1** então **n** é adicionado ao grupo de um vértice **m** tal que existe a aresta **(m, n)** no diagrama O-ER. Nesse caso, o vértice **n** corresponde a uma entidade envolvida em um relacionamento um-para-um ou em relacionamento um-para-muitos em que **n** tem a multiplicidade inferior.

Seguindo o projeto do banco de dados para o blog, e aplicando as regras da segunda fase, o particionamento é apresentado na Figura 20.

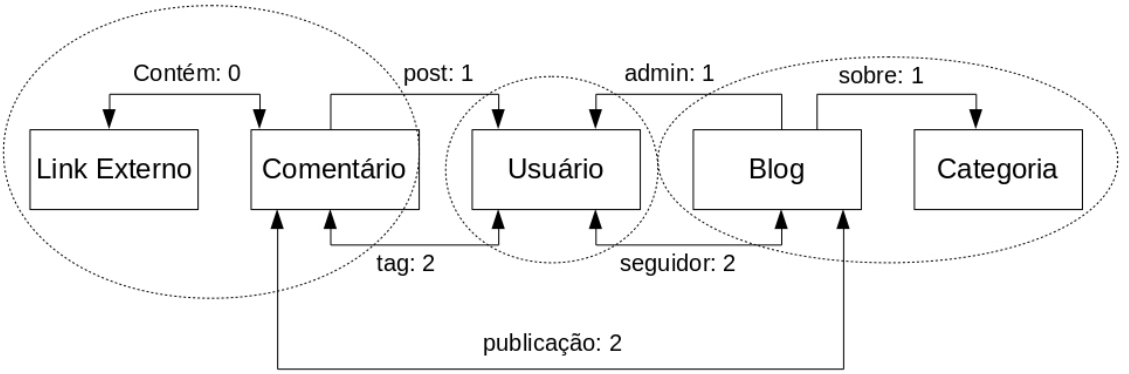

Figura 20: Particionamento durante a modelagem (Virgilio *et al.*, 2014)

Na terceira e última fase, um modelo para o banco de dados grafos é definido. O modelo que representa um esquema onde os vértices e as arestas estão definidas com seus atributos. Cada nome do atributo tem adicionado o nome da entidade originária. Nas arestas são atribuídos o rótulo e os atributos.

O resultado da terceira fase é apresentado na Figura 21.

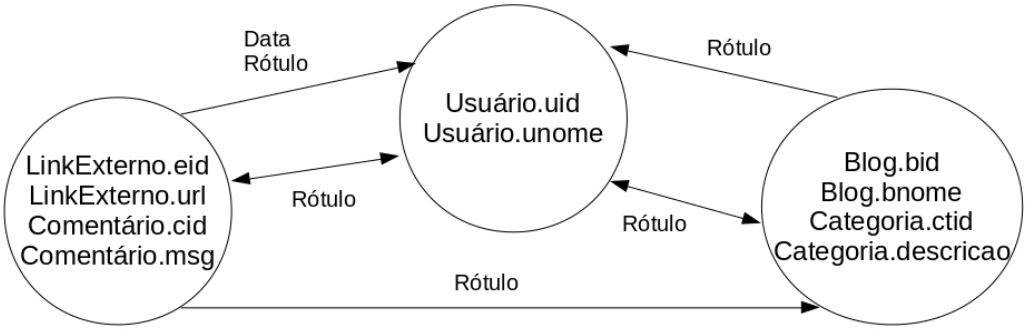

Figura 21: Template da Projeto de BD (Virgilio *et al.*, 2014)

O trabalho é concluído mencionando que utilizando avaliações, a metodologia oferece vantagens consideráveis em termos de desempenho de consulta em relação a abordagens simples. Porém, a migração do modelo conceitual para o modelo lógico não é automática, pois exige uma representação intermediária.

4.5.3 Mapeamento do modelo lógico de grafos em Ghrad *et al.* (2016)

O artigo apresenta o modelo GRAD com sua definição, restrições de integridade e uma álgebra de grafos. O mapeamento do nível conceitual para o modelo GRAD pode ser realizado utilizando os modelos UML ou EER, que utilizam conceitos de entidades, relacionamentos e atributos. No trabalho não são abordados os elementos utilizados no nível conceitual ou regras para o mapeamento.

4.6 Comparação entre os mapeamentos conceitual-lógico

Após apresentação e comparação dos modelos lógicos, foi realizada a apresentação de cada mapeamento neste capítulo. Para melhor análise, os mapeamentos foram comparados e dispostos na Tabela 3. Os itens utilizados estão relacionados com o tema da pesquisa do trabalho. Foi analisado se é abordado um modelo conceitual de dados e uma respectiva representação conceitual. Desse modelo conceitual, é analisado se o relacionamento entre entidades é apenas do tipo binário. Também se analisou se é ofertada uma maneira de realizar mapeamento conceitual-lógico.

Tabela 3: Comparativo entre os mapeamentos das propostas

	(Pokorný, 2016)	(Virgilio *et al.*, 2014)	(Ghrab *et al.*, 2016)
Modelo ER, EER ou UML	**X**	**X**	**X**
Definição formal do modelo conceitual	**X**		
Relacionamento Binário	**X**	**X**	**X**
Regras de mapeamento	**X**	**X**	

4.7 Conclusão do capítulo

Em relação aos modelos lógicos, todas as propostas consideram rótulos e atributos para vértices e em arestas. Já as restrições de integridades e a cardinalidade são consideradas apenas em Pokorný (2016) e Ghrab *et al*. (2016). O relacionamento NxM são abordados em Pokorný (2016) e Virgilio *et al.* (2014). Ghrab *et al.* (2016) se diferencia dos outros dois com o suporte a álgebra de grafos.

Já em relação ao projeto conceitual e mapeamento para o modelo lógico, as três propostas utilizam modelos conceituais, ou extensões, que também são utilizadas para projeto de bancos de dados tradicionais, como ER, EER ou UML. Apenas Pokorný (2016) representa formalmente o modelo conceitual. Todas as propostas consideram apenas relacionamentos binários. Pokorný (2016) e Virgilio *et al.* (2014) expõem suas regras do mapeamento do modelo conceitual para o modelo lógico.

Em uma comparação prévia, que analisa interesses particulares deste presente trabalho, e conforme as Tabelas 2 e 3, pode-se considerar que Pokorný (2016) é o trabalho mais completo. Porém salienta-se que os outros trabalhos têm outras características interessantes como tentativa de redução de operação de acesso e também, a apresentação de uma álgebra para grafos.

5 PROJETO LÓGICO DE BANCO DE DADOS NOSQL DE GRAFOS A PARTIR DE UM MODELO CONCEITUAL BASEADO NO MODELO ENTIDADE-RELACIONAMENTO

Este capítulo propõe o modelo EB-ER (*Extend Binary – Entity-Relationship*), em português Entidade-Relacionamento – Binário Estendido, que se trata de um modelo conceitual de dados baseado no modelo Entidade-Relacionamento. O EB-ER visa a modelagem conceitual para bancos de dados orientados a grafos. Propõe-se também uma metodologia para o mapeamento de esquema conceitual EB-ER para modelo lógico de grafos utilizando algoritmos de alto nível, que também considera restrições de integridade ao modelo lógico.

O modelo lógico de grafos é baseado no modelo de grafos de propriedades, apresentado no Capítulo 3. A partir das restrições de integridade do modelo lógico, propõe-se um conjunto de comandos para linguagem Cypher do sistema Neo4.

Na seção 5.1 será apresentado o EB-ER, sua definição formal e a justificativa da não utilização de alguns elementos do modelo no qual o mesmo é baseado. Na seção 5.2 é apresentado o modelo lógico de grafos, este que fornece uma visão do esquema lógico de dados e suas restrições. O conjunto de algoritmos de alto nível que fornece mapeamento do esquema EB-ER para o modelo lógico de grafos é apresentado na seção 5.3. Um conjunto de comandos e propostas de comandos para a linguagem Cypher do sistema Neo4j é apresentado na seção 5.4. Conclui-se o capítulo na seção 5.5.

Na Figura 22 são exibidas detalhadas as fases de projeto de banco de dados de grafos baseado nesta proposta. Para cada nível do projeto (conceitual, lógico e físico), são exibidas suas respectivas características.

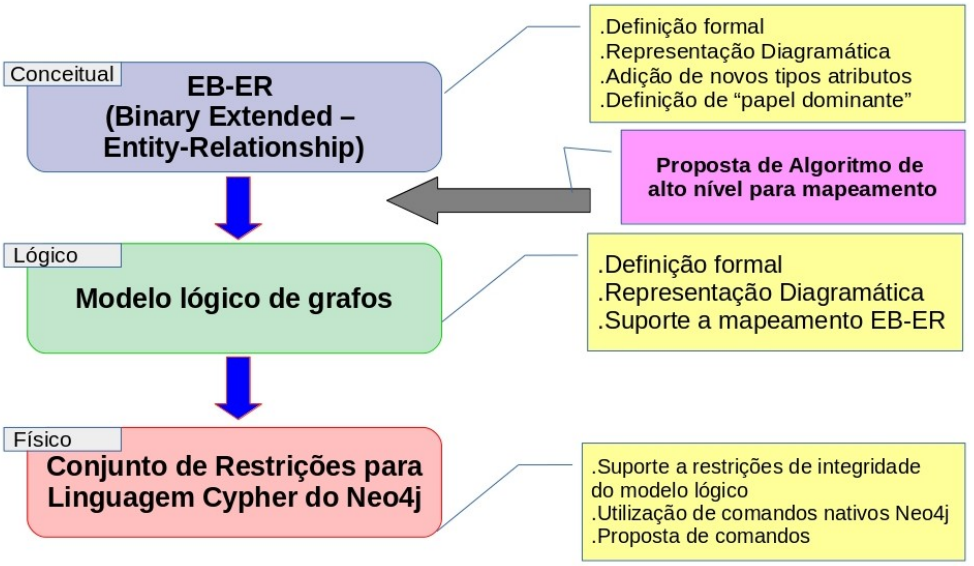

Figura 22: Ilustração das fases do projeto de banco de dados de grafos desta proposta

Os conceitos desta proposta são experimentados em dois estudos de caso no restante do presente trabalho.

5.1 Binary Extended – Entity-Relationship (EB-ER)

EB-ER é baseado no modelo ER, apresentado na seção 2.3. O EB-ER utiliza diversos conceitos como entidades, atributos e relacionamentos (considerando o tipo binário, a cardinalidade e os papéis das entidades envolvidas).

Uma definição formal em um modelo conceitual possibilitará a construção de operações sobre a estrutura de um esquema conceitual e seus respectivos dados. A seguir é apresentada uma definição formal para esquemas conceituais do modelo EB-ER.

Um modelo conceitual de dados EB-ER pode ser definido formalmente como o esquema **C = (E, R, A)**, onde:

- **E** representa um conjunto de entidades. Uma entidade $\mathbf{e_i} \in \mathbf{E}$ representa um grupo de objetos semelhantes de um domínio.
- **R** representa um conjunto de relacionamentos. Um relacionamento $\mathbf{r_i} \in \mathbf{R}$ é do tipo binário (ou grau dois), sendo $\mathbf{r_i} = \mathbf{<e_1, ca_1, \{ro_1\}, \{ro_2\}, ca_2, e_2, P>}$. As entidades envolvidas são $\mathbf{e_1}$, $\mathbf{e_2} \in \mathbf{E}$. As cardinalidades são $\mathbf{ca_1}$ e $\mathbf{ca_2}$, que utilizam os mesmos conceitos do ER, onde podem ser definidas como pares (min, max),

onde *min* pode ser 0 ou 1 e *max* pode ser 1 ou N. Os papéis de cada entidade no relacionamento são $\mathbf{ro_1}$ e $\mathbf{ro_2}$**,** estes são opcionais, ou seja, possivelmente ∅. **P** representa um dos papéis do relacionamento como papel dominante do relacionamento, **P** pode ser $\mathbf{ro_1}$, $\mathbf{ro_2}$ ou **∅**. Caso $\mathbf{e_1} = \mathbf{e_2}$, trata-se de um relacionamento recursivo.

  - A definição de um papel dominante é justificada para representar uma entidade principal no relacionamento, que beneficiará na modelagem lógica, onde poderá ser utilizada para definir o sentido do rótulo de aresta.

- **A** representa um conjunto de atributos do esquema conceitual. Neste conjunto pode se encontrar: $\mathbf{e_i.a_j} \in \mathbf{A}$ e $\mathbf{r_i.a_j} \in \mathbf{A}$, onde o primeiro representa um atributo de uma entidade e o segundo representa um atributo de relacionamento. Um atributo pode ser definido entre um dos seguintes tipos:

  - Fixo: O atributo deve estar presente em cada entidade ou relacionamento com o qual está associado. Um atributo fixo é definido com um símbolo # como prefixo do nome do atributo. Exemplos: $\mathbf{e_i.\#a}$ ou $\mathbf{r_i.\#a_j}$;

  - Identificador: Corresponde a um atributo identificador e obrigatório que possui um valor diferente para cada entidade ou relacionamento com o qual está associado. Um atributo identificador é definido com um símbolo ***** como prefixo do nome do atributo. Exemplos: $\mathbf{e_i.*a_j}$ ou $\mathbf{r_i.*a_j}$;

  - Alternativo: Corresponde a uma lista de atributos, em que algum deles deve estar presente em cada entidade ou relacionamento com o qual está associado. Um atributo alternativo é definido com um símbolo **@** como prefixo da lista de atributos. Exemplos: $\mathbf{e_i.@(at_1|at_2|...|at_n)}$ ou $\mathbf{r_i.@(at_1|at_2|...|at_n)}$;

  - Único: Atributo que define o valor de determinada propriedade de uma entidade, ou relacionamento, não pode se repetir. Este atributo utiliza o símbolo **§** como prefixo do nome do atributo. Exemplo $\mathbf{e_i.§a_j}$ ou $\mathbf{r_i.§a_j}$.

Os tipos de atributos Fixo e Identificador possuem semelhança na obrigatoriedade da existência de valor, porém, se diferem, onde o primeiro permite a repetição de valores para instâncias (de entidades ou de relacionamentos) e no segundo não permite a repetição de valores. Um exemplo para ilustração seria uma aplicação *e-*

Commerce, em que uma entidade “Cliente” pode possuir um atributo fixo “endereço” e atributo identificador “CPF” (ou RG, CNH, etc.).

O atributo do tipo Alternativo fornece a flexibilidade de utilizar uma sequência de atributos possíveis. Para exemplificar, pode ser utilizado um atributo para armazenar um contato para uma entidade “Cliente”, onde poderia ser utilizado: e-mail, telefone, *website*, endereço, conta em rede social, entre outros.

