Processo de Mapeamento do Modelo Conceitual Entidade-Relacionamento Estendido para o Modelo Lógico de Grafos

Luiz Sergio Velasques Urquiza Junior

Dezembro / 2020

Dissertação de Mestrado em Ciência da Computação

Processo de Mapeamento do Modelo Conceitual Entidade-Relacionamento Estendido para o Modelo Lógico de Grafos

Esse documento corresponde à Dissertação apresentada à Banca Examinadora para Defesa de Dissertação no curso de Mestrado em Ciência da Computação da Faculdade Campo Limpo Paulista.

Campo Limpo Paulista, 21 de dezembro de 2020.

Luiz Sergio Velasques Urquiza Junior

Dr Luis Mariano del Val Cura (Orientador)

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Ficha catalográfica elaborada pela

Biblioteca Central da Unifaccamp

U77p

Urquiza Junior, Luiz Sergio Velasques

Processo de mapeamento do modelo conceitual entidade-relacionamento estendido para o modelo de dados NoSQL / Luiz Sergio Velasques Urquiza Junior. Campo Limpo Paulista, SP: Unifaccamp, 2020.

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. Processo. 3. Mapeamento. 4. Grafos. 5. NoSQL. 6. Modelagem conceitual. 7. Modelagem lógica I. Del Val Cura, Luis Mariano. II. Centro Universitário Campo Limpo Paulista. III. Título.

CDD-005.75

Dedicatória

Dedico este trabalho a Mauni, minha amada companheira, melhor amiga e maravilhosa mãe de meu filho Bernardo, que são fonte de minha motivação e felicidade. Dedico também a meu pai, Luiz Sergio, e à memória de minha mãe, Luiza, que me mostraram que uma grande jornada sempre começa com o primeiro passo.

Agradecimentos

Agradeço aos professores do Programa de Pós Graduação da FACCAMP, ao meu orientador, Luis Mariano del Val Cura, pelo apoio e compreensão e aos meus amigos Igor, Renato, André, Leandro e Tatiana pelas animadas tardes de estudo.

Resumo. *Esta é uma proposta de processo de mapeamento do Modelo Conceitual Entidade-Relacionamento Estendido para o Modelo Lógico NoSQL de Grafo de Propriedades. Para cada elemento do diagrama EER, algoritmos de mapeamento são executados para gerar um grafo de esquema. Este grafo de esquema representa a estrutura e as restrições de integridade para instâncias de banco de dados grafo. Nesta abordagem, o modelo conceitual considera entidades, atributos simples/compostos e monovalorados/multivalorados, relacionamentos binários e n-ários e relações de generalização/especialização. O modelo lógico resultante é um multigrafo rotulado de propriedades. Este multigrafo pode ser uma ferramenta útil para validação e verificação de instâncias de banco de dados de grafos, e também na interação de projetistas e usuários. Além disso, essa abordagem sugere a implementação de restrições de integridade em bancos de dados de grafos. As restrições de cardinalidade são discutidas, bem como as regras de participação e disjunção na modelagem de generalização/especialização para garantir que o diagrama EER e o grafo de esquema sejam semanticamente equivalentes. Por fim, os conceitos propostos são exemplificados em um estudo de caso. Um diagrama EER é mapeado para um gráfico de esquema e uma instância de banco de dados é implementada em Neo4j. Este sistema de gerenciamento de banco de dados de grafos foi escolhido por sua robustez, facilidade de entendimento e grande popularidade entre designers, desenvolvedores e a comunidade acadêmica. O processo de mapeamento foi implementado na linguagem Java e ambos, o modelo conceitual e o modelo lógico são representados por estruturas XML e são respectivamente a entrada e saída do aplicativo.*

Palavras chave: *Bancos de Dados, NoSQL, Projeto de Bancos de Dados, Grafos, Modelagem de Dados*

Abstract. *This is a mapping process proposal from Extended Entity-Relationship Conceptual Model to Property Graph NoSQL Logical Model. For each EER diagram element, mapping algorithms are executed to generate a schema graph. This schema graph represents the structure and the integrity constraints for graph database instances. In this approach, conceptual model considers entities, simple/composite and monovalued/-multivalued attributes, binary and n-ary relationships and generalization/specialization relations. Resulting logical model is a labeled properties multigraph. This multigraph can be a useful tool for graph database instances validation and verification, also in designers and users interation. In addition, this approach suggests integrity constraints implementation in graph databases. Cardinality constraints are discussed, as well as participation and disjunction rules in generalization/specialization modeling to ensure that EER diagram and schema graph are semantically equivalent. Finally, the proposed concepts are exemplified in a study case. An EER diagram is mapped to a schema graph, and a database instance is implemented in Neo4j. This graph database management system was chosen for its robustness, ease of understanding and great popularity among designers, developers and the academic community. The mapping procces was implemented in Java language and both, conceptual model and logical model are represented by XML structures and are respectively the application input and output.*
Keywords: *Database, NoSQL, Database Design, Graph, Data Modeling*

Sumário

1 Introdução **17**

1.1 Objetivos 20

1.2 Contribuições 20

1.3 Organização do trabalho 21

2 Fundamentação Teórica **22**

2.1 Modelagem Conceitual 23

2.1.1 O Modelo Entidade-Relacionamento 23

2.1.2 O Modelo Entidade-Relacionamento Estendido (EER) 31

2.2 Projetos Lógicos de Bancos de Dados 36

2.2.1 O Modelo de Dados Relacional 37

2.2.2 Modelos de Dados *NoSQL* 38

2.2.3 O Modelo de Dados em Grafos 41

2.3 Sistemas Gerenciadores de Bancos de Dados em Grafos 52

2.3.1 Análise Comparativa 57

2.4 Trabalhos Correlatos 59

3 Processo de Mapeamento do Modelo Conceitual EER para O Modelo Lógico de Grafos **64**

3.1 Mapeamento de Entidades 65

3.1.1 Restrições de integridade no mapeamento de entidades 66

3.2 Mapeamento de Relacionamentos 67

3.2.1 Relacionamentos binários 67

3.2.2 Restrições de integridade em relacionamentos binários 69

3.2.3 Relacionamentos n-ários 71

3.2.4 Restrições de integridade em relacionamentos n-ários 73

3.3 Mapeamento de Especializações 75

3.3.1 Superclasses e Subclasses 76

3.3.2 Especializações e Generalizações 77

3.3.3 Representação de Árvores de Especializações como Grafos Aninhados 81

3.3.4 Restrições de integridade em árvores de espelializações 84

3.4 Algoritmos de Mapeamento do Modelo EER para o Modelo de BDG . . . 86

4 Estudo de Caso 98

4.1 Contextualização 98

4.2 O Banco de Dados com uma Abordagem Relacional 100

4.3 Geração do Esquema Lógico de Grafos 104

4.4 Uma Instância do BDG 113

4.4.1 Definições das Restrições de Integridade no BDG 121

5 Implementação do Processo de Mapeamento 125

5.1 Representação em XML 125

5.2 Representação do Modelo Conceitual em XML 127

5.3 Representação do Grafo de Esquema em XML 132

5.3.1 Restrições de Integridade em Listas de Adjacências 137

5.3.2 Representação de Especializações em XML 138

5.4 Protótipo de Software 140

5.4.1 Implementação do Modelo Conceitual 141

5.4.2 Implementação do Modelo de Grafos 142

5.4.3 Implementação do Processo de Mapeamento 143

6 Conclusões e Trabalhos Futuros 144

6.1 Contribuições 145

6.2 Trabalhos Futuros 145

Lista de Tabelas

2.1 Comparativo entre os sistemas gerenciadores de grafos 59

2.2 Comparativo entre os trabalhos correlatos 63

Lista de Figuras

2.1 A entidade *"Funcionário"* e seu conjunto de atributos 26

2.2 Relacionamento entre *"Funcionário"* e "Departamento" 27

2.3 Relacionamentos entre *"Funcionário"* e *"Departamento"* 29

2.4 Relacionamento ternário 30

2.5 Entidade fraca *"Dependente"* 31

2.6 A superclasse *"Funcionário"* e a subclasse *"Gerente"* 33

2.7 A especialização de *"Funcionário"* em *"Horista"* e *"Mensal"* 34

2.8 A especialização de *"Funcionário"* em *Técnico* e *"Engenheiro"* 34

2.9 Especializações de *"Funcionário"* 35

2.10 Ocorrências de *"Funcionário"* e suas subclasses 36

2.11 Uma rede social representada por um grafo 41

2.12 Grafo de Petersen 43

2.13 Grafo Simples (a), Multigrafo (b) e Grafo Dirigido (c) 44

2.14 Grafo de Propriedades 46

2.15 Hipergrafo (a) e Grafo Aninhado (b) 48

2.16 Modelo Conceitual do BDG Empresa 49

2.17 Grafo de Esquema do BDG Empresa 50

2.18 Instância Válida para o BDG Empresa 52

3.1 Mapeamento da Entidade *"Funcionário"* para Grafo de Esquema 66

3.2 Relacionamentos "*Trabalha_em*" e "*Gerencia*" 69

3.3 Mapeamento dos Relacionamentos "*Trabalha_em*" e "*Gerencia*" 70

3.4 Relacionamento "*Atua em*" entre as entidades *"Funcionário"*, *"Projeto"* e *"Local"* 73

3.5 Mapeamento do Relacionamento "*Atua_em*" para o grafo de esquema . . 73

3.6 Relacionamento "*Atua em*" remodelado como entidade fraca 75

3.7 Mapeamento de Subclasses e de Especializações/Generalizações 77

3.8 Mapeamento da superclasse *Funcionário* em *"Horista"* e *"Mensal"* . . . 79

3.9 Mapeamento da superclasse *"Funcionário"* em *"Técnico"* e *"Engenheiro"* 80

3.10 Árvore de especializações de *"Funcionário"* 81

3.11 Relacionamentos da árvore de especializações de *"Funcionário"* 83

3.12 Versão simplificada do modelo apresentado na Figura 3.11 84

3.13 Instância de BDG baseado nos grafos de esquema das Figuras 3.11 e 3.12 85

3.14 Descrição em UML das Classes Entidade, Relacionamento, e Especialização 88

3.15 Descrição em UML das Classes Nó e Aresta 89

4.1 Projeto Conceitual do Banco de Dados de Controle de Eventos 100

4.2 Mapeamento das Entidades, Especializações e Relacionamentos Binários do Modelo EER 103

4.3 Mapeamento dos Atributos Multivalorados do Modelo EER 103

4.4 Mapeamento do Relacionamento Ternário "Avalia" 104

4.5 Esquema Relacional do Sistema de Gerenciamento de Eventos do IFMS . 105

4.6 Mapeamento das Entidades do Modelo Relacional da Figura 4.1 107

4.7 Mapeamento dos Relacionamentos Binários do Modelo Relacional da Figura 4.1 109

4.8 Mapeamento do Relacionamento N-ário “Avalia” do Modelo Relacional da Figura 4.1 111

4.9 Mapeamento da Especialização de “Pessoa” em “Orientador”, “Estudante” e “Avaliador” 112

4.10 Versão Simplificada do Grafo de Esquema da Figura 4.9 113

4.11 Código *Cypher* que Cria um Nó no BDG 114

4.12 *Script* para Criação dos Quesitos de Avaliação 114

4.13 *Script* que Associa os Quesitos ao Evento Criado 115

4.14 Evento e Quesitos na Instância do BDG 115

4.15 Criando Instâncias de *“Pessoa”* no BDG 116

4.16 Criando Instâncias de *“Trabalho”* no BDG 116

4.17 Definindo a ligação entre os trabalhos e o evento 117

4.18 Definindo as participações dos estudantes nos trabalhos 117

4.19 Estabelecendo as orientações em cada trabalho 117

4.20 Instância do trabalho *“CET001”* 118

4.21 Criação de nós *“Avalia”* 118

4.22 Estabelecendo associações entre *“Quesito”* e *“Avalia”* 119

4.23 Estabelecendo associações entre *“Avaliador”* e *“Avalia”* 119

4.24 Estabelecendo associações entre *“Trabalho”* e *“Avalia”* 119

4.25 Avaliações do trabalho *“CET001”* 120

5.1 Árvore da Estrutura XML 5.1 127

5.2 Grafo de Esquema do Modelo da Figura 2.5 134

Lista de Algoritmos

1 Mapeia_Entidade (Entidade: e) 93

2 Mapeia_Relacionamento_Binario (Relacionamento: r) 94

3 Mapeia_Relacionamento_Binario_Multi (Relacionamento: r) 95

4 Mapeia_Relacionamento_N_Ario (Relacionamento: r) 96

5 Mapeia_Especialização (Especialização: p) 97

Lista de Códigos

5.1 Exemplo de Estrutura XML 126

5.2 Definição de Esquema XML do Modelo Conceitual 128

5.3 Definição de Esquema XML do Modelo de Grafos 132

5.4 Estrutura XML do grafo da Figura 5.2 135

5.5 Exemplo de Especializações 138

Capítulo 1

Introdução

A ideia de bancos de dados relacionais foi introduzida inicialmente por Edgar Frank Codd em seu artigo "*A relational model of data for large shared data banks*" em junho de 1970 (Codd; 1970). Neste artigo foi proposta uma abordagem baseada em teoria dos conjuntos e lógica de predicados para tratamento de dados de forma independente da aplicação, reduzindo o número de redundâncias e garantindo a consistência dos dados armazenados em cada estado do banco de dados (Elmasri et al.; 2011). A partir desta abordagem relacional, diversos Sistemas Gerenciadores de Bancos de Dados (SGBD) foram desenvolvidos e, não por acaso, dentre os dez SGBDs mais utilizados atualmente, seis são relacionais. O primeiro SGBD não relacional, o *MongoDB*, um SGBD de Documento, aparece apenas na quinta posição do ranking (DB-Engines; 2019).

Apesar da importância dos bancos relacionais, é cada vez maior a demanda por grandes volumes de dados e uma mudança de paradigma acaba se fazendo necessária. Segundo Pokorny (2013), a *web* possibilitou o surgimento de grandes repositórios de dados distribuídos que criaram demandas que tornaram os tradicionais bancos de dados relacionais ineficientes em seu tratamento. Poffo (2016) destacam ainda que o enorme volume de dados, quebrando a barreira dos petabytes, cria um cenário bastante desafiador quanto ao gerenciamento e manipulação destes dados, demandando uma mudança das tradicionais metodologias de projetos de bancos de dados.

Outro fator preponderante é que os dados passaram a vir de diversas fontes, muitas vezes com formatos e estruturas diferentes, acrescentando um alto grau de heterogeneidade e

complexidade no seu tratamento. Diante deste cenário, duas abordagens distintas tem sido desenvolvidas a fim de tratar de grandes volumes de dados heterogêneos distribuídos (*Big Data*): bancos de dados não relacionais, denominados *Not Only SQL* (*NoSQL*); e também uma nova classe de bancos de dados relacionais horizontalmente escaláveis, denominada *New SQL* (Binani et al.; 2016).

Bancos de dados *NoSQL* tem sido uma alternativa interessante para lidar com o *Big Data* em detrimento aos bancos de dados relacionais, uma vez que oferecem um modelo lógico de dados mais simplificado e um conjunto de regras menos restritivo. Além disso, Unal & Oguztuzun (2018) ainda acrescentam que bancos de dados relacionais não escaláveis perdem desempenho ao lidar com grandes quantidades de dados, sobretudo quando esses dados são altamente conectados, já que as consultas SQL são baseadas em custosas operações de junção (*join*) para recuperar e manipular dados. Já bancos de dados *New SQL* representam um esforço para agregar características *NoSQL* aos bancos de dados relacionais, como alta disponibilidade e capacidade de escalabilidade horizontal, porém ,sem deixar de dar suporte as propriedades *ACID* (Binani et al.; 2016).

Tradicionalmente, um projeto de banco de dados cria um conjunto de definições sobre como os dados devem ser estruturados. Como parte desta estrutura são definidas suas *restrições de integridade*, que devem ser respeitadas a fim de garantir a consistência em qualquer situação em que os dados sejam modificados. Este conjunto de definições é chamado de *esquema* de banco de dados e é parte fundamental de um banco de dados relacional. Bancos de dados *NoSQL*, em muitos casos, abrem mão de um esquema rígido a fim de evoluir de forma mais flexível e dinâmica ao longo do tempo (Sousa; 2018).

De acordo com (Abramova et al.; 2014), bancos de dados *NoSQL* são divididos em quatro categorias sendo: *chave/valor*, orientados a *documentos*, organizados em *colunas* (ou *colunares*) e orientados a *grafo*. Cada uma dessas categorias organiza e gerencia os dados de maneira distinta uma da outra, de acordo com características como consistência, disponibilidade, escalabilidade e desempenho.

No momento de se projetar um banco de dados é necessário conhecer a natureza dos dados a serem armazenados e a sua finalidade a fim de escolher qual é o melhor modelo de dados a ser seguido, seja este modelo relacional ou não. Algumas características que

podem ser levadas em consideração para uma escolha mais segura (Unal & Oguztuzun; 2018): a quantidade de relacionamentos *N para N* presentes no modelo conceitual, que podem levar a consultas mais custosas computacionalmente para um modelo de dados relacional; a possibilidade de que o sistema possa frequentemente passar por mudanças em seu esquema de dados também é outro fator que pode levar a um modelo de dados *NoSQL*. Outro fator relevante é a presença de entidades de dados com uma grande quantidade de atributos não obrigatórios, que, eventualmente, podem ficar com valores nulos no banco de dados. Em um banco *NoSQL*, esses atributos podem ser suprimidos quando não preenchidos. Outro fator relevante a ser considerado é sobre o próprio comportamento dos dados no banco de dados. Dados que se comportam como grafos, por exemplo, em um rede social, podem ser melhor servidos por um banco de dados *NoSQL* de grafos.

A partir do clássico artigo de Codd (Codd; 1970) foram desenvolvidos vários processos abordando temas sobre como abstrair, projetar e construir sistemas de bancos de dados relacionais, porém o mesmo não acontece com os bancos de dados *NoSQL* (Atzeni et al.; 2016; Poffo; 2016). Comyn-Wattiau & Akoka (2017) afirmam que a modelagem de dados para um sistema de banco de dados *NoSQL* ainda é feita, na maioria das vezes, de forma empírica, muitas vezes baseada em uma política de boas práticas e ignorando algumas fases de projeto de bancos de dados.

Embora bancos de dados *NoSQL* estejam atualmente ganhando espaço ao lidar com os desafios impostos por sistemas que manipulam *Big Data*, ainda não existem processos bem definidos sobre como projetar esses tipos de bancos de dados. Enquanto que projetos de bancos de dados relacionais possuem um conjunto de algoritmos bem definidos para mapeamento a partir de um modelo conceitual, o mesmo não ocorre quanto aos bancos de dados *NoSQL*. Neste contexto, este trabalho propõe um processo de mapeamento de um esquema lógico de banco de dados de grafo (BDG) a partir da modelagem conceitual Entidade-Relacionamento Extendido (EER). O modelo Entidade-Relacionamento (ER) é o modelo conceitual de dados mais utilizado em projetos de banco de dados (Elmasri et al.; 2011) e possui uma extensa literatura tratando de seus conceitos e fundamentos. O modelo EER é uma versão aprimorada do modelo ER que acrescenta uma série de elementos semânticos além de oferecer soluções mais precisas e representativas do domínio a ser representado no banco de dados.

1.1. Objetivos

Este trabalho tem como principal objetivo estabelecer um processo de mapeamento a partir de um projeto conceitual baseado no modelo EER para um modelo lógico de BDG. Especificamente, para se chegar a um BDG é necessário propor uma notação específica para o seu esquema lógico. Neste trabalho o esquema lógico de um BDG será denominado como *grafo de esquema* que pode ser entendido como um grafo que representará os conjuntos de objetos do mundo real e suas associações dentro do BDG, bem como seu conjunto de restrições. Também é necessário propor uma série de algoritmos em alto nível para o mapeamento dos elementos do modelo EER para o grafo de esquema.

Os algoritmos propostos para mapeamento do modelo conceitual EER para o modelo lógico de grafos serão implementados em linguagem *Java*, tendo como entrada um arquivo XML, produzido manualmente, representando um diagrama EER e gerando como saída um grafo de esquema também representado em formato XML.

1.2. Contribuições

Este trabalho propõe um projeto de bancos de dados *NoSQL* de grafos a partir de um processo de mapeamento do modelo EER para o grafo de esquema e sugere formas de se implementar um conjunto mais completo de restrições de integridade em BDGs, dando suporte a restrições de cardinalidade em relacionamentos, bem como restrições de completude e disjunção em especializações. Além disso, espera-se que este trabalho possa ajudar a promover discussões mais aprofundadas a respeito de projetos de bancos de dados *NoSQL*, especialmente os modelos de grafos de propriedades.

Este projeto de banco de dados *NoSQL* parte da modelagem conceitual utilizando conceitos do modelo entidade-relacionamento estendido. A proposta de modelagem lógica representa a estrutura do banco de dados como um grafo de propriedades, que transforma entidades em nós, relacionamentos em arestas e especializações em hiperarestas agrupadas em hipernós. Este modelo lógico também leva em consideração as restrições impostas no modelo conceitual (restrições de cardinalidade em relacionamentos e restrições de participação e disjunção, inerentes às especializações) para garantir instâncias coerentes do BDG.

O processo de transformação do modelo conceitual para o modelo lógico de grafos de propriedades também é um dos objetivos deste trabalho, apresentando uma série de algoritmos de mapeamento, que, uma vez implementados, podem oferecer ferramentas para validação e análise instâncias de BDGs geradas a partir do projeto de bancos de dados proposto.

1.3. Organização do trabalho

A organização deste trabalho segue uma linha de raciocínio a partir das definições gerais de projetos de bancos de dados, dando ênfase à modelagem conceitual utilizando diagrama ER e EER, bem como aos conceitos de projetos lógico de BDG. O capítulo 2 apresenta uma revisão bibliográfica com os conceitos de modelagem conceitual considerados neste trabalho, além de uma definição de projeto lógico, em especial o modelo de grafos de propriedades. O Capítulo 2 apresenta ainda uma comparação entre Sistemas Gerenciadores de Bancos de Dados (SGBDs) que implementam o modelo *NoSQL* de grafos e termina com uma relação de trabalhos correlatos. No Capítulo 3 é proposta uma série de algoritmos em alto nível para a geração do grafo de esquema do BDG a partir do processo de mapeamento do modelo conceitual EER. O Capítulo 4 apresenta um estudo de caso com o mapeamento de um diagrama EER para modelo lógico de grafos proposto com uma instância do BDG implementada utilizando SGBD de grafos *Neo4j*, onde serão discutidas as vantagens e as limitações práticas desta abordagem. No Capítulo 5 serão apresentados os detalhes de implementação da ferramenta de mapeamento propostos neste trabalho. Finalmente, o Capítulo 6 mostra as conclusões e trabalhos futuros considerados.

Capítulo 2

Fundamentação Teórica

Tradicionalmente, o projeto de um banco de dados acontece em três fases distintas. A primeira fase é a de modelagem conceitual, que consiste de uma descrição abstrata da estrutura do banco de dados, descrevendo os tipos de entidades que serão armazenadas e seus atributos, sendo uma fase independente da implementação do SGBD (Heuser; 2009). A modelagem conceitual pode ser feita atualmente utilizando ferramentas CASE (*Computer-Aided Software Engeneering*), e ter um formato diagramático ou ser um documento meramente descritivo. O mais usual em termos de modelagem conceitual é adotar o modelo ER e suas variações, já que este é um modelo bastante popular que apresenta o banco de dados de maneira diagramática e cujos conceitos são empregados em diversas ferramentas CASE (Date; 2004; Elmasri et al.; 2011; Heuser; 2009). Os diagramas ER e suas variações são hoje uma importante ferramenta de modelagem conceitual em um projeto de bancos de dados.

A segunda fase de um projeto de banco de dados é chamada de projeto lógico e consiste no mapeamento do modelo conceitual para um esquema que leva em consideração os detalhes de implementação do banco de dados no SGBD, ou seja, a transformação do modelo conceitual (um diagrama ER, por exemplo) em um *modelo de implementação* (Elmasri et al.; 2011), um conjunto de definições para os elementos da estrutura do banco de dados dentro do SGBD. Este modelo de implementação é, portanto, dependente do modelo de dados (relacional ou não) implementado pelo SGBD. Em outras palavras, em um SGBD relacional este mapeamento gera como modelo de implementação o conjunto

de definições de suas tabelas, enquanto que, para um modelo *NoSQL* de BDG, esta modelagem leva a um *grafo de esquema* (Sousa; 2018) que serve de orientação para o projetista de banco de dados no momento de criar uma instância do banco de dados. Segundo Heuser (2009), o projeto lógico pode ser entendido como o uma representação da estrutura de dados de um banco de dados, seja esta estrutura representada por tabelas de um banco de dados relacional, nós e arestas em um banco de dados de grafo, ou qualquer outro tipo de estrutura que seja conveniente às demais categorias de bancos de dados *NoSQL*.

A última fase do projeto de banco de dados é a modelagem física. Esta fase procura organizar fisicamente os arquivos do banco de dados no SGBD para otimização do desempenho de acesso aos dados. Na prática é um processo contínuo que não altera aspectos funcionais do banco de dados e não interfere no seu uso. Este processo também é conhecido como sintonia ou *tuning* de banco de dados e pode ser controlado automaticamente pelo próprio SGBD (Heuser; 2009).

2.1. Modelagem Conceitual

A modelagem conceitual é o primeiro passo em um projeto de banco de dados, geralmente obtida após um processo inicial de levantamento e análise de requisitos (Elmasri et al.; 2011). Nesta fase, identificam-se os elementos que serão representados no banco de dados, bem como suas associações. Embora a modelagem conceitual possa ser representada de maneira textual, na grande maioria das vezes ela resulta em um diagrama, dando uma noção mais ampla e simplificada da organização dos dados no banco de dados. Como modelo diagramático, o mais utilizado é o Modelo Entidade-Relacionamento (ER), proposto por Chen (1976), e suas variações.

2.1.1. O Modelo Entidade-Relacionamento

Geralmente, um projeto de banco de dados começa com um processo de levantamento e análise de requisitos, que é o primeiro passo da modelagem conceitual (Elmasri et al.; 2011). No modelo ER o processo de análise dos requisitos leva a elaboração de um *diagrama ER* que mostra os detalhes estruturais dos dados, as maneiras como esses dados se associam e suas restrições em um alto nível de abstração, livre dos detalhes de implementação. Segundo Silberschatz et al. (2012), o modelo ER “normalmente é usado

para representar o projeto conceitual" e seu esquema "especifica as entidades que são representadas no banco de dados, os atributos das entidades, os relacionamentos entre as entidades e as restrições sobre as entidades".

A modelagem conceitual de dados utilizando modelo ER pode ser realizada tanto manualmente quanto auxiliada por ferramentas CASE e, neste último caso, o processo de mapeamento para um banco de dados pode ser feito automaticamente pela ferramenta. Este processo cria o *esquema* do banco de dados que deverá ser implementado no SGBD alvo, geralmente descrevendo um conjunto de tabelas de um banco de dados relacional bem como seu conjunto de restrições de integridade.

Entidades e atributos

No modelo ER as *entidades* (Teorey et al.; 2014) ou *tipos de entidade* (Elmasri et al.; 2011) representam os conjuntos de objetos sobre os quais se deseja armazenar informações no banco de dados (Heuser; 2009) e são identificadas através do processo de abstração do mundo real e da análise inicial do levantamento de requisitos. Uma entidade também pode ser descrita como a representação de uma coleção de objetos que pertençam a um mesmo grupo com propriedades e comportamentos similares. Segundo Teorey et al. (2014), "elas normalmente representam um pessoa, lugar, coisa ou evento de interesse informativo". Na prática, uma entidade pode ser utilizada para descrever tanto objetos concretos, como um grupo de funcionários em uma empresa, quanto objetos abstratos, como projetos em andamento da mesma empresa.

Cada entidade tem associada a si um conjunto de unidades de dados que descrevem as propriedades dos objetos representados (Elmasri et al.; 2011). Estas unidades de dados são chamadas de *atributos*. Quando necessário destacar a ocorrência de alguma entidade em particular basta listar os valores de seus atributos. Desta forma uma entidade pode ser melhor definida como uma conjunto de objetos que "compartilham os mesmos atributos, mas cada um tem *o(s) próprio(s) valor(es)* para cada atributo" (Elmasri et al.; 2011). Pode ocorrer em uma entidade atributos de vários tipos diferentes:

1 - **Simples ou Compostos**: Atributos simples, ou *atômicos*, são atributos cujos valores são indivisíveis, enquanto que atributos compostos tem valores que podem

ser divididos em atributos mais básicos ou simples (Elmasri et al.; 2011).

2 - **Monovalorados ou Multivalorados**: Em uma ocorrência de uma entidade em particular, um atributo monovalorado pode armazenar, no máximo, um único valor. Já um atributo multivalorado pode armazenar um conjunto de valores distintos para a mesma ocorrência (Elmasri et al.; 2011).

3 - **De preenchimento obrigatório (*NOT NULL*)**: São atributos que devem receber algum valor válido. O conjunto de valores válidos para um determinado atributo é chamado de *domínio do atributo* e é definido nas fases subsequentes ao projeto conceitual. O valor *NULL* representa o não preenchimento, ou nulidade do valor do atributo, e só pode ser atribuído a atributos definidos como não obrigatórios.

4 - **De valor exclusivo (*UNIQUE*)**: Esses atributos, quando preenchidos, devem ter um valor exclusivo dentro do conjunto de entidades cadastrado no banco de dados, ou seja, não podem existir duas ocorrências com o mesmo valor para um mesmo atributo de valor exclusivo. Atributos de valor exclusivo não são necessariamente de preenchimento obrigatório, porém, uma vez preenchidos, não podem haver valores repetidos para esses atributos.

5 - **Identificadores**: Atributos utilizados para identificar uma ocorrência em particular de uma entidade. Atributos identificadores são, por definição, de preenchimento obrigatório e de valor exclusivo.

Neste trabalho será utilizada a notação prposta por Chen (1976) para representação de diagramas ER. Nesta notação cada entidade é representada por um retângulo preenchido com o seu nome. Seus atributos são representados por elipses ligadas à entidade por linhas.

A Figura 2.1 mostra a representação da entidade *"Funcionário"* e seus atributos em um diagrama ER. Os atributos *"Nome"* e *"Data de Nascimento"* são atributos simples e monovalorados. O atributo *"CPF"* está sublinhado, indicando ser um atributo identificador. O atributo *"Telefones"*, denotado por uma dupla elipse, é um atributo multivalorado, enquanto que o atributo *"Endereço"* é composto por *"Logradouro"*, *"Número"* e *"Bairro"*.

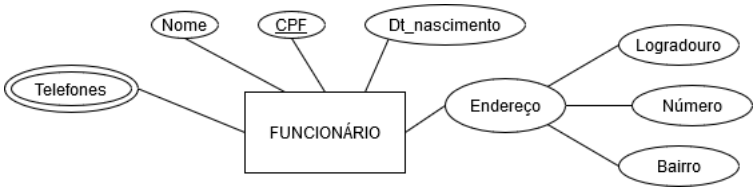

Figura 2.1. A entidade *"Funcionário"* e seu conjunto de atributos

Relacionamentos

Em um modelo ER várias entidades podem ser modeladas e, eventualmente, algumas de suas ocorrências podem se associar mutuamente. A representação da associação entre ocorrências de duas ou mais entidades é chamada de *relacionamento*. Heuser (2009) descreve o relacionamento como um "conjunto de associações entre ocorrências de entidades", isto é, um relacionamento indica que a ocorrência de uma entidade em particular pode referenciar ou ser referenciada por uma ou mais ocorrências distintas.

O grau de um relacionamento indica o número de entidades que participam do relacionamento. Os relacionamentos mais comuns são os de grau dois ou *binários* no qual se associam ocorrências de duas entidades. Entretanto, podem ser modelados relacionamentos de qualquer grau, geralmente mais complexos, mas que, em alguns casos, podem oferecer uma interpretação mais específica do que relacionamentos binários. Um relacionamento de grau 3 é chamado *ternário*, enquanto que um relacionamento de grau quatro é denominado *quaternário* e assim por diante. Para uma nomenclatura mais geral, um relacionamento de grau maior que dois também pode ser chamado de *relacionamento n-ário* e, de forma genérica, é dessa maneira que eles serão tratados nesta abordagem.

No modelo ER a notação de relacionamento é um losango ligando as entidades participantes, onde o nome do relacionamento é preenchido no centro do losango. A Figura 2.2 mostra um exemplo de relacionamento entre entidades de *"Funcionário"* e *"Departamento"*.