O atributo do tipo Único possui característica semelhante do Identificador, onde o esquema preserva a unicidade de uma propriedade a uma entidade ou relacionamento. Porém o atributo Único difere em permitir a inexistência de valor.

O EB-ER, como mencionado anteriormente, é baseado no ER. A diferença deste consiste na adição de alguns elementos e a inutilização de outros.

A representação diagramática do EB-ER é apresentada na Figura 22. As entidades utilizarão a mesma representação do ER, já os relacionamentos terão a diferença da exibição dos papéis dos relacionamentos, mesmos os binários ou de grau maior, e com adição da marcação do papel dominante. Na representação de atributos, considera-se o mesmo exemplo para atributo identificador do ER. Para os demais, propostos no EB-ER, são definidas novas representações.

5.1.1 Representação diagramática do EB-ER

Conforme utilizado no ER e demais modelos conceituais, utiliza-se uma representação gráfica para representar um esquema conceitual de uma aplicação. O EB-ER utiliza os recursos do ER, com a diferença de apresentar o papel de cada entidade no relacionamento. Normalmente esta característica é utilizada em relacionamentos recursivos. A utilização do papel é para representar o papel dominante do relacionamento, sendo este marcado com uma cor de realçe no nome do papel. As demais diferenças são na representação de atributos, onde são propostos três novos tipos (alternativo, fixo e único). O atributo identificador utiliza as características do ER. Na Figura 23 são apresentados todos elementos gráficos utilizados no EB-ER.

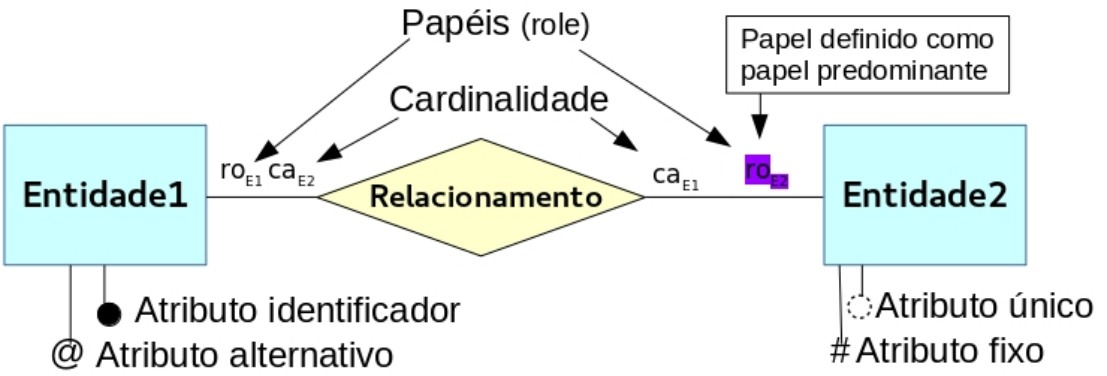

Figura 23: Elementos para representação diagramática do EB-ER

5.1.2 Justificativa da não inclusão de alguns elementos do Entidade-Relacionamento

Esta seção detalhará a não inclusão de alguns elementos do ER. Resumidamente, justifica-se na proposta de um modelo conceitual simplificado (utilização de poucos elementos), flexível (fornecimento de atributos alternativos), que tenha suporte a restrições (prover consistência nas entidades e relacionamentos) e ampla utilização de relacionamentos (consequências do desuso de alguns elementos do ER original), a retirada de alguns elementos que serão listados a seguir.

- A entidade fraca e a entidade associativa não foram utilizadas para oferecer mais simplicidade ao modelo conceitual. Para estes elementos, pode-se utilizar as entidades ofertadas no EB-ER.
- Os relacionamentos n-ários não são considerados, onde a alternativa é a substituição por entidades, em que a mesma se relaciona com todos as entidades que supostamente participariam de um relacionamento n-ário.
- A especialização é um conceito muito útil no ER, e também possibilita diversas alternativas posteriores para modelagem lógica. A especialização pode ser substituída considerando todas entidades que seriam as “especializadas” e com adição dos atributos que seriam da entidade “genérica”.
- O desuso dos atributos composto e multivalorado. O primeiro pode ser substituído pela decomposição de atributos, e utilizando, se necessário, os tipos

ofertados pelo EB-ER. Já o segundo pode ser substituído pela criação de uma entidade para armazenar os atributos e ainda, possibilitar o relacionamento com outras entidades.

- Cardinalidade de atributos não é utilizada no EB-ER. Justifica-se já no desuso dos atributos multivalorados.

Conforme as justificativas listadas acima, ressalta-se que, embora esses elementos tenham sido retirados, sua função prática permanece como o modelo ER original, assim se demonstra a simplificação sem perda de poder de representação.

Na seção seguinte é apresentado o conceito de modelo lógico de grafos, que proporcionará uma visão lógica, onde suas restrições de integridade fornecerão a possibilidade de construir um esquema mínimo para banco de dados de grafos.

5.2 Modelo lógico de grafos

Atualmente não existe uma definição padronizada de um modelo lógico de grafos, porém são encontradas diversas definições apresentadas em diversos trabalhos, alguns destes trabalhos foram apresentados no capítulo 4. Como mencionado anteriormente, para os BDs NoSQL de grafos, é considerado o grafo de propriedades. O modelo lógico de grafos considerado neste trabalho é apresentado a seguir.

Um modelo lógico de grafos pode ser definido formalmente como o esquema **G = (LV, LE, PV, PE, CV, CE)**, onde:

- **LV** representa um conjunto de rótulos de vértices. Um rótulo de vértice é representado por $\mathbf{lv_i} \in \mathbf{LV}$.
- **LE** representa um conjunto de rótulos de arestas. Cada rótulo de aresta é definido como $\mathbf{le_i} = \langle \mathbf{lv_1}, \mathbf{lv_2} \rangle$. Onde $\mathbf{le_i} \in \mathbf{LE}$ é o rótulo de aresta propriamente dito, $\mathbf{lv_1}$ se trata do rótulo de vértice inicial e já $\mathbf{lv_2}$, o rótulo de vértice final.
- **PV** representa o conjunto de propriedades de rótulos de vértices. Uma propriedade de rótulo de vértice $\mathbf{lv_i.p_j} \in \mathbf{PV}$, é a associação de uma propriedade $\mathbf{p_j}$ com um rótulo de vértice $\mathbf{lv_i}$. Uma propriedade de vértice é de um dos tipos: Fixo, Identificador, Alternativo ou Único, onde estes utilizam os mesmos conceitos dos tipos de atributos do EB-ER.

- **PE** representa o conjunto de propriedades de rótulos de arestas. Uma propriedade de rótulo de aresta $\mathbf{le_i.p_j} \in \mathbf{PE}$, é a associação de uma propriedade $\mathbf{p_j}$ com um rótulo de vértice $\mathbf{le_i}$. Uma propriedade de aresta é de um dos tipos: Fixo, Identificador, Alternativo ou Único, onde estes utilizam os mesmos conceitos dos tipos de atributos do EB-ER.
- **CV** consiste no conjunto de restrições em rótulo de vértices.
  - **exists (v, s)** onde **v ∈ LV** e **s ∈ PV**. Restrição de propriedade do tipo fixo, onde obriga a existência de uma propriedade de **s**, em que esta é vinculada ao **v**, um rótulo de vértice.
  - **disjoint (v, S)** onde **v ∈ LV** e $\mathbf{S \subseteq PV, S = \{s_1, s_2, ..., s_n\}}$. Restrição de propriedade do tipo alternativo, onde obriga a existência de uma propriedade, pertencente a uma lista de propriedades de **S**, esta lista de propriedades está vinculada à **v**, um rótulo de vértice.
  - **unique (v, s)** onde **v ∈ LV** e **S ∈ PV**. Restrição de propriedade do tipo único, onde obriga a unicidade do valor de uma propriedade de **s**, em que esta está vinculada à **v**, um rótulo de vértice.
  - **identifier (v, s)** onde **v ∈ LV** e **s ∈ PV**. Restrição de propriedade do tipo identificador, onde é obrigatória a existência da propriedade de **s** e a unicidade do seu respectivo valor, esta propriedade está vinculada à **v**, um rótulo de vértice.
  - **edgeExists (v, x)**, onde **v ∈ LV** e **x = <v,w> ∈ LE**. Essa restrição define que é obrigatório que cada vértice do banco de dados rotulado **v** seja vinculado a um vértice do banco de dados rotulado **w**.
  - **edgeRestricted (v, x)**, onde **v ∈ LV** e **x = < v,w> ∈ LE**. Essa restrição define que cada vértice do banco de dados rotulado como **v** pode ser vinculado, no máximo, a um vértice do banco de dados rotulado **w**.
- **CE** consiste no conjunto de restrições em rótulos de arestas.
  - **exists (e, t)** onde **e ∈ LV** e **t ∈ PE**. Restrição de propriedade do tipo fixo, onde obriga a existência de uma propriedade de **S**, em que esta é vinculada ao **e**, um rótulo de aresta.

- **disjoint (e, S)** onde **e ∈ LE** e **S ⊆ PE, S = {s_1, s_2, ..., s_n}**. Restrição de propriedade do tipo alternativo, onde obriga a existência de uma propriedade, pertencente a uma lista de propriedades de **S**, esta lista de propriedades está vinculada à **e**, um rótulo de aresta.
- **unique (e, t)** onde **e ∈ LE** e **t ∈ PE**. Restrição de propriedade do tipo único, onde obriga a unicidade do valor de uma propriedade de **t**, em que esta está vinculada à **e**, um rótulo de aresta.
- **identifier (e, t)** onde **e ∈ LE** e **t ∈ PE**. Restrição de propriedade do tipo identificador, onde se obriga a existência da propriedade de **t** e a unicidade do seu respectivo valor, esta propriedade está vinculada à **e**, um rótulo de aresta.

O modelo lógico de grafos apresentado nesta seção não descreve elementos, ou ocorrências, de vértices e/ou arestas. Ou seja, não são apresentadas as instâncias, e sim apenas o esquema lógico. Exemplos de instâncias serão ilustradas no estudo de caso.

5.2.1 Representação diagramática para o modelo lógico de grafos.

Nesta seção é apresentada uma representação diagramática para o modelo lógico de grafos. Para tal representação utiliza-se um multigrafo direcionado, rotulado e com propriedades. As restrições de integridade também são consideradas na representação gráfica.

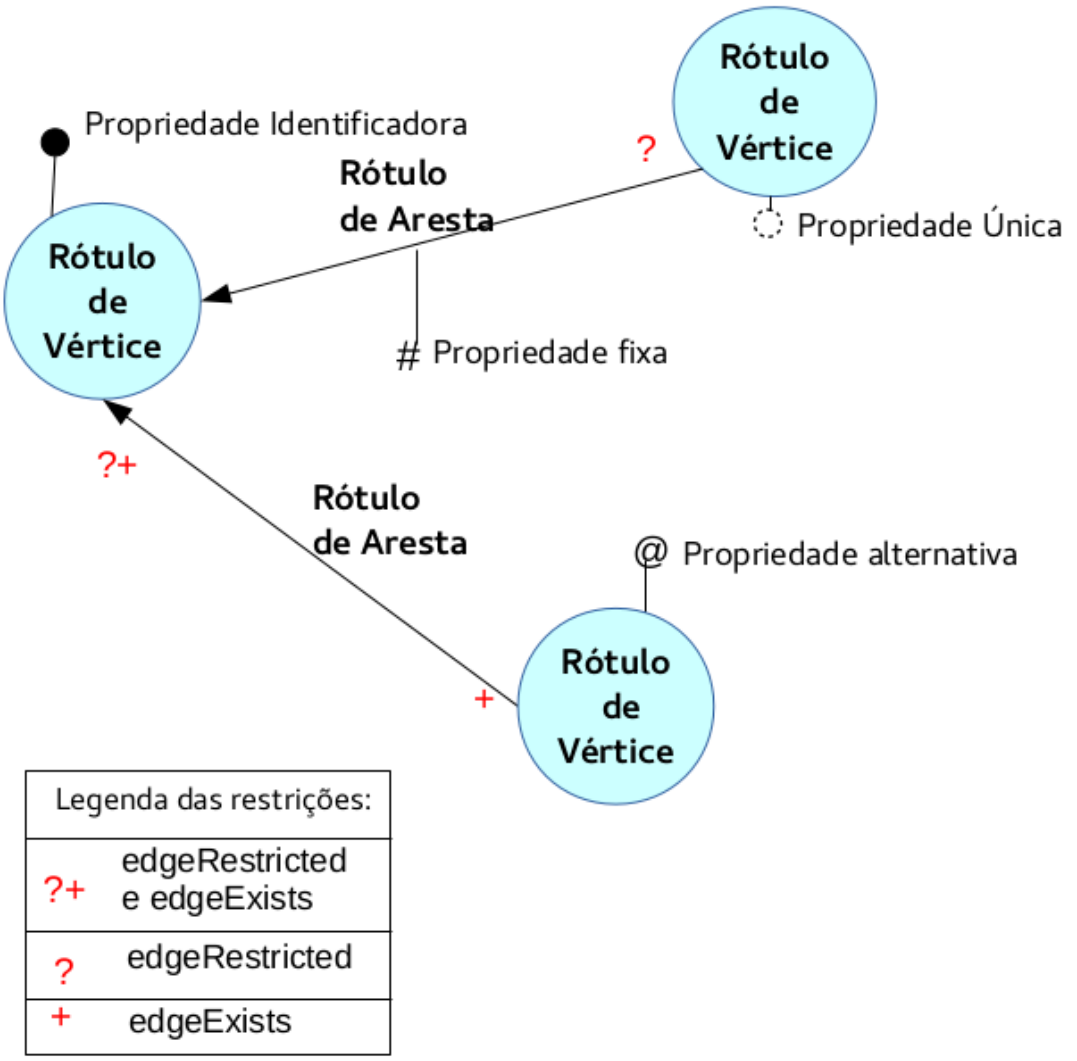

Figura 24: Representação diagramática do modelo lógico de grafos

Os rótulos de vértices são representados por círculos e nome dentro do círculo correspondente. Os rótulos de arestas são representados por setas e com nome no meio da aresta correspondente.

A propriedade do tipo identificador utiliza um círculo preenchido e ao seu lado seu nome. A propriedade do tipo alternativo utiliza o símbolo @ e ao seu lado uma lista de nomes de propriedade. A propriedade do tipo fixo utiliza o símbolo # e ao seu lado seu respectivo nome. A propriedade do tipo único utiliza um círculo pontilhado e ao seu lado seu nome. Cada propriedade é ligada ao seu respectivo elemento, um rótulo de vértice ou um rótulo de aresta.