A Figura 2.2 representa a modelagem de um relacionamento binário entre as entidades *"Funcionário"* e *"Departamento"*. Isto quer dizer que ocorrências de *"Funcionário"* podem se associar a ocorrências de *"Departamento"* através de um relacionamento chamando "*Trabalha_em*" indicando que existe uma regra semântica que diz que funcionários

eventualmente *trabalham em* departamentos.

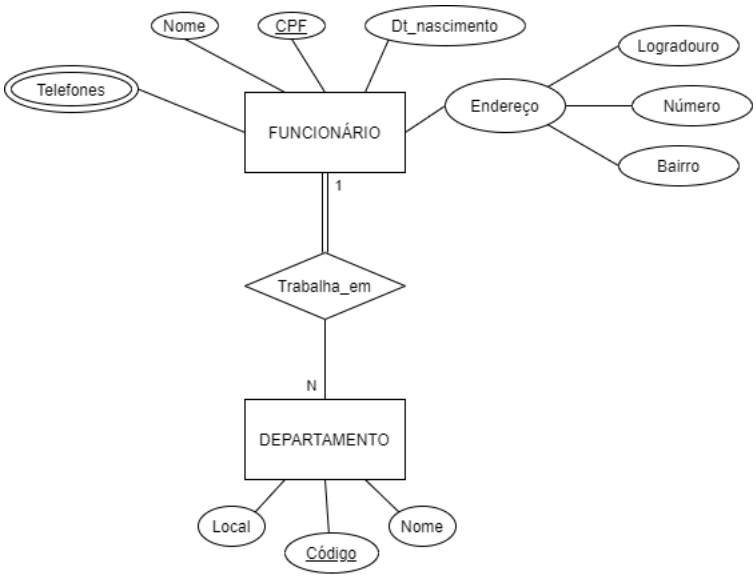

Figura 2.2. Relacionamento entre *"Funcionário"* e "Departamento"

Entretanto, seguindo o exemplo da Figura 2.2, em um modelo ER é importante indicar a quantidade de ocorrências de *"Departamento"* a qual uma ocorrência de *"Funcionário"* pode se associar e vice-versa, ou seja, indicar a quantidade de departamentos em que um funcionário em particular pode trabalhar e indicar a quantidade de funcionários que podem trabalhar em um determinado departamento. A essas quantidades dá-se o nome de *cardinalidades máximas* de *"Funcionário"* e de *"Departamento"*, respectivamente, no relacionamento "*Trabalha_em*. Para o exemplo do relacionamento "*Trabalha_em*", pode-se partir da premissa que cada funcionário trabalha em *um único* departamento e que em um departamento podem trabalhar *muitos* funcionários. Neste caso, a cardinalidade máxima de *"Funcionário"* é igual a *N* (indicando muitos ou "mais que um"), enquanto que a cardinalidade máxima de *"Departamento"* é definida como *1* no relacionamento "*Trabalha_em*". As cardinalidades máximas são definidas junto a linha que liga as entidades ao relacionamento no modelo ER.

Em relação a participação de uma entidade em um dado relacionamento, esta pode ser classificada como *total* ou *parcial*. Para uma entidade com participação total é obrigatório que *todas* as suas ocorrências façam parte do relacionamento, isto é, cada ocorrência deve estar associada a, *no mínimo uma* ocorrência da entidade parceira no relacionamento, enquanto que para as entidades com participação parcial essa obrigatoriedade não existe, ou

seja, podem existir ocorrências da entidade que não façam parte do conjunto representado pelo relacionamento em questão. No exemplo da Figura 2.2, é possível afirmar que *"Funcionário"* tem participação total no relacionamento "*Trabalha_em*", já que todo funcionário precisa trabalhar em *um, e somente um* departamento. Já *"Departamento"* tem participação parcial neste mesmo relacionamento, uma vez que podem existir departamentos vazios em um primeiro momento, ou seja, em um departamento podem trabalhar *zero* ou *N* funcionários. O tipo de participação de uma entidade no relacionamento define a *cardinalidade mínima* de sua parceira no relacionamento. Neste caso, entidades com participação total em um relacionamento tem entidades parceiras com cardinalidade mínima igual a *um*, enquanto que entidades com participação parcial tem entidades parceiras com cardinalidade mínima igual a *zero*. No exemplo da Figura 2.2 a participação total de *"Funcionário"* no relacionamento "*Trabalha_em*" é indicada por uma linha dupla, enquanto que a participação parcial de *"Departamento"* no mesmo relacionamento é denotada pela linha simples.

Outro aspecto importante sobre os relacionamentos é que eles permitem que se guarde informações sobre a associação entre as entidades envolvidas. Nestas situações, um relacionamento pode conter atributos próprios para armazenar os dados pertinentes a esta associação. Os atributos de um relacionamento seguem a mesma classificação dos atributos de entidades e não pertencem a nenhuma das entidades participantes e sim à associação entre elas, portanto só fazem sentido a partir do momento que as ocorrências destas entidades formam uma ocorrência do relacionamento em particular.

Em um diagrama ER, caso haja necessidade de guardar certos atributos para as associações entre as entidades, eles são ligados diretamente ao relacionamento representado pela associação. A Figura 2.3 mostra outro exemplo de relacionamentos binários entre as entidades *"Funcionário"* e *"Departamento"*.

Na Figura 2.3 é possível descrever dois tipos de relacionamentos entre *"Funcionário"* e *"Departamento"*. O relacionamento "*Trabalha_em*" indica que funcionários devem *trabalhar em* algum departamento. Neste caso as cardinalidades mínima e máxima indicam, respectivamente, que um departamento pode empregar *zero ou vários (0, N)* funcionários. Por outro lado, cada funcionário deve trabalhar em *um, e somente um (1, 1)*, departamento.

Neste segundo caso, a participação de *"Funcionário"* no relacionamento "*Trabalha_em*" é total, já que todo funcionário deve trabalhar em algum departamento. Para este relacionamento, o atributo "*Dt_início*" armazena a data em que um determinado funcionário passou a trabalhar em um departamento específico, ou seja, é um atributo que pertence às *ocorrências* do relacionamento "*Trabalha_em*". O relacionamento "*Gerencia*" indica que um funcionário *pode gerenciar* um único departamento (0, 1), que, por sua vez, pode ter um gerente (0, 1). Assim como em "*Trabalha_em*", o atributo "*Dt_início*" pertence a uma ocorrência do relacionamento "*Gerencia*" e especifica a data em que o funcionário passou a gerenciar o departamento em uma linha de tempo.

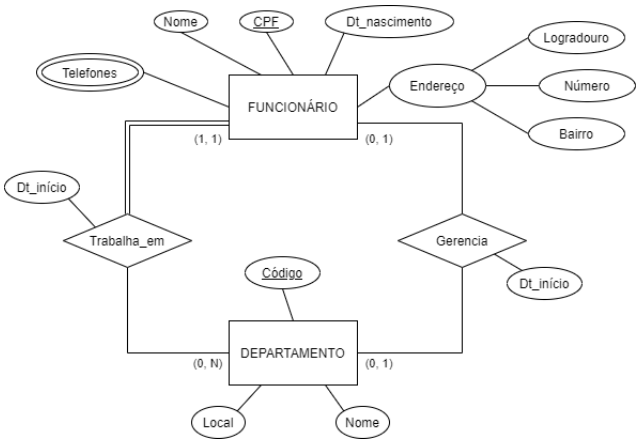

Figura 2.3. Relacionamentos entre *"Funcionário"* e *"Departamento"*

Relacionamentos n-ários

Relacionamentos de grau maior que dois (ternários, quartenários, etc) formam uma classe especial que permite a participação de três ou mais entidades. Teorey et al. (2014) destacam que este tipo de relacionamento deve ser utilizado quando é necessária uma representação semântica mais precisa e que não possa ser representada apenas com relacionamentos binários, entretanto, quando puder ser decomposto em relacionamentos binários sem perda de representatividade, estes tipos de relacionamentos devem ser evitados, em favor de uma maior simplicidade na modelagem.

Conceitos como cardinalidade e participação em relacionamentos ternários ainda são aplicáveis, embora sejam uma "extensão não trivial do conceito de cardinalidade em relacionamentos binários" (Heuser; 2009). A Figura 2.4 mostra o exemplo de um relaciona-

mento ternário.

A Figura 2.4 exemplifica um relacionamento ternário entre as entidades *"Funcionário"*, *"Projeto"* e *"Local"*. Uma ocorrência em particular do relacionamento "*Atua em*" associa uma ocorrência *f* de *"Funcionário"*, uma ocorrência *p* de *"Projeto"* e uma ocorrência *l* de *"Local"* e indica para cada projeto, quem são os funcionários atuando e o local do projeto. Em relação as cardinalidades das entidades participantes, pode-se dizer que:

1 - A cardinalidade *N* de *"Funcionário"* indica que para cada par distinto (*p*, *l*) de ocorrências de *"Projeto"* e *"Local"* podem existir *muitas* ocorrências de *"Funcionário"*, ou seja, *vários* funcionários podem trabalhar em um mesmo projeto/ local;

2 - A cardinalidade *1* de *"Projeto"* indica que para cada par de ocorrências (*f*, *l*) de *"Funcionário"* e *"Local"* existe *apenas uma* ocorrência de *"Projeto"*, isto é, um funcionário trabalhando em um determinado local atua em um *único* projeto;

3 - A cardinalidade *N* de *"Local"* incica que para cada par (*p*, *f*) de *"Projeto"* e *"Funcionário"* existem *várias* ocorrências de *"Local"*, ou seja, um funcionário pode executar o mesmo projeto em vários locais diferentes.

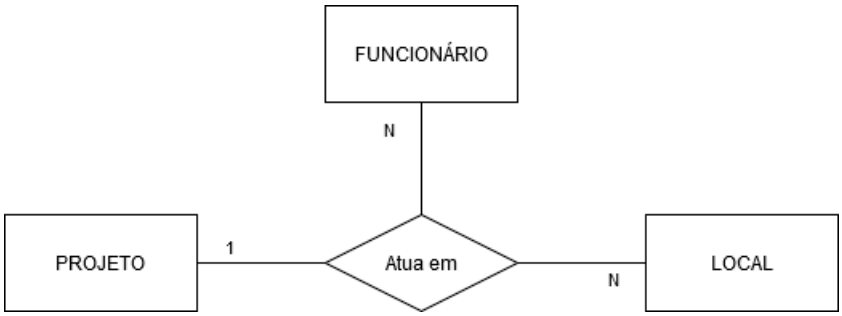

Figura 2.4. Relacionamento ternário

Entidades fracas

Entidades fracas são um tipo especial de entidades que representam conjuntos de ocorrências cuja existência depende da ocorrência de alguma entidade *forte* associada. Segundo Teorey et al. (2014), "entidades fracas derivam sua identidade dos atributos de identificação de uma ou mais *'entidades-pai'*", portanto para localizar uma ocorrência específica é necessário recorrer a ocorrência de sua entidade-pai associada.

A Figura 2.5 mostra a entidade fraca *"Dependente"*. As ocorrências de *"Dependente"* somente fazem sentido se estiverem associadas a alguma ocorrência de *"Funcionário"*. Em outras palavras, não existe dependente sem algum funcionário com quem ele se associe, ou ainda, os dependentes só podem ocorrer condicionados a existência prévia de algum funcionário do qual ele dependa. Neste caso, para identificar um dependente, é preciso recorrer ao identificador da *entidade-forte "Funcionário"* (*CPF*) e combiná-lo com o *Código* do dependente.

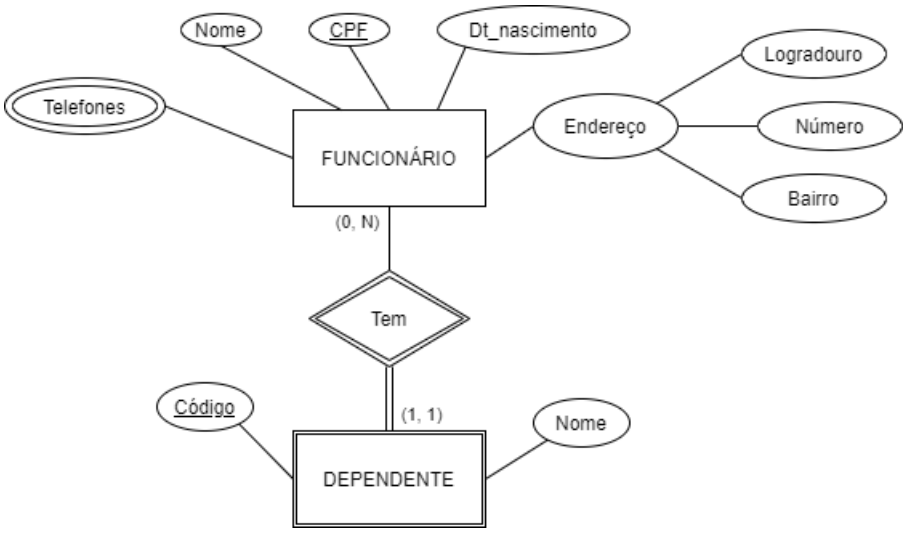

Figura 2.5. Entidade fraca *"Dependente"*

As entidades fracas são denotadas por retângulos duplos e devem estar ligadas a sua entidade-forte através de um relacionamento com participação total, indicando que todas as suas ocorrências devem estar ligadas a alguma ocorrência de sua entidade forte. No exemplo da Figura 2.5, cada ocorrência da entidade fraca *Dependente* deve se ligar a uma e somente uma ocorrência de sua entidade forte *"Funcionário"*.

2.1.2. O Modelo Entidade-Relacionamento Estendido (EER)

O modelo EER foi proposto como um aprimoramento do tradicional modelo ER com a inclusão de novos recursos semânticos. Esta notação acrescenta os conceitos de superclasse/subclasse, especialização/generalização e herança, além de apresentar um conjunto semântico mais preciso do que o modelo ER visando atender a sistemas de bancos de dados com um conjunto mais complexo de requisitos Elmasri et al. (2011). Além disso, esta é uma forma de modelagem extremamente importante no projeto de bancos de dados

relacionais, sendo uma das mais largamente utilizadas juntamente com a notação UML (Vágner; 2018).

Subclasses e superclasses

O conceito de *subclasse* é utilizado para representar *subtipos* ou *subagrupamentos* dentro de um conjunto representando por uma entidade quando esses subtipos "são significativos e precisam ser representados explicitamente, por causa de seu significado para a aplicação de banco de dados" (Elmasri et al.; 2011). As subclasses podem, eventualmente, representar um subconjunto com características e propriedades particulares em relação aos demais elementos e essas particularidades podem ser representadas tanto como atributos específicos, quanto como relacionamentos que somente são pertinentes a esses elementos. Cada subclasse definida para uma entidade representa um subconjunto da entidade, portanto uma ocorrência de alguma subclasse é também representada pela entidade como um todo já que representam o mesmo objeto do mundo real. Neste contexto, a entidade que foi subdividida em subclasses é definida como *superclasse*.

Na Figura 2.6, a entidade *"Gerente"* representa um subgrupo de *"Funcionário"* que difere dos demais por ser gerente de algum departamento. No modelo EER é possível representar os gerentes como uma subclasse da entidade *"Funcionário"*, que neste contexto, se torna uma superclasse para *"Gerente"*. Os gerentes são um subconjunto do conjunto dos funcionários da empresa. As subclasses são ligadas diretamente a sua superclasse por uma linha com uma seta apontando para a superclasse. Ainda neste exemplo, o relacionamento *Gerencia*, mostrado na Figura 2.3, associando *"Funcionário"* e *"Departamento"*, pode, neste contexto, passar a ser entre *"Gerente"* e *"Departamento"*, uma vez que somente os funcionários gerentes é que gerenciam algum departamento. Outra modificação é que este relacionamento passa a ser de participação total por parte de *"Gerente"*, indicando que todo gerente, de fato, está associado ao departamento gerenciado.

No exemplo da Figura 2.6 a entidade *"Funcionário"* apresenta um subagrupamento especial de funcionários que são gerentes.

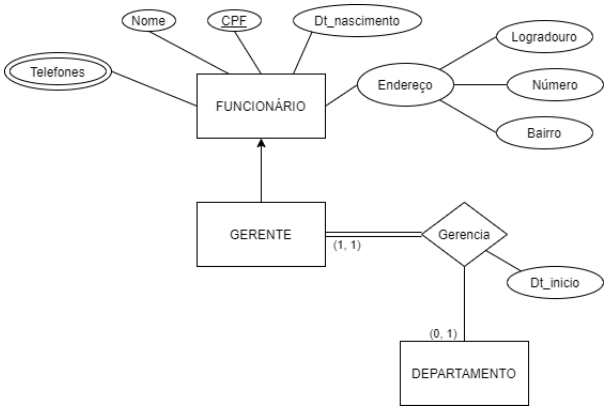

Figura 2.6. A superclasse *"Funcionário"* e a subclasse *"Gerente"*

Especializações e generalizações

Uma *especialização* é um processo que acontece quando uma entidade é dividida, por alguma característica ou critério específico, em um conjunto de subclasses que enfatizem as diferenças que sejam consiceradas significativas entre as ocorrências de uma superclasse. Neste contexto, as subclasses de especialização podem formar subconjuntos disjuntos da superclasse, ou seja, nestes casos não existem ocorrências que participem de mais do que uma subclasse da mesma especialização. Outra possibilidade é que as subclasses permitam uma sobreposição de ocorrências. Ou seja, uma mesma ocorrência da superclasse pode ser representanda por ocorrências em duas ou mais subclasses no mesmo processo de especialização.

A Figura 2.7 mostra a divisão de *"Funcionário"* em *"Horista"* e *"Mensal"*. Esta situação evidencia que os funcionários da empresa são subdivididos em dois tipos: funcionários mensais, que recebem um *salário* fixo pago mensalmente e funcionários horistas, que possuem uma *escala de pagamento* que define como e quanto será pago. Neste exemplo, a linha dupla ligada a FUNCIONÁRIO denota que a superclasse tem *participação total* na especialização, isto é, todo funcionário da empresa obrigatoriamente se enquadra como um funcionário mensal ou como um funcionário horista. A união entre os elementos das subclasses *"Horista"* e *"Mensal"* representa todos os funcionários da empresa. O círculo anotado com a letra "*D*" indica que *"Mensal"* e *"Horista"* correspondem a *subconjuntos disjuntos* de funcionários, ou seja, uma ocorrência de funcionário em *"Mensal"* não será observada em *"Horista"* e vice-versa.

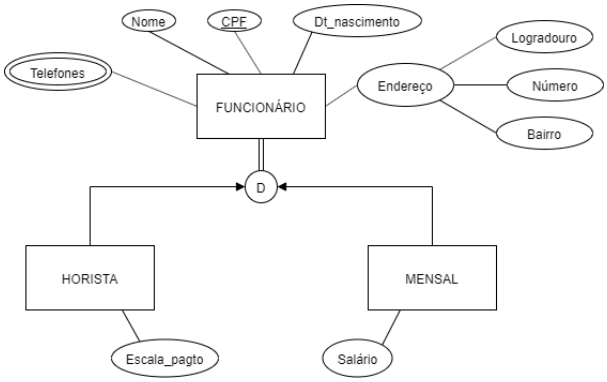

Figura 2.7. A especialização de *"Funcionário"* em *"Horista"* e *"Mensal"*

O exemplo da Figura 2.8 mostra que a entidade *"Funcionário"* pode ser dividida em *"Técnico"* e *"Engenheiro"*, entretanto a participação de *"Funcionário"* na especialização não é total, denotado pela linha simples ligada à superclasse. Neste contexto, uma ocorrência de *"Funcionário"* pode não pertencer nem a *"Técnico"* e nem a *"Engenheiro"*. As subclasses nesse exemplo ainda formam subconjuntos disjuntos de *"Funcionário"*, porém a união entre os conjuntos representados pelas subclasses não abrange todos os elementos da superclasse *"Funcionário"*.

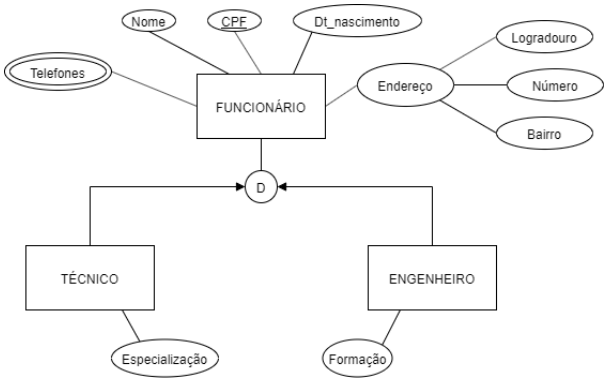

Figura 2.8. A especialização de *"Funcionário"* em *Técnico* e *"Engenheiro"*

O modelo EER permite que vários processos de especialização possam ser modelados para uma mesma superclasse, de acordo com as necessidades do projeto do banco de dados, verificadas após a análise de requisitos. Neste caso, subclasses ligadas a uma mesma superclasse, mas de especializações diferentes podem ter elementos em comum.

A Figura 2.9 mostra a organização dos funcionários da empresa de acordo com as subclasses identificadas e modeladas a partir de um levantamento de requisitos prévio. Os

atributos neste exemplo foram suprimidos para melhor visualização da superclasse *"Funcionário"* e suas subclasses. A especialização mais à esquerda indica que *todo* funcionário da empresa se enquadra como *"Horista"* ou *"Mensal"*, sem exceção, sendo estas subclasses representadas por subconjuntos disjuntos de funcionários. A subclasse *"Gerente"*, ao centro, indica que alguns funcionários são gerentes de departamento. Já a especialização mais à direita, indica que funcionários podem ser técnicos ou engenheiros, mas não necessariamente um dos dois. As especializações são todas independentes umas das outras, indicando que uma ocorrência de funcionário, por exemplo, o *"João da Silva"*, pode ser um técnico horista e não gerente. Em um outro exemplo, é possível obter a ocorrência da funcionária *"Isabel Fernandes"* como sendo uma engenheira gerente e mensalista. Como a superclasse *"Funcionário"* não tem participação total na especialização *Técnico/Engenheiro*, ainda é possível exemplificar o caso do funcionário Miguel Souza como um simples funcionário de pagamento mensal.

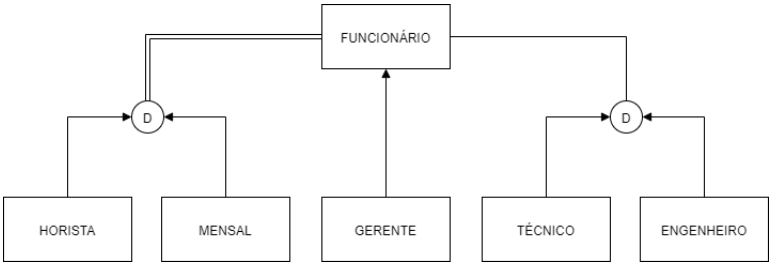

Figura 2.9. Especializações de *"Funcionário"*

Na Figura 2.10 a entidade *"Funcionário"* é apresentada como um conjunto de funcionários (F1, F2, ..., F8). Em *(a)* pode-se perceber que os funcionários foram divididos entre funcionários horistas e funcionários mensais e que todos os funcionários se enquadram em um dos subgrupos. Nesta situação fica evidenciada a participação total de *"Funcionário"* na especialização *Horista/Mensal*. Em *(b)* a superclasse *"Funcionário"* apresenta um subgrupo de gerentes. Todo gerente é um funcionário, mas nem todo funcionário é um gerente. Já em *(c)*, o conjunto de funcionários apresenta dois subconjuntos disjuntos de técnicos e engenheiros. Neste caso *"Funcionário"* não tem participação total na especialização *Técnico/Engenheiro*, portanto existem alguns funcionários (F1, F2 e F4) que não são nem técnicos e nem engenheiros.

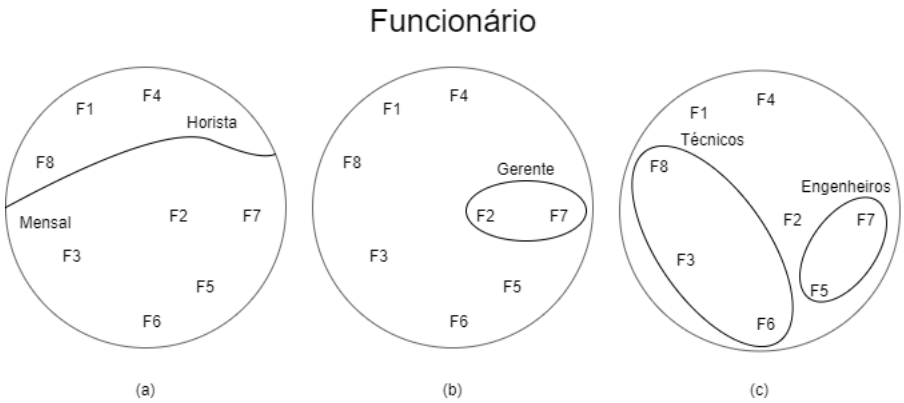

Figura 2.10. Ocorrências de *"Funcionário"* e suas subclasses

2.2. Projetos Lógicos de Bancos de Dados

Um modelo de dados é uma coleção de conceitos que descrevem a estrutura de um banco de dados e oferecem um nível de abstração que permite um melhor entendimento dos dados (Elmasri et al.; 2011). Em termos de modelagem conceitual de dados, o mais tradicional é utilizar o modelo ER e suas variações por se tratar de um modelo em alto nível, portanto livre de paradigmas de implementação, de representação do banco de dados que pode ser utilizado por projetistas de bancos de dados tanto no diálogo com os usuários, quanto na definição dos detalhes técnicos juntamente ao analista responsável pela implementação.

A grande vantagem da utilização de modelagem conceitual de dados é que ela garante uma conversação mais fluida entre os usuários e analistas, pois, neste nível de abstração, um modelo conceitual é de fácil assimilação por parte dos usuários, mesmo aqueles que não têm experiência com bancos de dados. Ao mesmo tempo, um modelo conceitual pode ser mapeado tanto para bancos de dados relacionais, num processo já bastante conhecido, quanto para bancos de dados *NoSQL*, onde a categoria do banco de dados a ser utilizado dependerá dos requisitos do sistema a ser implementado.

Portanto, em uma abordagem mais tradicional de desenvolvimento de um sistema de banco de dados, a primeira fase trata da modelagem conceitual, podendo ou não utilizar ferramentas CASE, para elaboração de um diagrama ER. A partir deste diagrama, a segunda fase trata do projeto lógico, que, por sua vez, considera os detalhes técnicos da implementação e define o tipo do banco de dados a ser modelado, seja este banco de dados

relacional ou não.

2.2.1. O Modelo de Dados Relacional

Um banco de dados relacional organiza seus dados em relações. Já uma relação é um conjunto de tuplas, onde cada tupla possui um conjunto de atributos associados a um determinado tipo ou a um domínio específico e representa um registro de algum objeto do mundo real. Cada relação é descrita por um *esquema de relação*, que define quais são e de que tipo/domínio são seus atributos (Elmasri et al.; 2011). De forma mais prática, uma relação é análoga a uma tabela, onde as linhas representam os registros (tuplas) e as colunas representam os atributos. Para Date (2004), um banco de dados relacional é aquele onde os dados são percebidos pelos usuários como um conjunto de tabelas que satisfazem a certas regras de integridade definidas pelo projetista e que são manipulados por operadores que derivam em novas tabelas. Bancos de dados relacionais seguem estritamente os esquemas definidos para suas relações e qualquer mudança na estrutura de dados ou na maneira como os dados são tratados e armazenados deve ser refletida nestes esquemas.

O modelo de dados relacional pode ser representado por um conjunto de diagramas utilizados pelos projetistas de bancos de dados para descrever sua estrutura, isto é, seu conjunto de relações, atributos, relacionamentos e restrições de integridade. É comum a utilização de ferramentas CASE e diagramas de esquema, (Elmasri et al.; 2011) para uma melhor visualização desta estrutura e verificar os detalhes de cada relação, seu conjunto de atributos, seus inter relacionamentos e as restrições estabelecidas. Também é bastante comum obter a estrutura de tabelas do modelo relacional executando um conjunto de algoritmos de mapeamento a partir do modelo conceitual, geralmente diagramas ER e variações (Elmasri et al.; 2011; Elfaki et al.; 2019). Finalmente, a partir do modelo relacional é possível gerar o esquema completo do banco de dados relacional e criar sua estrutura dentro do SGBD.

Para manter a integridade das informações no banco de dados, ou seja, garantir que não hajam inconsistências nos dados armazenados, o modelo relacional utiliza um conjunto bastante rígido de *restrições de integridade*, que são provenientes do domínio que o banco de dados representa no mundo real (Elmasri et al.; 2011). Essas restrições definem desde o tipo/domínio de cada atributo das relações até as regras de integridade referencial e de

exclusividade sobre valores (Elmasri et al.; 2011) e fazem parte do esquema lógico de um banco de dados relacional.

Bancos de dados relacionais geralmente são apoiados por uma arquitetura centralizada, o que ajuda a manter a consistência dos dados, entretanto prejudica sua capacidade de aumento de desempenho (escalabilidade) e de tolerância a falhas. A maneira mais usual de aumentar o desempenho em bancos de dados relacionais é a *escalabilidade vertical*, que, segundo Pokorny (2013), significa investir em servidores cada vez mais caros e muitas vezes pouco confiáveis em termos de ganho de desempenho. Uma alternativa é adaptar os bancos de dados relacionais para a *escalabilidade horizontal*, ou seja, distribuir o banco de dados em um cluster de servidores com redimensionamento dinâmico, permitindo que mais nós possam ser adicionados ao cluster, reorganizando os recursos compartilhados sem perder desempenho e com tolerância a falhas. Uma solução alternativa destacada por Pokorny (2013) é a utilização de bancos *NoSQL*, que já são projetados para escalabilidade horizontal, oferecendo uma melhora significativa no desempenho investindo em clusters de servidores mais baratos.

2.2.2. Modelos de Dados *NoSQL*

Desde o início dos anos 2000 o *Big Data* vem se tornando um dos grandes desafios para a Ciência da Computação em virtude do volume de dados cada vez maior a ser gerenciado pelos bancos de dados aliado a exigência por desempenho cada vez mais rápido (Sousa & Pereira; 2015). Além disso, novas aplicações da *Web* têm trazido à tona um novo conjunto de demandas como alta concorrência em operações tanto de leitura quanto de escrita mantendo uma baixa latência, a necessidade por alta escalabilidade e alta disponibilidade ao mesmo tempo em que devem ser mantidos baixos os custos operacionais e de gerenciamento da estrutura física dos servidores do banco de dados (Roy-Hubara et al.; 2017).

Além das novas demandas, contribui para a complexidade do *Big Data* a enorme variedade apresentada por esses dados, uma vez que podem ser provenientes de diversas fontes como, por exemplo, planilhas, documentos XML e até mesmo de bancos de dados relacionais. Grolinger et al. (2013) definem *Big Data* como "um termo usado para se referir a conjuntos de dados massivos e complexos compostos de uma variedade de estruturas

de dados, incluindo dados estruturados, semiestruturados e não estruturados". Segundo Alexandre & Cavique (2013), *Big Data* pode ser caracterizado por um conjunto de propriedades denominado 3Vs:

- **Volume**: O volume extremamente massivo de dados que cresce exponencialmente com o tempo. Armazenar e gerenciar esses dados de maneira sustentável passa a ser um requisito importante ao lidar com *Big Data*;
- **Velocidade**: A transmissão e análise dos dados deve ser feita de forma extremamente rápida, se possível, em tempo real. Um canal de acesso de alta velocidade aos dados pode oferecer vantagens de competitividade aos clientes.
- **Variedade**: O formato dos dados armazenados pode variar muito, desde dados tabulares, passando por estruturas hierárquicas como XML e chegando a arquivos binários.

Neste cenário, os SGBDs relacionais que não atentam para estas características passaram a apresentar limitações para atender as novas demandas apresentadas. A partir daí, surgiram os bancos de dados *NoSQL*, como alternativa para lidar com essa nova problemática, oferecendo suporte a um novo conjunto de recursos e mudando o foco para propriedades como alta disponibilidade de serviços e escalabilidade horizontal visando uma maior eficiência e melhoria de desempenho das consultas (Abramova et al.; 2014).

Para evidenciar uma mudança de paradigma em relação aos bancos de dados relacionais e atender aos novos requisitos impostos, algumas características são associadas a bancos *NoSQL*, como a flexibilização em relação a esquemas rígidos a fim de lidar com a variedade dos tipos de dados e permitir uma evolução mais dinâmica ao longo do tempo; a possibilidade de escalabilidade horizontal, acrescentando dinamicamente novos nós a um cluster de servidores com o propósito de melhorar o desempenho e o tempo de resposta; o foco na alta disponibilidade e na tolerância ao particionamento em detrimento à consistência e em favor de um alto grau de distribuição; a prevalência das propriedades BASE em detrimento aos princípios ACID a fim de permitir uma maior flexibilidade a mudanças e aumento de desempenho no gerenciamento dos dados armazenados (Sousa & Pereira; 2015).