A restrição *edgeExists* é representada pelo símbolo + e a restrição *edgeRestricted* é representada pelo símbolo ?. Os símbolos de representação de restrição ficam próximos dos rótulos de vértice e rótulos de arestas correspondentes. Em casos onde existam as duas restrições, os dois símbolos são utilizados: +? ou vice-versa.

5.3 Mapeamento de esquema EB-ER para esquema lógico de grafos

Para realização do mapeamento de um esquema conceitual EB-ER para um esquema lógico de grafos, é essencial a utilização de um conjunto de regras de mapeamento. Esta afirmação pode ser fundamentada pelos quesitos listados em Necaský (2006), e também através da definição de regras de mapeamento, nas propostas de (Pokorný, 2016), (Virgilio *et al.*, 2014) analisadas no Capítulo 4.

Sendo assim, a proposta oferece um conjunto de regras para mapeamento completo de um esquema conceitual EB-ER para um modelo lógico de grafos utilizando os modelos apresentados anteriormente. Estas regras são algoritmos, apresentados a seguir.

5.3.1 Algoritmo Mapeamento

O algoritmo Mapeamento, apresentado na Figura 25, pode ser considerado o principal algoritmo do mapeamento do esquema conceitual EB-ER para o modelo lógico de grafos, pois o mesmo iniciará e finalizará o processo de mapeamento. Durante a execução do algoritmo Mapeamento, ele “acionará” outros três algoritmos responsáveis pela criação de restrições, se necessário, no modelo lógico de grafos.

Algorithm 1: MAPPING ALGORITHM

Input: $Conceptual\ Schema\ EB-ER:\ C=(E,R,A)$
Output: $Logical\ Graph\ Schema:\ G=(LV,LE,PV,PE,CV,CE)$
begin
$LV=\emptyset$
$LE=\emptyset$
$PV=\emptyset$
$PE=\emptyset$
$CV=\emptyset$
$CE=\emptyset$

for $each\ entity\ e_i \in E$ **do**
$LV=LV\cup\{lv_i\}$
end
for $each\ relationship\ r_i =< e_1,ca_1,ro_1,ro_2,ca_2,e_2,P >, r_i \in R$ **do**
if $P==\{e_2\}$ **then**
$LE=LE\cup\{le_2,le_1\}$
end
else
$LE=LE\cup\{le_1,le_2\}$
end
end
for $each\ relationship\ r_i =< e_1,ca_1,ro_1,ro_2,ca_2,e_2,P >, r_i \in R$ **do**
$CV=CV\cup CardinalityConstraints(r_i)$
end
for $each\ attribute\ of\ entity\ e_i.a_j \in A, e_i \in E$ **do**
$PV=PV\cup\{lv_i.p_j\}$
$CV=CV\cup VertexConstraints(e_i.a_j)$
end
for $each\ attribute\ of\ relationship\ r_i.a_j \in A, r_i \in R$ **do**
$PE=PE\cup\{le_i.p_j\}$
$CE=CE\cup EdgeConstraints(r_i.a_j)$
end
return G
end

Figura 25: Algoritmo Mapeamento

O algoritmo Mapeamento considera em sua entrada um esquema EB-ER C = (E, R, A) e um modelo lógico de grafos G = (LV, LE, PV, PE, CV, CE) para sua saída. Após início, na sequência de linhas 2 a 7 é atribuído “vazio” para todos elementos de G. Em seguida, nas linhas 8 e 9 cada entidade $e_i \in E$ é mapeada para um rótulo de vértice $lv_i \in LV$. Na sequência de linhas 10 a 14, para cada relacionamento $r_i \in R$ é mapeado um rótulo de aresta $le_i \in LE$, os rótulos de vértices inicial e final são definidos da seguinte

forma: se P = {e_2} lv_2 será o rótulo de vértice inicial e lv_1 o final, senão, utiliza-se lv_1 para inicial e lv_2 para final. Na linha 15, para cada relacionamento $r_i \in R$, o algoritmo acionará a verificação através do algoritmo Restrições de Cardinalidade, após verificação o resultado é unido com estado atual de CV. Na linha 17, para cada atributo de entidade são realizadas duas ações nas linhas 18 e 19, respectivamente: I. adiciona-se uma propriedade de rótulo de vértice; II aciona o algoritmo Restrições de Vértice, e seu resultado é unido com o estado atual de CV. Na linha 20, para cada atributo de relacionamento são realizadas duas ações nas linhas 21 e 22, respectivamente: I. adiciona-se uma propriedade de rótulo de aresta; II aciona o algoritmo Restrições de Aresta, e seu resultado é unido com o estado atual de CE. Na linha 23 é retornado o estado do modelo lógico de grafos **G** e na última linha, finaliza-se a execução do algoritmo.

5.3.2 Algoritmo Rótulos de Cardinalidade

O segundo algoritmo a ser apresentado é também o segundo a ser executado no processo de mapeamento, trata-se do algoritmo Rótulos de Cardinalidade, ilustrado na Figura 26. De maneira geral, são analisadas as cardinalidades de cada relacionamento, e conforme definição, são definidas restrições de Rótulos de Vértices.

Algorithm 2: CARDINALITY CONSTRAINTS ALGORITHM

Input: $r_i =< e_1, ca_1, ro_1, ro_2, ca_2, e_2, P >, r_i \in R$
Output: $X : \ Set\ of\ Constraints$
begin
  $X = \emptyset$
  if $ca_1 == (0,1)\ or\ ca_1 == (1,1)$ **then**
    $X = X \cup edgeRestricted(lv_1, le_i)$
  end
  if $ca_1 == (1,1)\ or\ ca_1 == (1,N)$ **then**
    $X = X \cup edgeExists(lv_1, le_i)$
  end
  if $ca_2 == (0,1)\ or\ ca_2 == (1,1)$ **then**
    $X = X \cup edgeRestricted(lv_2, le_i)$
  end
  if $ca_2 == (1,1)\ or\ ca_2 == (1,N)$ **then**
    $X = X \cup edgeExists(lv_2, le_i)$
  end
  return X
end

Figura 26: Algoritmo Restrições de Cardinalidade

O algoritmo Restrições de Cardinalidade considera em sua entrada um relacionamento $r_i \in R$ enviado pelo algoritmo Mapeamento e para sua saída um conjunto de restrições X, possivelmente vazio. Após início na linha 1, atribui-se “vazio” para X na linha 2. Na linha 3 verifica-se a cardinalidade ca_1 é do tipo (0,1) ou (1,1), sendo verdade, na linha 4 adiciona-se uma restrição edgeRestricted(lv_1,le_i) no conjunto X. Na linha 5 verifica-se a cardinalidade ca_1 é do tipo (1,1) ou (1,N), sendo verdade, na linha 6 adiciona-se uma restrição edgeExists(lv_1,le_i) no conjunto X. Na linha 7 verifica-se a cardinalidade ca_2 é do tipo (0,1) ou (1,1), sendo verdade, na linha 8 adiciona-se uma restrição edgeRestricted(lv_2,le_i) no conjunto X. Na linha 9 verifica-se a cardinalidade ca_2 é do tipo (1,1) ou (1,N), sendo verdade, na linha 10 adiciona-se uma restrição edgeExists(lv_2,le_i) no conjunto X. Na linha 11 é retornado o conjunto de restrições X para o algoritmo Mapeamento e na sequência, finaliza-se a execução do algoritmo Restrições de Cardinalidade.

5.3.3 Algoritmo Restrições de Vértice

O algoritmo Restrições de Vértice é acionado pelo Algoritmo Principal a cada mapeamento de atributo de entidade para propriedade de rótulo de vértice. Resumidamente, para cada propriedade mapeada cria-se uma restrição de rótulo de vértice. O algoritmo Restrições de Vértice é ilustrado na Figura 27, apresentada a seguir.

Algorithm 3: VERTEX CONSTRAINTS ALGORITHM

Input: $Attribute\ e_i.a_j \in A, e_i \in E$
Output: $Constraints\ of\ Vertice\ Labels$
begin
  if $a_j == \#a_j$ **then**
    return $exists(lv_i, pv_j)$
  end
  if $a_j == @(at_1|at_2|...|at_n)$ **then**
    return $disjoint(lv_i, (pv_1|pv_2|...|pv_n)$
  end
  if $a_j == \S a_j$ **then**
    return $unique(lv_i, pv_j)$
  end
  if $a_j == *a_j$ **then**
    return $identifier(lv_i, pv_j)$
  end
end

Figura 27: Algoritmo Restrições de Vértice

O algoritmo Restrições de Vértice considera em sua entrada um atributo de entidade, $e_i.a_j \in A$, $e_i \in E$ enviado pelo algoritmo Mapeamento e para sua saída uma restrição. Após início na linha 1, a linha 2 verifica se o atributo é do tipo fixo. Se sim, na linha 3 retorna a restrição exists(lv_1,pv_i). A linha 5 verifica se o atributo é do tipo alternativo. Se sim, na linha 6 retorna a restrição disjoint(lv_1,($pv_1|pv_2|...|pv_n$). A linha 8 verifica se o atributo é do tipo único. Se sim, na linha 9 retorna a restrição unique(lv_1,pv_i). A linha 11 verifica se o atributo é do tipo identificador. Se sim, na linha 12 retorna a restrição identifier(lv_1,pv_i). Na linha 14 é finalizada a execução do algoritmo.

5.3.4 Algoritmo Restrições de Aresta

O algoritmo Restrições de Aresta é acionado pelo Algoritmo Principal a cada mapeamento de atributo de relacionamento para propriedade de rótulo de aresta. Resumidamente, o funcionamento do algoritmo é semelhante ao anterior, porém neste para cada propriedade mapeada cria-se uma restrição de rótulo de aresta. O algoritmo Restrições de Aresta é apresentado na Figura 28.

Algorithm 4: EDGE CONSTRAINTS ALGORITHM

Input: $attribute\ r_i.a_j \in A, r_i \in R$
Output: $Constraints\ of\ Edge\ Labels$
begin
  if $a_j == \#a_j$ **then**
    return $exists(le_i, pe_j)$
  end
  if $a_j == @(at_1|at_2|...|at_n)$ **then**
    return $disjoint(le_i, (pe_1|pe_2|...|pe_n)$
  end
  if $a_j == \S a_j$ **then**
    return $unique(le_i, pe_j)$
  end
  if $a_j == *a_j$ **then**
    return $identifier(le_i, pe_j)$
  end
end

Figura 28: Algoritmo Restrições de Aresta

O algoritmo Restrições de Aresta considera em sua entrada um atributo de relacionamento $r_i.a_j \in A$, ri ∈ R enviado pelo algoritmo Mapeamento e para sua saída uma restrição. Após início na linha 1, a linha 2 verifica se o atributo é do tipo fixo, se sim, na linha 3 retorna a restrição exists(le_1,pe_i). A linha 5 verifica se o atributo é do tipo alternativo, se sim, na linha 6 retorna a restrição disjoint(le_1,(pe_1|pe_2|...|pe_n). A linha 8 verifica se o atributo é do tipo único, se sim, na linha 9 retorna a restrição unique(le_1,pe_i). A linha 11 verifica se o atributo é do tipo identificador, se sim, na linha 12 retorna a restrição identifier(le_1,pe_i). Na linha 14 é finalizada a execução do algoritmo.

A Figura 29 ilustra o processo de mapeamento através da execução dos algoritmos envolvidos. Destaca-se que o mapeamento é iniciado pelo Algoritmo Mapeamento. Este aciona todos os outros três algoritmos que executam e retornam conforme a ordem: Algoritmo de Restrições de Cardinalidade, Algoritmo Restrições de Vértice e por fim, Algoritmo Restrições de Aresta.

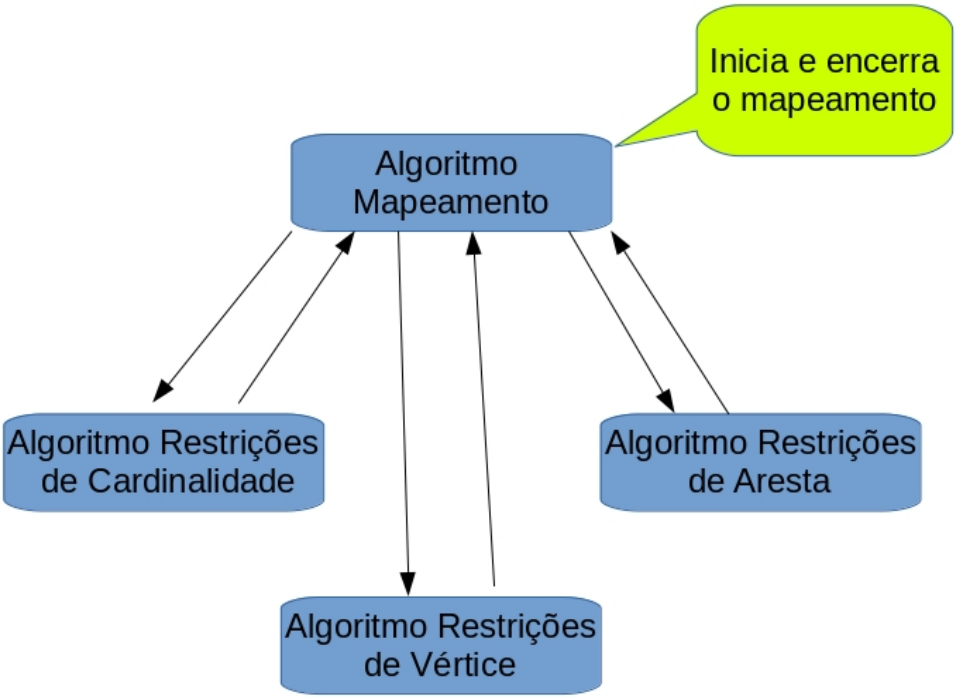

Figura 29: Exemplificação do processo de mapeamento através dos algoritmos

A execução retorna para o algoritmo Mapeamento, onde o mesmo retorna o modelo lógico de grafos e encerra o processo de mapeamento.

5.4 Proposta de conjunto de restrições de integridade para a linguagem Cypher do Neo4j

A linguagem Cypher oferece algumas restrições de integridade (apresentadas no apêndice A) que estão presentes no modelo lógico de grafos da proposta. Sendo assim, torna-se importante apresentar extensões para que o Cypher suporte as demais restrições de integridade da proposta.