Sousa & Pereira (2015) destacam ainda que, apesar das características elencadas acima,

não existe consenso sobre o que, de fato, constitui um modelo de dados *NoSQL*. Outro contraponto é apresentado por Comyn-Wattiau & Akoka (2017), que destacam a importância dos esquemas conceitual e lógico em um projeto de banco de dados e que geralmente estão ausentes em um banco de dados *NoSQL*, podendo trazer dificuldades tanto no entendimento de sua estrutura quanto na elaboração de consultas com algum grau de complexidade.

Além disso, ao contrário do que acontece com os SGBDs relacionais, que seguem uma modelagem e conjunto de regras padronizados e são essencialmente muito parecidos, SGBDs *NoSQL* podem abordar soluções distintas entre si e são subdivididos em quatro grandes categorias (Abramova et al.; 2014; Atzeni et al.; 2016):

- **Chave-valor**: Nesta abordagem o banco de dados é organizado como uma tabela *hash* onde os dados são armazenados em pares *chave-valor*. Para acessar o valor de algum registro armazenado, basta fornecer a sua chave, que no banco de dados, tem um valor único (Abramova et al.; 2014).
- **De Documentos**: O banco de dados é organizado em uma coleção de documentos, que utilizam um padrão de formatação como *XML* ou *JSON*. Cada documento organiza os dados em estruturas hierárquicas contendo texto simples e referências a outros documentos do banco de dados (de Lima & dos Santos Mello; 2015).
- **Colunares**: Bancos de dados colunares podem ser entendidos como uma estrutura de dois níveis, onde o primeiro nível é organizado como uma estrutura chave-valor, que identifica, num segundo nível, um mapa de valores denominado coluna. Os valores nas colunas são indexados por um nome e possuem um *timestamp* (Poffo; 2016).
- **De Grafo**: Os dados armazenados neste tipo de banco de dados são organizados em formato de grafo, onde os nós representam instâncias das entidades representadas e as arestas representam as associações que as instâncias mantêm entre si. Bancos de dados de grafo são bastante comuns para representar redes sociais, e sistemas de mapas e rotas e dão especial atenção aos relacionamentos entre as entidades. Os SGBDs de grafo são o foco deste trabalho.

2.2.3. O Modelo de Dados em Grafos

O modelo de dados em grafos representa o banco de dados como um grande grafo, onde as ocorrências das entidades são representadas pelos nós enquanto que as ocorrências dos relacionamentos são representadas pelas arestas. Particularmente, os Bancos de dados de Grafos (BDGs) podem armazenar não somente dados sobre as entidades, mas dão uma atenção especial para os dados associados aos relacionamentos. Estes relacionamentos podem ser uma alternativa muito interessante para sistemas onde os dados são altamente interligados entre si, ou seja, estes relacionamentos têm tanta importância quanto as entidades interligadas (Neo4j; 2020b). Unal & Oguztuzun (2018) destacam que BDGs oferecem uma maneira eficiente de percorrer entre dados altamente conectados, uma vez que essas conexões são armazenadas como arestas, o que não gera uma alto custo computacional como as operações de *join* executadas por bancos de dados relacionais.

O exemplo clássico para aplicação de um BDG é a implementação de um sistema de rede social, onde os nós representam as pessoas enquanto as arestas representam as conexões entre as pessoas.

A Figura 2.11 mostra a interação entre usuários de uma rede social. Cada nó representa um usuário da rede, enquanto que os relacionamentos são representados por arestas direcionadas indicando uma ordem de usuários "seguidores" e "seguidos". No exemplo, o usuário *João* é *seguidor* de *Lucas* que, por sua vez, é *seguidor* de *João* e de *Aline*. Por fim, *Aline* é *seguidora* de *João*.

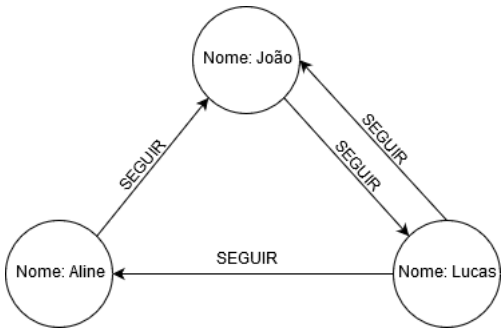

Figura 2.11. Uma rede social representada por um grafo

O exemplo da Figura 2.11 denota que uma rede social segue naturalmente o formato e comportamento de um grafo, evidenciando mais uma vantagem deste modelo de dados.

Embora a amostragem de usuários e relacionamentos seja bastante pequena, uma rede social no mundo real, com possivelmente milhões de usuários conectados, segue basicamente o mesmo comportamento do exemplo dado (Robinson et al.; 2013). Ainda que este exemplo possa ser modelado como um banco de dados relacional, a possibilidade de ter um grande número de usuários com, potencialmente, um número muito grande de relacionamentos, torna esta alternativa, mais tradicional, um entrave do ponto de vista do desempenho. Neste caso, precisamos executar consultas que envolvam junções em tabelas extremamente povoadas e que podem, eventualmente, retornar um número exorbitante de resultados. Por outro lado, a abordagem de um modelo de grafos trata os dados de maneira mais natural e fluida para este exemplo, já que a própria rede social é, de fato, um grafo e as soluções que envolvem BDGs priorizam a resolução de problemas onde as entidades participantes possam apresentar uma grande quantidade de relacionamentos.

Entretanto, para compreender melhor o funcionamento de um BDG é necessário estar ciente dos conceitos básicos sobre Teoria dos Grafos que norteiam este trabalho.

Conceitos Básicos de Grafos

Um grafo pode ser descrito como um conjunto *V*, não vazio, de nós (ou vértices) e um conjunto *A* de arestas as quais conectam pares, não necessariamente distintos, de nós (Costa; 2011). De maneira menos formal, um grafo é composto por dois conjuntos, sendo o primeiro um conjunto contendo um ou mais nós, que podem ser representados graficamente como pontos ou círculos. O segundo conjunto, possivelmente vazio, contém arestas, onde cada aresta é representada como uma linha que conecta dois nós do grafo. Considerando os nós *v1* e *v2* e a aresta *a1* conectando estes dois nós, diz-se que *v1* e *v2* são *adjacentes* e que *a1* incide sobre *v1* e sobre *v2*. Caso *v1* e *v2* sejam o mesmo nó, então a aresta *a1* é chamada de *laço*.

A Figura 2.12 mostra um exemplo de grafo. Graficamente, um grafo é apenas um conjunto de pontos que podem ser interligados por arestas e segundo Robinson et al. (2013) "representam entidades como nós e as maneiras pelas quais essas entidades se relacionam com o mundo como relacionamentos". Semanticamente, grafos podem ser usados para representar problemas do mundo real, como por exemplo, as ligações químicas entre os

átomos de uma molécula, um conjunto de cidades e as rotas e distâncias entre elas ou até mesmo um grupo de pessoas e seus graus de parentesco. A depender da natureza do problema a ser representado por um grafo, ele pode apresentar propriedades especiais para seus conjuntos de nós e de arestas.

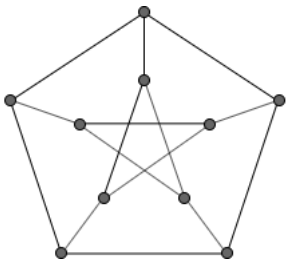

Figura 2.12. Grafo de Petersen

Alguns grafos apresentam como característica não possuírem laços e nem *arestas múltiplas*, ou seja para quaisquer dois nós adjacentes e distintos *v1* e *v2*, existe *uma, e somente uma,* aresta conectando *v1* e *v2* (Costa; 2011). Estes são denominados *grafos simples*. Entretanto, podem haver grafos, denominados *multigrafos*, que admitem múltiplas arestas diferentes conectando o mesmo par de nós *v1* e *v2* (Feofiloff et al.; 2011). Cada uma destas arestas pode representar, neste contexto, uma associação de natureza distinta entre *v1* e *v2*. Com relação especificamente às arestas, é possível admitir um terceiro tipo de grafo, cujas arestas são *dirigidas*, ou seja, mantêm uma ordem entre o par de nós adjacentes indicando qual é o nó de *origem* e qual é o nó de *destino*. Arestas dirigidas são chamadas de *arcos* e têm uma seta apontando para o nó de destino (Prestes; 2016). Uma representação com arestas dirigidas pode ser utilizada, por exemplo, para indicar relacionamentos hierárquicos, estabelecendo uma ordem entre o nó de origem e o nó de destino. Para efeitos práticos, uma aresta não dirigida pode ser, em alguns casos, representada como um par de arcos ligando os mesmos nós, porém com sentidos opostos (Pokorny; 2016). Também é possível representar uma aresta comum como um arco com setas em ambos os lados.

A Figura 2.13 exemplifica os três tipos de grafos definidos anteriormente. A Figura 2.13 *(a)* mostra um grafo simples representando um conjunto de moléculas de água (H_2O) se conectando para se organizar em estado líquido. As arestas representadas por linhas contínuas ligam os elementos (hidrogênio e oxigênio) que formam cada molécula de água

em particular, enquanto que as arestas representadas por linhas pontilhadas representam as ligações entre as moléculas. O multigrafo representado em 3.3 *(b)* indica a existência de rotas entre cidades do estado de São Paulo. Neste caso, as linhas contínuas indicam que existe uma rota que pode ser feita de carro, enquanto que as linhas pontilhadas indicam que existe um trajeto que pode ser feito de trem. Por fim, em 3.3 *(c)* exemplifica-se um grafo dirigido representando algumas pessoas e seus graus de parentesco. O uso de arcos em lugar de arestas neste grafo é necessário para denotação exata da relação entre duas pessoas, como, por exemplo, denotar que *Lourival* é pai de *Geovane* e não o contrário, já que o arco rotulado como *pai* segue esta ordem. Eventualmente, existem arcos bidirecionados, neste caso, representados pelas relações entre *Fábio* e *Jéssica*, que são *primos* e entre *Marcelo* e *Rose*, que são esposo e esposa, indicando que nestas relações não existe um relacionamento mandatório, ou seja, Jéssica é prima de Fabio e Fabio é primo de Jéssica; e Marcelo é esposo de Rose e Rose é esposa de Marcelo.

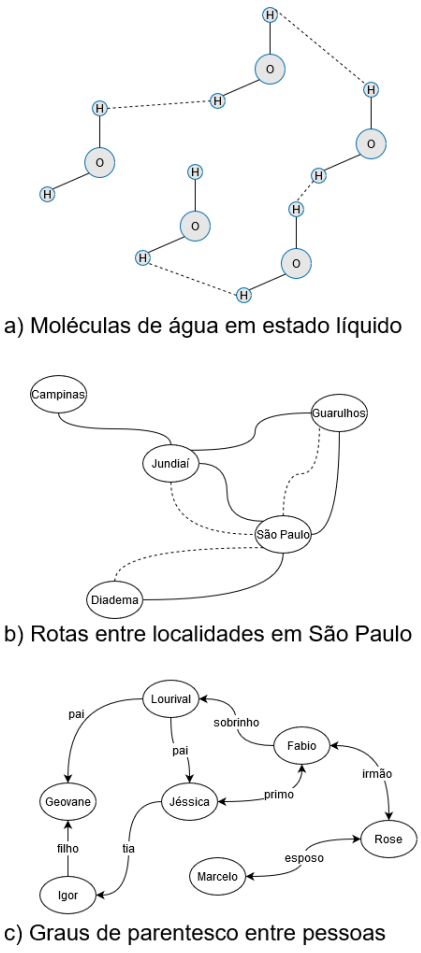

Figura 2.13. Grafo Simples (a), Multigrafo (b) e Grafo Dirigido (c)

O estudo sobre grafos compõe um grande ramo da Matemática e existe uma extensa literatura lidando com seus conceitos, aplicações, problemas clássicos e suas soluções. As suas aplicações podem permear áreas como a Biologia e a Química, além de serem bastante utilizadas na Ciência da Computação nas subáreas de Robótica, Redes Neurais, Autômatos e Estruturas de Dados (Prestes; 2016).

Grafos de Propriedades

Segundo Pokorny (2016), a modelagem de dados utilizando grafos, na grande maioria dos casos, recorre a um tipo específico de grafo: o chamado *grafos de propriedades*. De acordo com Robinson et al. (2013), um grafo de propriedades é uma ferramenta bastante intuitiva e de fácil entendimento, além de possuir as seguintes características:

- **Nós rotulados**: Todos os nós contêm pelo menos um rótulo, que indica a(s) entidade(s) representada(s) pelo nó. Conjuntos de nós com o mesmo rótulo representam ocorrências distintas da mesma entidade;
- **Nós contêm propriedades**: cada propriedade contida em um nó representa uma unidade de dados e é identificada como um par chave-valor;
- **Arestas rotuladas**: as arestas devem possuir um único rótulo e uma direção, indicando respectivamente o relacionamento que elas representam e os nós de origem e de destino;
- **Arestas podem conter propriedades**: Caso necessário, as arestas podem conter uma ou mais propriedades, assim como os nós.

A Figura 2.14 mostra um grafo de propriedades representando ocorrências das entidades/-relacionamentos modelados de acordo com o diagrama da Figura 2.2. Neste exemplo existem dois nós representando os funcionários "*João da Silva*" e "*Isabel Fernandes*", além do nó representando o departamento de "*Produção*" localizado em "*Campinas*". Nota-se que, mesmo nós com o mesmo rótulo podem possuir conjuntos de atributos distintos. No exemplo, o funcionário "*João da Silva*" possui os atributos "*Nome*" e "*CPF*", enquanto que a funcionária "*Isabel Fernandes*" conta com os atributos "*Nome*" e "*Dt_nascimento*". As arestas rotuladas como "*Trabalha_em*" representam os relacionamentos de ambos os funcionários com o departamento em que trabalham e mantêm a propriedade "*Dt_inicio*",

indicando a data em que começaram a trabalhar no departamento. A aresta entre a funcionária "*Isabel Fernandes*" e o departamento de "*Produção*", rotulada como "*Gerencia*", indica que esta funcionária é a gerente deste departamento e iniciou sua gestão em "*31/08/2017*".

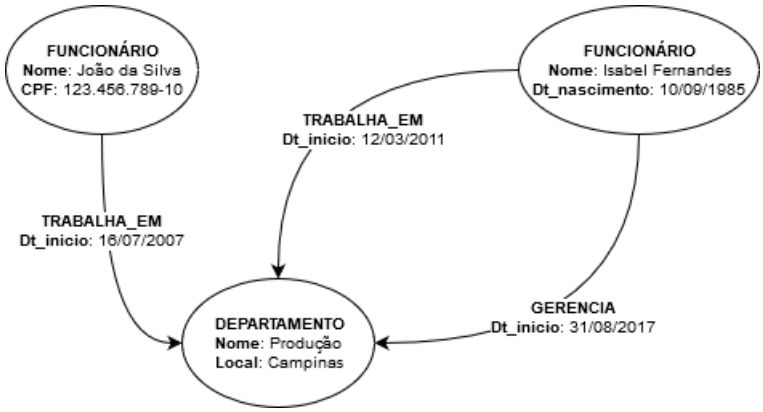

Figura 2.14. Grafo de Propriedades

Resumidamente, um grafo de propriedades pode ser descrito como um "multigrafo dirigido, rotulado e com atributos" (Pokorny; 2016). Este tipo de grafo agrega conjuntos de propriedades aos nós e arestas para armazenar dados e suas arestas dirigidas dão um maior sentido semântico aos relacionamentos, fazendo deste uma ferramenta bastante útil para o projeto de BDGs. Além disso, Kusu & Hatano (2018) destacam que grafos de propriedades fornecem uma maneira eficiente, gerenciável e estável de fazer travessias mesmo em conjuntos massivos de dados.

Hipergrafos e Grafos Aninhados

Em ocasiões específicas é necessário recorrer a classes especiais de grafos a fim de ganhar representatividade e maior significado semântico na modelagem de dados. Neste contexto, faz-se necessário apresentar os conceitos fundamentais de *hipergrafos* e de *grafos aninhados*. De acordo com Gallo et al. (1993) e Pokorny (2016), hipergrafos são uma generalização dos grafos *padrão* que permitem que uma mesma aresta, neste caso chamada de *hiperaresta*, conecte arbitrariamente mais de dois nós.

Formalmente, um hipergrafo é um par $\mathcal{HG} = \{\mathcal{V}, \mathcal{H}\}$, onde $\mathcal{V}$ representa um conjunto de n nós (ou vértices) e $\mathcal{H}$ representa um conjunto de m hiperarestas. A *cardinalidade* de

uma hiperaresta h_i, i = $\{1, 2, ..., m\}$, denotada por $|h_i|$, é a quantidade de nós em que ela incide. Obviamente, um hipergrafo onde $|h_i|$ = 2 para todo h_i pertencente a $\mathcal{H}$, denota um grafo padrão, onde todas as arestas são binárias.

Uma variante do hipergrafo é o *hipergrafo dirigido*, onde as hiperarestas são *dirigidas*, ou também denominadas *hiperarcos*. Segundo Gallo et al. (1993), uma hiperaresta dirigida é denotada pelo par $h = \{\mathcal{X}, \mathcal{Y}\}$ de subconjuntos disjuntos de nós do hipergrafo, onde $\mathcal{X}$ representa o conjunto de nós que estão na *cauda* da hiperaresta *h*, enquanto que $\mathcal{Y}$ representa o conjunto de nós que estão na *cabeça* da aresta *h*. Hiperarestas dirigidas possuem uma seta que aponta da *cauda* em direção a *cabeça*.

Um grafo que contenha *grafos aninhados* ou *hipernós* é um tipo especial de grafo onde alguns de seus vértices podem ser definidos como grafos menores (Pokorny; 2016; Catarino; 2017), de acordo com a conveniência e para uma melhor visualização do problema. Este recurso pode ser bastante útil em diminuir a complexidade de representação do problema como um todo e, quando necessário, permitir uma análise mais detalhada ao expandir um hipernó para sua forma de grafo.

A Figura 2.15 *(a)* mostra um hipergrafo dirigido que contém uma única hiperaresta ligando os nós *1*, *2* e *3*, aos nós *4* e *5*. Neste caso a cauda da hiperaresta é composta pelos nós *1*, *2* e *3* enquanto que a cabeça da hiperaresta é dada pelos nós *4* e *5*. O grafo apresentado em *(b)*, contém 5 nós, onde os nós *4* e *5* são considerados hipernós, pois, de forma expandida, esses nós são definidos como grafos menores.

Modelagem Lógica com Grafo de Propriedades

A modelagem de um BDG pode ser entendida como a definição de seu esquema lógico, neste caso representado por um grafo que descreve como a instância do banco de dados deve ser construída e como os dados são organizados. Segundo Pokorny (2016), para ser considerado um modelo de BDG completo é necessário oferecer os recursos elencados abaixo:

- Tanto o esquema do BDG quanto o próprio BDG devem ser representados como grafos e suas generalizações (hipergrafos e grafos aninhados);
- As operações de manipulação dos dados do BDG devem ser implementadas como

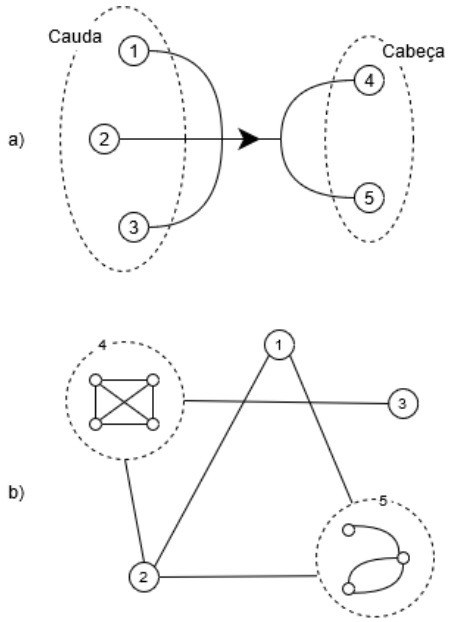

Figura 2.15. Hipergrafo (a) e Grafo Aninhado (b)

operações típicas de grafos, como travessia, inserção, alteração e exclusão de nós e arestas;

•O modelo deve dar suporte a um conjunto de *restrições de integridade* para garantir a consistência dos dados armazenados no grafo.

Entretanto, Pokorny (2016) enfatiza que a maioria dos modelos implementados comercialmente costuma ignorar pelo menos uma das características acima, em especial o tratamento mais detalhado de restrições de integridade e de dependências funcionais.

A proposta de modelagem lógica de BDG deste trabalho consiste na definição de uma série de algoritmos de alto nível para transformação do modelo conceitual EER, o qual considera entidades, relacionamentos binários e n-ários, generalizações/especializações e restrições de integridade, em um esquema de grafo de propriedades que represente a estrutura, organização e restrições sobre as entidades e relacionamentos dentro do BDG. Sendo assim, qualquer instância do BDG pode ser validada por este *grafo de esquema* e garantir a consistência e a representatividade dos dados armazenados no BDG.

A Figura 2.16 mostra um exemplo de modelo conceitual EER a ser mapeado para um modelo lógico de grafo de propriedades. Este modelo exemplifica o banco de dados da empresa, representando a entidade *"Funcionário"* e seus relacionamentos com a entidade fraca *"Dependente"* (Funcionário *tem* Dependente) e a com a entidade *"Departamento"*

(Funcionário *trabalha em* Departamento). Neste exemplo, a entidade *"Funcionário"* se especializa na subclasse *"Gerente"*, e esta se associa a *"Departamento"* através do relacionamento *"Gerencia"*. O diagrama da Figura 2.16 pode ser mapeado para o grafo de esquema mostrado na Figura 2.17.

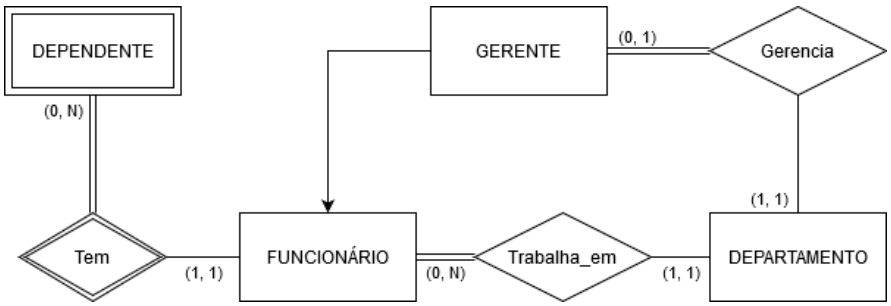

Figura 2.16. Modelo Conceitual do BDG Empresa

O grafo de esquema da Figura 2.17 pode ser entendido como um dos possíveis resultados do mapeamento a partir do modelo conceitual EER mostrado na Figura 2.16. Neste esquema, as entidades do modelo EER se convertem em nós do grafo de esquema, enquanto que os relacionamentos são mapeados para arestas. A subclasse *"Gerente"* é mapeada como um nó e ligado ao nó *"Funcionário"*, que representa a sua superclasse no modelo EER, através de uma aresta rotulada como "*É um*" no grafo de esquema. Como o modelo da Figura 2.16 não apresenta atributos, estes não foram mapeados para o grafo de esquema. A elipse pontilhada envolvendo os nós *"Funcionário"* e *"Gerente"* indica que seu conteúdo pode ser modelado como um *grafo aninhado*, uma vez que qualquer aresta incidente do/para o subgrafo composto por *"Funcionário"*, *"Gerente"* e seus relacionamentos representam uma única ocorrência do mundo real em uma instância deste modelo lógico.

A modelagem lógica usando grafos apresenta algumas limitações em relação ao modelo conceitual EER. Segundo Pokorny et al. (2017), o resultado do mapeamento de um modelo conceitual para um modelo lógico de grafos apresenta uma noção mais fraca de esquema de banco de dados por conta da necessidade de relaxar algumas restrições que são inerentes ao modelo conceitual a fim de satisfazer as notações gerais de grafos de propriedades. Neste contexto, Pokorny et al. (2017) destacam três tipos de restrições de integridade:

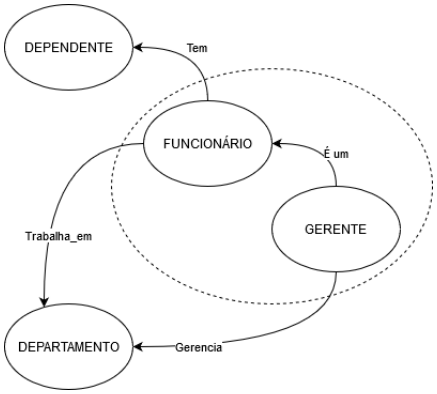

Figura 2.17. Grafo de Esquema do BDG Empresa

- **Restrições inerentes**: São regras inerentes ao modelo de dados utilizado e, sendo assim, não necessitam ser especificadas explicitamente no esquema do banco de dados;
- **Restrições explícitas**: São regras que devem ser especificadas no esquema do banco de dados utilizando sua *Linguagem de Definição de Dados (DDL)*;
- **Restrições implícitas**: São resultados da combinação lógica entre restrições inerentes e explícitas.

Uma dessas limitações dizem respeito à definição das *cardinalidades* que fazem parte do modelo EER, mas que não são inerentes ao modelo de grafos. Teoricamente não existem restrições quanto ao número de arestas que podem incidir a partir de ou finalizando em um nó do grafo. Em consequência disso, o modelo lógico da Figura 2.17 pode, por exemplo, validar instâncias onde uma determinada ocorrência de *"Funcionário"* se conecte a várias ocorrências de *"Departamento"* com arestas rotuladas como "*Trabalha_em*", causando uma inconsistência em relação ao modelo EER da Figura 2.16, o qual indica que cada funcionário trabalha em um único departamento por vez. Outra limitação é quanto à *participação* de uma entidade em um relacionamento. No diagrama da Figura 2.16, a participação de *"Funcionário"* no relacionamento "*Trabalha_em*" é *total*, conforme indicado pela linha dupla no diagrama. Isto indica que cada funcionário deve, obrigatoriamente, estar associado ao departamento em que trabalha. Por outro lado, é permitido que um departamento, eventualmente, não possua funcionários trabalhando, pois *"Departamento"* tem participação *parcial* no relacionamento "*Trabalha_em*". Estas restrições também não são inerentes aos modelos de grafos, entretanto são fundamentais

para a consistência dos dados e validação das instâncias do BDG.

A fim de garantir que o modelo lógico de grafos seja fiel ao modelo EER, é necessário garantir que todas as restrições que são inerentes ao modelo EER sejam implementadas como *restrições explícitas* no modelo lógico de grafos, portanto, para o exemplo do modelo apresentado na Figura 2.16 as seguintes restrições devem ser explicitadas no esquema lógico de grafos:

- De cada instância de *"Funcionário" deve partir* uma única aresta rotulada como *"Trabalha_em"*, a qual deve incidir em uma instância de *"Departamento"*;
- De cada instância de *"Gerente" deve partir* uma única aresta rotulada como *"Gerencia"*, a qual deve incidir em uma instância de *"Departamento"*;
- Em cada instância de *"Departamento" deve chegar* uma única aresta rotulada como *"Gerencia"*, a qual deve partir de uma instância de *"Gerente"*;
- Em cada instância de *"Dependente" deve chegar* uma única aresta rotulada como *"Tem"*, a qual deve partir de uma instância de *"Funcionário"*;

A partir do resultado do mapeamento do modelo EER exemplificado na Figura 2.16 para o modelo de grafos da Figura 2.17 e considerando as restrições explícitas listadas acima, uma possível instância do GDB é mostrada na Figura 2.18.

Para o exemplo da Figura 2.18 foi considerado o atributo *"Nome"* para as entidades *"Funcionário"*, *"Departamento"* e *"Dependente"*. Nesta instância do BDG, o departamento de "*Produção*" conta com os funcionários *"Miguel Souza"* e *"Isabel Fernandes"* e está sob a gerência desta última. Já no departamento de *"Engenharia"*, estão alocados os funcionários *"João da Silva"* e *"Geovane Santos"*, o qual é o gerente e tem *"Igor Santos"* como dependente. É importante notar que embora os departamentos tenham gerentes, não existe uma ocorrência explícita de *"Gerente"* no BDG, uma vez que todo gerente *é um* funcionário, representando o mesmo objeto do mundo real, porém com características específicas se comparado a um funcionário comum. Uma alternativa plausível seria acrescentar o rótulo *"Gerente"* aos funcionários *"Isabel Fernandes"* e *"Geovane Santos"* para indicar explicitamente que esses funcionários fazem parte de um subgrupo específico de funcionários que atuam como gerentes.

O capítulo 3 trata em detalhes o processo de mapeamento do modelo conceitual EER para

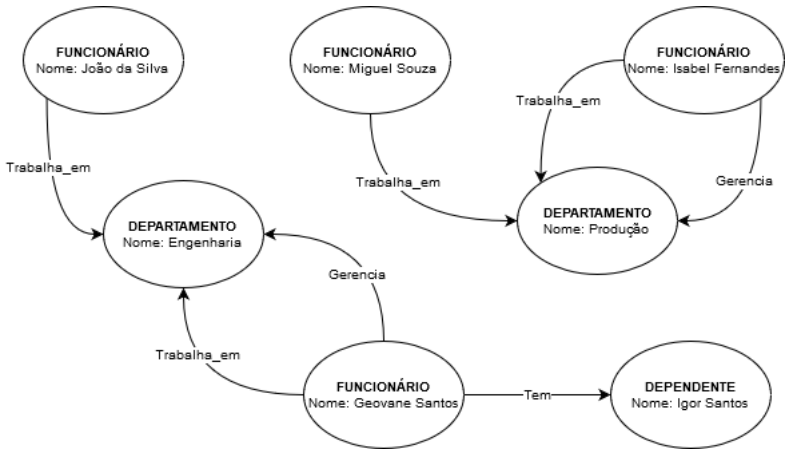

Figura 2.18. Instância Válida para o BDG Empresa

o modelo lógico de grafo de propriedades admitindo entidades, relacionamentos binários e n-ários, especializações/generalizações e considerando atributos monovalorados/multivalorados e simples/compostos. Os algoritmos propostos para o mapeamento para o modelo de BDG serão apresentados em alto nível, podendo ser implementados em qualquer linguagem de programação que for mais conveniente.

2.3. Sistemas Gerenciadores de Bancos de Dados em Grafos

A medida que a problemática envolvendo *Big Data* toma proporções cada vez maiores e que foi constatada a dificuldade dos SGBDs relacionais em fornecer uma solução viável, uma série de abordagens alternativas levaram a implementação de SGBDs que pudessem lidar com grandes quantidades de dados e que dessem suporte a escalabilidade horizontal com alguma política de replicação de dados a fim de manter uma alta disponibilidade aos dados e de melhorar o desempenho na transmissão e análise dos dados. Um subconjunto destas abordagens alternativas ficaram conhecidas como *NoSQL* e possuem características específicas para demandas específicas dos aplicativos aos quais dão suporte.

Alvarez et al. (2015) destaca SGBDs *NoSQL* de primeira e de segunda geração, sendo os de primeira geração aqueles que implementam apenas uma única categoria dos modelos de dados *NoSQL* (chave-valor, de documentos, colunares ou grafos), priorizando características e oferecendo recursos inerentes aos respectivos modelos lógicos. Já os SGBDs *NoSQL* de segunda geração combinam características de dois ou mais modelos lógicos a fim de agregar suas características e aumentar sua versatilidade ao lidar com a grande

variedade estrutural e lógica das fontes de dados. Por este motivo, os SGBDs de segunda geração também são conhecidos como *multimodelo*.

Entre os SGBDs *NoSQL* desenvolvidos, merecem destaque as soluções implementadas para dar suporte a bancos de dados em grafos (SGBDGs), uma vez que a *web* evoluiu rapidamente a partir do acesso a páginas meramente estáticas (Liptchinsky et al.; 2013) para lidar com uma rede que proporciona experiências altamente interativas, atribuindo grande relevância às redes sociais, cuja estrutura de dados pode ser naturalmente representada por grafos.

De acordo com Penteado et al. (2014), os SGBDGs podem ser classificados como *nativos* ou *não nativos*. Sistemas de grafos nativos tratam os dados fisicamente como grafos, mantendo listas de adjacências para cada nó, enquanto que sistemas não nativos podem utilizar outras estruturas de dados e modelos, dentre eles o relacional, para armazenar fisicamente os dados do grafo.

Este capítulo apresenta uma análise comparativa entre os principais SGBDGs que apresentam soluções nativas de armazenamento físico dos dados.