Na Tabela 4, são apresentados comandos para restrições de rótulos de vértice. As restrições exists (a,S), unique (c,S) e identifier (d, S), já são suportadas por comandos nativos do Cypher. Sendo assim, estas restrições utilizam os mesmos comandos da linguagem.

Tabela 4: Restrições de Rótulos de Vértices respectivos comandos para o Cypher

Restrição de Rótulo de Vértice	**Comando ou extensão para o Cypher**
exists (v, s)	CREATE CONSTRAINT ON **(v:s)** ASSERT **exists (v.s)**
disjoint (v, S)	CREATE CONSTRAINT ON **(v:v)** ASSERT DISJOINT **(v.S_1, v.S_2, ..., v.S_N)**
unique (v, S)	CREATE CONSTRAINT ON (**v:v**) ASSERT **v.S** IS UNIQUE
identifier (v, S)	CREATE CONSTRAINT ON **(v:v)** ASSERT **v.S** IS NODE KEY
edgeExists (v, x)	CREATE CONSTRAINT ON **(v)-[x]-()** IS EXISTS
edgeRestricted (v, x)	CREATE CONSTRAINT ON **(v)-[x]-()** IS RESTRICTED

Na tabela anterior foram apresentados comandos do Cypher e extensões para o mesmo. As propostas de comandos são para as restrições: disjoint (v, S), edgeExists (v, x) e edgeRestricted (v, x).

A Tabela 5 apresenta comandos para restrições de rótulos de aresta. Apenas o comando para restrição exists (e, s) é nativa no Cypher.

Tabela 5: Restrições de Rótulos de Arestas respectivos comandos para o Cypher

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

As propostas de comandos são para as restrições de rótulos de arestas: disjoint (e, S), unique (e, t) e identifier (e,t) foram apresentadas anteriormente. As duas últimas só são ofertadas para rótulos de vértices.

As propostas de comandos para o Cypher permitirão recursos para oferecer mais consistência, se necessário, em um sistema Neo4j. As restrições não estão limitadas a apenas ao Cypher. O exemplo foi escolhido por se tratar de um dos sistemas mais utilizados atualmente.

Em busca de validação dos algoritmos de alto nível e de agilidade para mapeamento entre os níveis conceitual e lógico, e também para geração de comandos de restrições de integridade, apresenta-se um sistema para a proposta deste trabalho. Este sistema foi desenvolvido utilizando linguagem de programação e banco de dados relacional. O sistema foi desenvolvido para realização de projeto lógico de BDG a partir de esquema EB-ER e geração de comandos para as restrições na linguagem Cypher. Por questões de organização, o sistema é apresentado no Capítulo 6.

5.5 Considerações do capítulo

Neste capítulo foi apresentada a proposta de projeto lógico de banco de dados NoSQL de Grafos a partir de um modelo conceitual baseado no ER. O modelo conceitual é chamado de EB-ER. Seu esquema conceitual pode ser mapeado para um modelo lógico de grafos utilizando um conjunto de algoritmos de alto nível, ambos apresentados neste capítulo. Para os dois níveis, conceitual e lógico, foram propostas representações formais e diagramáticas.

O modelo lógico de grafos oferta restrições de integridade, algumas das quais são suportadas na linguagem Cypher do sistema de banco de dados de grafos Neo4j. Para as restrições não suportadas, são propostas extensões para a linguagem Cypher. E no Capítulo 7 e no apêndice B serão apresentados estudos de caso utilizando os recursos apresentados na proposta.

6 APLICAÇÃO PARA MAPEAMENTO AUTOMÁTICO DE ESQUEMA EB-ER PARA MODELO LÓGICO DE GRAFOS.

A aplicação oferta automatização de mapeamento de esquema conceitual EB-ER para esquema lógico de grafos. A aplicação possibilita a inserção de um esquema conceitual utilizando todos os recursos do EB-ER. O mapeamento entre o nível conceitual e o nível lógico é construído a partir do conjunto de algoritmos ilustrado em 5.3.

Conforme ilustrado no Capítulo 5, a proposta oferece comandos da linguagem Cypher, do Neo4j, a partir das restrições de integridade apresentadas no esquema lógico de grafos. Assim, esta funcionalidade também é ofertada pela aplicação deste trabalho. Na Figura 30 é ilustrada uma esquematização da aplicação.

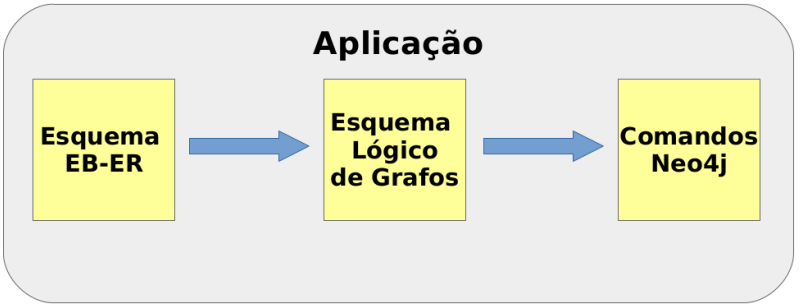

Figura 30: Esquematização da aplicação

A aplicação foi desenvolvida com a linguagem de programação Java, e para armazenamento dos dados dos esquemas conceitual e lógico, foi utilizado o SGBD MySQL Server.

Para apresentar o banco de dados de uma forma simplificada, são apresentados dois diagramas relacionais. O primeiro ilustrado na Figura 31 representa as tabelas utilizadas para esquema conceitual.

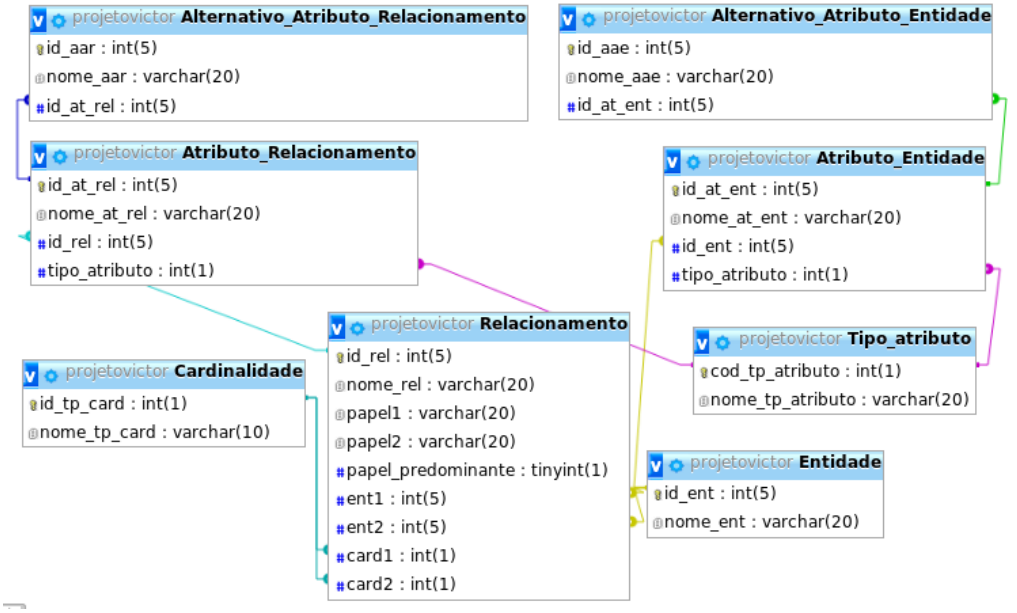

Figura 31: Tabelas utilizadas para armazenar o esquema EB-ER no Banco de Dados da aplicação

As tabelas utilizadas para o esquema conceitual são: Entidade, Atributo_Entidade, Atributo_Alternativo_Entidade, Relacionamento, Atributo_Relacionamento, Atributo_Alternativo_Relacionamento, Cardinalidade e Tipo_atributo.

O diagrama relacional que representa o esquema lógico de grafos é apresentado na Figura 32.

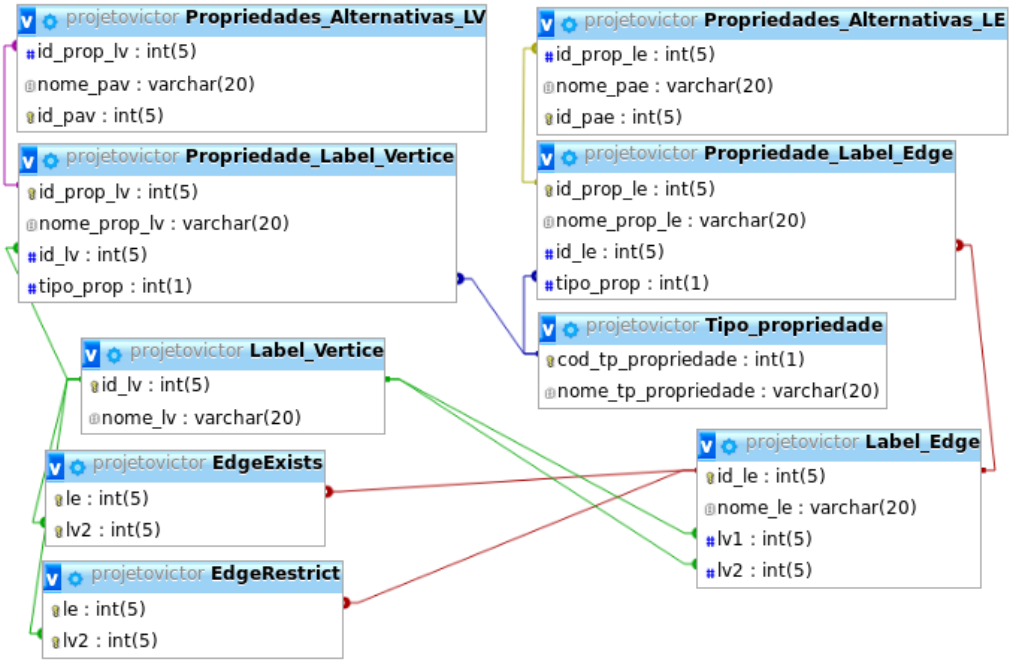

Figura 32: Tabelas utilizadas para armazenar o esquema lógico de grafos no Banco de Dados da aplicação

As tabelas utilizadas para o esquema lógico são: Label_Vertice, Propriedade_Label_Vertice, Propriedades_Alternativas_LV, Label_Edge, Propriedade_Label_Edge, Propriedades_Alternativas_LE, Tipo_propriedade, EdgeExists e EdgeRestrict. As restrições Exists, Disjoint, Unique e Identifier são consideradas a partir das tabelas de propriedades.

A aplicação apresenta diversas funcionalidades, como:

- Cadastro e visualização de entidades e seus respectivos atributos;
- Cadastro e visualização de relacionamentos e seus respetivos atributos;
- Função para excluir um esquema conceitual EB-ER;
- Criação do esquema lógico de grafos a partir de mapeamento do esquema EB-ER cadastrado;
- Visualização dos elementos do esquema lógico de grafos;
- Visualização das restrições de integridade do esquema lógico de grafos;
- Geração de comandos para linguagem Cypher, utilizada no sistema Neo4j.

A Figura 33 apresenta a tela inicial com o submenu Modelo Conceitual EB-ER selecionado.

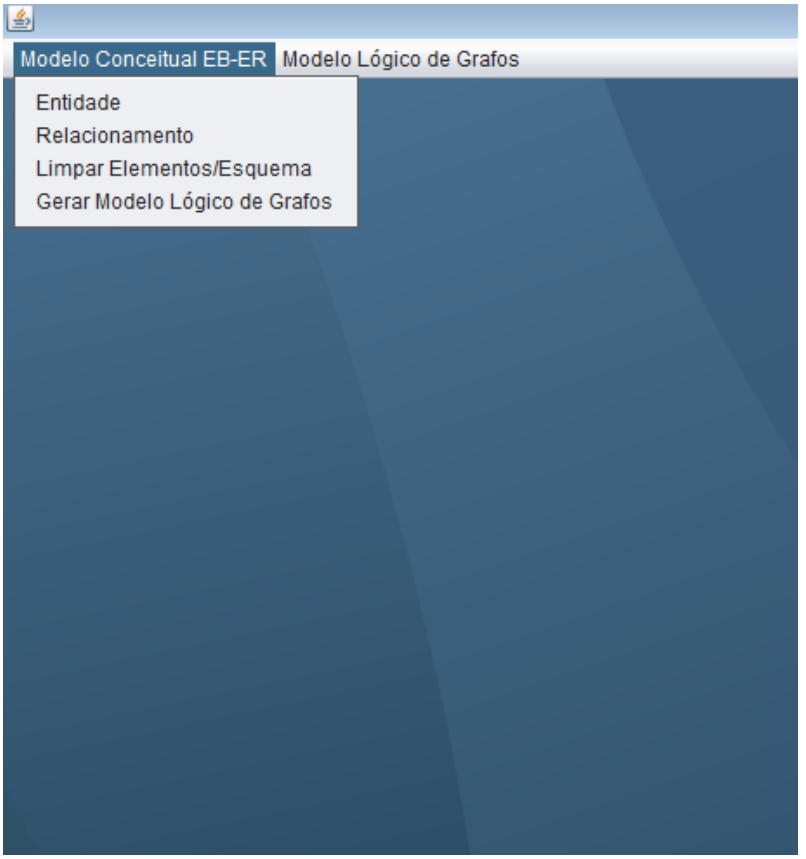

Figura 33: Aplicação: Submenu Modelo Conceitual EB-ER

As telas para gerenciamento de entidades e seus respectivos atributos são exibidas na Figura 34.

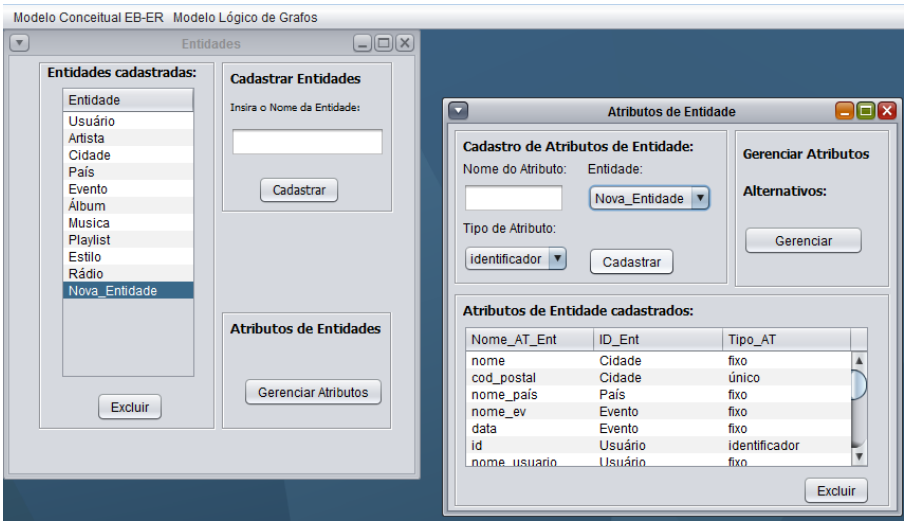

Figura 34: Telas Entidades e Atributos Entidade

As telas de gerenciamento dos relacionamentos do esquema EB-ER são semelhantes às telas para as entidades. A tela para visualização das restrições de rótulos de vértices é ilustrada na Figura 35.