Comparação entre SGBDGs

No mundo real, em aplicações em que os dados podem ser representados naturalmente como grafos, e também considerando a heterogeneidade das estruturas de dados envolvidas e seu volume cada vez maior, alguns recursos podem ser considerados fundamentais na escolha de um SGBD que possa atender as necessidades demandadas:

- **Esquemas flexíveis** são importantes em um contexto em que os requisitos de sistemas mudam frequentemente e exigem uma grande capacidade de evolução e adaptação;
- **Alto desempenho** para operações de busca e manipulação de dados no banco de dados também tem sido uma característica bastante desejável, principalmente em aplicações que lidam com *Big Data*;
- **Escalabilidade horizontal** permite que o banco de dados possa ser distribuído em um *cluster* de servidores, aumentando a disponibilidade e o desempenho das aplicações;

- **Linguagem de consulta** que permita que as consultas possam ser feitas de forma simples e natural no banco de dados, proporcionando um rápido processo de aprendizagem dos desenvolvedores.

Considerando a abordagem tratada nesta dissertação, uma característica interessante é o tratamento do banco de dados como um grafo de propriedades.

Vale ressaltar que os todos os SGBDs considerados atendem aos recursos listados acima, permitindo alguma política de flexibilidade em seus esquemas, garantindo alto desempenho com o gerenciamento nativo dos dados, proporcionando escalabilidade, seja em *clusters* de servidores, seja dando suporte ao armazenamento em nuvem, e oferecendo alguma forma de manipulação de dados viável.

OrientDB

O *OrientDB* é um SGBD de grafos de segunda geração que implementa os modelos de dados de grafos, de documentos, chave/valor e orientado a objetos. É portanto um SGBD multimodelo, uma vez que oferece suporte a mais de um modelo de dados.

Em se tratando de bancos de dados de grafos, ele mantém listas de adjacências entre os nós para garantir um alto grau de desempenho em consultas. Entretanto, os dados dos nós e das arestas são armazenados como documentos, o que confere grande flexibilidade quanto a estrutura dos dados armazenados (Fernandes & Bernardino; 2018).

Este SGBD também dá suporte a bancos de dados em modo *schemaless* como padrão, o que permite que as ocorrências das mesmas entidades possam armazenar conjuntos de atributos distintos entre si. Entretanto, possibilita soluções tanto em *schemafull* quanto em *schema-hybrid*, sendo o primeiro caso utilizado quando as entidades tem um conjunto fechado de atributos obrigatórios, e o segundo sendo uma combinação entre os modos *schemaless* e *schemafull*, ou seja, possui um conjunto de atributos obrigatórios, mas permite que sejam inseridos atributos próprios (OrientDB; 2020).

O acesso e manipulação dos dados no *OrientDB* podem ser feitos com *APIs* do *Java* ou com uma linguagem baseada em *SQL* que não utiliza operações de junção (*join*) (Fernandes & Bernardino; 2018) e que executa independentemente do modelo de dados empre-

gado pelo banco de dados. No caso do modelo de dados de grafo, ainda é possível utilizar a *Gremlin API* para executar consultas, que oferece uma linguagem especializada para lidar com grafos de propriedades.

ArangoDB

O *ArangoDB* também é um SGBD multimodelo nativo que implementa os modelos de grafo, documento e chave-valor e oferece recursos de escalabilidade vertical e horizontal. Além disso, oferece uma linguagem própria para acesso e manipulação de dados: *Arango Query Language* ou simplesmente *AQL* (ArangoDB; 2020). A linguagem *AQL* permite uma maior fluidez entre instâncias de bancos de dados de modelos diferentes, o que garante que os dados podem ser acessados e manipulados independentemente do modelo de dados implementado.

Como principais vantagens, o *Arango* permite uma interface simplificada de acesso e manipulação de dados com a linguagem *AQL*, suporte a transações *ACID* em instâncias simples do banco de dados rodando em um servidor único e operações atômicas quando em modo *cluster*. Além disso permite uma arquitetura modular dando suporte a modelos de dados diferentes em cada cluster, se necessário (Fernandes & Bernardino; 2018).

Em caso de falha em algum *cluster*, o *ArangoDB* prioriza as características de consistência em detrimento à disponibilidade. Isto significa que os usuários deste SGBD têm sempre a mesma visão do banco de dados, independente do *cluster* ao qual estão conectados. Fernandes & Bernardino (2018) destacam ainda que o *ArangoDB* é considerado um banco de dados multimodelo e distribuído.

Microsoft Azure Cosmos DB

O *Microsoft Azure Cosmos DB* é o único SGBD comercial e é intitulado "o serviço de banco de dados multimodelo distribuído globalmente da Microsoft" (Microsoft; 2020). O seu conceito de distribuição global oferece um serviço de replicação possibilitando que os aplicativos possam acessar uma cópia dos dados fisicamente próxima dos usuários.

Como sistema multimodelo, o *Azure Cosmos* implementa os modelos de grafos, de do-

cumentos, chave-valor e da família de colunas. O serviço de BDG é fornecido através da API do *Gremlin* pelo *Azure Cosmos* prometendo dar suporte a "grandes grafos com bilhões de vértices e arestas" (Microsoft; 2020). O modelo de grafos utilizado por este SGBD é o de grafo de propriedades, com nós e arestas rotulados que mantêm as unidades de dados armazenadas em estruturas chave-valor.

O *Azure Cosmos DB* também gerencia os bancos de dados sem a necessidade de definição de um esquema (*schemaless*). Além disso, não oferece nenhum tipo de suporte a definição de esquemas ou regras mais rígidas quanto à definição da estrutura de dados de suas entidades e relacionamentos.

Neo4j

Segundo (Fernandes & Bernardino; 2018), o *Neo4j* é o SGBD de grafos mais utilizado por usuários e desenvolvedores, tendo a comunidade de bancos de dados de grafos mais ativa no mundo. Trata-se de um sistema gerenciador de primeira geração, que implementa exclusivamente o modelo de grafos de propriedades, em contraste com os outros SGBDs citados anteriormente, que são multimodelo e implementam mais de um modelo de dados distinto (Alvarez et al.; 2015).

Além disso, este SGBD implementa o banco de dados fisicamente como um grafo (Penteado et al.; 2014), mantendo listas de adjacências para cada nó instanciado, fornecendo alto desempenho às consultas (Neo4j; 2020a). Como características técnicas, o *Neo4j* implementa uma política de *schema-free*, não exigindo um esquema rígido para definição da estrutura e das restrições do banco de dados. Entretanto, permite que algumas restrições sejam definidas sobre as propriedades em nós com rótulos determinados, dando suporte a regras de obrigatoriedade, unicidade e identidade.

A manipulação de dados pode ser feita com uma linguagem própria chamada *Cypher*, que, segundo (Robinson et al.; 2013), foi projetada para ser uma ferramenta intuitiva e de fácil compreensão tanto para desenvolvedores quanto para projetistas de bancos de dados e é considerada uma linguagem bastante expressiva e compacta para lidar com BDGs. Sendo estruturada em cláusulas e utilizando-se de predicados, esta linguagem define padrões estruturais que são usados como parâmetros de busca no BDG.

2.3.1. Análise Comparativa

Dadas as variadas opções de sistemas gerenciadores que implementam o modelo de grafos, a exemplo os quatro SGBDGs citados nesta dissertação, faz-se necessário estabelecer um paralelo entre eles a fim de compreender suas características em comum, bem como as vantagens e desvantagens apresentadas por cada um. A fim de garantir uma opção segura e que dê suporte a um estudo de caso realmente significativo, pode-se elencar algumas características desejáveis em um SGBDG a fim de justificar sua escolha entre os demais.

1 - **Disponibilidade de informações:** neste tópico é considerada a quantidade de informações disponíveis sobre um determinado SGBD, sejam exemplos práticos, documentação técnica, livros, artigos, etc;
2 - **Plataforma de testes:** acesso facilitado a alguma plataforma para testes e aprendizagem;
3 - **Licença de uso:** neste caso, opta-se por aquelas ferramentas que ofereçam o máximo de recursos de forma livre e gratuita;
4 - **Esquemas:** considera-se que ferramentas que ofereçam previamente alguma política de definição de esquemas e restrições levam alguma vantagem sobre as demais;
5 - **Facilidade de manipulação:** consideram-se as linguagens utilizadas pelos sistemas gerenciadores que traduzem de maneira mais natural possível as operações de travessia em grafos.

No quesito *disponibilidade de informações*, o *Neo4j* tem um destaque muito grande em relação aos demais sistemas gerenciadores, tendo sido citado cerca de treze mil vezes em um motor de busca de material acadêmico na internet, enquanto que os outros gerenciadores de grafos somam cerca de quatro mil referências.

Em relação à *plataforma de testes*, o *Neo4j* apresenta uma ferramenta *online* para testes e aprendizagem, o que permite que o estudo dos recursos da ferramenta possam ser estudados sem a necessidade de instalação do *software* localmente. Este recurso também permite que os bancos de teste possam ser acessados em diferentes estações de trabalho.

Em relação à *licença de uso*, somente o *Microsoft Azure Cosmos DB* possui somente licença comercial. Os demais sistemas gerenciadores oferecem ao menos uma versão

com licença *open source*. Destes últimos, o *Arango DB* oferece uma licença comercial denominada *Enterprise* com uma gama de recursos focados em otimização de desempenho, escalabilidade e segurança.

Dentre os sistemas gerenciadores, tanto o *Arango DB* quanto o *Microsoft Aruze Cosmos* não disponibilizam nenhum nível de esquematização do banco de dados. O *Orient DB* oferece os recursos de *schemafull*, que considera um conjunto fechado e obrigatório de propriedades para para nós com um determinado rótulo; e *schema-hybrid*, onde é possível elencar propriedades obrigatórias, mas permite a customização dos nós adicionando propriedades particulares para cada ocorrência do nó. A definição de esquema do *Neo4j* permite configurar propriedades de determinados nós como *obrigatórias*, *únicas* ou como parte de um identificador para o nó.

Finalmente, a *facilidade de manipulação* de dados implica diretamente da maneira como as consultas podem ser estruturadas e executas pelo SGBD. Neste caso, considera-se que a linguagem *Cypher* do *Neo4j* possui as melhores características, por possuir uma sintaxe que se aproxima visualmente dos elementos do grafo e por possibilitar subconsultas aninhadas que facilitam a extração e manipulação de dados do grafo.

A Tabela 2.1 mostra um conjunto de características básicas consideradas relevantes em SGBDGs. Segundo esta tabulação, pode-se constatar que o sistema *Neo4j* pode ser considerado bastante adequado para testes e estudos de caso, já que possui uma licença *Open Source* que permite algumas opções de restrições de integridade e ter o suporte da linguagem *Cypher*. Além disso, os testes podem ser realizados em plataforma *online*, dando mais celeridade ao estudo de caso.

	Modelo de Dados	**Licença**	**Esquemas**	**Implementação**	**Manipulação e Acesso**
Orient DB	Documentos; Grafos; Chave-Valor	Open Source	Schemaless; Schemafull; Schema-hybrid	Java	Linguagem baseada em SQL; Gremlin; Java API
Arango DB	Documentos; Grafos; Chave-Valor	Open Source	Schemaless	C++	Linguagem AQL; Gremlin
Microsoft Azure Cosmos DB	Documentos; Grafos; Chave-Valor; Colunares	Comercial	Schemaless	Não informado	Linguagem baseada em SQL; Gremlin
Neo4j	Grafos	Open Source	Schemaless; Schema-optional	Java, Scala	Cypher; Gremlin

Tabela 2.1. Comparativo entre os sistemas gerenciadores de grafos

2.4. Trabalhos Correlatos

Mesmo oferecendo soluções alternativas bastante interessantes ao lidar com os desafios do *Big Data*, ainda não são muito comuns abordagens que tratam de projetos de bancos de dados *NoSQL*. E mesmo entre os projetos de pesquisa que tratam de alguma forma de projetos de bancos de dados, ainda são minoria aqueles que tratam das restrições de integridade inerentes aos modelos conceituais. Além disso, entre os trabalhos encontrados existem duas estratégias distintas: na primeira são tratados os processos de migração de dados de um sistema relacional para um sistema *NoSQL*; já na segunda estratégia, são levados em consideração os projetos de bancos de dados que iniciam com uma modelagem conceitual para ser mapeado para um esquema lógico baseado em um modelo de dados *NoSQL*.

Em relação à primeira estratégia de tratamento de bancos de dados *NoSQL*, podem ser citados alguns trabalhos que fazem o mapeamento direto de uma instância de banco de dados relacional para uma instância de banco de dados *NoSQL* em um processo de migração de dados. Exemplo deste tipo de abordagem podem ser visto em Stanescu et al. (2016), que desenvolveram e implementaram uma série de algoritmos para migração de bancos

de dados *MySQL* para bancos de dados *MongoDB*. Abordagem semelhante pode ser apreciada no trabalho de Kuderu & Kumari (2016), em que foi desenvolvido um *software* para lidar com a migração de um esquema de dados relacional para um banco de dados *NoSQL*. Schreiner et al. (2020) propõe a utilização de uma camada intermediária entre a aplicação e o SGBD para tornar menos custosa a transição dos dados do modelo relacional para algum modelo *NoSQL*, considerando a grande popularidade dos bancos de dados relacionais e os recursos de bancos de dados *NoSQL* ao lidar com o *Big Data*. Focada na automatização do processo de migração de bancos de dados relacionais para bancos de dados *NoSQL* de documento, a pesquisa de Liyanaarachchi et al. (2016) resultou em um *software* que converte tabelas e relacionamentos do *MySQL* em documentos *JSON* para o *MongoDB*.

Em relação à segunda estratégia, a qual lida com a modelagem conceitual e posterior mapeamento para um modelo lógico *NoSQL*, pode-se destacar algumas abordagens. A exemplo disto, Pokorny (2016) propõe um projeto de BDG começando com um modelo conceitual e sua relação com um modelo lógico de BDG. Neste trabalho são discutidos os problemas e desafios ao lidar com a modelagem de BDGs. O modelo conceitual de grafo proposto pelo autor é uma variação binária do modelo ER que conta com entidades fortes e fracas, relacionamentos, atributos, chaves de identificação parcial e total, hierarquias do tipo "É um" e restrições de cardinalidade mínima e máxima. São consideradas regras de transição do modelo conceitual para o esquema de BDG, transformando entidades em nós, relacionamentos e hierarquias "É um" em arestas e, finalmente, restrições inerentes ao modelo ER em regras explícitas no modelo de grafos. Entretanto, nesta abordagem somente são considerados relacionamentos binários e atributos simples em um formato *chave-valor*. Além disso, processos de especialização em que as superclasses são divididas em mais de uma subclasse não chegam a ser discutidos.

Daniel et al. (2016) apresentam uma abordagem em que foi desenvolvido um *framework* capaz de executar um processo de mapeamento de um esquema conceitual para uma representação de grafos. O esquema conceitual é representado em UML (Unified Modeling Language) e considera regras de negócio que são verificadas por consultas geradas automaticamente pelo *framework*, que, por sua vez, atua como uma camada intermediária entre o BDG e as aplicações cliente garantindo um acesso seguro e a integridade dos da-

dos armazenados. Ainda que seja um projeto bastante amadurecido, este ainda não lida com restrições de participação e disjunção em especializações que derivam em mais de uma subclasse.

A proposta apresentada por De Virgilio et al. (2014) corresponde a uma metodologia geral para projetos de bancos de dados de grafos. Esta metodologia considera um diagrama conceitual como entrada para geração automática de um esquema lógico de grafos. Os autores destacam que qualquer modelo conceitual pode ser utilizado como ponto de partida, porém, como estudo de caso, foi utilizado uma versão simplificada do modelo ER. A transformação do diagrama ER em um esquema de grafo inicialmente converte as entidades em nós e os relacionamentos em arestas com pesos atribuídos de acordo com a cardinalidade dos participantes. Uma análise nas conexões entre os nós é feita em um segundo momento e, em casos específicos, alguns conjuntos de nós podem formar uma agregação. Esta segunda etapa é justificada como sendo necessária para diminuir o número de acesso aos dados no SGBD, melhorando o desempenho das consultas. Ao final, uma análise de desempenho foi feita com uma instância do BDG comparando a abordagem tratada no trabalho (com elementos agregados) com as abordagens padrão de bancos de dados. Os resultados evidenciaram algumas vantagens de desempenho para a abordagem defendida pelos autores. No entanto, este trabalho não considera elementos que garantam que as restrições de integridade definidas no modelo conceitual serão implementadas no modelo lógico.

Atzeni et al. (2016) propõe uma metodologia de projeto de bancos de dados *NoSQL* baseada em uma ferramenta denominada *NoAM* (*NoSQL Abstract Model*), a qual oferece suporte a modelagem lógica de alto nível para bancos de dados *NoSQL*. Esta ferramenta pode ser entendida como um representação intermediária entre as fases de modelagem conceitual e lógica, oferecendo uma metodologia de projeto lógico independente do modelo de dados implementado pelo sistema gerenciador alvo. Os autores concluem ponderando sobre a importância da adoção de técnicas tradicionais de projetos de bancos de dados, provenientes dos bancos de dados relacionais, neste contexto aplicadas a bancos de dados *NoSQL*.

O trabalho de Sousa & Cura (2018) faz uma revisão na literatura buscando ferramentas e

metodologias que abordam sobre modelagem lógica de bancos de dados *NoSQL*, tentando encontrar contribuições no que tange ao projeto destes tipos de bancos de dados. Os autores concluem que existem poucos trabalhos que tratam de alguma forma de mapeamento do modelo conceitual para um modelo lógico *NoSQL* em qualquer das quatro categorias preexistentes. Paralelamente a isto, propõem uma metodologia de projeto de banco de dados que parte da ideia de mapeamento de uma variação simplificada do modelo conceitual ER para um modelo lógico de grafos.

Por fim, o trabalho proposto por Roy-Hubara et al. (2017) propõe um projeto de banco de dados de grafo iniciando com a modelagem conceitual representada por um diagrama entidade-relacionamento (DER). Este diagrama inicial passa por uma fase de refinamento, onde relacionamentos *n-ários*, especializações e relacionamentos do tipo *todo-parte* são transformados em relacionamentos binários semanticamente equivalentes. Após esta fase inicial, o diagrama refinado é, enfim, mapeado para um modelo lógico representado por um grafo. Esta abordagem admite que restrições de cardinalidade sejam definidas nos grafos, entretanto não apresenta uma solução automatizada para o processo de mapeamento.

A Tabela 2.2 mostra uma série de características dos trabalhos citados comparando com as propostas apresentadas neste trabalho. O trabalho de Daniel et al. (2016) traz uma proposta bastante abrangente, porém não leva em consideração as restrições de participação de superclasses em processos de especialização e não distingue entre disjunção e sobreposição entre as subclasses. Já Pokorny (2016) parte para uma metodologia de projeto de bancos de dados de grafos a partir de uma simplificação do modelo ER, que possui somente relacionamentos binários, não trata atributos multivalorados e considera apenas especializações simples, em que a subclasse indica apenas a existência de um subconjunto relevante na superclasse. Atzeni et al. (2016) traz uma poposta de projeto de banco de dados *NoSQL* a partir de uma representação conceitual utilizando UML a ser convertida para um modelo lógico *NoSQL* abstrato. Este modelo não define qual a categoria de banco *NoSQL* será implementada, mas propõe uniformizar os projetos de bancos de dados *NoSQL*. A abordagem mostrada por De Virgilio et al. (2014) é de um projeto de bancos de dados a partir do modelo ER básico no qual elementos semanticamente próximos são transformados em um elemento agregado a fim de minimizar o número de nós e arestas

nas instâncias do BDG e priorizar um maior desempenho em relação as técnicas mais comuns em BDGs.

	Conceitual	**Lógico**	**Restrições**	**Mapeamento por Software**
Daniel et al. (2016)	UML	Multigrafo de Propriedades	Cardinalidades e Regras de Negócio	Sim
Pokorny (2016)	ER Binário	Multigrafo de Propriedades	Cardinalidades	Não
Atzeni et al. (2016)	UML	NoSQL Abstrato	-	Não
De Virgilio et al. (2014)	ER	Multigrafo de Propriedades	-	Não
Sousa & Cura (2018)	ER Binário	Multigrafo de Propriedades	Propriedades e Existência/Obrigatoriedade de Arestas	Não
Roy-Hubara et al. (2017)	DER	Multigrafo de Propriedades	Cardinalidades	Não
Nossa Abordagem	ER Estendido	Multigrafo de Propriedades com Hipernós e Hiperarestas	Cardinalidades, Totalidade e Disjunção em Especializações	Sim

Tabela 2.2. Comparativo entre os trabalhos correlatos

Conforme mostrado na Tabela 2.2, fica evidente que existem trabalhos relevantes que não tratam na sua totalidade as restrições de integridade em bancos de dados *NoSQL*. Entretanto, é quase um consenso entre os autores a grande importância da modelagem conceitual em projetos de bancos de dados não relacionais, uma vez que estes modelos facilitam na compreensão e no tratamento dos dados armazenados no banco de dados. Neste trabalho, também é considerado muito importante a fase de modelagem conceitual onde a estrutura dos dados, bem como um conjunto mais amplo de restrições de integridades são transportadas para o modelo lógico, trazendo uma contribuição em relação aos trabalhos ora citados.

Capítulo 3

Processo de Mapeamento do Modelo Conceitual EER para O Modelo Lógico de Grafos

O projeto lógico de um banco de dados é um modelo em um nível de abstração em que já são considerados detalhes técnicos da implementação do banco de dados. Em uma abordagem de banco de dados relacional, o projeto lógico envolve a modelagem do banco de dados em termos de relações e relacionamentos com base na definição de chaves primárias e chaves estrangeiras. Já em um banco de dados de grafos, a estrutura do banco de dados pode ser modelada como um grafo onde tanto os nós como as arestas podem agregar conjuntos de propriedades.

Nesta dissertação é proposto um processo de mapeamento a partir dos conceitos de modelagem EER apresentados na seção 2.1 para um modelo lógico de grafos de propriedades, cujos conceitos foram introduzidos na seção 2.2.3. Este processo inicia com o mapeamento de todas as entidades do modelo conceitual para nós no grafo de esquema. Em seguida, são mapeados os relacionamentos do modelo conceitual, que podem dar origem a arestas ou outros nós no grafo. Por fim, as especializações são mapeadas para hiperarestas.

3.1. Mapeamento de Entidades

No primeiro passo do processo, as entidades do modelo EER são mapeadas como nós para o grafo de propriedades. Entretanto, deve-se fazer uma análise das restrições inerentes do modelo conceitual para transformação em restrições explícitas do modelo de grafos quando necessário. Para definir o algoritmo de mapeamento de entidades, deve-se considerar:

- O conjunto $E = \{e_1, e_2, ...\}$ das entidades do modelo EER a serem mapeadas;
- Cada entidade $e_i \in E$, com $i = \{1, 2, ...\}$, possui:
  - um conjunto de atributos simples $S = \{s_1, s_2, ...\}$;
  - um conjunto de atributos compostos $C = \{c_1, c_2, ...\}$;
  - um conjunto de atributos multivalorados $M = \{m_1, m_2, ...\}$

Para cada entidade e_i do modelo EER cria-se um nó α_i no grafo de esquema.

1 - O nome da entidade e_i deve ser mapeado como rótulo para o nó α_i;

2 - Para cada atributo simples $s_j \in S$, $j = \{1, 2, ...\}$, deve ser criada uma propriedade equivalente em α_i.

3 - Para cada atributo composto de $c_j \in C$, $j = \{1, 2, ...\}$, deve-se mapear cada um de seus componentes simples como propriedades de α_i. Em alguns casos, para evitar redundâncias, a propriedade mapeada pode ser nomeada com o nome do atributo multivalorado, seguido de um ponto e finalizando com o nome do componente simples. Por exemplo, o componente simples "Logradouro" do atributo multivalorado "Endereço" pode ser mapeado para a propriedade "Endereço.Logradouro" em α_i.

4 - Cada atributo multivalorado $m_j \in M$, $j = \{1, 2, ...\}$, deve ser mapeado como um novo nó β_j

4.1 - O nó β_j deve ser conectado ao nó α_i com uma aresta rotulada como "*Pertence a*" com direção a α_i;

4.2 - O nó β_j deve ser rotulado com o nome do atributo m_j;

4.3 - Os componentes simples do atributo m_j devem ser mapeados como propriedades de β_j.

A Figura 3.1 exemplifica o processo de mapeamento da entidade *"Funcionário"*. A Figura 3.1 *(a)* mostra a entidade *"Funcionário"* conforme foi modelada no exemplo da Figura 2.1. Em 3.1 *(b)* é mostrado o resultado do mapeamento da entidade *"Funcionário"* para o nó rotulado como *"Funcionário"* no grafo de esquema. Os atributos simples "*Nome*", "*CPF*" e "*Dt_nascimento*" foram mapeados como propriedades do nó *"Funcionário"*, assim como, "*Logradouro*", "*Número*" e "*Bairro*", que são os componentes simples do atributo composto "*Endereço*" no modelo EER. O atributo multivalorado "*Telefones*" foi mapeado como o nó rotulado como *"Telefone"* no grafo de esquema contendo apenas a propriedade "*Telefone*" e associado a *"Funcionário"* com uma aresta rotulada como "*Pertence a*" apontando para *"Funcionário"*. Geralmente, atributos multivalorados são nomeados no plural no modelo conceitual, entretanto no mapeamento para o modelo lógico de grafos pode ser considerado o nome no singular para rotular os nós gerados a partir deste tipo de atributo.

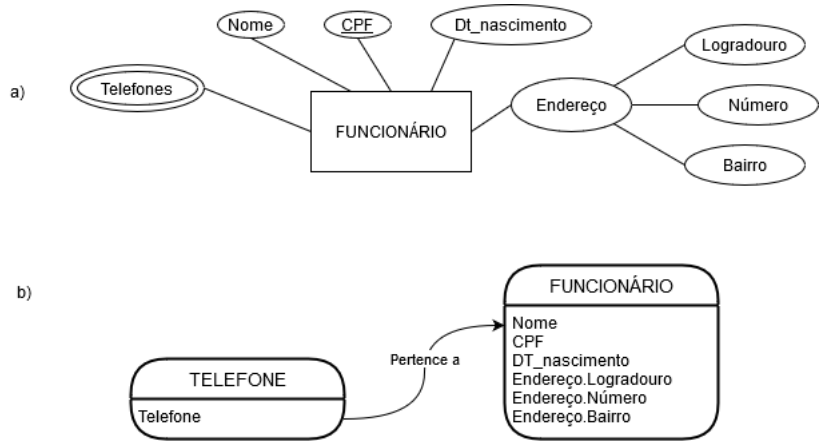

Figura 3.1. Mapeamento da Entidade *"Funcionário"* para Grafo de Esquema

3.1.1. Restrições de integridade no mapeamento de entidades

No caso do mapeamento da entidade *"Funcionário"*, exemplificado na Figura 3.1, o nó *"Telefone"* é mapeado a partir de um atributo multivalorado da entidade *"Funcionário"* no modelo EER. Neste caso fica bastante óbvio um relacionamento *1:N* entre *"Funcionário"* e *"Telefone"* com participação total de *"Telefone"*, já que cada funcionário pode ter *vários* telefones e cada telefone pertence exclusivamente a um *único* funcionário. Em um modelo lógico relacional esta restrição pode ser implementada com a inclusão da chave primária de *"Funcionário"* como chave estrangeira de preenchimento obrigatório

em *"Telefone"*. Entretanto, esta restrição deve ser implementada como uma *restrição explícita* no modelo de grafo de propriedades.

Para implementação deste tipo de restrição de integridade no grafo de propriedades deve-se considerar:

- Os nós α e β que representam respectivamente a entidade mapeada e seu atributo multivalorado no grafo de esquema;
- A aresta ϵ, que parte de β em direção a α, rotulada como "*Pertence a*".

A partir daí a restrição de integridade entre *"Funcionário"* (nó α) e *"Telefone"* (nó β) deve ser explicitada genericamente como: toda ocorrência do nó β deve ser conectada a uma única ocorrência do nó α através de uma ocorrência da aresta ϵ. Esta restrição indica que cada ocorrência de β deve ter, partindo de si, uma, e somente uma, ocorrência de ϵ ligando-a a uma ocorrência de α. Isto permite que as ocorrências de α se conectem simultaneamente a várias ocorrências de β, mas restringe que cada ocorrência de β esteja conectada exclusivamente a uma única ocorrência de α.

3.2. Mapeamento de Relacionamentos

Nos casos mais simples, o mapeamento de um relacionamento cria uma aresta dirigida ligando os nós equivalentes às entidades envolvidas no relacionamento. As restrições inerentes relativas às cardinalidades e participações no relacionamento no modelo EER devem ser implementadas como restrições explícitas no grafo de esquema para refletir a semântica do modelo conceitual e não permitir inconsistências no BDG. Além disso, o mapeamento de relacionamentos binários, por ser mais natural ao grafo, tem um algoritmo diferente dos relacionamentos n-ários. Outro fator relevante é com relação ao mapeamento de relacionamentos que possuem atributos multivalorados, que exigem um processo mais complexo.

3.2.1. Relacionamentos binários

Para o mapeamento de relacionamentos binários no modelo EER, deve-se considerar:

- O conjunto $R = \{r_1, r_2, ...\}$ dos relacionamentos binários do modelo EER;
- Os nós α_o e α_d, que representam respectivamente os mapeamentos das entidades de origem e de destino participantes do relacionamento r_i no modelo EER. Neste

caso, a definição das entidades de origem e de destino de um relacionamento dependem do sentido semântico definido para o relacionamento.

Para cada relacionamento $r_i \in R$, $i = \{1, 2, ...\}$, que não possua atributos multivalorados, cria-se a aresta ϵ_i no grafo de esquema, ligando os nós α_o e α_d.

1 - O nome do relacionamento r_i deve ser mapeado como rótulo para a nova aresta ϵ_i;

2 - A direção da aresta ϵ_i deve apontar em direção ao nó de destino α_d;

3 - Os atributos simples e componentes simples dos atributos compostos de r_i devem ser mapeados como propriedades da aresta ϵ_i.

Entretanto é possível que um relacionamento seja modelado com um ou mais atributos multivalorados. Neste caso, para cada relacionamento $r_i \in R$, $i = \{1, 2, ...\}$, que contenha um conjunto de atributos multivalorados $M = \{m_1, m_2, ...\}$, cria-se o nó μ_i.

1 - O nome do relacionamento r_i deve ser mapeado como rótulo para o novo nó μ_i;

2 - Cria-se uma aresta rotulada como "*Participa de*" partindo do nó α_o e chegando ao nó μ_i.

3 - Cria-se uma aresta rotulada como "*Participa de*" partindo do nó α_d e chegando ao nó μ_i.

4 - Os atributos simples de r_i devem ser mapeados como propriedades do nó μ_i, assim como os componentes simples de seus atributos compostos.

5 - Cada atributo multivalorado $m_j \in M$, $j = \{1, 2, ...\}$, de r_i deve ser mapeado como um novo nó β_j

5.1 - O nó β_j deve ser conectado a μ_i com uma aresta rotulada como "*Pertence a*" com seta em direção a μ_i;

5.2 - O nó β_j deve ser rotulado com o nome do atributo m_j, considerando a flexão de número para o singular, quando for o caso;

5.3 - Os componentes simples de m_j devem ser mapeados como propriedades do nó β_j.

A Figura 3.2 mostra a modelagem conceitual das entidades *"Funcionário"* e *"Departamento"*, cujos relacionamentos "*Trabalha_em*" e "*Gerencia*" serão mapeados para o grafo de esquema. Os relacionamentos "*Trabalha_em*" e "*Gerencia*" são mapeados para o grafo

de esquema de maneiras diferentes, uma vez que o relacionamento "*Gerencia*" não apresenta atributos multivalorados, enquanto que o relacionamento "*Trabalha_em*" apresenta o atributo "*Horários*", que representa a escala de horários que um funcionário trabalha em um dado departamento.

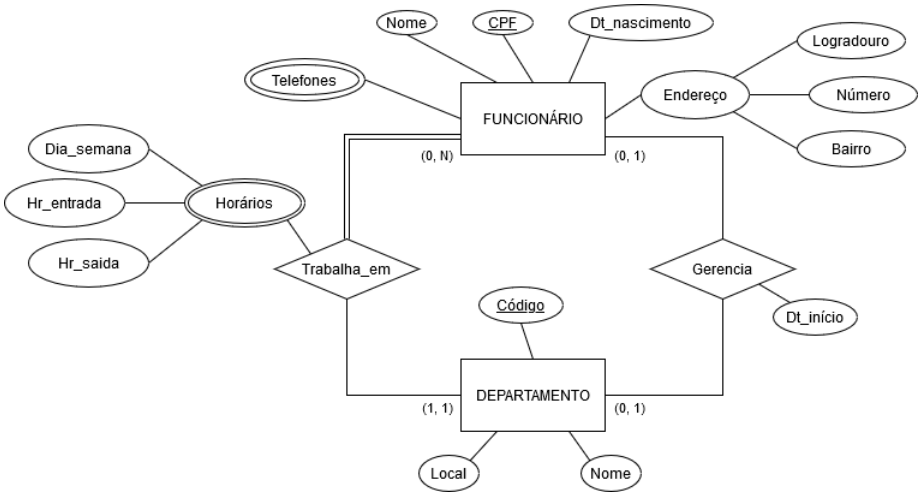

Figura 3.2. Relacionamentos "*Trabalha_em*" e "*Gerencia*"

A Figura 3.3 mostra o resultado do mapeamento do modelo da Figura 3.2. No exemplo da Figura 3.3, o relacionamento "*Gerencia*" foi mapeado diretamente como uma aresta rotulada como *"Gerencia"* no grafo de esquema. O sentido da aresta vai do nó *"Funcionário"* (α_o) para o nó *"Departamento"* (α_d), pois, pela semântica do modelo da Figura 3.2, um funcionário *gerencia* um departamento e não o contrário. O atributo simples "*Dt_inicio*" foi mapeado como uma propriedade da aresta "*Gerencia*". Já o relacionamento "*Trabalha_em*" do modelo EER foi mapeado como um nó rotulado como *"Trabalha_em"*, uma vez que apresenta o atributo multivalorado "*Horários*". Este nó é conectado tanto a *"Funcionário"* como a *"Departamento"*, com arestas rotuladas como "*Participa de*". O atributo multivalorado "*Horários*" é transformado em um nó, rotulado como *"Horário"*, e seus componentes simples "*Dia_semana*", "*Hr_entrada*" e "*Hr_saida*" são transformados em propriedades. Por fim, o nó *"Horário"* é conectado ao nó *"Trabalha_em"* com uma aresta rotulada como "*Pertence a*" apontando para este último.

3.2.2. Restrições de integridade em relacionamentos binários

Basicamente, as restrições em relacionamentos binários têm a ver com as cardinalidades e a participação (total ou parcial) das entidades envolvidas. Em um modelo relacional,

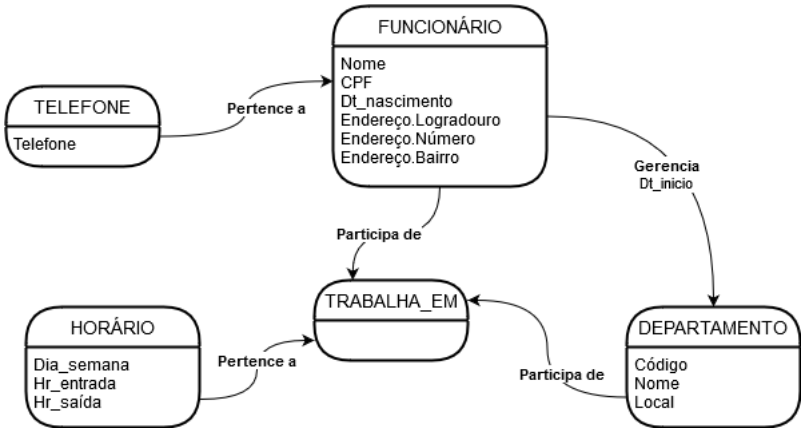

Figura 3.3. Mapeamento dos Relacionamentos "*Trabalha_em*" e "*Gerencia*"

estas restrições são inerentes, definidas com a inserção de chaves primárias e chaves estrangeiras, que podem ser definidas como obrigatórias ou não, nas relações e, portanto, não precisam ser explicitadas. Mas em um modelo de grafos, que permite que um determinado nó tenha um número ilimitado de conexões, estas restrições devem ser explícitas e definidas usando a DDL do gerenciador de BDG para manter a integridade dos dados e se manter consistente com o modelo conceitual.

Sem uma definição de restrições de integridade, o exemplo mostrado na Figura 3.3 pode permitir situações inconsistentes como, por exemplo, uma ocorrência de *"Departamento"* conectada a várias ocorrências distintas de *"Funcionário"* com arestas rotuladas como "*Gerencia*", representando um departamento com mais de um gerente, o que, de acordo com o modelo conceitual da Figura 3.2, não é permitido.

A implementação das regras de integridade para definição de cardinalidades e participações das entidades envolvidas em um relacionamento binário depende de como o relacionamento foi mapeado para o grafo de esquema e definem o número mínimo e máximo de arestas que podem incidir sobre os nós envolvidos no relacionamento.

Considerando que o relacionamento r, entre as entidades de origem e_o e de destino e_d tenha sido mapeado para o grafo de esquema como uma aresta simples ϵ conectando os nós α_o e α_d, equivalentes a e_o e e_d respectivamente:

- Caso a participação de e_o no modelo EER seja *total*, deve-se definir que em toda ocorrência de α_o deve incidir *pelo menos uma* ocorrência da aresta ϵ. A mesma regra se aplica às ocorrencias do nó α_d;

•Caso a cardinalidade máxima de e_o no modelo EER seja *um*, deve-se definir que em toda ocorrência de α_o deve incidir *no máximo uma* ocorrência da aresta ϵ. A mesma regra se aplica às ocorrencias de α_d.

Por outro lado, deve-se considerar os casos como o do relacionamento r entre as entidades e_o e e_d que tenha atributos multivalorados e que, portanto, tenha sido mapeado como um nó μ conectado aos nós α_o e α_d com arestas rotudadas como "*Participa de*":

•Caso a participação de e_o no modelo EER seja *total*, define-se que o toda ocorrência do nó α_o deve se conectar a pelo menos uma ocorrência do nó μ com uma aresta rotulada como "*Participa de*". O mesmo se aplica às ocorrências de α_d.

•Caso a cardinalidade máxima de e_o tenha sido definida como *um* no modelo EER, define-se que para qualquer ocorrência de α_o é permitida a conexão com *no máximo uma* ocorrência de μ com uma aresta rotulada como "*Participa de*". O mesmo se aplica às ocorrências de α_d.

•Sobre cada ocorrência de μ devem incidir *exatamente duas* arestas rotuladas como "*Participa de*" sendo uma partindo de uma ocorrência de α_o e outra partindo de uma ocorrência de α_d.

•Os nós relativos aos atributos multivalorados do relacionamento r seguem as mesmas regras de restrição dos nós relativos a atributos multivalorados de entidades comuns.

3.2.3. Relacionamentos n-ários

Relacionamentos n-ários são representações de associações de ocorrências pertencentes a três ou mais entidades em um modelo ER e suas variações. Embora relacionamentos binários sejam o tipo mais comum e muitos sistemas de modelagens oferecerem suporte apenas a relacionamentos de grau dois (Teorey et al.; 2014), algumas situações somente podem ser representadas fielmente pela a modelagem de relacionamentos de grau maior.

O mapeamento de um relacionamento n-ário em uma abordagem de banco de dados relacional produz uma relação tendo como chaves estrangeiras as chaves primárias das relações participantes do relacionamento. Considerando uma abordagem de BDG, cada relacionamento n-ário é mapeado como um nó conectado aos nós representantes das entidades participantes do relacionamento. Cada ocorrência de um relacionamento n-ário

é uma instância que associa ocorrências de cada uma de suas *n* entidades participantes e seus atributos, caso houverem, representam informações que só fazem sentido dentro desta associação. Para o mapeamento dos relacionamentos n-ários de um modelo EER é necessário considerar:

- O conjunto *R* = $\{r_1, r_2, ...\}$ dos relacionamentos n-ários do modelo EER;
- O conjunto *N* = $\{\alpha_1, \alpha_2, ..., \alpha_n\}$ dos *n* nós representantes das entidades participantes do relacionamento n-ário $r_i \in R$, i = $\{1, 2, ..., n\}$.

Para cada relacionamento n-ário $r_i \in R$, i = $\{1, 2, ...\}$, do modelo EER deve ser criado um nó μ_i no grafo de esquema.

1 - Para cada nó $\alpha_j \in N$, j = $\{1, 2, ..., n\}$, cria-se uma aresta rotulada como "*Participa de*" partindo de α_j em direção a μ_i;

2 - Tanto os atributos simples, como os componentes simples dos atributos compostos do relacionamento r_i devem ser mapeados como propriedades para μ_i;

3 - Caso r_i tenha atributos multivalorados, estes devem ser mapeados como nós para o grafo de esquema. Seja *M* = $\{m_1, m_2, ...\}$ o conjunto dos atributos multivalorados do relacionamento r_i:

3.1 - Para cada atributo multivalorado $m_j \in M$, cria-se o nó β_j;

3.2 - Os componentes simples do atributo m_j devem ser mapeados como propriedades para o nó β_j;

3.3 - O nó β_j deve ser conectado ao nó μ_i com uma aresta rotulada como "*Pertence a*" apontando para μ_i.

A Figura 3.4 exemplifica um relacionamento ternário em modelo EER a ser mapeado para o grafo de esquema. O relacionamento "*Atua_em*" representa a associação entre as entidades *"Funcionário"*, *"Projeto"* e *"Local"* e indica os funcionários que dedicam uma certa carga horária trabalhando em um local executando algum projeto. Neste modelo existem uma restrição quanto à cardinalidade *1* de projeto em relação a cada par distinto de funcionário e local. Neste caso específico, pode-se dizer que para um determinado funcionário atuando em um dado local define o projeto que está em execução. É importante ressaltar que cada ocorrência do relacionamento "*Atua em*" representa um trio de ocorrências (*f*, *p*, *l*), sendo *f* um funcionário, *p* um projeto e *l* um local.

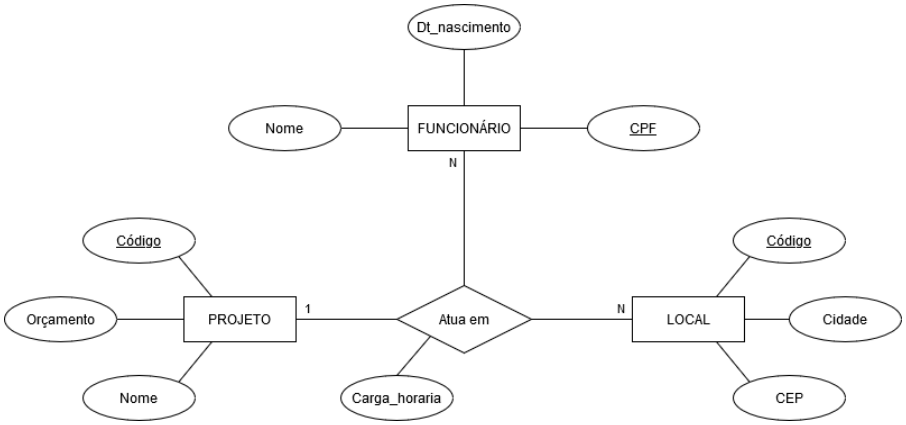

Figura 3.4. Relacionamento "*Atua em*" entre as entidades *"Funcionário"*, *"Projeto"* e *"Local"*

O resultado do mapeamento deste modelo para o grafo de esquema pode ser visto na Figura 3.5, que mostra o esquema lógico de grafo para o modelo da Figura 3.4. No processo de mapeamento, as entidades *"Funcionário"*, *"Projeto"* e *"Local"* são convertidas para nós do grafo de esquema juntamente com seus respectivos atributos, que são mapeados para propriedades dos respectivos nós. O relacionamento "*Atua em*" é transformado em um nó rotulado como *"Atua em"* e conectado aos outros nós com arestas rotuladas como "*Participa de*". Por fim, o atributo "*Carga_horaria*" é mapeado como uma propriedade do nó *"Atua em"* no grafo de esquema.

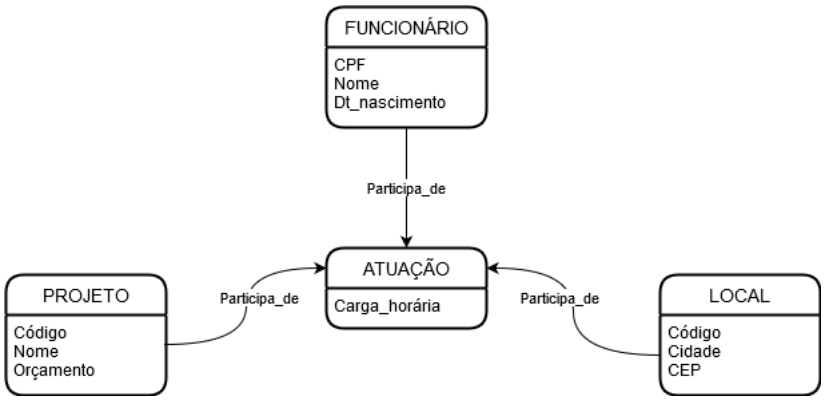

Figura 3.5. Mapeamento do Relacionamento "*Atua_em*" para o grafo de esquema

3.2.4. Restrições de integridade em relacionamentos n-ários

A despeito das cardinalidades modeladas para um relacionamento n-ário em um modelo ER, cada ocorrência deste tipo de relacionamento referencia exatamente uma única

ocorrência de cada uma das *n* entidades participantes. Por outro lado, cada ocorrência de qualquer entidade participante de um relacionamento n-ário pode ser referenciada por alguma das ocorrências do relacionamento. Isso evidencia que cada ocorrência do relacionamento mantém uma *participação total* em relação cada uma das *n* entidades participantes.

Em alguns casos, é interessante considerar que relacionamentos n-ários como o da Figura 3.4 possam ser modelados como uma *entidade fraca* ligada às outras entidades participantes com relacionamentos binários. Neste caso específico, o relacionamento ternário *"Atua em"* pode ser substituído pela entidade fraca *"Atuação"* e ter como proprietárias as entidades *"Funcionário"* e *"Local"*, já que estas entidades tem cardinalidade *N* no relacionamento ternário original (Elmasri et al.; 2011; Silberschatz et al.; 2012). Isto quer dizer que cada ocorrência da entidade fraca *"Atua em"* pode ser identificada por um par (*f*, *l*) de ocorrências de *"Funcionário"* e de *"Local"*. A entidade *"Projeto"* também deve se associar a nova entidade *"Atua em"*, entretanto não como entidade forte, já que no relacionamento ternário ela mantém uma cardinalidade *1*. Entretanto, as cardinalidades *1-N* mantidas entre as entidades *"Funcionário"*, *"Projeto"* e *"Local"* não representam com naturalidade a cardinalidade *1-N-N* modelada pelo relacionamento ternário da Figura 3.4 (Silberschatz et al.; 2012).

Esse processo de *binarização* do relacionamento *"Atua em"* pode ser visto na Figura 3.6. Esta figura mostra um conjunto de relacionamentos binários que podem ser utilizados em substituição ao modelo ternário apresentado pela Figura 3.4. O processo de mapeamento do modelo da Figura 3.6 produz, neste caso, o mesmo grafo de esquema mostrado na Figura 3.5. Neste caso, basta definir as regras de integridade para o grafo de esquema para garantir a integridade dos dados nas instâncias do BDG baseadas neste esquema. No modelo mostrado na Figura 3.5, as restrições de integridade devem garantir que cada ocorrência do nó rotulado como "*Atua em*" deve se conectar com uma, e somente uma, ocorrência de cada nó relativo as entidades *"Funcionário"*, *"Projeto"* e *"Local"*. Já em relação as entidades participantes, nenhuma restrição será necessária.

De maneira genérica, seja o relacionamento n-ário r o qual tenha sido mapeado para o grafo de esquema como o nó μ , e o conjunto P = $\{e_1, e_2, ..., e_n\}$ das *n* entidades

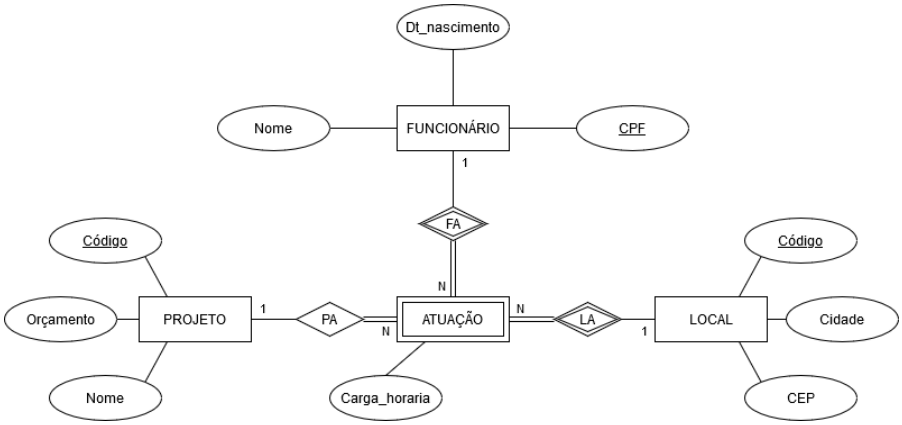

Figura 3.6. Relacionamento "*Atua em*" remodelado como entidade fraca

participantes do relacionamento r, o qual tenha sido mapeado para o conjunto N = $\{\alpha_1$, α_2, ..., $\alpha_n\}$ de nós do grafo de esquema.

- Toda ocorrência do nó μ em uma instância do BDG deve se conectar a *pelo menos uma* ocorrência de cada um dos nós $\alpha_i \in N$, i = $\{1, 2, ..., n\}$ com arestas rotuladas como "*Participa de*" com direção a μ;
- Toda ocorrência do nó μ deve se conectar a *no máximo uma* ocorrência de cada um dos nós $\alpha_i \in N$, i = $\{1, 2, ..., n\}$.

Estas duas restrições garantem que cada ocorrência de μ associe uma única ocorrência de cada um dos nós do conjunto N, os quais representam as entidades que participam do relacionamento n-ário r no modelo EER.

3.3. Mapeamento de Especializações

Subclasses são representações de subagrupamentos que são considerados relevantes na modelagem de uma entidade no modelo EER. Elas indicam que o conjunto representado por uma determinada entidade pode conter elementos específicos que se destacam dos elementos comuns agregando um conjunto próprio de atributos e/ou relacionamentos. A modelagem conceitual usando modelo EER prevê a definição de entidades próprias para representar esses subagrupamentos e associá-los com a sua *entidade pai*, formando uma hierarquia com origem na entidade modelada como superclasse e estabelecendo um caminho para as entidades modeladas como subclasses. As subclasses, por sua vez, podem manter uma relação semântica entre si, definindo uma disjunção/sobreposição de seus ele-

mentos e estabelecendo a participação da superclasse como total ou parcial nestes tipos de associação.

Situações onde apenas um subagrupamento é identificado dentro do conjunto representado pela entidade são mais simples de serem mapeados, mas quando existe um processo de especialização, onde são identificadas várias subclasses formando subconjuntos disjuntos/sobrepostos de elementos, o mapeamento para um projeto lógico de grafos pode exigir atenção especial.

3.3.1. Superclasses e Subclasses

Um subagrupamento simples em uma determinada entidade pode ser modelado como uma associação direta de superclasse-subclasse no modelo conceitual EER, onde se define que uma ocorrência de uma subclasse *é uma* ocorrência da superclasse. Na prática, tanto a ocorrência da subclasse quanto a ocorrência da superclasse referenciam o mesmo objeto do mundo real (Elmasri et al.; 2011). O mapeamento dessas situações para o modelo lógico de grafos é bastantes simples, bastando estabelecer a relação superclasse-subclasse entre os nós representantes das entidades no modelo EER. Para este processo, deve-se considerar:

- O nó α, que representa o mapeamento da superclasse do modelo EER para o grafo de esquema;
- O nó β, que representa o mapeamento da subclasse do modelo EER para o grafo de esquema.

Para estabelecer a relação entre os nós α e β basta criar uma aresta rotulada como "*É um*" partindo de β e chegando em α. Isto indica que toda ocorrência de β é, por consequência, uma ocorrência de α, já que β representa um subagrupamento de elementos pertencentes a α. A Figura 3.7 exemplifica este mapeamento a partir da relação *"Gerente" é um "Funcionário"*. A Figura 3.7 *(a)* mostra a modelagem da superclasse *"Funcionário"* e da subclasse *"Gerente"*. Neste contexto, um gerente é também um funcionário já que a subclasse *"Gerente"* representa um subconjunto de funcionários que gerenciam algum departamento. O grafo de esquema mostrado em *(b)* é o resultado do mapeamento proposto para o modelo conceitual apresentado em *(a)*. Levando em consideração os nós *"Funcionário"* e *"Gerente"*, o processo de mapeamento simplesmente acrescentou a aresta

dirigida rotulada como "*É um*" em direção a *"Funcionário"*. Neste exemplo, como a entidade *"Gerente"* não possui atributos próprios, estes não foram mapeados para o grafo de esquema em *(b)*.

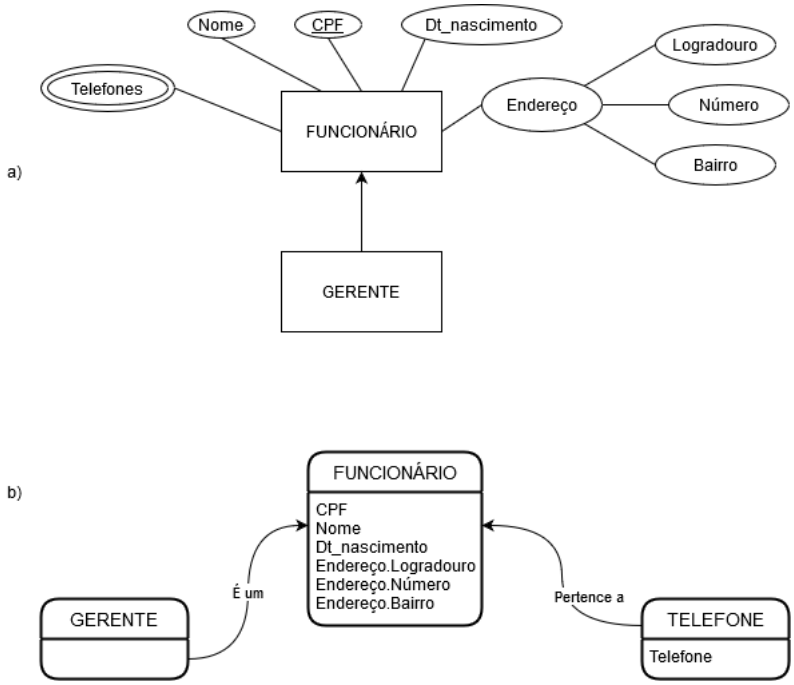

Figura 3.7. Mapeamento de Subclasses e de Especializações/Generalizações

3.3.2. Especializações e Generalizações

O mapeamento de subclasses que são modeladas a partir de um processo de especialização é um processo mais complexo, já que deve considerar as relações semânticas inerentes a este tipo de modelagem, como a disjunção/sobreposição entre as ocorrências das subclasses e a participação das superclasses em cada processo de especialização. Pokorny (2016) trata apenas de hierarquias do tipo "*is a*", as quais não lidam com a disjunção/sobreposição e não levam em consideração a participação da superclasse nas subclasses, tratando apenas de situações similares a exemplificada pela Figura 3.7 de subagrupamentos simples dentro de uma superclasse. Neste tópico serão tratadas as situações em que uma entidade é subdividida em dois ou mais subagrupamentos, considerando a participação da superclasse nas subclasses e as regras de disjunção e sobreposição das subclasses.

Em um modelo EER, uma entidade pode passar por mais de um processo de especialização. É importante salientar que nesta abordagem considera-se que cada processo de

especialização divide uma entidade, neste contexto denominada superclasse, em dois ou mais subagrupamentos de elementos, denominados subclasses de especialização. No processo de mapeamento para o grafo de esquema, a superclasse e as subclasses são mapeados como nós. Já cada especialização em si é mapeada como uma *hiperaresta* no grafo de esquema a fim de distinguir cada processo de especialização para uma mesma superclasse. Para cada processo de especialização do modelo EER deve-se considerar:

- O nó α, o qual representa o mapeamento da superclasse para o grafo de esquema;
- O conjunto $B = \{\beta_1, \beta_2, ...\}$, que representam os nós do grafo de esquema equivalentes às subclasses do processo de especialização.

As situações que requerem cuidado nesta abordagem são:

1 - Especializações com participação total da superclasse;

2 - Especializações que indicam disjunção entre as subclasses.

Para cada processo de especialização, cria-se a hiperaresta θ no grafo de esquema, colocando a superclasse α na cauda e cada uma das subclasses $\beta_i \in B = \{\beta_1, \beta_2, ...\}$ na cabeça da hiperaresta θ. Desta forma, uma hiperaresta de especializações "corre" da superclasse em direção às subclasses modeladas.

1 - Caso a participação da superclasse seja *total* e as subclasses sejam definidas como disjuntas neste processo de especialização, a hiperaresta θ deve ser rotulada como "*Disjunção Total*";

2 - Caso haja apenas a participação total da superclasse, a hiperaresta θ deve ser rotulada como "*Sobreposição Total*";

3 - No caso de modelo EER definir apenas a disjunção entre as subclasses, a hiperaresta θ deve ser rotulada como "*Disjunção Parcial*";

4 - Por fim, no caso de não haver participação total da superclasse, e o processo de especialização permitir a sobreposição entre as subclasses, a hiperaresta θ deve ser rotulada como "*Sobpreposição Parcial*".

No exemplo da Figura 3.8, o mapeamento da especialização *"Funcionário"* para *"Horista"/"Mensal"* gera uma hiperaresta, rotulada como *"Disjunção Total"*, que representa o próprio processo de especialização. Esta hiperaresta não gera ocorrências na instância do banco de dados, porém indica as regras semânticas para esta especialização:

a disjunção entre as subclasses (a letra *"D"* no círculo da especialização) e a participação total da superclasse na especialização (linha dupla). Além disso, os nós ligados à cabeça da hiperaresta define quais são as subclasses envolvidas neste processo de especialização, neste caso, *"Horista"* e *"Mensal"*, enquanto que a cauda define a superclasse. No caso de a especialização definir uma disjunção parcial ou uma sobreposição total nas subclasses, o processo é bastante semelhante, mudando apenas o rótulo da hiperaresta representante da especialização, conforme exemplo da Figura 3.9.

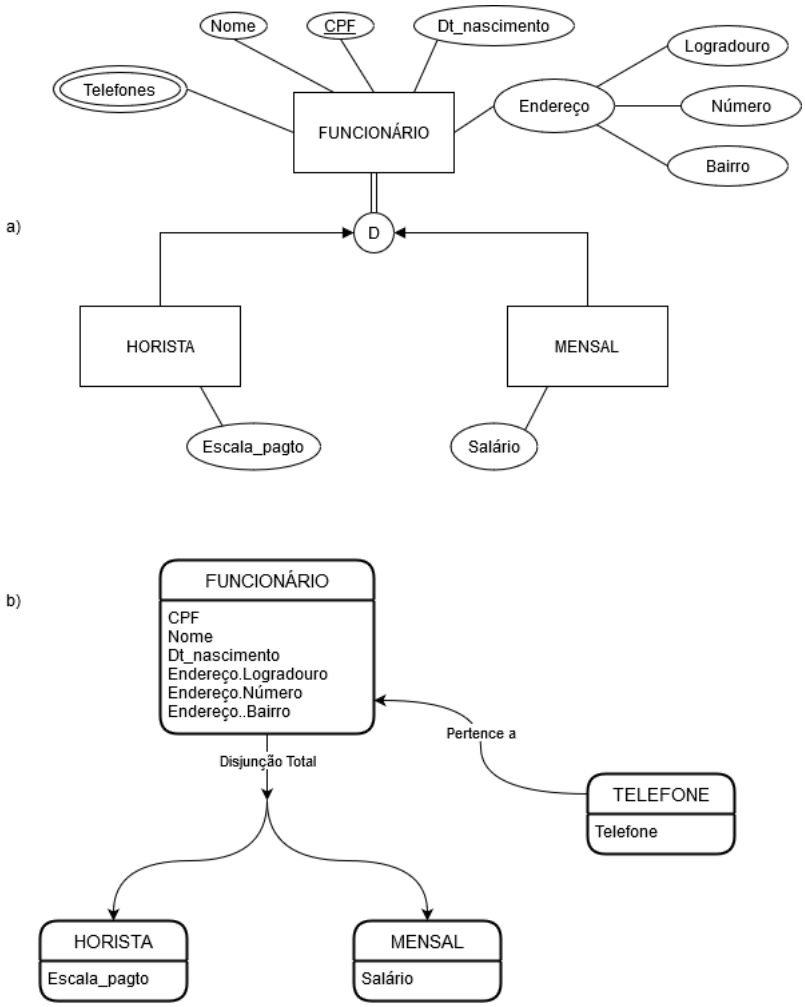

Figura 3.8. Mapeamento da superclasse *Funcionário* em *"Horista"* e *"Mensal"*

No exemplo da Figura 3.9, é definida uma especialização de *"Funcionário"* em *"Técnico"/"Engenheiro"* como uma disjunção parcial, ou seja, é permitido que algumas ocorrências de *"Funcionário"* não se especializem em *"Técnico"* nem em *"Engenheiro"* . Isto indica que dentro do conjunto representado por *"Funcionário"* existem dois subconjuntos próprios e disjuntos entre si de funcionários que são técnicos, representando

o primeiro subconjunto e outro de funcionários que são engenheiros. A diferença para o modelo exemplificado na Figura 3.8 é que esta especialização permite que hajam funcionários que não sejam nem técnicos e nem engenheiros.

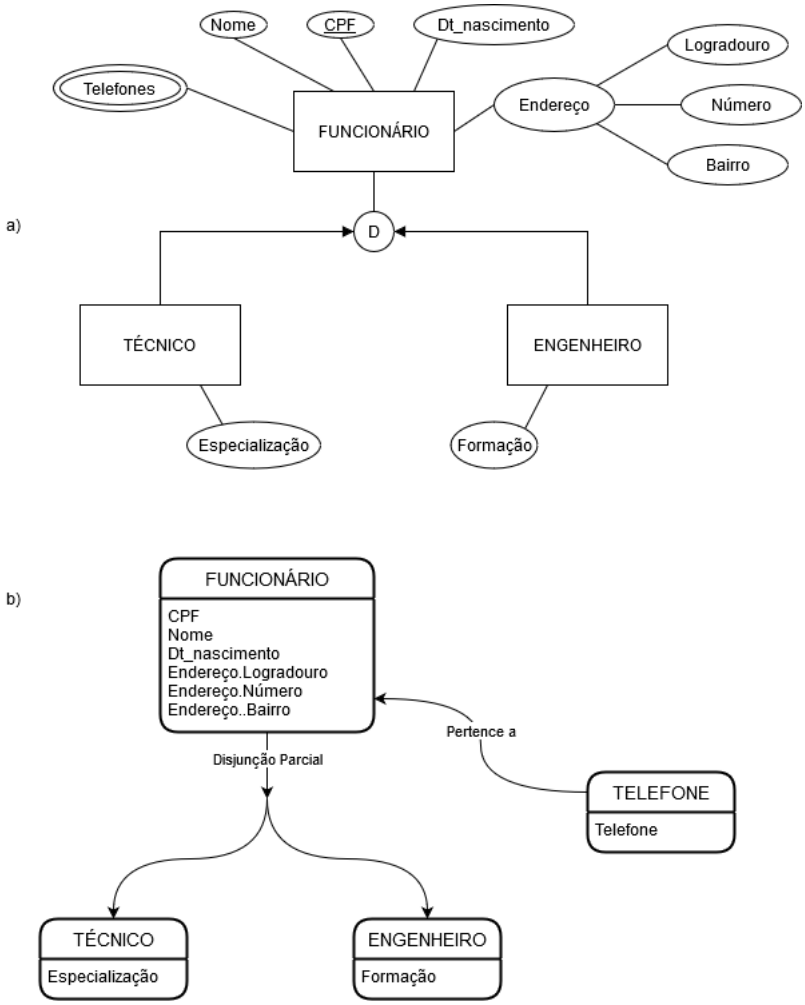

Figura 3.9. Mapeamento da superclasse *"Funcionário"* em *"Técnico"* e *"Engenheiro"*

Como exemplificado pelas Figuras 3.7, 3.8 e 3.9, a entidade *"Funcionário"* pode ser subdividida em vários conjuntos de subclasses. O mapeamento da entidade *"Funcionário"* e seus conjuntos de subclasses formam uma *árvore de especializações* com raiz em *"Funcionário"* e cujas folhas representam as subclasses modeladas. As hiperarestas internas representam as especializações e suas regras semânticas. A Figura 3.10 mostra o resultado do mapeamento do modelo EER apresentado na Figura 2.9 representando uma árvore de especializações com raiz em *"Funcionário"*. A eventual ocorrência de qualquer um dos nós folha em uma instância do banco de dados faz referência a um funcionário, porém

com características específicas relativas a subclasse representada por esta folha.