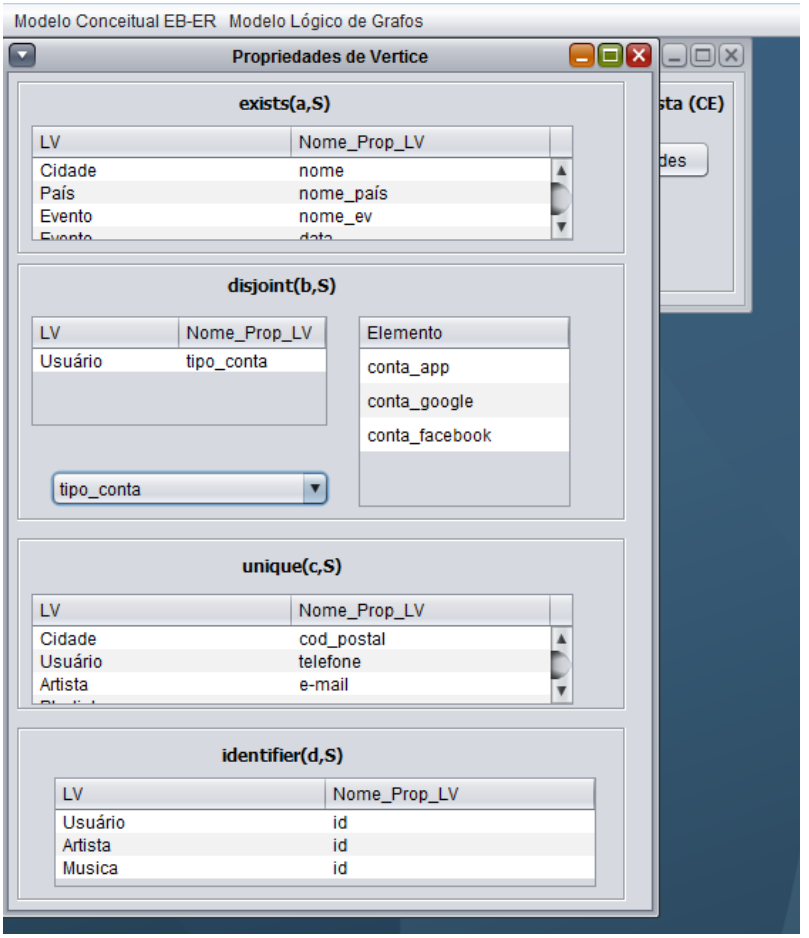

Figura 35: Tela de Restrições de Rótulos de Vértices

A partir das restrições de integridade do esquema lógico de grafos, a proposta oferece comandos de restrições para o sistema Neo4j. Na aplicação, os comandos são gerados em um arquivo de texto. Um exemplo de arquivo de texto contendo os comandos Neo4j é ilustrado na Figura 36.

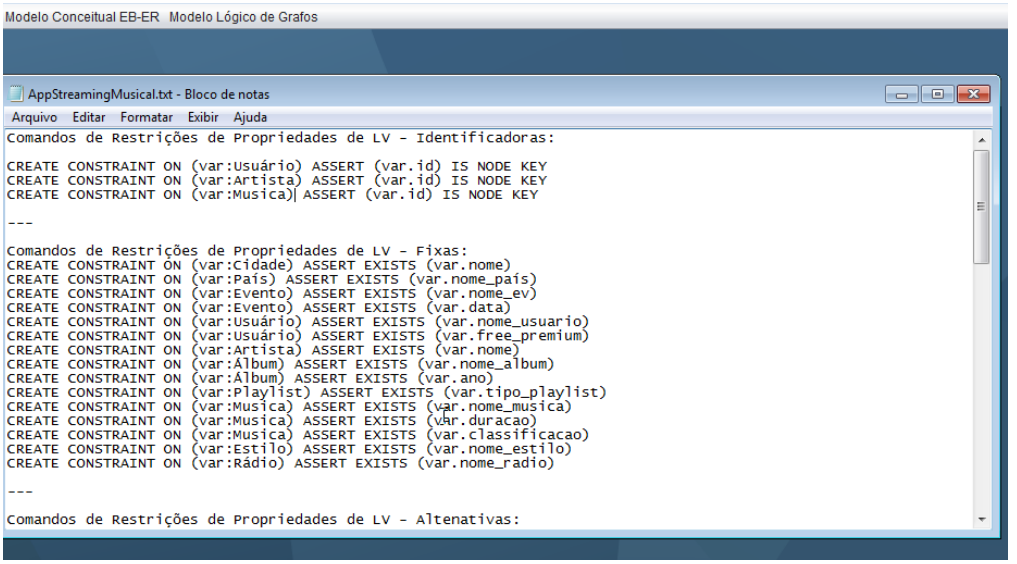

Figura 36: Comandos Neo4j gerados a partir das restrições de integridade do esquema lógico de grafos

A aplicação foi construída com o intuito de automatizar o esquema de mapeamento desta proposta. Sendo assim, para considerar uma aplicação totalmente apropriada para a comunidade.

7 ESTUDO DE CASO: APLICAÇÃO DE STREAMING DE MÚSICAS

A evolução das tecnologias de transmissão de dados e o surgimento de *smartphones*, *smartTVs*, *tablets* entre outros eletrônicos, motivaram o surgimento de diversas aplicações. Um tipo de aplicativo que cada vez mais é utilizado e suportado em diversos eletrônicos é o de *streaming* de música.

Uma aplicação de *streaming* de música oferece aos usuários milhares de músicas, vinculadas a álbuns ou aos seus artistas preferidos, de maneira ágil e intuitiva. As músicas podem ser pesquisadas pelo nome ou através do artista.

As aplicações de *streaming* musical normalmente oferecem serviços diferenciados para o tipo de conta utilizada pelo usuário. A maioria destas aplicações são do tipo *freemium*, onde possui acesso com contas do tipo "*free*", em que o usuário utiliza alguns recursos de maneira gratuita e do tipo "*premium*", onde o usuário tem acesso a todos recursos da plataforma.

Paralelamente, aplicações atuais como as redes sociais são muito utilizadas por artistas e seus seguidores. Alguns recursos destas aplicações também são inseridos nas aplicações de streaming musical e oferecem a possibilidade dos usuários seguirem seus artistas favoritos e salvar músicas e listas de músicas de sua preferência. Adicionalmente, as aplicações permitem que o usuário crie e compartilhe suas listas de músicas para reprodução, além de reproduzir listas de reprodução de outros usuários da plataforma.

Considerando a grande utilização e os diversos recursos que uma aplicação de *streaming* musical pode oferecer, foi escolhida uma aplicação deste tipo para representar como estudo de caso desta dissertação.

A seção a seguir apresenta um esquema conceitual EB-ER para uma aplicação do estudo de caso.

7.1 Esquema conceitual do estudo de caso

Considerando uma aplicação de *streaming* musical, um esquema conceitual baseado no EB-ER é apresentado na Figura 37.

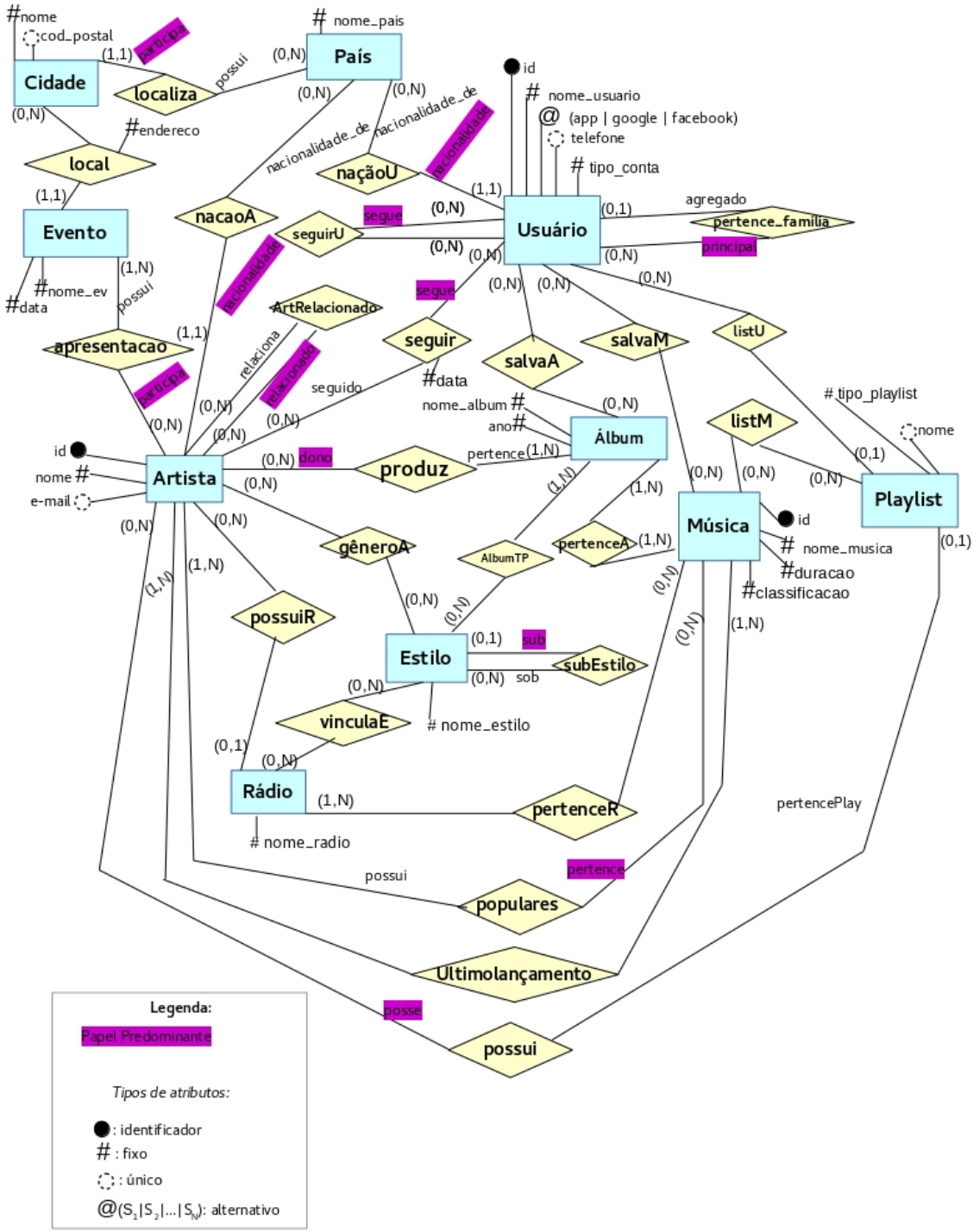

Figura 37: Esquema Conceitual para aplicação de streaming de músicas

O esquema conceitual, ilustrado na figura anterior, utiliza todos recursos do EB-ER. As entidades refletem informações mínimas para aplicação. Os relacionamentos definidos são considerados essenciais para aplicação. O papel dominante foi definido em alguns relacionamentos, como por exemplo “seguir” entre as entidades Usuário e Artista, que pode ser considerado mais indicado o papel dominante no lado de Usuário. Outro

exemplo de papel dominante é utilizado no relacionamento "naçãoU", que entre as entidades País e Usuário, foi definida a configuração no lado de Usuário. No relacionamento “vinculaE”, que possui cardinalidade idêntica nos dois lados, não foi definido um papel dominante, assim não terá uma configuração para decisão durante o mapeamento para o nível lógico. Um atributo alternativo foi utilizado na entidade Usuário, em que este deve ter um dos tipos de acesso à aplicação, configuração muito utilizada em aplicações. Atributos do tipo fixo foram definidos em várias entidades, assim, garantirá que esta propriedade deverá existir nos próximos níveis de modalagem.

7.2 Esquema Lógico do estudo de caso

Após definição do esquema conceitual, a próxima fase é realizar o projeto lógico, utilizando os conceitos da proposta desta dissertação. O esquema lógico gerado é ilustrado na Figura 38.