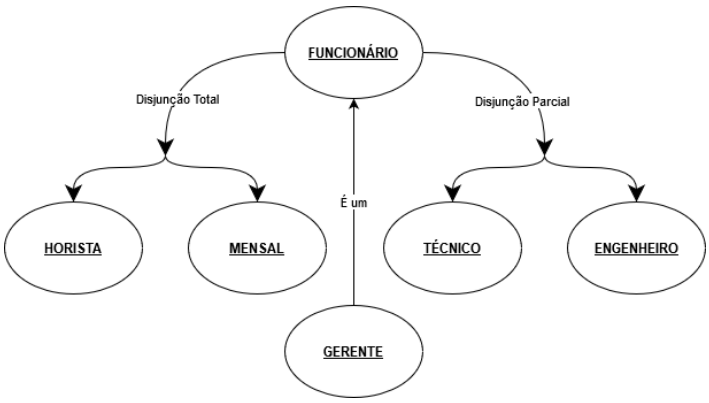

Figura 3.10. Árvore de especializações de *"Funcionário"*

Apesar do grafo de esquema modelar nós que representam subclasses no modelo EER, somente ocorrências da superclasse serão instanciadas no BDG, uma vez que o modelo de grafos permite que nós com o mesmo rótulo, portanto ocorrências da mesma entidade, possam ter conjuntos de propriedades e arestas distintos entre si. Outra característica interessante dos grafos de propriedades é que eles permitem que um nó possa assumir mais de um rótulo. Na prática, isso permite que um determinado nó representante de alguma ocorrência de um *funcionário horista* na instância do BDG possa agregar as propriedades básicas de *"Funcionário"* como "*CPF*", "*Nome*", etc, com a propriedade "*Escala_pagto*", inerente a *"Horista"* em um único nó rotulado como *"Funcionário"* e como *"Horista"*. Se este mesmo nó representar um funcionário técnico, ele ainda deve agregar a propriedade "*Especialização*" e o rótulo *"Técnico"* na instância do BDG.

3.3.3. Representação de Árvores de Especializações como Grafos Aninhados

Ao instanciar um BDG no qual tenha sido modelada uma árvore de especializações em seu grafo de esquema, conforme os exemplos das Figuras 3.7 a 3.10, que apresentam a superclasse *"Funcionário"* e seu conjunto de subclasses, é possível pensar nesta árvore como um *grafo aninhado* ou *hipernó* dentro do grafo de esquema. Na prática, a ocorrência deste hipernó nas instâncias do BDG gera apenas um nó simples do tipo *"Funcionário"*, porém agregando características e rótulos específicos de acordo com a modelagem especificada na árvore de especialilzações.

De forma geral, qualquer árvore de especializações definida em um grafo de esquema

pode ser representada por um hipernó, o qual gera ocorrências simples do mesmo tipo da superclasse raiz da árvore nas instâncias do BDG, porém agregando propriedades, relacionamentos e rótulos definidos para qualquer nó pertencente à árvore, desde que respeite as regras de disjunção e participação definidas no grafo de esquema.

Um bom exemplo é a ocorrência de *"Funcionário"*, que agrega a propriedade "*Salário*" e o rótulo "*Mensal*" e que automaticamente não poderá agregar a propriedade "*Escala_pagto*" e o rótulo "*Horista*" e vice-versa, já que as subclasses *"Mensal"* e *"Horista"* foram definidas como disjuntas entre si no modelo conceitual e no grafo de esquema. Além disso, deverá agregar características de uma das duas subclasses, já que a participação de *"Funcionário"* foi definida como *total* nesta especialização. Em outro exemplo, a ocorrência de *"Funcionário"* que gerencia algum departamento deve agregar o rótulo *"Gerente"* e a aresta rotulada como "*Gerencia*" definida no grafo de esquema como uma conexão entre os nós *"Gerente"* e *"Departamento"* e estabelecem a relação dos funcionários que gerenciam algum departamento.

A Figura 3.11 mostra a árvore de especializações de *"Funcionário"* e alguns de seus relacionamentos. Este exemplo mostra a árvore de especializações de *"Funcionário"* delimitada pelo círculo pontilhado e seus relacionamentos com as entidades *"Dependente"*, *"Departamento"* e *"Telefone"*, sendo esta última justificada pelo mapeamento do atributo multivalorado "*Telefones*" de *"Funcionário"*. Os atributos foram suprimidos a fim de simplificar as diagramação deste modelo, mas assume-se que sejam os mesmos que tem sido utilizados ao longo deste trabalho.

Uma instância de BDG baseado neste esquema somente poderá gerar nós representantes de uma das quatro entidades a seguir: *"Funcionário"*, *"Dependente"*, *"Departamento"* e *"Telefone"*. Um nó referente a um funcionário que for gerente de algum departamento apenas agregará a aresta "*Gerencia*" e o rótulo *"Gerente"*, o conectando com o nó relativo ao departamento gerenciado. O nó relativo a qualquer funcionário horista agrega o atributo "*Escala_pagto*" juntamente com o rótulo *"Horista"*, e não permite o atributo "*Salario*" e nem o rótulo *"'Mensal"*. Analogamente, as ocorrências de funcionários técnicos agregam o atributo "*Especialização*" juntamente com o rótulo *"Técnico"* e não admitem o atributo "*Formação*" e o rótulo *"Engenheiro"*.

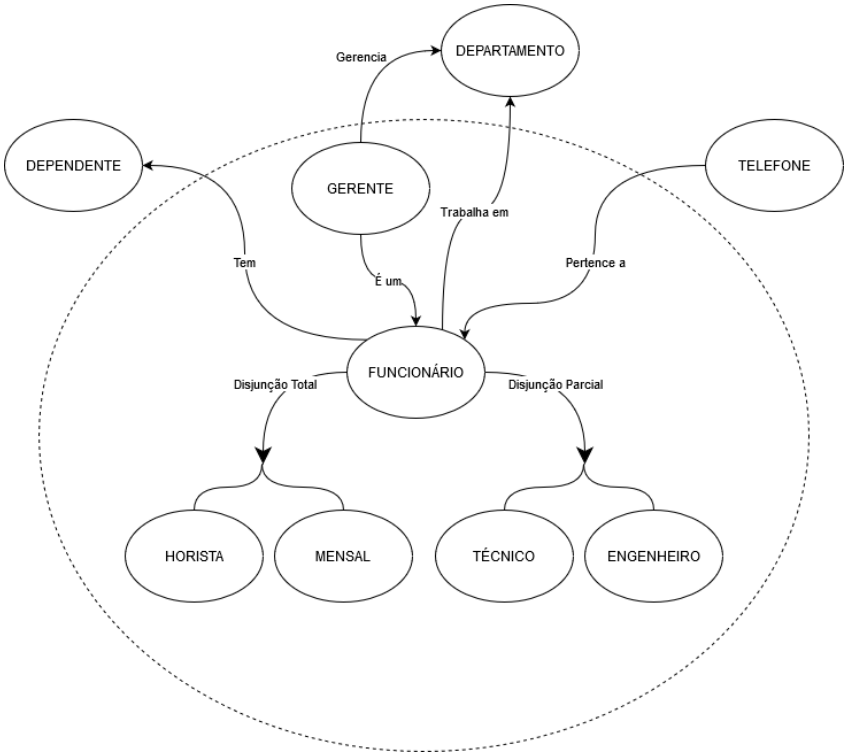

Figura 3.11. Relacionamentos da árvore de especializações de *"Funcionário"*

Uma versão mais simplificada do modelo acima é mostrada na Figura 3.12, onde a árvore de especializações de *"Funcionário"* foi comprimida em um único nó rotulado como *"Funcionário"* com a finalidade de simplificar a diagramação do grafo de esquema. A linha dupla, neste caso, denota que *"Funcionário"* é um *hipernó* que pode ser expandido para sua forma de árvore sempre que houver a necessidade de visualizar detalhes sobre suas especializações. Pode-se chamar esta versão de esquema de grafo *simplificado* do BDG, enquanto que a versão apresentada na Figura 3.11 pode ser denominada como esquema de grafo *expandido* para o mesmo BDG.

Na Figura 3.13 está exemplificada uma possível instância de BDG baseada nos esquemas mostrados nas figuras 3.11 e 3.12. Vale ressaltar que neste exemplo, existem apenas ocorrências dos nós que aparecem na versão do grafo de esquema simplificado, porém eles seguem as regras definidas na versão expandida quanto às subclasses. Por exemplo, todos os nós do tipo *"Funcionário"* devem agregar a propriedade "*Salário*" ou a propriedade "*Escala_pagto*" respectivamente com os rótulos *"Mensal"* ou *"Horista"*, entretanto, em nenhum desses nós ocorre simultaneamente as duas propriedades ou os dois rótulos. Isso respeita tanto a participação total de *"Funcionário"* na especialização em *"Men-*

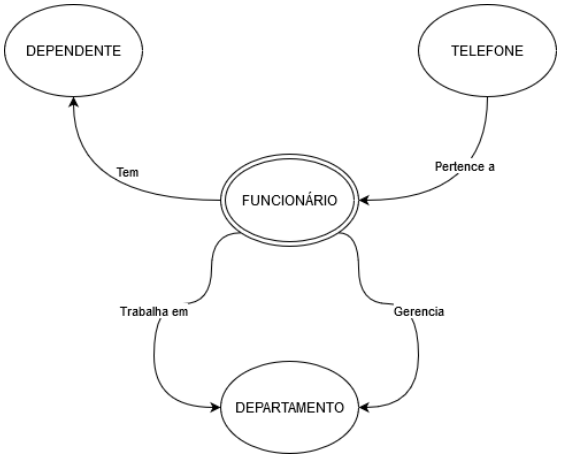

Figura 3.12. Versão simplificada do modelo apresentado na Figura 3.11

sal"/"Horista", quanto à disjunção entre estas subclasses. A partir daí, pode-se afirmar que o funcionário "*Pedro Cunha*" é um funcionário horista, pois é rotulado como tal e possui uma escala de pagamento semanal. Da mesma maneira, pode-se dizer que "*João Guerra*" e "*Isabel Souza*" são funcionários mensalistas, já que agregam a propriedade "*Salário*" além do rótulo *"Mensal"*. Também é possível dizer que a funcionária "*Isabel Souza*" é a gerente do departamento de "*Produção*", já que agregou o rótulo *"Gerente"* juntamente com a aresta "*Gerencia*", apontando para este departamento. Por fim, ainda é possível afirmar que "*Pedro Cunha*" é um funcionário técnico em alvenaria e que "*Isabel Souza*" é uma engenheira civil, entretanto "*João Guerra*" não pode ser classificado nem como técnico e nem como engenheiro. Esta última situação é permitida, já que a participação de *"Funcionário"* na especialização em *"Técnico"/"Engenheiro"* é apenas parcial.

3.3.4. Restrições de integridade em árvores de espelializações

A verificação de restrições de integridade de dados sobre ocorrências de hipernós de especialização se refere a um conjunto de regras para dar garantias às condições de participação total da superclasse e/ou de disjunção entre as subclasses participantes do processo. Estas duas condições, quando definidas no modelo conceitual, devem ser explicitadas no modelo lógico de grafos a fim de manter a consistência dos dados nas eventuais instâncias do BDG.

Como o modelo de grafos permite que seus nós, eventualmente, possam assumir mais de

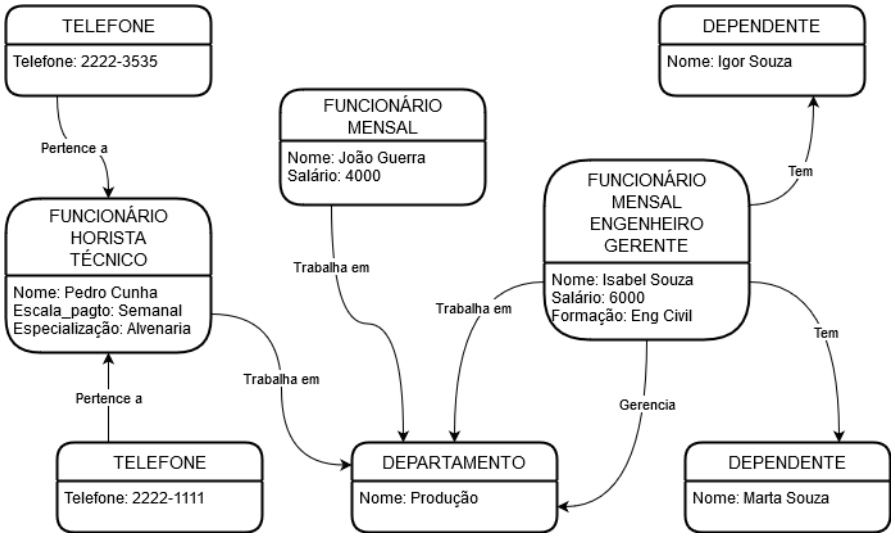

Figura 3.13. Instância de BDG baseado nos grafos de esquema das Figuras 3.11 e 3.12

um rótulo simultaneamente, pode-se usar uma estratégia para manipulação de rótulos dos nós para garantir as regras de participação total e/ou de disjunção estejam de acordo com as definições implementadas no modelo EER.

A estratégia consiste em definir que qualquer nó que tenha mais de um rótulo representa a ocorrência de um objeto do mundo real que pertence a pelo menos um subagrupamento dentro de uma superclasse. Por exemplo, na Figura 3.13, o nó que representa o funcionário *"João Guerra"* tem os rótulos *"Funcionário"* e *"Mensal"*, indicando que este nó representa uma ocorrência da entidade *"Funcionário"* que se especializa em *"Mensal"*. Da mesma forma, o nó que representa *"Isabel Souza"* referencia uma ocorrência de *"Funcionário"* que se especializa em *"Mensal"*, *"Engenheiro"* e *"Gerente"*.

Caso houvesse uma ocorrência de algum nó que assumisse, por exemplo, os rótulos *"Técnico"* e *"Engenheiro"* simultaneamente, este nó não seria validado pelo grafo de esquema da Figura 3.13, pois pode-se dizer que estes rótulos são *divergentes* entre si, já que representam subclasses que foram definidas como disjuntas na especialização de *"Funcionário"* em *"Técnico"*/*"Engenheiro"*. De maneira similar, a ocorrência de um nó assumindo *"Funcionário"* como único rótulo também não deve ser validado pelo grafo de esquema, uma vez que a entidade *"Funcionário"* tem participação total na especialização *"Horista"*/*"Mensal"* e, portanto, deve assumir um dos dois como um segundo rótulo.

Garantir que as restrições de integridade para especializações/generalizações sejam respeitadas no modelo de grafos consiste em garantir que os nós nas instâncias do BDG somente admitam conjuntos de rótulos que não sejam divergentes entre si e que caso haja algum processo de especialização com participação total da superclasse, esses nós agreguem pelo menos um rótulo referente às subclasses da especialização.

Para implementar uma validação de restrições de integridade de processos de especialização deve-se considerar:

- O nó α que representa uma superclasse em um processo de especialização;
- O conjunto $B = \{\beta_1, \beta_2, ...\}$ dos nós que representam as subclasses do mesmo processo de especialização.

Caso a especialização seja definida como *participação total* da superclasse α, então qualquer instância de α deve incluir o rótulo de α juntamente com o rótulo de pelo menos um nó $\beta_i \in B$. Em outras palavras, qualquer ocorrência de α deve ter pelo menos dois rótulos, sendo o primeiro o rótulo do nó α e o segundo, o rótulo de algum nó β_i.

Caso a especialização seja definida como uma *disjunção*, então qualquer instância de α pode incluir, além do rótulo de α, o rótulo relativo a, no máximo, um nó $\beta_i \in B$. Neste segundo caso, os nós β_i são ditos divergentes entre si e, portanto, em uma ocorrência de α não é possível haver rótulos de duas ou mais subclasses β_i.

Nota-se que, caso as duas regras sejam aplicadas ao mesmo processo de especialização, qualquer ocorrência do nó α nas instâncias do BDG devem incluir, além do rótulo de α, o rótulo de *exatamente um* nó $\beta_i \in B$, pois este caso exemplifica um processo de especialização com *participação total* da superclasse e *disjunção* entre as subclasses, configurando uma especialização com *disjunção total*.

3.4. Algoritmos de Mapeamento do Modelo EER para o Modelo de BDG

Nesta seção serão apresentados formalmente os algoritmos de mapeamento utilizados para geração de um modelo lógico de BDG a partir do modelo conceitual EER. A notação dos algoritmos é de pseudocódigo, entretanto a participação de alguns objetos deve ser melhor detalhada para uma melhor compreensão. Basicamente, consideram-se objetos de cinco classes: Entidade, Relacionamento, Especialização, Nó e Aresta. As três primeiras

classes representam objetos do modelo EER, enquanto que as últimas tratam objetos do modelo de grafos.

A classe *Entidade* apresenta como primeiro atributo de objeto o seu nome, que pode ser definido tão logo um objeto seja instanciado. Além disso, esta classe mantém uma lista de objetos *Atributo* que permite a definição de atributos simples/compostos e monovalorados/multivalorados da entidade. Por fim, caso a entidade representada pelo objeto já tenha sido mapeada, ela mantém uma referência para o objeto da classe Nó, que representa o nó mapeado no grafo de esquema.

Assim como a classe *Entidade*, a classe *Relacionamento* mantém um atributo que armazena seu nome e uma lista de objetos *Atributo* do relacionamento. Esta classe mantém também uma lista de *entidades participantes* do relacionamento. No caso de relacionamentos binários, esta lista deve conter referências tanto para o objeto *Entidade* de origem quanto para o objeto *Entidade* de destino do relacionamento. Para os casos de relacionamentos n-ários, esta lista mantém referências a pelo menos três objetos *Entidade*.

A classe *Especialização* modela objetos que representam os eventuais processos de especialização do modelo EER. Basicamente, esta classe possui um atributo que define a participação da superclasse (total ou parcial), um atributo para indicar se a especialização exige disjunção ou sobreposição entre as subclasses, uma referência para o objeto *Entidade* da superclasse e outra referência para a lista de entidades de subclasses. Baseado nisso, objetos desta classe podem executar o método *getSuperclasse* que retorna o objeto *Entidade* correspondente à superclasse no modelo. Analogamente, executa o método *getSubclasses* para retornar a lista dos entidades que representam as subclasses da especialização.

A Figura 3.14 mostra as definições das classes *Entidade*, *Relacionamento*, *Especialização*, usando notação UML.

Objetos da classe *Nó* mantém um rótulo e uma lista com as propriedades do nó. Para inserir uma nova propriedade em um objeto Nó, basta executar o método *insere_propriedade*, que toma como parâmetro o nome da propriedade a ser inserida. Também mantém listas de adjacências para arestas que saem do objeto (o objeto é origem da aresta) e para arestas que chegam ao objeto (o objeto é destino da aresta). Para inserir uma aresta na lista de

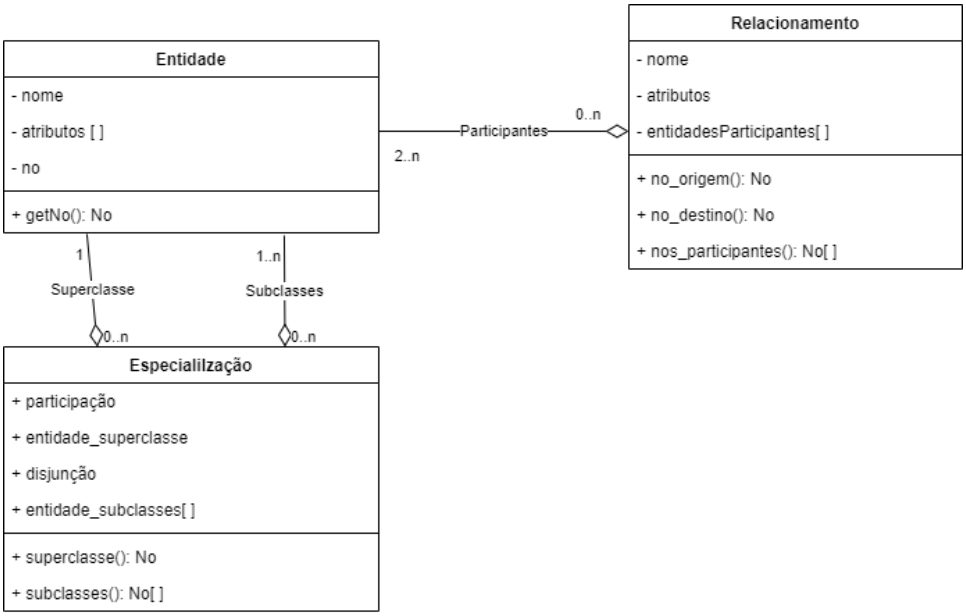

Figura 3.14. Descrição em UML das Classes Entidade, Relacionamento, e Especialização

arestas que saem do objeto, executa-se o método *cria_aresta* passando o rótulo da aresta a ser criada e uma referência para o nó de destino ou para uma lista de nós de destino (no caso de uma hiperaresta). Neste caso, os objetos referenciados como nós de destino adicionam a aresta criada em sua lista de arestas que chegam ao objeto. Sendo assim, o método *cria_aresta* é a maneira pela qual se estabelece conexões entre os nós do grafo.

Por fim, a classe *Aresta* gerencia objetos que representam as arestas do modelo lógico de grafos. Um objeto de aresta pode ser criado pelo método *cria_aresta* da classe *Nó* e possui um rótulo e uma lista de propriedades. Para inserir uma propriedade em uma aresta, executa-se o método *insere_propriedade* passando o nome da propriedade como parâmetro.

A Figura 3.15 mostra as definições das classes *Nó* e *Aresta*, usando notação UML. Os algoritmos a seguir partem de um mundo abstrato populado por objetos das classes *Entidade*, *Relacionamento* e *Especialização* e, a partir daí, geram automaticamente os objetos das classes *Nó* e *Aresta* equivalentes, criando um modelo de grafo de propriedades.

O Algoritmo 1 mapeia uma entidade do modelo conceitual EER. Ele considera o objeto *e*, da classe Entidade, como entrada. Nas linhas *1* e *2*, o algoritmo instancia o objeto *alpha*, da classe *Nó*, preenchendo seu rótulo com o nome de *e*. Da linha *3* até a linha *11* o

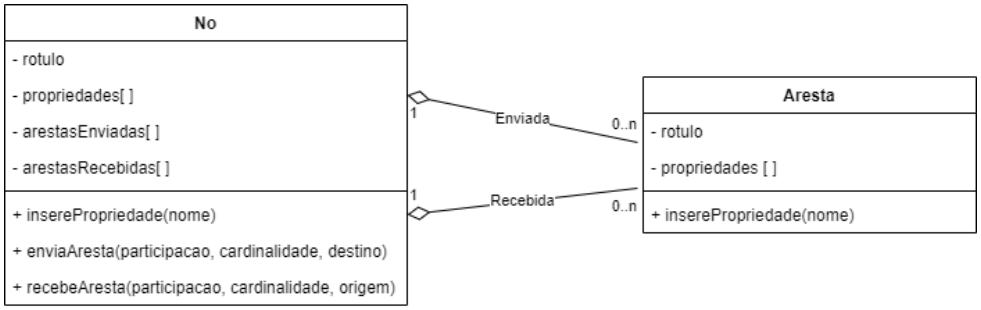

Figura 3.15. Descrição em UML das Classes Nó e Aresta

algoritmo mapeia todos os atributos monovalorados simples e compostos como propriedades de *alpha*. Nas linhas *12* até a *22*, cada atributo multivalorado de *e* é mapeado como um nó *beta*, transformando cada componente simples do atributo em propriedades para *beta*. Por fim, o nó *beta* cria uma aresta rotulada como "Pertence a" apontando para o nó *alpha*. Para este algoritmo e os próximos considera-se que, atributos compostos podem iterar entre seus componentes simples, a fim de mapeá-los como propriedades. A saída deste algoritmo é dada pelo objeto *alpha* da classe *Nó* e suas conexões com os nós *beta*, provenientes de atributos multivalorados.

O mapeamento de relacionamentos binários conta com dois algoritmos distintos, sendo o primeiro para o caso de relacionamentos que não possuem atributos multivalorados (Algoritmo 2) e o segundo para o caso contrário (Algoritmo 3).

O Algoritmo 2, que considera apenas os relacionamentos sem atributos multivalorados, toma como entrada um objeto *r* da classe *Relacionamento*. Como saída, este algoritmo produz uma aresta referenciada pelos nós de origem e de destino. Para executar este algoritmo, considera-se que todas as entidades básicas do modelo EER já tenham sido mapeadas para o modelo de grafos. Sendo assim, nas linhas *1* e *2* o relacionamento *r* atribui a *alpha_o* e *alpha_d* os objetos da classe *No* que representam respectivamente os nós de origem e de destino do relacionamento. Na linha *3*, o nó de origem *alpha_o* cria a aresta *epson* com destino ao nó *alpha_d*. Nas linhas *4* a *12* são mapeados os atributos simples e compostos de *r* como propriedades para *epson*.

Caso o relacionamento *r* possua algum atributo multivalorado, então deve ser executado o Algoritmo 3, onde o relacionamento é mapeado para o grafo como o nó *mu* nas linhas *3* e *4*. Os atributos monovalorados de *r* são convertidos em propriedades para *mu* nas

linhas *5* a *13*. Todos os atributos multivalorados do relacionamento *r* são mapeados como nós *beta* e seus componentes são transformados em propriedades nas linhas *14* até *23*. Cada nó *beta* é ligado ao nó *mu* com uma aresta rotulada como "Pertence a" na linha *24*. Finalmente, nas linhas *26* e *27* os nós *alpha_o* e *alpha_d* são conectados ao nó *mu* com arestas rotuladas como "Participa de".

O mapeamento de relacionamentos n-ários segue um algoritmo distinto do algoritmo de mapeamento de relacionamentos binários. Como entrada, ele considera o objeto *r* da classe *Relacionamento*. Como saída, este algoritmo produz um objeto da classe *Nó*, que representa o relacionamento *r* no modelo lógico. Todos os nós da lista de participantes devem ser conectados a este *nó de relacionamento* com arestas rotuladas como "Participa de". As linhas *1* e *2* criam o nó *mu* preenchendo o seu rótulo com o nome do relacionamento. A variável *participantes* é preenchida com a lista de todos os nós representantes das entidades participantes do relacionamento *r*. Para cada participante *alpha* do relacionamento, é criada uma aresta rotulada como "Participa de" cujo destino é o nó *mu*, nas linhas *4* a *6*. Todos os atributos monovalorados do relacionamento *r* são mapeados como propriedades de do nó *mu* nas linhas *7* a *15*. Da linha *16* até a linha *24* são mapeados os atributos multivalorados do relacionamento *r* como nós *beta*. Cada nó *beta* é ligado ao nó *mu* com uma aresta rotulada como "Pertence a" na linha *25*.

O mapeamento de especializações também é feito de maneiras distintas, dependendo da quantidade de subclasses envolvidas no processo. Para o caso mais simples, no qual existe apenas uma subclasse, basta criar uma aresta rotulada como *"É um"* partindo da subclasse em direção à superclasse.

Nos casos mais complexos, onde o processo de especialização envolve duas ou mais subclasses, deve ser verificado se a especialização define uma disjunção ou uma sobreposição e se a participação da superclasse é total ou parcial. Caso a especialização seja uma sobreposição com participação parcial da superclasse, então cada uma das subclasses pode criar uma aresta simples rotulada como *"É um"* com destino à superclasse.

Caso contrário, deve ser criada uma *hiperaresta* partindo da superclasse em direção a todas as subclasses relacionadas. Neste caso, a participação (total ou parcial) da superclasse e a disjunção/sobreposição das subclasses na especialização definem o rótulo da hipera-

resta. Para isso, utiliza-se o Algoritmo 5. Este algoritmo recebe como entrada um objeto *p* da classe *Especialização*. Como saída, este algoritmo pode produzir uma aresta rotulada como "É um" partindo do nó que representa a subclasse, no primeiro caso, ou uma *hiperaresta* criada a partir do nó que representa a superclasse (*alpha*) e indo em direção até cada as subclasses (nós *beta*), para o segundo caso. O rótulo desta *hiperaresta* depende da participação da superclasse e da disjunção/sobreposição das subclasses na especialização: no caso de participação total e disjunção, a hiperaresta será rotulada como *"Disjunção Total"*; caso a participação seja parcial e haja disjunção, o rótulo será *"Disjunção Parcial"*. Por fim, em caso de participação total e sobreposição, o rótulo será *"Sobreposição Total"*.

Na linha *1* a variável *alpha* é preenchida com o nó representante da superclasse da especialização a ser mapeada. Já na linha *2* ocorre o preenchimento da lista de subclasses da especialização. A linha *3* verifica a quantidade de subclasses da especialização *p*. Caso esta quantidade seja igual a *1*, a linha *4* é executada, fazendo com que a única subclasse da lista crie uma aresta rotulada *"É um"* com destino a superclasse *alpha*. Caso haja mais que uma subclasse, é executado o bloco de código das linhas *6* a *21*. A linha *6* verifica os casos de participação parcial e sobreposição, fazendo com que, nas linhas *7*, *8* e *9*, cada uma das subclasses crie uma aresta rotulada como *"É um"* apontando para a superclasse. Caso contrário, a linha *11* verifica se a superclasse ter participação total na especialização, nesses caso é criada uma hiperaresta rotulada como *"Disjunção Total"* se a especialização for definida como disjunta nas linhas *12* e *13*, ou então a hiperaresta é criada com o rótulo *"Sobreposição Total"* no caso da especialização admitir sobreposições entre as subclasses nas linhas *14* e *15*. Por fim, caso a especialização seja de participação parcial, a hiperaresta criada deverá ter o rótulo de *"Disjunção Parcial"* na linha *18*.

Observa-se que nas linhas *13*, *15* e *18*, o segundo parâmetro para o método *cria_aresta* é uma lista de objetos da classe nó, fazendo com que a aresta criada tenha um nó de origem (o nó *alpha*) e vários nós de destino (os nós da lista de subclasses). Sendo assim, a aresta criada a partir destes comandos pode ser considerada uma *hiperaresta* originando na superclasse (cauda) e chegando a cada uma das subclasses da especialização (cabeça).

O capítulo 4 apresenta um estudo de caso baseado em um modelo conceitual EER de

banco de dados para controle e gerenciamento de eventos acadêmicos do Instituto Federal de Mato Grosso do Sul - Campus Corumbá. Este capítulo apresenta dois modelos lógicos para o modelo conceitual, sendo o primeiro um esquema relacional, obtido pela metodologia de mapeamento conceitual-relacional apresentada por (Elmasri et al.; 2011), e o segundo um modelo de grafo de propriedades, obtido a partir da submissão dos elementos do modelo conceitual aos algoritmos de mapeamento propostos por este trabalho.