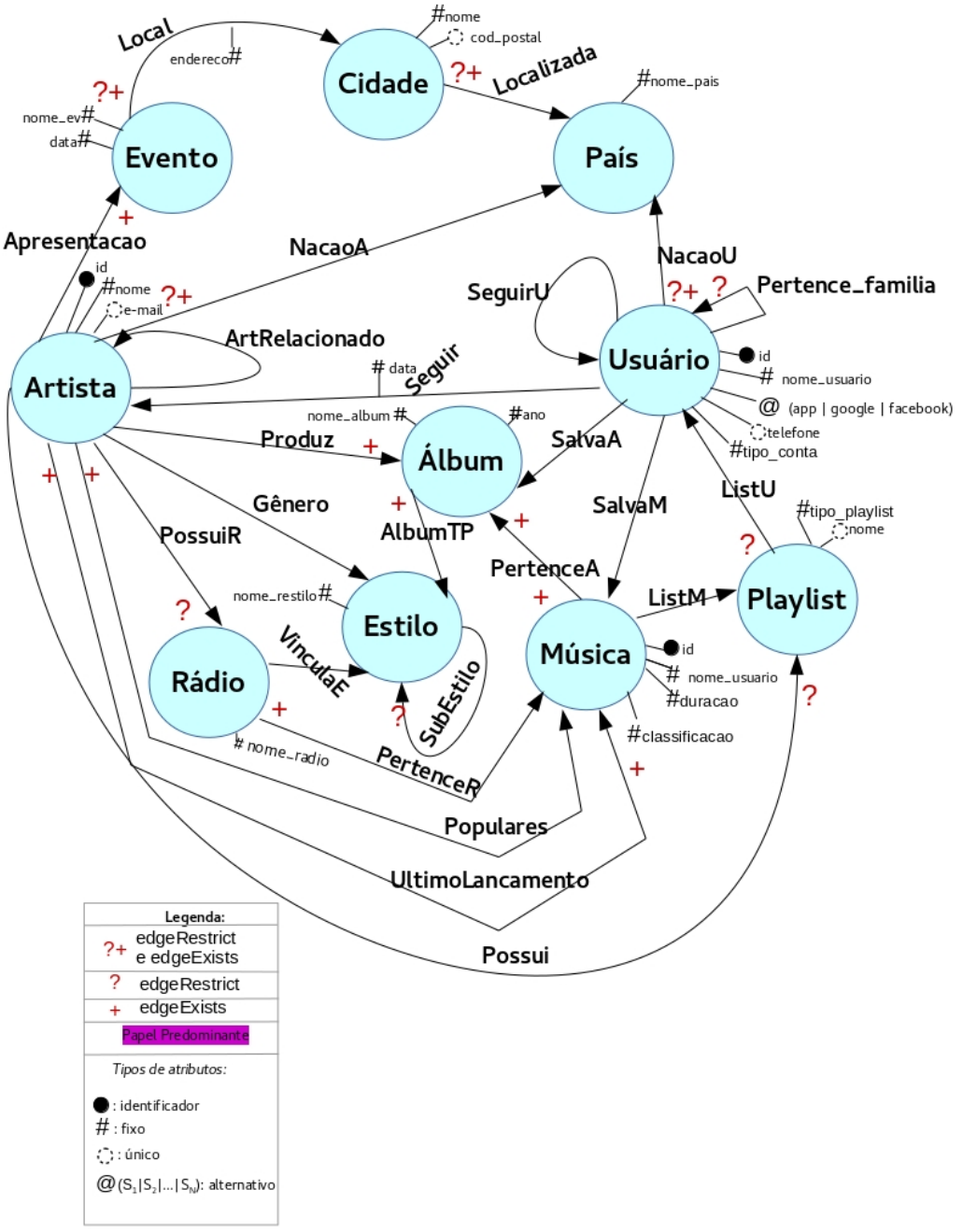

Figura 38: Esquema Lógico de Grafos para aplicação de streaming de músicas

O esquema lógico gerado reflete todos rótulos de vértices, rótulos de arestas, propriedades e restrições de integridade. O papel dominante no relacionamento “seguir” do esquema conceitual garantiu o sentido do rótulo de aresta “seguir” iniciando em Usuário e encerrando em Artistas. O rótulo de aresta "naçãoU" tem o rótulo de vértice inicial Usuário e rótulo de vértice final País, o sentido foi garantido com a definição do papel dominante no esquema conceitual. As propriedades mantiveram as características

oriundas dos atributos. As restrições de rótulos de vértices vinculadas a rótulos de arestas são exibidas no esquema lógico.

7.3 Conjunto de Restrições para comandos Neo4j

Nesta seção é apresentado o conjunto de comandos gerados a partir das restrições do esquema lógico do estudo de caso. Os comandos são baseados nas definições da seção 5.4, onde se listam os comandos já ofertados pelo Neo4j, e também são sugeridos comandos para as demais restrições.

Comandos de Restrições de Propriedades de LV - Identificadoras:
CREATE CONSTRAINT ON (var:Usuário) ASSERT (var.id) IS NODE KEY
CREATE CONSTRAINT ON (var:Artista) ASSERT (var.id) IS NODE KEY
CREATE CONSTRAINT ON (var:Musica) ASSERT (var.id) IS NODE KEY

Comandos de Restrições de Propriedades de LV - Fixas:
CREATE CONSTRAINT ON (var:Cidade) ASSERT EXISTS (var.nome)
CREATE CONSTRAINT ON (var:País) ASSERT EXISTS (var.nome_país)
CREATE CONSTRAINT ON (var:Evento) ASSERT EXISTS (var.nome_ev)
CREATE CONSTRAINT ON (var:Evento) ASSERT EXISTS (var.data)
CREATE CONSTRAINT ON (var:Usuário) ASSERT EXISTS (var.nome_usuario)
CREATE CONSTRAINT ON (var:Usuário) ASSERT EXISTS (var.free_premium)
CREATE CONSTRAINT ON (var:Artista) ASSERT EXISTS (var.nome)
CREATE CONSTRAINT ON (var:Álbum) ASSERT EXISTS (var.nome_album)
CREATE CONSTRAINT ON (var:Álbum) ASSERT EXISTS (var.ano)
CREATE CONSTRAINT ON (var:Playlist) ASSERT EXISTS (var.tipo_playlist)
CREATE CONSTRAINT ON (var:Musica) ASSERT EXISTS (var.nome_musica)
CREATE CONSTRAINT ON (var:Musica) ASSERT EXISTS (var.duracao)
CREATE CONSTRAINT ON (var:Musica) ASSERT EXISTS (var.classificacao)
CREATE CONSTRAINT ON (var:Estilo) ASSERT EXISTS (var.nome_estilo)
CREATE CONSTRAINT ON (var:Rádio) ASSERT EXISTS (var.nome_radio)

Comandos de Restrições de Propriedades de LV - Altenativas:
CREATE CONSTRAINT ON (var:Usuário) ASSERT DISJOINT (var.conta_app var.conta_google var.conta_facebook)

Comandos de Restrições de Propriedades de LV - Únicas:

CREATE CONSTRAINT ON (var:Cidade) ASSERT (var.cod_postal) IS UNIQUE

CREATE CONSTRAINT ON (var:Usuário) ASSERT (var.telefone) IS UNIQUE

CREATE CONSTRAINT ON (var:Artista) ASSERT (var.e-mail) IS UNIQUE

CREATE CONSTRAINT ON (var:Playlist) ASSERT (var.nome) IS UNIQUE

Comandos de Restrições de Propriedades de LE - Identificadoras:

Comandos de Restrições de Propriedades de LE - Fixas:

CREATE CONSTRAINT ON ()-[var:local]-() ASSERT EXISTS (var.endereco)

CREATE CONSTRAINT ON ()-[var:seguir]-() ASSERT EXISTS (var.data)

Comandos de Restrições de Propriedades de LE - Altenativas:

Comandos de Restrições de Propriedades de LE - Únicas:

Comandos de Restrições de LE Obrigatórios para LV:

CREATE CONSTRAINT ON (Usuário)-[nacaoU]-() IS EXISTS

CREATE CONSTRAINT ON (Usuário)-[dominante]-() IS EXISTS

CREATE CONSTRAINT ON (Artista)-[nacaoA]-() IS EXISTS

CREATE CONSTRAINT ON (Artista)-[populares]-() IS EXISTS

CREATE CONSTRAINT ON (Artista)-[ultimoLancamento]-() IS EXISTS

CREATE CONSTRAINT ON (Cidade)-[localiza]-() IS EXISTS

CREATE CONSTRAINT ON (Evento)-[local]-() IS EXISTS

CREATE CONSTRAINT ON (Evento)-[apresentacao]-() IS EXISTS

CREATE CONSTRAINT ON (Álbum)-[cria]-() IS EXISTS

CREATE CONSTRAINT ON (Álbum)-[pertenceA]-() IS EXISTS

CREATE CONSTRAINT ON (Álbum)-[albumTP]-() IS EXISTS

CREATE CONSTRAINT ON (Musica)-[pertenceA]-() IS EXISTS

CREATE CONSTRAINT ON (Musica)-[ultimoLancamento]-() IS EXISTS

CREATE CONSTRAINT ON (Rádio)-[pertenceR]-() IS EXISTS

CREATE CONSTRAINT ON (Nova_Entidade)-[dominante]-() IS EXISTS

Comandos de Restrições de LE Restritas para LV:
CREATE CONSTRAINT ON (Usuário)-[pertence_familia]-() IS RESTRICTED
CREATE CONSTRAINT ON (Cidade)-[localiza]-() IS RESTRICTED
CREATE CONSTRAINT ON (Evento)-[local]-() IS RESTRICTED
CREATE CONSTRAINT ON (Playlist)-[listaU]-() IS RESTRICTED
CREATE CONSTRAINT ON (Playlist)-[possui]-() IS RESTRICTED
CREATE CONSTRAINT ON (Estilo)-[subEstilo]-() IS RESTRICTED
CREATE CONSTRAINT ON (Rádio)-[possuirR]-() IS RESTRICTED
------FIM-------

Os comandos listados acima foram gerados automaticamente em um arquivo texto, este gerado pela aplicação desenvolvida para esta dissertação. Os comandos possuem a sintaxe apresentada na seção 5.4, em que apresenta todos comandos e propostas para cobrir as restrições de nível lógico na linguagem Cypher do sistema Neo4j.

Alguns tipos de comandos para rótulos de arestas não foram gerados pois não existiam restrições deste tipo no nível lógico, por exemplo, propriedades identificadoras, alternativas e únicas.

O rótulo de vértice Evento possui dois comandos vinculados ao rótulo de aresta “local”, em um se obriga a existência e em outro restringe-se em até uma associação. Estas restrições foram ilustradas no nível lógico, onde se tinha "?+".

7.4 Considerações do capítulo

Analisando o esquema lógico do estudo de caso, pode-se verificar que, para as entidades, criaram-se os rótulos de vértices. Para os relacionamentos criaram-se os rótulos de arestas. Enquanto que, para os atributos, criaram-se as propriedades.

As restrições do esquema lógico, representadas pelos símbolos “?+”, “?” e “+”, foram geradas a partir das cardinalidades dos relacionamentos do esquema conceitual. Os sentidos dos rótulos de aresta foram definidos de maneira aleatória, ou então através do papel dominante definido no relacionamento do esquema conceitual.

O estudo de caso abordou uma aplicação com alto grau de relacionamentos entre as entidades, característica de aplicações que normalmente utilizam bancos de dados de grafos. Um estudo de caso para um cenário acadêmico, com menor grau de relacionamento entre entidades, é apresentado no apêndice B.

8 CONSIDERAÇÕES FINAIS

Os Bancos de Dados NoSQL estão cada vez mais presentes nas aplicações atuais, que normalmente usam o armazenamento de dados na Web, utilizando predominantemente a computação na nuvem. Os Bancos de Dados Orientados a Grafos (BDGs), pertencentes ao grupo de BDs NoSQL, possuem diversas opções de sistemas gerenciadores e também, são pesquisados em diversas áreas. Esta categoria é amplamente utilizada quando se possui um alto grau de relacionamento entre os dados. Alguns exemplos são aplicações para redes sociais, gerenciamento de dados geográficos, entre outras.

O projeto de bancos de dados tradicionais já se encontra consolidado na bibliografia. Enquanto que para projeto de BDs NoSQL, o tema ainda está em discussão. Considerando os BDGs, pode-se considerar que pela existência de sistemas que utilizam diferentes modelos de dados de grafos, e também pela oferta de flexibilidade que normalmente as aplicações atuais necessitam.

Recentes trabalhos apresentam propostas de projetos para BDGs. Percebe-se que os trabalhos ainda apresentam alguns desafios para o projeto lógico. Alguns não detalham o mapeamento do nível conceitual para o nível lógico, e/ou não abordam questões consolidadas para restrições de integridade.

O presente trabalho aborda uma proposta de projeto lógico de banco de dados NoSQL orientados a grafos. O mapeamento é realizado a partir de um esquema conceitual EB-ER, modelo conceitual baseado no modelo ER. Este mapeamento é realizado utilizando um conjunto de algoritmos de alto nível. Estes algoritmos fornecerão um esquema lógico de grafos. Este esquema ilustrará um conjunto de restrições de integridade, que quando o banco de dados de grafos for implementado, o mesmo deverá respeitar estas regras.

Algumas das restrições de integridade do modelo lógico de grafos desta proposta não são suportadas pela linguagem Cypher do Neo4j (atualmente, um dos sistemas de BDG mais utilizados). Sendo assim, no Capítulo 5, são propostos comandos para esta linguagem.

Para ilustrar a proposta deste trabalho, apresenta-se um estudo de caso para uma aplicação de *streaming* de músicas, a qual é amplamente utilizada na atualidade e requer de diversos recursos, com uma considerável complexidade de relacionamentos, tornado um banco de dados NoSQL orientado a grafos uma considerável alternativa para implementação. Adicionalmente, é apresentado um estudo de caso para um cenário acadêmico, no apêndice B, onde é possível analisar a proposta em uma aplicação com menor complexidade. Os dois estudos de caso utilizam todos os conceitos abordados nesta proposta: modelo conceitual EB-ER, modelo lógico de grafos e geração de comandos ofertados pelo Neo4j, e complementarmente, os comandos propostos nesta dissertação.

Esta dissertação se baseou em diversos trabalhos: Poffo *et al.* (2016), Bugiotti *et al.* (2014), Lima *et al.* (2015), Pokorný (2016), Ghrab *et al.* (2016) e Virgilio *et al.* (2014) que enfatizavam a importância de um esquema de um banco de dados e apresentavam um projeto de banco de dados NoSQL a partir de um esquema conceitual de dados. Assim, destaca-se a importância de, pelo menos, um esquema mínimo de banco de dados e o projeto conceitual, onde este deve oferecer modelagem lógica para qualquer tipo de aplicação.

8.1 Contribuições do trabalho

Nesta seção serão listadas as principais contribuições deste trabalho:

- **Restrições de integridades para modelo lógico de grafos:** Diversos trabalhos oferecem definições de modelos lógicos, porém nestas definições existem poucas contribuições para restrições de integridade. A proposta apresentou novas restrições de integridade, que inclusive cobrem todas as regras definidas no projeto conceitual.
- **Novas propostas de extensão ao modelo ER:** O modelo ER e sua extensão EER já estão consolidadas no ambiente acadêmico. Porém foram propostas duas novas extensões para o ER ou qualquer outro modelo conceitual de dados semelhante. A primeira proposta é um atributo alternativo, onde o projetista define um atributo obrigatório dentro de uma lista de possíveis valores pré-definidos. Já a segunda foi a marcação de um papel dominante em um

relacionamento de grau dois, ou do tipo recursivo, onde este fornece semântica para definição de direção de uma aresta no momento da modelagem lógica.

- **Desenvolvimento de ferramenta de automatização do mapeamento conceitual-lógico:** Nos trabalhos de Lima *et al.* (2015) e Poffo *et al.* (2016), foram mencionadas como projeções futuras o desenvolvimento de uma ferramenta para realizar o projeto lógico de maneira automática, que foi produzida nesta dissertação.
- **Discussão acerca de um esquema mínimo de banco de dados** – Não se trata de uma restrição limitante, onde nada de diferente poderá ser feito. Na verdade trata-se de um esquema mínimo requirido para oferecer certa consistência necessária para algum tipo de manipulação ou integração de dados posterior. O esquema mínimo é construído no nível conceitual e modelado em restrições de integridade no nível lógico de grafos. Em nível de banco de dados, a linguagem de consulta necessitará suportar as restrições de integridade do nível lógico.

8.2 Trabalhos futuros

Para trabalhos futuros, são sugeridas três atividades: 1) Apresentar inclusão de comandos em uma linguagem de sistema de banco de dados de grafos (Neo4J ou outro) que suportem todas as restrições de integridades oferecidas no modelo lógico de grafos apresentado neste trabalho; 2) Realizar projeto lógico de esquema conceitual EB-ER para outros modelos de dados NoSQL, assim para assegurar o conceito de um modelo conceitual independente do tipo de aplicação; 3) Para aplicação desenvolvida, adicionar recursos para que aplicação seja mais intuitiva para o usuário como, por exemplo:

- Possibilitar a criação do esquema conceitual através de recursos diagramáticos apresentados na seção 5.1.1. Acredita-se, que desta forma, o usuário poderá representar um esquema conceitual de maneira intuitiva.
- Representação do esquema lógico de grafos gerado através de um grafo, que utilize os recursos ilustrados na seção 5.2.1.
- Inclusão de mapeamento das restrições de integridade do esquema de lógico de grafos para outras linguagens de sistemas de banco de dados de grafos.

REFERÊNCIAS BIBLIOGRÁFICAS

Almeida, M. B. 2002. Uma introdução ao XML, sua utilização na Internet e alguns conceitos complementares. Ciência da informação 31.2 (2002): 5-13.

Angelotti, E. S. 2010. Banco de dados. – Curitiba: Editora do Livro Técnico, 2010.

Angles, R. 2012. A Comparison of Current Graph Database Models. In 2012 IEEE 28th International Conference on Data Engineering Workshops (ICDEW), pages 171#177, April 2012.

Badia, A. 2002. Conceptual Modeling for Semistructured Data. In Proceedings of the 3rd International Conference on Web Information Systems Engineering Workshops (WISE 2002 Workshops), p. 170-177. Singapore, December 2002.

Banerjee, S., Shaw, R., Sarkar, A. and Debnath, N. C. 2015. Towards logical level design of Big Data. In: Towards logical level design of Big Data. In 2015 IEEE 13th International Conference on Industrial Informatics (INDIN) p. 1665-1671.

Bugiotti, F., Cabibbo, L., Atzeni, P. and Torlone, R. 2014. Database design for NoSQL systems. In: International Conference on Conceptual Modeling. Springer International Publishing. p. 223-231.

Chartrand, . 2006. Introduction to graph theory. Tata McGraw-Hill Education, 2006.

Chebotko, A., Kashlev, A. and Lu, S. 2015. A Big Data Modeling Methodology for Apache Cassandra. In: IEEE. Big Data (BigData Congress), 2015 IEEE International Congress. [S.l.], 2015. p. 238–245.

Chen, P. P. 1976. The entity-relationship model—toward a unified view of data. In: ACM Transactions on Database Systems (TODS) 1.1 : 9-36.

Daniel, G., Suný, G. and Cabot J. 2016. UMLtoGraphDB: Mapping Conceptual Schemas to Graph Databases. The 35th International Conference on Conceptual Modeling (ER2016), Nov 2016, Gifu, Japan.

Dobbie, G., Xiaoying, W., Ling, T.W. and Lee, M.L. 2000. ORA-SS: An Object-Relationship-Attribute Model for Semi-Structured Data. In: Technical Report, Department of Computer Science, National University of Singapore. December, 2000

Elmasri, R. and Navathe, S. B. 2011. Fundamentals of Database Systems. 6th edition, Pearson Addison Wesley, 2011.

Erven, G. C. G. V. 2015. MDG-NoSQL: Modelo de Dados para Bancos NoSQL Baseados em Grafos. 105 p. Dissertação (Mestrado) Universidade de Brasília, Brasília, 2015.

Feng, W., Gu, P., Zhang, C. and Zhou, K. 2015. Transforming UML Class Diagram into Cassandra Data Model with Annotations. In: 2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity). IEEE. p. 798-805.

Ghrab, A., Romero, O., Skhiri, S., Vaisman, A. and Zimányi, E. 2016. Grad: On graph database modeling arXiv preprint arXiv:1602.00503 (2016).

Heuser, C. A 2008. Projeto de Banco de Dados. 6ª edição. Porto alegre: Editoria Bookman, 2008.

Izquierdo, J. L. C. and Cabot, J. 2013. Discovering implicit schemas in JSON data. International Conference on Web Engineering. Springer, Berlin, Heidelberg, 2013.

Kapsammer, E., Kusel, A., Mitsch, S., Proll, B., Retschitzegger, W., Schwinger, W., Schonbock, J., Wimmer, M., Wischenbart M. and Lechner, S. 2012. User profile integration made easy: Model-driven extraction and transformation of social network schemas. In: Proceedings of the 21st International Conference on World Wide Web . New York, NY, USA: ACM, 2012. (WWW ’12 Companion), p. 939–948. ISBN 978-1-4503-1230-1.

Kaur, K. and Rani, R. 2013. Modeling and querying data in NoSQL databases. In: Big Data, 2013 IEEE International Conference on IEEE. p. 1-7.

Lima, C. and Mello, R. S. 2015. A Workload-Driven Logical Design Approach for NoSQL Document Databases. In: XVII International Conference on Information Integration and Web-based Applications & Services (iiWAS 2015), 2015, Bruxelas. Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services, Brussels, Belgium, December 11-13, 2015. New York: ACM, 2015.

Necaský, M. 2006. Conceptual Modeling for XML: A Survey. In: Dateso.

Okman L., Gal-oz. N., Gonen, Y., Gudes E. and Abramov, J. 2011. Security issues in NoSQL databases. In Proc. of the 10th TrustCom Conference, pages 541–547. IEEE, 2011.

Penteado, R. R., Schroeder, R., Hoss, D., Nande, J., Maeda, R. M., Couto, W. O. and Hara, C. S. 2014. Um estudo sobre bancos de dados em grafos nativos. X Escola Regional de Banco de Dados (ERBD). 2014

Poffo, J. P. and Mello R. S. 2016. A Logical Design Process for Columnar Databases. In: International Conference on Internet and Web Applications and Services (ICIW), 2016, Valência. 11th International Conference on Internet and Web Applications and Services, 2016.

Pokorný, J. 2016. Conceptual and Database Modelling of Graph Databases. In: Proceedings of the 20th International Database Engineering & Applications Symposium. ACM, 2016. p. 370-377.

Pokorný, J, Valenta, M. & and Kovačič, J. 2017. Integrity constraints in graph databases. Procedia Computer Science, v. 109, p. 975-981, 2017.

Rabuzin, K., Sestak, M. and Konecki, M. 2016. "Implementing UNIQUE Integrity Constraint in Graph Databases." The Eleventh International Multi-Conference on Computing in the Global Information Technology.

Sadalage, P. J. and Fowler, M. 2013. NoSQL essencial: um guia conciso para o mundo emergente da persistência poliglota. São Paulo: Novatec, 2013. 220p.

Schwartz, B. 2015. Schemaless Databases don’t exist. Acessado em 14 de novembro de 2017.Disponível em: https://vividcortex.com/blog/2015/02/24/schemaless-databases-dont-exist.

Sestak, M., Rabuzin, K. and Novak, M. 2016. "Integrity constraints in graph databases-implementation challenges." Central European Conference on Information and Intelligent Systems. Faculty of Organization and Informatics Varazdin, 2016.

Silberschatz, A, Korth, H. F. and Sudarshan, S. 2012. Sistema de banco de dados. 6. ed. São Paulo, SP: Pearson Makron Books, 2012. 860 p. ISBN 9788534610735.

Souza, A., Prado, E., Sun and V. Fantinato, M. 2014. “Critérios para Seleção de SGBD NoSQL: o Ponto de Vista de Especialistas com base na Literatura.” In Simpósio Brasileiro de Sistemas da Informação(SBSI) 2014.

Stonebraker, M. 2012. What Does ’Big Data’ Mean. Acessado em 05 de janeiro de 2018. Disponível em: https://cacm.acm.org/blogs/blog-cacm/155468-what-does-big-data-mean/fulltext.

Tiwari, S. 2011. “Professional NoSQL”, Indianapolis: John Wiley & Sons, Inc.

Virgilio, R., Maccioni, A. and Torlone, R. 2014. "Model-driven design of graph databases." International Conference on Conceptual Modeling. Springer, Cham, 2014.

Wischenbart, M., Mitsch, S., Kapsammer, E., Kusel, A., Pröll, B., Retschitzegger, W., Schwinger, W., Schonbock, J., Wimmer, M. and Lechner, S. 2012. User profile integration made easy: model-driven extraction and transformation of social network schemas. In Proceedings of the 21st International Conference on World Wide Web (pp. 939-948). ACM.

Wood, P. T. 2012. Query languages for graph databases. In: ACM SIGMOD Record 41.1 p. 50-60.

APÊNDICE A – Restrições de integridade do modelo de banco de dados de grafos suportadas na linguagem Cypher

Serão apresentadas as restrições de integridade do modelo de banco de dados de grafos (5.3) que podem ser suportadas na linguagem Cypher, utilizada para o sistema Neo4j. Este apêndice é construído a partir da documentação[9] da linguagem Cypher. Ressalta-se que algumas restrições estão disponíveis na versão *Enterprise*, onde se exige licença de uso.

1 Atributo identificador

Pode-se definir um (ou mais atributos) como atributo "chave". A sintaxe é apresentada a seguir:

```
CREATE CONSTRAINT ON (LV:Rótulo_de_Vértice) ASSERT (LV.nome_atributo) IS NODE KEY
```

No primeiro par de parênteses é definido o rótulo de vértice/aresta que receberá a restrição, sendo que LV é uma variável representando o rótulo para simplificar o comando. No segundo par de parênteses define-se o(s) atributo(s) de LV que receberá(ão) a restrição "NODE KEY".

2 Atributo único

Pode-se definir um atributo de um rótulo de vértice/aresta com valor único. A sintaxe é apresentada a seguir:

```
CREATE CONSTRAINT ON (LV:Rótulo_de_Vértice) ASSERT LV.nome_atributo IS UNIQUE
```

9 http://neo4j.com/docs/developer-manual/current/cypher/

No primeiro par de parênteses é definido o rótulo de vértice/aresta que receberá a restrição, sendo que LV é uma variável representando o rótulo para simplificar o comando. Após “ASSERT” é d efinido o atributo que receberá a restrição de “UNIQUE”.

3 Atributo fixo

Define-se um atributo fixo para um rótulo de vértice, como um atributo com a sua existência obrigatória. A sintaxe é apresentada a seguir.

CREATE CONSTRAINT ON (LV:Rótulo_de_Vértice) ASSERT exists(LV.nome_atributo)

No primeiro par de parênteses é definido o rótulo de vértice que receberá a restrição, sendo que LV é uma variável representando o rótulo para simplificar o comando. No segundo par de parênteses define-se o(s) atributo(s) de LV que deverá(ão) possuir um valor neste determinado campo.

Define-se um atributo fixo para um rótulo de aresta, como um atributo com a sua existência obrigatória. A sintaxe é apresentada a seguir.

CREATE CONSTRAINT ON ()-[LE:Rótulo_Aresta]-() ASSERT exists(LE.nome_atributo)

No primeiro par de colchetes é definido o rótulo de aresta que receberá a restrição, sendo que LE é uma variável representando o rótulo para simplificar o comando. No último par de parênteses define-se o(s) atributo(s) de LE que deverá(ão) possuir um valor neste determinado campo.

Das restrições apresentadas neste apêndice, apenas a de atributo único é ofertada na versão *Community* . O restante é suportado apenas na versão *Enterprise*.

APÊNDICE B – Estudo de caso de um cenário acadêmico

Neste apêndice será apresentado um estudo de caso de projeto de banco de dados para uma aplicação de uma faculdade. Este estudo busca ilustrar o projeto de pesquisa de uma maneira sucinta e prática.

Considera-se que serão armazenadas as informações dos alunos e professores, e a cidades em que cada um reside. Para cadastro do aluno, serão armazenados os dados cpf, nome e meio de contato, que pode ser através de telefone ou e-mail. Os professores também deverão ter em seu respectivo cadastro com cpf, nome e e-mail, sendo que o último não poderá ser utilizado por mais de um profissional. Para as cidades, além do nome, deverá ser utilizado o CEP oficial para cadastro.

Para cada curso deverá possuir um nome e uma identificação interna. Além disso, o curso possuirá a informação do seu reconhecimento (através de órgão regulamentador). O reconhecimento também possuirá uma identificação e obrigatoriamente o ano em que o mesmo foi realizado. O curso também possuirá as informações de todos os alunos e ano de matrícula de cada um, o professor coordenador e as disciplinas vinculadas. Para cada disciplina será necessário realizar o armazenamento, se houver, de pré-requisito de outra disciplina e o professor que a leciona.

Através dos requisitos apresentados, foi construído um esquema conceitual EB-ER, o mesmo é ilustrado a Figura 39.