Algorithm 1 Mapeia_Entidade (Entidade: e)

```
1: Cria o nó alpha
2: alpha.rotulo ← e.nome
3: for All atr in e.atributos_monovalorados do
4:     if atr é simples then
5:         alpha.insere_propriedade(atr)
6:     else
7:         for All componente in atr do
8:             alpha.insere_propriedade(atr + "." + componente)
9:         end for
10:    end if
11: end for
12: for All atr in e.atributos_multivalorados do
13:    Cria o nó beta
14:    beta.rotulo ← atr
15:    if atr é simples then
16:        beta.insere_propriedade(atr)
17:    else
18:        for All componente in atr do
19:            beta.insere_propriedade(componente)
20:        end for
21:    end if
22:    beta.cria_aresta("Pertence a", alpha)
23: end for
```

Algorithm 2 Mapeia_Relacionamento_Binario (Relacionamento: r)

```
alpha_o ← r.no_origem()
alpha_d ← r.no_destino()
epson ← alpha_o.cria_aresta(r.nome, alpha_d)
for All atr in r.atributos_monovalorados do
    if atr é simples then
        epson.insere_propriedade(atr)
    else
        for All componente in atr do
            epson.insere_propriedade(componente)
        end for
    end if
end for
```

Algorithm 3 Mapeia_Relacionamento_Binario_Multi (Relacionamento: r)

```
alpha_o ← r.no_origem()
alpha_d ← r.no_destino()
Cria o nó mu
mu.rotulo ← r.nome
for All atr in r.atributos_monovalorados do
    if atr é simples then
        mu.insere_propriedade(atr)
    else
        for All componente in atr do
            mu.insere_propriedade(componente)
        end for
    end if
end for
for All atr in r.atributos_multivalorados do
    Cria o nó beta
    beta.rotulo ← atr
    if atr é simples then
        beta.insere_propriedade(atr)
    else
        for All componente in atr do
            beta.insere_propriedade(componente)
        end for
    end if
    beta.cria_aresta("Pertence a", mu)
end for
alpha_o.cria_aresta("Participa de", mu)
alpha_d.cria_aresta("Participa de", mu)
```

Algorithm 4 Mapeia_Relacionamento_N_Ario (Relacionamento: r)

```
1: Cria o nó mu
2: mu.rotulo ← r.nome
3: participantes ← r.nos_participantes()
4: for All alpha in participantes do
5:     alpha.cria_aresta("Participa de", mu)
6: end for
7: for All atr in r.atributos_monovalorados do
8:     if atr é simples then
9:         mu.insere_propriedade(atr)
10:    else
11:        for All componente in atr do
12:            mu.insere_propriedade(componente)
13:        end for
14:    end if
15: end for
16: for All atr in r.atributos_multivalorados do
17:    Cria o nó beta
18:    if atr é simples then
19:        beta.insere_propriedade(atr)
20:    else
21:        for All componente in atr do
22:            beta.insere_propriedade(componente)
23:        end for
24:    end if
25:    beta.cria_aresta("Pertence a", mu)
26: end for
```

Algorithm 5 Mapeia_Especialização (Especialização: p)

```
1: alpha ← p.superclasse()
2: subclasses ← p.subclasses()
3: if subclasses.quantidade = 1 then
4:     subclasses[0].cria_aresta("É um", alpha)
5: else
6:     if p.participação = "Parcial" , p.disjunção = false then
7:         for All subclasse in subclasses do
8:             subclasse.cria_aresta ("É um", alpha)
9:         end for
10:    else
11:        if p.participação = "Total" then
12:            if p.disjunção = true then
13:                alpha.cria_aresta("Disjunção Total", subclasses)
14:            else
15:                alpha.cria_aresta("Sobreposição Total", subclasses)
16:            end if
17:        else
18:            alpha.cria_aresta("Disjunção Parcial", subclasses)
19:        end if
20:    end if
21: end if
```

Capítulo 4

Estudo de Caso

Para validar o processo proposto, um estudo de caso foi conduzido, em forma de Prova de Conceito, para aplicar os algoritmos em um problema do mundo real. O processo de conversão parte de uma modelagem conceitual baseada nos conceitos de modelo EER apresentados neste trabalho, o qual será convertido para um grafo de esquema. Uma instância do BDG baseada no projeto lógico resultante da conversão será implementada usando o Neo4j.

4.1. Contextualização

Este estudo de caso é baseado na necessidade de implementação de um sistema de bancos de dados para gerenciamento de eventos acadêmicos do Instituto Federal de Mato Grosso do Sul (IFMS), câmpus Corumbá. São definidos como eventos acadêmicos as feiras de tecnologia e semanas temáticas onde os estudandes, tanto da instituição quanto de outras escolas da região, apresentam seus trabalhos e projetos de pesquisa, orientados e coorientados por docentes e pesquisadores.

Os trabalhos são organizados por nível de ensino (Fundamental ou Médio) e categorizados de acordo com a área de pesquisa: Ciências Biológicas e da Saúde; Ciências Exatas e da Terra; Ciências Humanas e Sociais Aplicadas; Engenharias e Ciências Agrárias; Multidisciplinar.

Os trabalhos concorrem entre si, de acordo com o seu nível de ensino e de sua área de pesquisa, embora haja um quadro geral de pontuação que ranqueia os trabalhos dentro de

cada nível independentemente de área de pesquisa.

O ranqueamento dos trabalhos é dado por um sistema de avaliações por quesitos, onde os quesitos de avaliação são predeterminados pelo edital de abertura do evento. Para cada quesito, é definido um peso que é utilizado no cálculo ponderado da pontuação de cada trabalho.

Por fim, a pontuação dos trabalhos é atribuída por um conjunto de avaliadores que lançam notas de zero a dez em todos os quesitos para cada trabalho em que são escalados para avaliar. A pontuação dada por um avaliador para um trabalho é calculada pelo somatório das notas multiplicadas pelo peso em cada quesito avaliado. A pontuação final de um trabalho é calculada pela média das pontuações dadas por todos os avaliadores que lançaram notas para o trabalho.

A Figura 4.1 mostra o projeto conceitual do banco de dados do sistema de controle de eventos do IFMS. Neste diagrama EER foram identificadas as principais entidades do banco de dados com seus atributos e relacionamentos. Para cada relacionamento binário foi acrescentada uma seta indicando o sentido semântico de sua leitura. O relacionamento "Orienta", por exemplo, que representa a associação entre orientadores e trabalhos, está marcado com uma seta apontando na direção da entidade "Trabalho", que indica que a semântica definida para este relacionamento denota que "*orientadores orientam trabalhos*" e não o contrário.

Neste modelo, a entidade "Pessoa" é modelada como uma superclasse de especialização para as subclasses "Orientador", "Avaliador" e "Estudante", porém com participação parcial no processo. A entidade central do banco de dados é a entidade "Trabalho", que existe condicionada a um ou mais orientadores, a um ou mais estudantes participantes e organizada dentro de um evento. Os eventos definem os quesitos de avaliação e organizam os trabalhos.

Por fim, as avaliações foram modeladas como um relacionamento ternário entre "Trabalho", "Avaliador" e "Quesito". Neste contexto, um trabalho recebe de um avaliador um conjunto de valores relativos às notas nos quesitos definidos pelo evento. A pontuação dada pelo avaliador é um valor calculado pelo somatório de todas as notas atribuídas pelo avaliador multiplicadas pelo peso do respectivo quesito indicado.

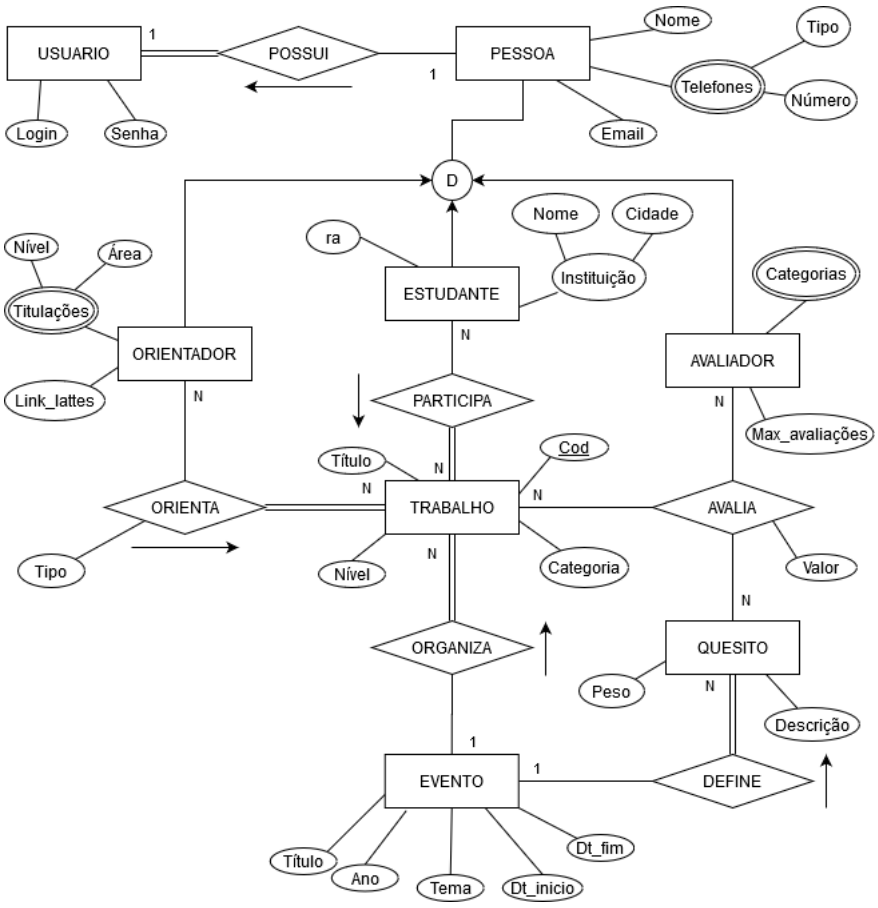

Figura 4.1. Projeto Conceitual do Banco de Dados de Controle de Eventos

Neste esquema conceitual não foram considerados detalhes sobre os usuários do sistema, os quais gerenciam os eventos e que envolvem um esquema de atores e controles de acesso. Somente as entidades que representam o evento em si, trabalhos e seus relacionamentos foram detalhados.

4.2. O Banco de Dados com uma Abordagem Relacional

Para efeitos de comparação entre os modelos de dados relacional e de grafos, esta seção apresenta o resultado da conversão do modelo conceitual da Figura 4.1 para um esquema de relações, atributos e relacionamentos representados por chaves estrangeiras do modelo relacional. Para gerar um esquema relacional a partir do modelo conceitual apresentado serão utilizados os algoritmos de mapeamento do modelo conceitual EER para o modelo relacional apresentados por Elmasri et al. (2011) .

Resumidamente, os algoritmos de mapeamento convertem cada entidade em uma relação,

onde são incluídos todos os seus atributos simples e componentes simples de seus atributos compostos. Um dos atributos identificadores modelados deve ser escolhido como chave primária da relação gerada. Caso contrário, pode ser incluído um atributo identificador no modelo relacional para atuar como chave primária.

Considerando a primeira etapa de mapeamento, geram-se as relações "Usuário", "Pessoa", "Trabalho", Evento" e "Quesito". Apenas o atributo "Cod", da entidade "Trabalho" foi modelado como identificador no modelo conceitual, sendo assim, foi acrescentado ao modelo lógico o atributo "id" como chave primária em todas as relações geradas até este ponto.

Já em uma segunda etapa de geração do modelo relacional é feito o mapeamento da especialização de "Pessoa" em "Orientador", "Estudante" e "Avaliador". Para esta fase do mapeamento, optou-se por criar uma relação para cada subclasse do modelo conceitual. Neste processo foi incluida como chave estrangeira a chave primária de "Pessoa" e os atributos próprios de cada subclasse em suas respectivas relações. Além disso, para cada relação gerada neste processo, a chave estrangeira "Pessoa_id" foi definida como chave primária. Esta opção de conversão de especializações pode ser usada para representar tanto especializações totais como parciais e permite disjunção ou sobreposição entre as subclasses (Elmasri et al.; 2011).

O próximo passo indicado é o mapeamento dos relacionamentos binários do modelo conceitual. Em sua forma mais simples, estes relacionamentos são convertidos utilizando-se de chaves estrangeiras nos relacionamentos de cardinalidade 1:1 e 1:N. Relacionamentos N:N exigem uma "relação de referência cruzada" (Elmasri et al.; 2011), que cria uma relação intermediária entre as relações participantes do relacionamento.

No modelo conceitual apresentado, apenas o relacionamento "*Pessoa possui Usuário*" é um relacionamento de cardinalidade 1:1 com participação total de "Usuário". Neste cenário, (Elmasri et al.; 2011) sugere como primeira opção incluir a chave primária de "Pessoa" como chave estrangeira em "Usuário".

Outros dois relacionamentos possuem cardinalidade 1:N, que são: "*Evento organiza Trabalho*" e "*Evento define Quesito*". A entidade "Evento" tem cardinalidade igual a *1* tanto no relacionamento "organiza" quanto no relacionamento "define", enquanto que as en-

tidades “Trabalho” e “Quesito”, respectivamente, tem cardinalidade igual a *N*. Nestes casos, as chaves primárias das relações de cardinalidade *1* são incluídas como chaves estrangeiras nas relações de cardinalidade *N*. Portanto, neste estudo de caso, a entidade “Trabalho” recebe a chave primária de “Evento” como chave estrangeira. O mesmo ocorre com a entidade “Quesito”, que também recebe a chave primária de “Evento” como chave estrangeira.

O último tipo de relacionamento binário envolve ambas as entidades participantes com cardinalidade *N*. Neste caso, o relacionamento é transformado em uma nova relação e recebe as chaves primárias das relações participantes como chaves estrangeiras. Caso o relacionamento tenha atributos, estes são incluídos na nova relação. No modelo conceitual os relacionamentos N:N são “*Orientador orienta Trabalho*” e “*Estudante participa de Trabalho*”. Nestes casos o relacionamento “orienta” gera a relação “Orientação”, enquanto que o relacionamento “participa” gera a relação “Participação”. As chaves primárias de “Orientador” e de “Trabalho” são incluídas em “Orientação” como chaves estrangeiras. Similarmente, na relação “Participação” devem ser incluídas como chaves estrangeiras as chaves primárias de “Trabalho” e de “Estudante”.

A Figura 4.2 mostra o esquema relacional gerado até este ponto do processo de mapeamento. Este esquema mostra as relações “Usuário”, “Pessoa”, “Trabalho”, “Quesito” e “Evento”, que foram geradas na primeira etapa do processo de mapeamento. Na segunda etapa, foram geradas as subclasses “Orientador”, “Estudante” e “Avaliador”, destacadas em azul. Por fim, foram incluídas as chaves estrangeiras em “Usuário”, “Trabalho” e “Quesito” (com destaque em vermelho) representando os relacionamentos binários 1:1 e 1:N e as relações “Orientação” e “Participação” (também em vermelho) indicando relacionamentos N:N. Nota-se que até este momento não foram mapeados os atributos multivalorados e o relacionamento n-ário “Avalia”.

Segundo Elmasri et al. (2011), para cada atributo multivalorado *A* do modelo conceitual, uma nova relação é criada no modelo lógico relacional. Deve ser incluída nesta relação o atributo (ou conjunto de atributos, caso *A* seja composto) que representem *A*, além da chave primária da relação proprietária do atributo *A* no modelo relacional. A chave primária de *A* no modelo relacional é composta por sua chave estrangeira combinada com

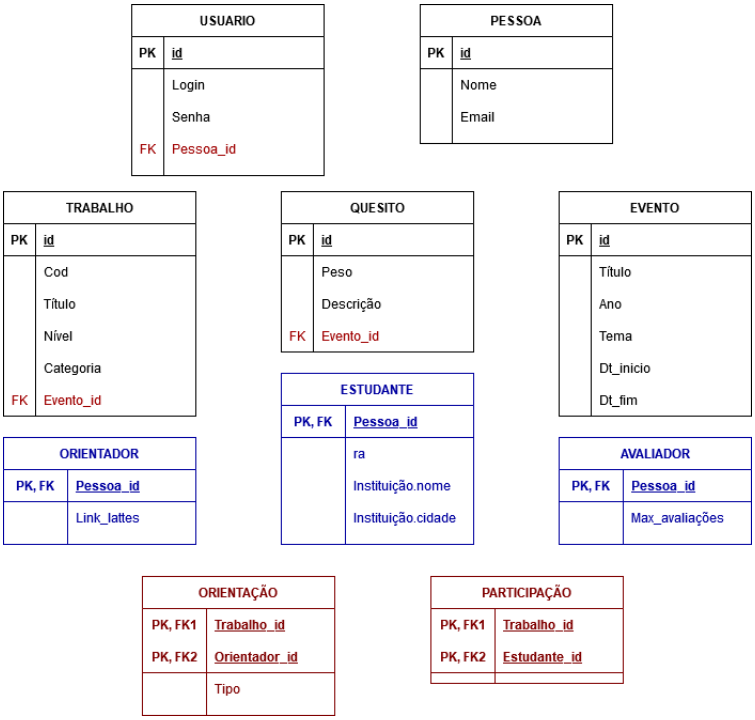

Figura 4.2. Mapeamento das Entidades, Especializações e Relacionamentos Binários do Modelo EER

os atributos de *A* que forem incluídos no processo.

Em nosso estudo de caso os atributos multivalorados são: “Telefones” pertencente a “Pessoa”; “Categorias” ligado a “Avaliador”; e “Titulações”, que pertence a “Orientador”. Estes atributos multivalorados devem ser mapeados como relações no modelo lógico conforme mostra a Figura 4.3, que mostra o mapeamento dos atributos multivalorados do modelo conceitual da Figura 4.1. A relação “Telefone” representa o atributo “Telefones” de “Pessoa” e possui como atributos a chave primária de “Pessoa” e os campos “Tipo” e “Número”, que compõem o atributo multivalorado no modelo conceitual. Analogamente, “Titulação” e “Categoria” seguem as mesmas regras de mapeamento.

Figura 4.3. Mapeamento dos Atributos Multivalorados do Modelo EER

Por fim, os relacionamentos n-ários devem ser mapeados como relações no modelo relacional. As chaves primárias das relações participantes do relacionamento devem ser

incluídas como chaves estrangeiras nesta nova relação, assim como os atributos próprios do relacionamento. A chave primária da relação é dada pela combinação de todas as chaves estrangeiras que referenciam as entidades participantes que tenham cardinalidade *N* no relacionamento n-ário (Elmasri et al.; 2011).

No modelo conceitual, o único relacionamento n-ário é o relacionamento "Avalia", que associa avaliadores, trabalhos e quesitos para composição das notas. Neste caso, o relacionamento "Avalia" pode ser mapeado como a relação "Avaliação", que inclui as chaves primárias de "Trabalho", "Avaliador" e "Quesito" como chaves estrangeiras e as define como chave primária da relação, uma vez que todas as relações participantes tem cardinalidade *N* do relacionamento n-ário. O resultado pode ser visto na Figura 4.4.

AVALIAÇÃO	
PK, FK	Trabalho_id
PK, FK	Avaliador_id
PK, FK	Quesito_id
	Valor

Figura 4.4. Mapeamento do Relacionamento Ternário "Avalia"

O esquema completo do modelo lógico gerado a partir do mapeamento do modelo conceitual da Figura 4.1 é mostrado pela Figura 4.5.

O esquema relacional do sistema de gerenciamento de eventos do IFMS conta com quatorze relações que foram geradas a partir do processo de mapeamento do esquema conceitual EER mostrado na Figura 4.1. Para a maioria das entidades mapeadas foi incluído como chave primária o atributo "id", além da inclusão de atributos de chave estrangeira para representação de relacionamentos 1:1 e 1:N. Os atributos multivalorados foram mapeados como relações ligadas as relações proprietárias. Finalmente, para os relacionamentos N:N e para o relacionamento ternário foram geradas relações intermediárias onde foram incluídas as chaves primárias das relações participantes.

4.3. Geração do Esquema Lógico de Grafos

O esquema lógico de BDG para o modelo conceitual da Figura 4.1, proposto para este estudo de caso, pode ser descrito como um grafo que representa as entidades e seus relacionamentos respectivamente como nós e arestas. Algumas regras que são inerentes ao

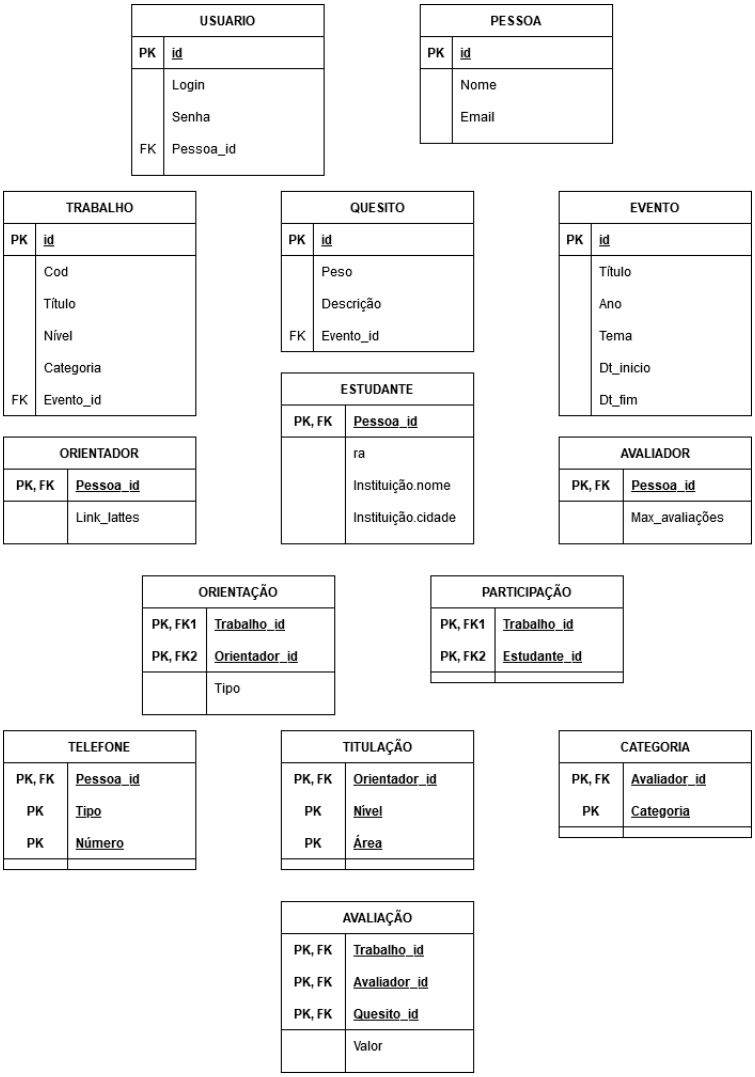

Figura 4.5. Esquema Relacional do Sistema de Gerenciamento de Eventos do IFMS

modelo conceitual, como cardinalidades e participação, devem ser definidas como regras explicitas em um modelo de grafos.

A primeira etapa do processo de mapeamento para um esquema de BDG trata da conversão das entidades do modelo conceitual em nós em um grafo de propriedades. Os atributos de entidade são convertidos em propriedades e os nós gerados devem ser rotulados com o nome das entidades de origem. Para processamento desta etapa deve-se executar o algoritmo 1 para o mapeamento de cada entidade do modelo conceitual.

Cada entidade de entrada do algoritmo de mapeamento carrega informações sobre sua lista de atributos e sobre seus relacionamentos. Para o modelo conceitual considerado, a

lista de entidades, com suas respectivas listas de atributos, para esta etapa do processo de mapeamento é composta por:

- **Usuário**
  - Atributos monovalorados: “Login” e “Senha”
- **Pessoa**
  - Atributos monovalorados: “Nome” e “Email”
  - Atributo multivalorado: “Telefones” (composto por “Tipo” e “Número”
- **Trabalho**
  - Atributos monovalorados: “Cod”, “Título”, “Nível” e “Categoria”
- **Quesito**
  - Atributos monovalorados: “Peso” e “Descrição”
- **Evento**
  - Atributos monovalorados: “Título”, “Ano”, “Tema”, “Dt_inicio” e “Dt_fim”
- **Orientador**
  - Atributo monovalorados: “Link_lattes”
  - Atributo multivalorado: “Titulações” (composto por “Nível” e “Área”)
- **Estudante**
  - Atributo monovalorados: “ra” e “Instituição” (composto por “Nome” e “Cidade”
- **Avaliador**
  - Atributo monovalorados: “Max_avaliações”
  - Atributo multivalorado: “Categorias”

Considerando que o processo de mapeamento para o projeto lógico de BDG siga a mesma ordem da lista acima, a primeira entidade a ser mapeada é a entidade “Usuário”. Para esta entidade o algoritmo 1 gera o nó rotulado como “Usuário” e nele são inseridas as propriedades “Login” e “Senha” a partir da lista de atributos monovalorados da entidade. Na sequência, deve ser mapeada a entidade “Pessoa”, para a qual é gerado o nó rotulado como “Pessoa”. Os atributos monovalorados da entidade são mapeados como propriedades do nó enquanto que o atributo multivalorado “Telefones” gera um nó rotulado como “Telefone”, onde são incluídos os componentes simples “Tipo” e “Número”. Por fim, o

nó “Telefone” é ligado ao nó “Pessoa” por uma aresta rotulada como “Pertence a” com direção apontada a “Pessoa”.

Os mapeamentos das entidades “Trabalho”, “Quesito” e “Evento” seguem a mesma sequência de passos de “Usuário”, enquanto que o mapeamento da entidade “Orientador” segue de forma bastante similar a “Pessoa”. Quanto ao mapeamento da entidade “Estudante”, este gera o nó rotulado como “Estudante”, onde é incluída a propriedade “ra” e os componentes simples do atributo composto “Instituição”, que são nomeados como as propriedades “Instituição.Nome” e “Instituição.Cidade” a fim de evitar possíveis redundâncias com nomes de outros atributos inerentes ao nó. Por fim, o mapeamento da entidade “Avaliador” gera no grafo de esquema o nó rotulado como “Avaliador”, ao qual é inserida a propriedade “Max_avaliações”. O atributo multivalorado “Categorias” gera o nó “Categoria”, que como não possui elementos de composição, agrega apenas uma propriedade também nomeada como “Categoria”. O nó “Categoria” deve ser ligado ao nó “Avaliador” com uma aresta rotulada como “Pertence a” apontando para “Avaliador”.

A Figura 4.6 mostra o resultado do processo de mapeamento ao fim desta primeira etapa, mostrando o grafo de esquema gerado ao fim da etapa de mapeamentos das entidades modeladas na Figura 4.1. Os nós e arestas que aparecem neste modelo lógico de BDG foram gerados a partir da execução do algoritmo 1 para cada entidade do modelo conceitual.

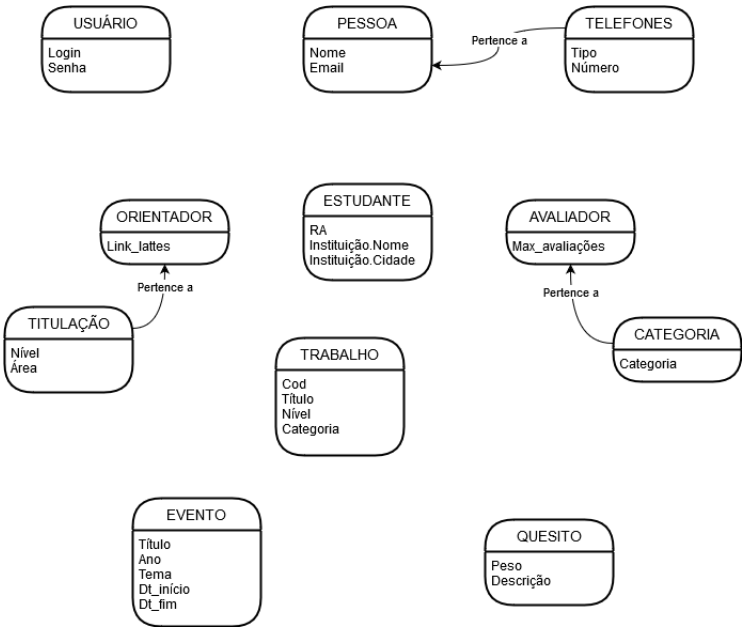

Figura 4.6. Mapeamento das Entidades do Modelo Relacional da Figura 4.1

Em uma próxima etapa deve ser executado o algoritmo 2, que toma como parâmetro um relacionamento binário *r* do modelo EER que não possua atributos multivalorados. Além disso, este algoritmo considera que as entidades participantes de *r* já foram mapeadas como nós para o modelo lógico de BDG. A lista de relacionamentos que atendem a estes quesitos são:

- **Pessoa possui Usuário**
  - Entidade de Origem: "Pessoa"
  - Entidade de Destino: "Usuário"
- **Orientador orienta Trabalho**
  - Entidade de Origem: "Orientador"
  - Entidade de Destino: "Trabalho"
  - Atributo: tipo
- **Estudante participa de Trabalho**
  - Entidade de Origem: "Estudante"
  - Entidade de Destino: "Trabalho"
- **Evento organiza Trabalho**
  - Entidade de Origem: "Evento"
  - Entidade de Destino: "Trabalho"
- **Evento define Quesito**
  - Entidade de Origem: "Evento"
  - Entidade de Destino: "Quesito"

Ao executar o algoritmo 2 para o relacionamento "Pessoa possui Usuário", o primeiro passo é definir os nós de origem e de destino do relacionamento. Neste caso, o nó "Pessoa" atua como nó de origem ($alpha_o$), enquanto que o nó "Usuário" assume o papel de nó de destino ($alpha_d$). Em seguida, o nó "Pessoa" cria a aresta "$theta$", rotulada como "Possui" e indicando o nó "Usuário" como destino. Os próximos passos tratam do mapeamento dos atributos do relacionamento, porém a lista de atributos deste relacionamento é vazia, o que encerra a execução do algoritmo para este primeio relacionamento.

O segundo relacionamento a ser mapeado é o relacionamento "Orientador orienta Trabalho". Novamente, o primeiro passo define o nó $alpha_o$ como sendo "Orientador" e o nó $alpha_d$ como sendo Trabalho. O nó "Orientador" cria a aresta "$theta$", rotulada como

“orienta”, o conectando a “Trabalho”. Finalmente, a aresta “$theta$” insere a propriedade “Tipo”, relativa ao atributo do relacionamento.

O mapeamento dos relacionamentos “Estudante participa de Trabalho”, “Evento organiza Trabalho” e “Evento define Quesitos” segue os mesmos passos do mapeamento de “Pessoa possui Usuário”.

Como este modelo conceitual não possui relacionamentos com atributos multivalorados, então o algoritmo 3 não será executado para este estudo de caso.

A Figura 4.7 mostra o resultado desta etapa do mapeamento do modelo conceitual. Esta figura mostra as arestas mapeadas na segunda etapa de mapeamento do modelo relacional da Figura 4.1 destacadas em vermelho. Cada aresta foi gerada a partir de uma execução do algoritmo 2 a partir de um relacionamento binário sem atributos multivalorados. Vale destacar que apenas a aresta “Orienta” possui uma propriedade, proveniente de um atributo do relacionamento de mesmo nome do modelo conceitual.

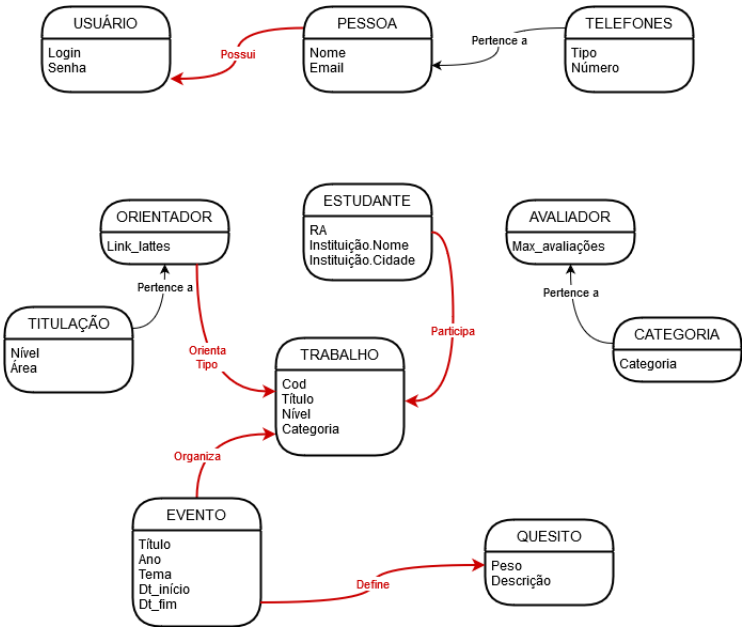

Figura 4.7. Mapeamento dos Relacionamentos Binários do Modelo Relacional da Figura 4.1

A próxima etapa do mapeamento é o processamento dos relacionamentos n-ários do modelo conceitual. Este tipo de relacionamento deve ser mapeado como um nó no grafo de esquema e conectado aos nós participantes. Para cada relacionamento n-ário deve ser exe-

cutado o algoritmo 4 passando o objeto que representa o relacionamento como parâmetro de entrada. No esquema conceitual apresentado na Figura 4.1 o único relacionamento n-ario é o relacionamento "Avalia" no qual avaliadores avaliam os trabalhos sobre um conjunto de quesitos pré estabelecidos.

- **Relacionamento "Avalia"**
  - Entidades Participantes: "Avaliador", "Trabalho" e "Quesito".
  - Atributo Monovalorado: "Valor"

Ao executar o algoritmo 4 passando como parâmetro o relacionamento *"Avalia"*, o primeiro passo é a geração do nó "mi", rotulado como *"Avalia"*, o qual representa o relacionamento *"Avalia"* no grafo de esquema. Em seguida, o algoritmo percorre a lista dos nós participantes do relacionamento, conectando cada um deles ao nó *"Avalia"* com arestas rotuladas como *"Participa de"*. O atributo *"Valor"*, inerente ao relacionamento *"Avalia"* é mapeado como uma propriedade para o nó *"Avalia"*. Como este relacionamento não possui atributos multivalorados, a execução do algoritmo 4 se encerra.