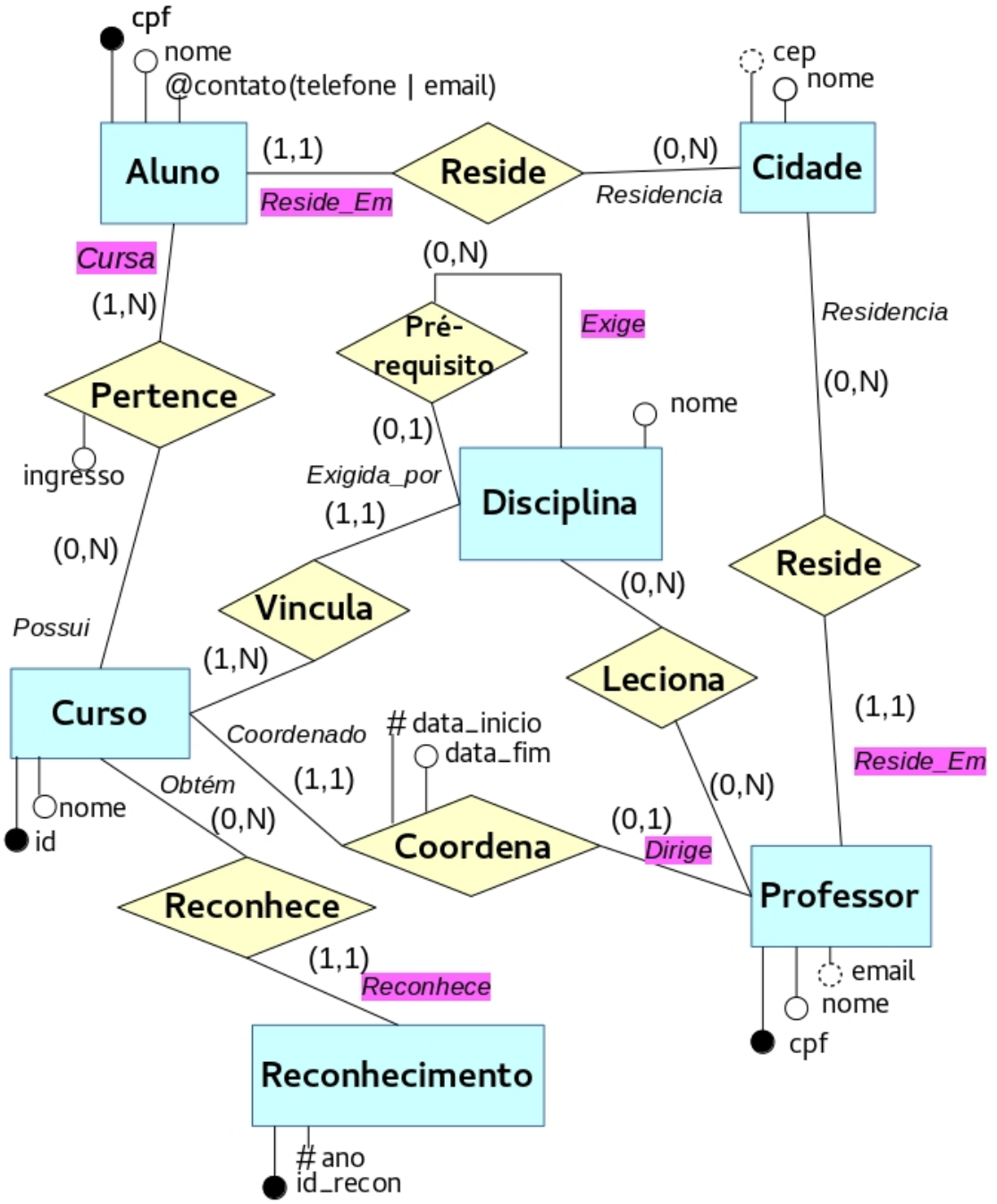

Figura 39: Esquema conceitual EB-ER do estudo de caso

B.1 Mapeamento do esquema conceitual para um modelo lógico de grafos

A ilustração do resultado do mapeamento do esquema conceitual EB-ER apresentado na Figura 39 para o modelo lógico de grafos é apresentado nesta seção. O diagrama do modelo lógico de grafo é apresentado na Figura 40.

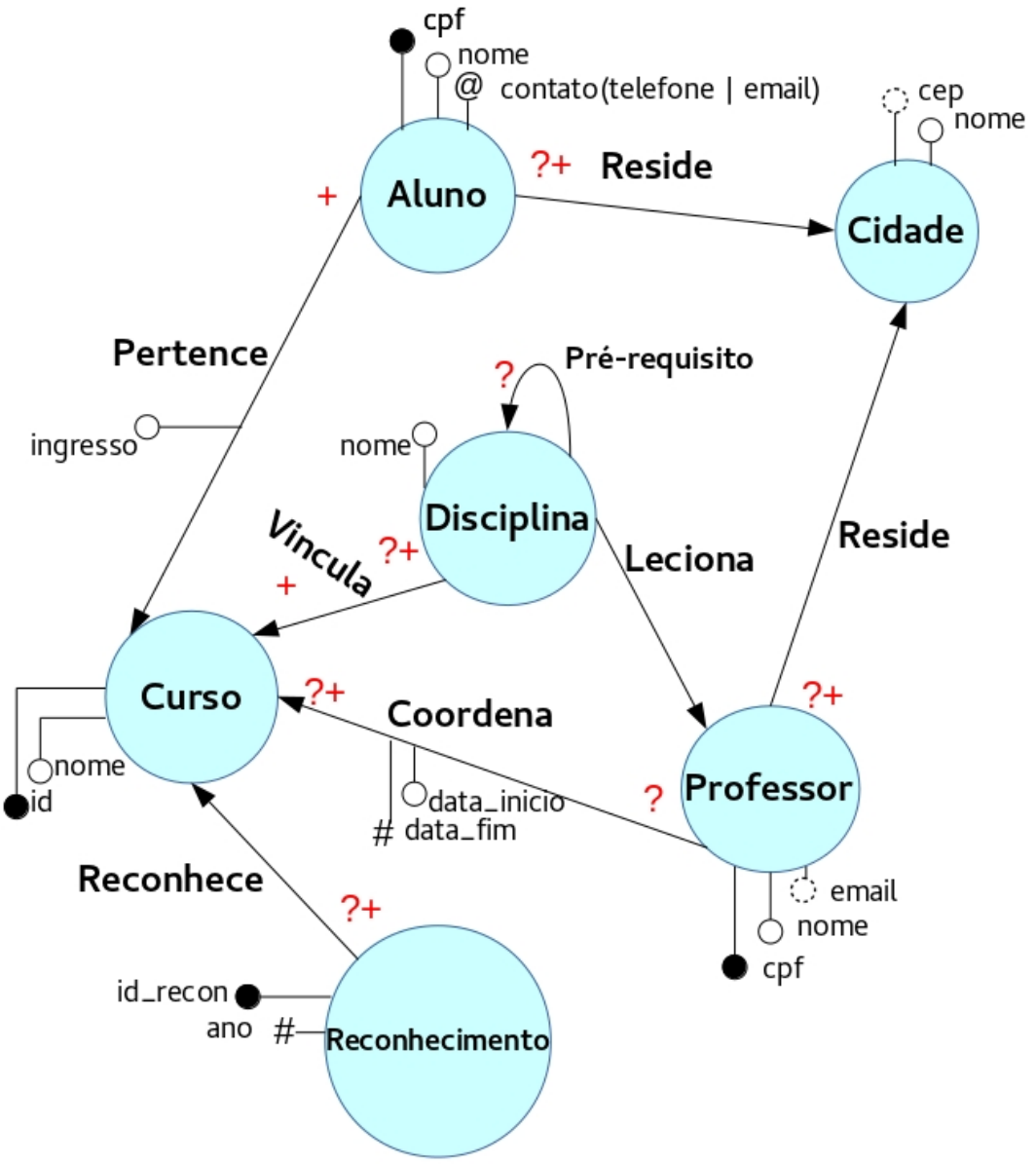

Figura 40: Esquema lógico de grafos do estudo de caso

Um exemplo de banco de dados de grafos para o estudo de caso é apresentado a na Figura 41. Sua criação foi realizada no Neo4j e os comandos utilizados são apresentados na seção seguinte.

Na imagem anterior, os rótulos de vértices estão em cores diferentes, e através destas cores são representados cada vértice no BD. Os rótulos de arestas são identificados com seus nomes em suas respectivas arestas. A seguir é apresentado o esquema do BD gerado pelo Neo4j.

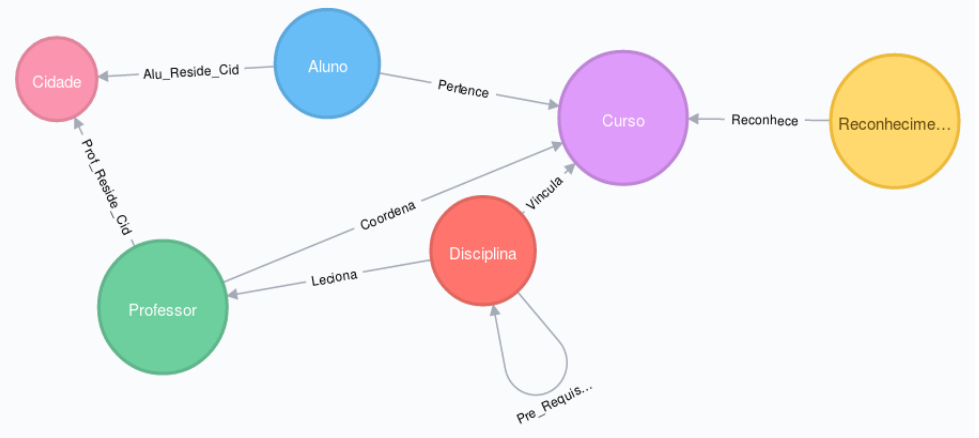

Figura 41: Esquema gerado Neo4j do Banco de Dados do Estudo de Caso

O esquema criado automaticamente, apresenta grande semelhança com o modelo lógico de grafos definido na Figura 40.

O esquema é apresentado a partir da execução do comando:

CALL db.schema

B.2 – Implementação do Estudo de Caso de um Cenário Acadêmico no sistema BDG Neo4j

Nesta seção é criado um exemplo de banco de dados no Neo4j a partir do modelo lógico de grafos apresentado no estudo de caso.

B.2.1 Criação dos vértices

Comandos utilizados para criação de vértices (também chamados de “nodes”) e suas respectivas restrições suportadas pelas versões do Neo4j.

B.2.1.1 Criação dos vértices agrupados no rótulo “Aluno”

CREATE CONSTRAINT ON (a:Aluno) ASSERT a.cpf IS UNIQUE

CREATE CONSTRAINT ON (a:Aluno) ASSERT EXISTS(a.cpf)

CREATE (:Aluno {nome: "Leonardo Bonfim", contato: "leo@mail.com", cpf: "12345678900"})

CREATE (:Aluno {nome: "Matheus Silveira", contato: "9999-7777", cpf: "13131313137"})

B.2.1.2 Criação dos vértices agrupados no rótulo “Curso”

CREATE CONSTRAINT ON (c:Curso) ASSERT c.id IS UNIQUE

CREATE CONSTRAINT ON (c:Curso) ASSERT (c.id, c.nome) IS NODE KEY

CREATE (:Curso {id: "001", nome: "Redes de Computadores"})

B.2.1.3 Criação dos vértices agrupados no rótulo “Disciplina”

CREATE (:Disciplina {nome: "Gerência de Redes"})

CREATE (:Disciplina {nome: "Serviços de Redes"})

CREATE (:Disciplina {nome: "Gerência de Redes Avançada"})

B.2.1.4 Criação dos vértices agrupados no rótulo “Cidade”

CREATE CONSTRAINT ON (City:Cidade) ASSERT City.cep IS UNIQUE

CREATE (:Cidade {nome: "Criciúma", cep: "1234"})

CREATE (:Cidade {nome: "Sombrio", cep: "1313"})

B.2.1.5 Criação dos vértices agrupados no rótulo “Professor”

CREATE CONSTRAINT ON (Prof:Professor) ASSERT EXISTS(Prof.cpf)

CREATE CONSTRAINT ON (Prof:Professor) ASSERT Prof.cpf IS UNIQUE

CREATE CONSTRAINT ON (Prof:Professor) ASSERT Prof.email IS UNIQUE

CREATE (:Professor {cpf: "10203040506", nome: "Luis Mariano", email: "luis@mail.com"})

CREATE (:Professor {cpf: "90807060504", nome: "Victor Sousa", email: "victor@mail.com"})

B.2.1.6 Criação dos vértices agrupados no rótulo Reconhecimento

```
CREATE (:Reconhecimento {id: "201706290302001", ano: "2017"})
```

B.2.2 Criação das arestas

Comandos utilizados para criação de arestas (também chamadas de “relationshps”) e suas respectivas restrições suportadas pelas versões do Neo4j.

B.2.2.1 Criação das arestas agrupadas no rótulo “Leciona”

```
MATCH (prof:Professor) WHERE prof.cpf = "10203040506"
MATCH (ds:Disciplina) WHERE ds.nome = "Serviços de Redes"
CREATE (prof)<-[:Leciona]->(ds)
MATCH (prof:Professor) WHERE prof.cpf = "90807060504"
MATCH (ds:Disciplina) WHERE ds.nome = "Gerência de Redes"
CREATE (prof)<-[:Leciona]-(ds)
```

B.2.2.2 Criação das arestas agrupadas no rótulo “Alu_Reside_Cid”

```
MATCH (al:Aluno) WHERE al.cpf = "12345678900"
MATCH (city:Cidade) WHERE city.nome =  "Sombrio"
CREATE (al)-[:Alu_Reside_Cid]->(city)
MATCH (al:Aluno) WHERE al.cpf = "13131313137"
MATCH (city:Cidade) WHERE city.nome =  "Sombrio"
CREATE (al)-[:Alu_Reside_Cid]->(city)
```

B.2.2.3 Criação das arestas agrupadas no rótulo “Prof_Reside_Cid”

```
MATCH (prof:Professor) WHERE prof.cpf = "10203040506"
MATCH (city:Cidade) WHERE city.nome =  "Criciúma"
```

CREATE (prof)-[:Prof_Reside_Cid]->(city)

MATCH (prof:Professor) WHERE prof.cpf = "90807060504"

MATCH (city:Cidade) WHERE city.nome = "Sombrio"

CREATE (prof)-[:Prof_Reside_Cid]->(city)

B.2.2.4 Criação das arestas agrupadas no rótulo “Pertence”

MATCH (al:Aluno) WHERE al.cpf = "12345678900"

MATCH (c:Curso) WHERE c.id = "001"

CREATE (al)-[:Pertence {ingresso: "20172"}]->(c)

MATCH (al:Aluno) WHERE al.cpf = "13131313137"

MATCH (c:Curso) WHERE c.id = "001"

CREATE (al)-[:Pertence {ingresso: "20172"}]->(c)

B.2.2.5 Criação das arestas agrupadas no rótulo “Vincula”

MATCH (c:Curso) WHERE c.nome = "Redes de Computadores"

MATCH (ds:Disciplina) WHERE ds.nome = "Serviços de Redes"

CREATE (ds)-[:Vincula]->(c)

MATCH (c:Curso) WHERE c.nome = "Redes de Computadores"

MATCH (ds:Disciplina) WHERE ds.nome = "Gerência de Redes"

CREATE (ds)-[:Vincula]->(c)

B.2.2.6 Criação das arestas agrupadas no rótulo “Coordena”

CREATE CONSTRAINT ON ()-[C:Coordena]-() ASSERT exists(C.data_inicio)

MATCH (c:Curso) WHERE c.nome = "Redes de Computadores"

MATCH (prof:Professor) WHERE prof.cpf = "90807060504"

CREATE (prof)-[:Coordena {data_inicio: "2017-01-01", data_fim: "0000-00-00"}]->(c)

B.2.2.7 Criação das arestas agrupadas no rótulo “Reconhece”

MATCH (c:Curso) WHERE c.id = "001"

MATCH (r:Reconhecimento) WHERE r.id = "201706290302001"

CREATE (r)-[:Reconhece]->(c)