A Figura 4.8 destaca em vermelho o nó "Avalia" gerado a partir do relacionamento "Avalia" do modelo conceitual. Os nós "Trabalho", "Avaliador" e "Quesito" se conectam ao nó "Avalia" com arestas rotuladas como "Participa de". O atributo de relacionamento "Valor", que representa o valor da nota que um avaliador atribui a um determinado trabalho em um quesito é inserido como propriedade de "Avalia".

Por fim, o processo de mapeamento deve executar o algoritmo 5 para mapear as especializações. No caso de o modelo conceitual mostrar uma especialização com apenas uma subclasse, este algoritmo gera uma aresta originando a partir da subclasse com destino a superclasse e rotulada como "É um". Já para o caso de uma superclasse ser dividida em várias subclasses disjuntas, este algoritmo gera uma aresta partindo da superclasse, porém referenciada por cada uma das subclasses em suas respectivas listas de arestas que chegam ao nó, isto é, a mesma aresta é definida como destino de vários nós, configurando uma hiperaresta dirigida da superclasse para as subclasses.

Para o modelo conceitual deste estudo de caso, o algoritmo 5 deve ser executado para a especialização de "Pessoa" em "Orientador", "Estudante" e "Avaliador". Como resultado, é gerada uma aresta rotulada como "Disjunção Parcial" partindo de "Pessoa" e chegando

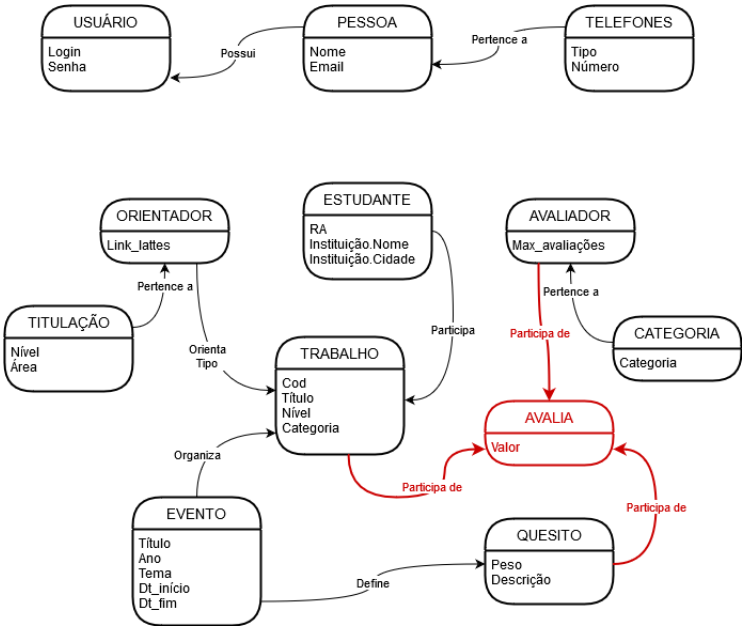

Figura 4.8. Mapeamento do Relacionamento N-ário "Avalia" do Modelo Relacional da Figura 4.1

em "Orientador", "Estudante" e "Avaliador".

Após o encerramento do algoritmo 5, o processo de mapeamento do modelo conceitual da Figura 4.1 chega ao fim.

A Figura 4.9 mostra o grafo de esquema resultante do processo de mapeamento do modelo conceitual exibido na Figura 4.1. Este grafo destaca em vermelho a hiperaresta gerada a partir da especialização de "Pessoa" em "Orientador", "Estudante" e "Avaliador". Esta hiperaresta foi rotulada como "Disjunção Parcial" dada a participação parcial da entidade "Pessoa" na especialização no modelo conceitual.

Ao fim deste processo de mapeamento, foi gerado um grafo de esquema contendo doze nós, onze arestas e uma hiperaresta. Embora este modelo possa parecer relativamente pequeno quanto ao número de elementos presentes, o uso de um banco dados de grafo pode ser vantajoso ao considerar o grande potencial de interconexões entre as instâncias destes nós em um BDG. Relacionamentos N:N são simplificados em comparação com o modelo relacional, já que no modelo lógico de grafos, relacionamentos com estas cardinalidades podem ser representados com apenas uma aresta ligando os nós, e, em um modelo relacional, o mapeamento destes casos gera relações intermediárias com chave primária

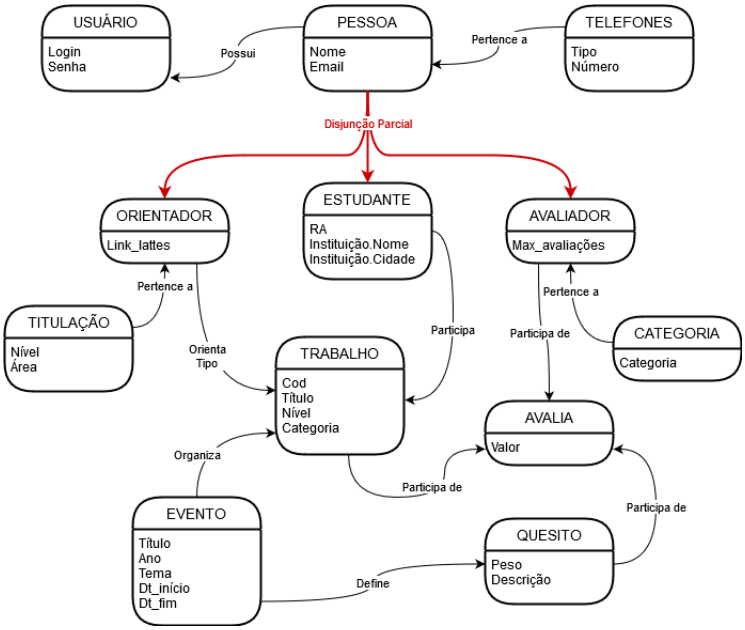

Figura 4.9. Mapeamento da Especialização de "Pessoa" em "Orientador", "Estudante" e "Avaliador"

composta pelas chaves estrangeiras das entidades participantes.

Vale ressaltar ainda que o grafo de esquema da Figura 4.9 representa a versão expandida do modelo lógico de grafos. Instâncias baseadas neste esquema lógico não produzirão nós rotulados como "Orientador", "Estudate" ou "Avaliador". Apenas instâncias de "Pessoa" deverão aparecer no grafo de dados, porém com a possibilidade de agregar atributos e relacionamentos definidos para suas subclasses.

A Figura 4.10 apresenta a versão simplificada do grafo de esquema gerado pelos algoritmos de mapeamento para o modelo lógico de grafos. Neste esquema, o nó "Pessoa" é destacado dos demais, indicando ser este um hipernó que representa a árvore de especializações de "Pessoa". Este hipernó agrega as propriedades e arestas dos nós "Orientador", "Estudante" e "Avaliador" e cada elemento agregado mostra entre parênteses o rótulo do nó proprietário no modelo expandido. Por exemplo, o atributo "Link_lattes" no modelo simplificado é um elemento agregado de "Orientador" no modelo expandido e a aresta "Participa" é também um elemento agregado, neste caso, do nó "Estudante" do modelo expandido. Este modelo simplificado é considerado por mostrar apenas os nós que deverão ser instanciados quando o BDG foi populado. Observando estas considerações é possível constatar que apenas nove nós do grafo de esquema serão instanciados em uma

eventual instância do BDG.

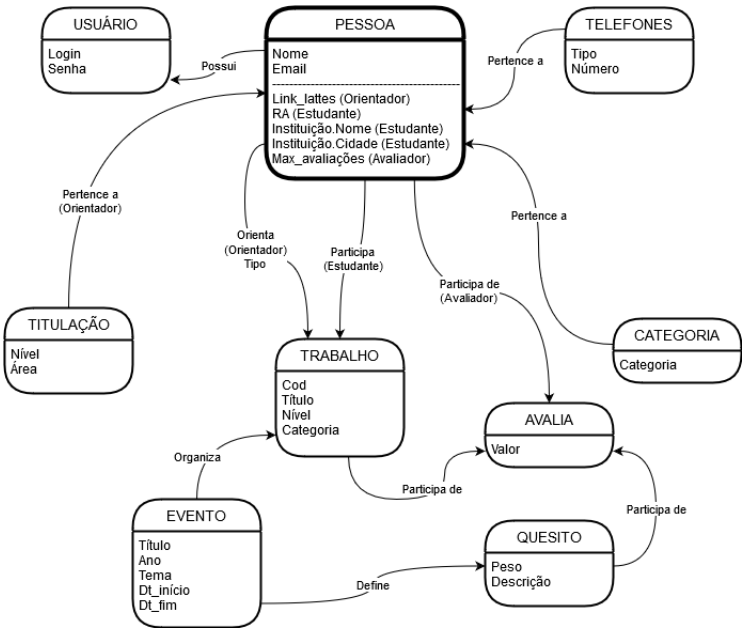

Figura 4.10. Versão Simplificada do Grafo de Esquema da Figura 4.9

4.4. Uma Instância do BDG

Nesta seção será implementada uma instância do BDG seguindo os parâmetros do projeto lógico apresentado pela Figura 4.9. Esta instância contém dados fictícios e será implementada utilizando o SGBD de grafos *Neo4j*.

Para manipular dados na instância do banco de dados, será utilizada a linguagem *Cypher*, a qual utiliza uma técnica de *ASCII art*, onde o código escrito se torna graficamente parecido com o diagrama de um grafo (Robinson et al.; 2013). Neste caso, a definição ou a referência a um nó é feita entre *parênteses*, enquanto que as definições das arestas são delimitadas por *traços* e *colchetes*. As propriedades de nós e arestas devem ser definidas entre *chaves* estabelecendo uma sequência de pares "*chave*:*valor*" separados por *vírgulas*.

A instância do BDG exemplificada para este estudo de caso e, seguindo o projeto lógico da Figura 4.9, contem:

- Uma instância de *"Evento"*;
- Quatro instâncias de *"Quesito"*;
- Quatro instâncias de *"Trabalho"*;
- Quatro instâncias de *"Estudante"*;

•Quatro instâncias de *"Orientador"*;

•Quatro instâncias de *"Avaliador"*;

Para povoar o grafo com dados pode-se fazer uso de um *script* em linguagem *Cypher* contendo as definições tanto dos nós quanto das arestas, representando objetos do mundo real e seus relacionamentos. Para criar cada novo elemento do grafo, utiliza-se o comando *CREATE* da linguagem *Cypher*. Os primeiros elementos criados no BDG foram os nós, começando com a instância que representa o próprio evento.

A Figura 4.11 mostra o trecho do *script* que cria a instância de *"Evento"* no BDG considerada para este caso de estudo. O *script* exemplificado cria um nó rotulado como *"Evento"* cujas propriedades *"Título"* e *"Ano"* armazenam respectivamente os valores *"Fecicamp"* e *"2020"*. O nó criado é armazenado temporariamente em uma *variável* chamada *"eve"* e pode ser referenciado posteriormente no *script* para definir as associações com o evento criado.

```
// Cria o Evento "Fecicamp" no ano de 2020
CREATE (eve :Evento {Titulo: "Fecicamp", Ano: "2020"})
```

Figura 4.11. Código *Cypher* que Cria um Nó no BDG

Na sequência, são instanciados os quesitos de avaliação definidos para o evento. O trecho de *script* que cria estas instâncias é mostrado na Figura 4.12, que cria os nós referentes aos quesitos de avaliação. As propriedades consideradas para cada instância de *"Quesito"* são *"Peso"* e *"Descrição"*. Entretanto, estes nós não estão associados ao evento criado anteriormente. Neste caso, se faz necessário estabelecer uma relação entre o evento e cada um dos quesitos criados, informando que o evento *"define"* estes quesitos de avaliação.

```
// Cria os quesitos de avaliação do evento
CREATE (q1 :Quesito {Peso: 3, Descricao: "Apresentação Oral"})
CREATE (q2 :Quesito {Peso: 1, Descricao: "Banner"})
CREATE (q3 :Quesito {Peso: 1, Descricao: "Relevância"})
CREATE (q4 :Quesito {Peso: 2, Descricao: "Resumo"})
```

Figura 4.12. *Script* para Criação dos Quesitos de Avaliação

A Figura 4.13 mostra a definição de arestas entre o evento representado pela variável *"eve"* e cada um dos quesitos *"qi"*, com $i = \{1, 2, 3, 4\}$. Neste caso, cada aresta é

rotulada como *"Define"* e a orientação tem origem no evento e destino no quesito. A Figura 4.14 mostra o estado do grafo após as definições do evento e seus quesitos.

```
// Estabelece que o evento "define" seus quesitos
CREATE (eve)-[:Define]->(q1)
CREATE (eve)-[:Define]->(q2)
CREATE (eve)-[:Define]->(q3)
CREATE (eve)-[:Define]->(q4)
```

Figura 4.13. *Script* que Associa os Quesitos ao Evento Criado

A Figura 4.14 mostra os nós e arestas que representam o evento (grafado em marrom) e seus quesitos de avaliação (grafados em azul). Nota-se a aresta rotulada como *"Define"* ligando o evento a cada quesito de avaliação, com as setas apontando para os quesitos.

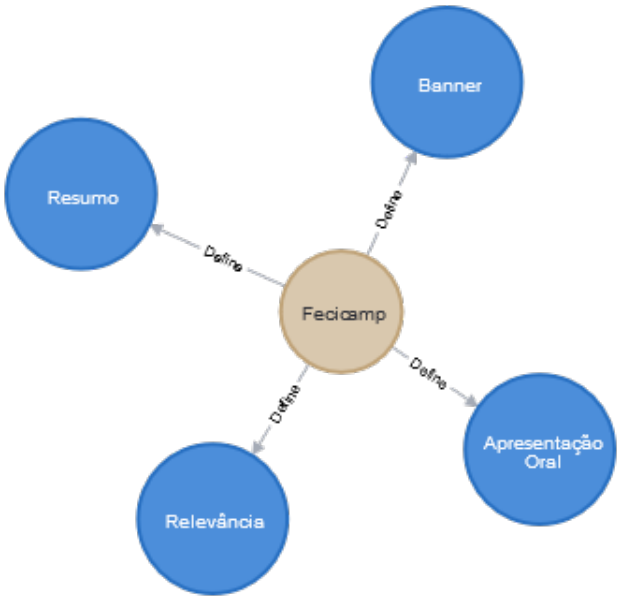

Figura 4.14. Evento e Quesitos na Instância do BDG

O próximo passo é instanciar as pessoas no BDG que devem lidar com os trabalhos de maneiras distintas. Conforme o modelo da Figura 4.9, estas pessoas dever ser agrupadas em subconjuntos disjuntos entre si de orientadores, estudantes e avaliadores. Portanto, neste caso, cada nó apresenta dois rótulos, sendo um de *"Pessoa"* e o segundo definindo o subtipo de *"Pessoa"* ao qual o nó se refere.

A Figura 4.15 mostra o *script* que cria pessoas no BDG. Cada novo nó criado pelo *script* possui dois rótulos para representar que estas instâncias representam objetos modelados no processo de especialização de *"Pessoa"* em *"Orientador"*, *"Estudante"* e *"Avalia-*

dor", conforme definido pelo modelo conceitual da Figura 4.1 e mapeado para o grafo de esquema da Figura 4.9.

```
// Cria os orientadores que participam do evento
CREATE (o1 :Pessoa :Orientador {Nome: "Alex Roberto"})
CREATE (o2 :Pessoa :Orientador {Nome: "Flavio Ribas"})
CREATE (o3 :Pessoa :Orientador {Nome: "Luciana Pereira"})
CREATE (o4 :Pessoa :Orientador {Nome: "Paulo Sergio"})
// Cria os estudantes
CREATE (e1 :Pessoa :Estudante {Nome: "Aline Alves", RA: "1912"})
CREATE (e2 :Pessoa :Estudante {Nome: "Breno Barros", RA: "8183"})
CREATE (e3 :Pessoa :Estudante {Nome: "Caio Coelho", RA: "9205"})
CREATE (e4 :Pessoa :Estudante {Nome: "Diana Dias", RA: "8119"})
// Cria os avaliadores
CREATE (a1 :Pessoa :Avaliador {Nome: "Andrew Bond", Max_avaliacoes: "3"})
CREATE (a2 :Pessoa :Avaliador {Nome: "James Potter", Max_avaliacoes: "3"})
CREATE (a3 :Pessoa :Avaliador {Nome: "Jane Warren", Max_avaliacoes: "3"})
CREATE (a4 :Pessoa :Avaliador {Nome: "Karl Newt", Max_avaliacoes: "3"})
```

Figura 4.15. Criando Instâncias de *"Pessoa"* no BDG

Uma vez criados os nós que representam as pessoas no banco de dados, chega a hora de instanciar os nós referentes aos trabalhos. Cada trabalho tem um conjunto básico de atributos para representar seu código interno, a categoria em que foi inscrito, o nível de ensino e seu título. A Figura 4.16 mostra como são instanciados os quatro trabalhos cadastrados no BDG. Cada nó é armazenado em uma variável ti onde $i = \{1, 2, 3, 4\}$ para referências futuras no *script*.

```
CREATE (t1 :Trabalho {Cod: "CET001", Categoria: "Ciências Exatas e da Terra",
    Nivel: "Medio", Titulo: "Motor Eletromagnético"})
CREATE (t2 :Trabalho {Cod: "CET002", Categoria: "Ciências Exatas e da Terra",
    Nivel: "Medio", Titulo: "Energia Eólica"})
CREATE (t3 :Trabalho {Cod: "CET003", Categoria: "Ciências Exatas e da Terra",
    Nivel: "Medio", Titulo: "Matemática Descomplicada"})
CREATE (t4 :Trabalho {Cod: "CET004", Categoria: "Ciências Exatas e da Terra",
    Nivel: "Medio", Titulo: "Fisica do Movimento"})
```

Figura 4.16. Criando Instâncias de *"Trabalho"* no BDG

Tão importante quanto instanciar os trabalhos é estabelecer as associações com o evento e com as pessoas. Estas associações definem o evento onde os trabalhos estão sendo organizados, quem são os alunos participantes de cada trabalho, quem são os orientadores e coorientadores e, por fim, estabelece quais avaliadores avaliam quais trabalhos. A Figura 4.17 mostra como são criadas as associações entre o evento e os trabalhos indicando que o evento criado a priori *"Organiza"* estes trabalhos

Além da conexão com o evento, os trabalhos estão ligados aos estudantes. A Figura 4.18 mostra o trecho do *script* que associa cada trabalho a seu grupo de estudantes.

```
CREATE (eve)-[:Organiza]->(t1)
CREATE (eve)-[:Organiza]->(t2)
CREATE (eve)-[:Organiza]->(t3)
CREATE (eve)-[:Organiza]->(t4)
```

Figura 4.17. Definindo a ligação entre os trabalhos e o evento

```
// Estabelece os estudantes
// que participam em cada trabalho
CREATE (e1)-[:Participa]->(t1)
CREATE (e2)-[:Participa]->(t1)
CREATE (e2)-[:Participa]->(t2)
CREATE (e3)-[:Participa]->(t2)
CREATE (e3)-[:Participa]->(t3)
CREATE (e4)-[:Participa]->(t3)
CREATE (e4)-[:Participa]->(t4)
CREATE (e1)-[:Participa]->(t4)
```

Figura 4.18. Definindo as participações dos estudantes nos trabalhos

O *script* mostrado pela Figura 4.18 define dois estudantes para cada trabalho criado. Entretanto como a participação dos estudantes, os trabalhos também são orientados (e coorientados) por pessoas que fazem parte do subconjunto de orientadores. Para estabelecer esta associação, basta executar o trecho do *script* conforme mostrado na Figura 4.19.

```
// Estabelece as orientações de cada trabalho
CREATE (o1)-[:Orienta {tipo: "Orientação"}]->(t1)
CREATE (o2)-[:Orienta {tipo: "Coorientação"}]->(t1)
CREATE (o3)-[:Orienta {tipo: "Orientação"}]->(t2)
CREATE (o4)-[:Orienta {tipo: "Coorientação"}]->(t2)
CREATE (o3)-[:Orienta {tipo: "Orientação"}]->(t3)
CREATE (o2)-[:Orienta {tipo: "Orientação"}]->(t4)
```

Figura 4.19. Estabelecendo as orientações em cada trabalho

O *script* da Figura 4.19 estabelece que os trabalhos *t1* e *t2* tem orientador e coorientador, enquanto que os trabalhos *t3* e *t4* tem apenas orientador. Após estas rodadas de *script*, já existe uma rede conectando o evento, os trabalhos, os orientadores e os estudantes. A Figura 4.20 mostra as conexões do trabalho *"CET001"*. Este trabalho está conectado aos estudantes *"Aline Alves"* e *"Breno Barros"*, é orientado por *"Alex Roberto"* e coorientado por *"Flavio Ribas"*. Por fim, está ligado ao evento *"Fecicamp"*. As conexões dos outros trabalhos têm um conjunto de conexões semelhantes a este.

Finalmente, as avaliações são instanciadas no BDG. Cada avaliação representa um relacionamento ternário entre instâncias de *"Trabalho"*, *"Quesito"* e *"Avaliador"*, e é repre-

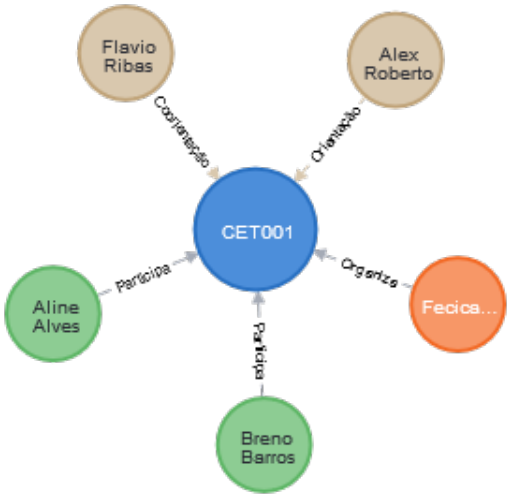

Figura 4.20. Instância do trabalho *"CET001"*

sentada como um nó rotulado como *"Avalia"*, que armazena o valor da nota dada por um avaliador para um determinado trabalho em um dado quesito. Neste exemplo, o evento define quatro quesitos, portanto é natural que um avaliador faça o lançamento de quatro notas para cada trabalho avaliado, sendo uma para cada quesito definido. A Figura 4.21 mostra o trecho do *script* que lança algumas avaliações no BDG.

```
// Cria algumas avaliações
CREATE (av1 :Avalia {Valor: 8.1})
CREATE (av2 :Avalia {Valor: 8.4})
CREATE (av3 :Avalia {Valor: 7.8})
CREATE (av4 :Avalia {Valor: 9.1})
CREATE (av5 :Avalia {Valor: 8.0})
CREATE (av6 :Avalia {Valor: 8.5})
CREATE (av7 :Avalia {Valor: 7.5})
CREATE (av8 :Avalia {Valor: 9.0})
```

Figura 4.21. Criação de nós *"Avalia"*

Entretanto, de acordo o grafo de esquema, cada nó rotulado como *"Avalia"* deve se conectar a uma instância de *"Trabalho"*, de *"Quesito"* e de *"Avaliador"* com arestas rotuladas como *"Participa de"*. A Figura 4.22 estabelece as associações entre cada nó *"Avalia"* com um nó *"Quesito"*, indicando qual quesito está pesando sobre cada nota.

De forma análoga ao passo anterior, cada nó *"Avalia"* deve se associar com um nó *"Avaliador"*, neste caso indicando qual o avaliador que lançou a nota no BDG. Estas associações são estabelecidas no *script* mostrado na Figura 4.23.

Por fim, cada nó *"Avaliação"* deve ser ligado a algum nó *"Trabalho"* no BDG, indicando

```
// Vincula cada avaliação a um
// determinado quesito
CREATE (q1)-[:Participa_de]->(av1)
CREATE (q2)-[:Participa_de]->(av2)
CREATE (q3)-[:Participa_de]->(av3)
CREATE (q4)-[:Participa_de]->(av4)
CREATE (q1)-[:Participa_de]->(av5)
CREATE (q2)-[:Participa_de]->(av6)
CREATE (q3)-[:Participa_de]->(av7)
CREATE (q4)-[:Participa_de]->(av8)
```

Figura 4.22. Estabelecendo associações entre *"Quesito"* e *"Avalia"*

```
// Associa cada avaliação a um avaliador
CREATE (a1)-[:Participa_de]->(av1)
CREATE (a1)-[:Participa_de]->(av2)
CREATE (a1)-[:Participa_de]->(av3)
CREATE (a1)-[:Participa_de]->(av4)
CREATE (a2)-[:Participa_de]->(av5)
CREATE (a2)-[:Participa_de]->(av6)
CREATE (a2)-[:Participa_de]->(av7)
CREATE (a2)-[:Participa_de]->(av8)
```

Figura 4.23. Estabelecendo associações entre *"Avaliador"* e *"Avalia"*

para qual trabalho foi lançada cada nota. O *script* mostrado na 4.24 demonstra como são criadas as arestas entre instâncias de *"Trabalho"* e de *"Avalia"*.

```
//Conecta uma avaliação a um trabalho
CREATE (t1)-[:Participa_de]->(av1)
CREATE (t1)-[:Participa_de]->(av2)
CREATE (t1)-[:Participa_de]->(av3)
CREATE (t1)-[:Participa_de]->(av4)
CREATE (t2)-[:Participa_de]->(av5)
CREATE (t2)-[:Participa_de]->(av6)
CREATE (t2)-[:Participa_de]->(av7)
CREATE (t2)-[:Participa_de]->(av8)
```

Figura 4.24. Estabelecendo associações entre *"Trabalho"* e *"Avalia"*

Com a criação das avaliações e suas conexões, já é possível ter uma visão geral sobre como os trabalhos são organizados no BDG, mantendo associações diretas com o evento, com estudantes, orientadores e avaliações, que, por sua vez, estão conectadas aos quesitos de avaliação e aos avaliadores. A Figura 4.25 mostra o trabalho *"CET001"* e suas associações. Este trabalho está associado aos estudantes (nós verdes) e orientadores (nós marrons). Neste caso, apenas o avaliador *"Andrew Bond"* (nó rosa) avaliou o trabalho. Cada quesito (em azul claro) tem uma nota (em vermelho) associada ao trabalho e dada pelo avaliador.

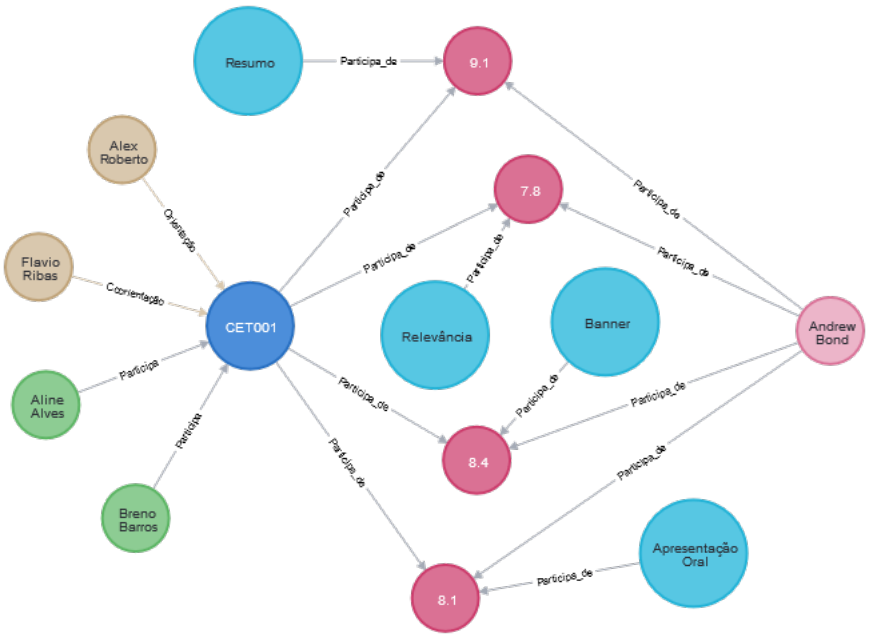

Figura 4.25. Avaliações do trabalho *"CET001"*

Mesmo um estudo de caso com um número pequeno de nós evidencia que o comportamento do BDG pode levar a um grande número de associações. Potencialmente, cada instância de *"Trabalho"* pode ter um grande número de arestas, especialmente se aumentar o número de avaliadores. Além disso, um evento pode ter um número considerável de trabalhos a serem organizados. Por fim, o BDG pode dar suporte a múltiplos eventos, cada um com um conjunto de quesitos e de trabalhos distintos. Estudantes podem participar de vários trabalhos em eventos distintos, assim como orientares podem orientar e coorientar qualquer número de trabalhos e avaliadores podem avaliar trabalhos em uma infinidade de eventos. Essas características reforçam a ideia de que um BDG pode atender a demanda em potencial que pode ser gerada a partir deste caso de uso, especialmente quando se considera que o volume de dados aumenta com o tempo, pois prioriza principalmente as operações de travessia no grafo de dados, que em um SGBD de grafos não registra perda significativa de desempenho, independentemente da quantidade de nós e arestas do BDG.

Por outro lado, a maioria dos SGBDs de grafos, por adotarem uma política *schema-free*, não implementa um grande conjunto de regras de integridade. A exemplo do *Neo4j*, somente restrições de unicidade e obrigatoriedade de propriedades podem ser definidas para os nós e arestas do grafo de dados. Definições de cardinalidade em relacionamentos e de participação/disjunção em modelagens superclasse/subclasse geralmente são deixadas

de lado em favor de uma maior liberdade, principalmente em projetos onde as regras de negócio tendem a mudar frequentemente com o passar do tempo. Entretanto, estabelecer um conjunto de restrições de integridade mais abrangente pode trazer garantias de um BDG mais consistente, principalmente em sistemas altamente conectados e com tendência a poucas mudanças ao longo do tempo.

4.4.1. Definições das Restrições de Integridade no BDG

Apesar das vantagens dos sistemas *NoSQL*, em especial o modelo de grafos, em relação aos bancos de dados relacionais quanto ao desempenho, flexibilidade e escalabilidade no tratamento de dados modelados como grafos, a implementação de restrições de integridade ainda representa um grande desafio no projeto destes modelos de dados. A grande maioria dos SGBDs de grafos não implementa, ou implementa um conjunto muito reduzido de restrições de integridade com a justificativa de manter a flexibilidade e a capacidade do banco de dados de evoluir livremente e gradativamente ao longo do tempo (Šestak et al.; 2016).

Entretanto, um projeto lógico de banco de dados que defina restrições de integridade, e não somente sua estrutura, é de extrema importância para manter a consistência dos dados armazenados. Além disso, Comyn-Wattiau & Akoka (2017) destacam que mesmo em se tratando de modelagem de dados *NoSQL* de grafos, um esquema lógico bem definido pode facilitar tanto o projeto do banco de dados quanto a implementação de consultas, principalmente as mais complexas, no banco de dados.

Em um projeto lógico de BDG, muitas restrições de integridade, que são inerentes ao modelo conceitual ER e suas variações, devem ser explicitamente definidas, uma vez que muitas destas regras não são inerentes ao modelo de grafos. Considerando o *Neo4j* como um exemplo prático, constata-se que este SGBD de grafos dá suporte a um conjunto reduzido de restrições de integridade, tratando apenas de restrições impostas a propriedades de nós ou de arestas arestas com rótulos específicos (Neo4j; 2020c). É possível garantir a existência de uma determinada propriedade em um nó/aresta de um determinado rótulo, similar a definição de atributos de preenchimento obrigatório (*NOT NULL*) em tabelas de bancos de dados relacionais. O *Neo4j* também dá suporte a restrição de unicidade (*UNIQUE*) de valores em propriedades de nó com um rótulo específico. Por fim, permite

definir um conjunto de propriedades chave em um nó, tal qual definir um conjunto de atributos como chave primária em relações.

No contexto de um BDG, onde os relacionamentos são representandos por arestas direcionadas, cada nó mantém duas listas distintas de referências: a primeira mantém o conjunto de todas as arestas que partem do nó (arestas enviadas); a segunda se refere ao conjunto de arestas que chegam ao nó (arestas recebidas). Estas são chamadas de listas de adjacências e são as principais responsáveis pela agilidade das consultas aos dados do grafo (Robinson et al.; 2013). Entretanto, nenhuma restrição é imposta às listas de adjacências no *Neo4j* para garantir que as restrições de cardinalidade sejam atendidas.

Sejam os nós α e β em uma instância do BDG, o qual representam respectivamente ocorrências das entidades A e B do modelo conceitual. Considerando que haja um relacionamento R entre A e B onde A tem participação total em R, então é necessário garantir que toda ocorrência α de A deve manter em suas listas de adjacências pelo menos uma aresta ϵ, representante do relacionamento R. Além disso, caso a cardinalidade máxima de B seja igual a *1*, então será necessário garantir que a aresta ϵ seja a única representante de R nas listas de adjacências de α, já que isto indica que α deve se conectar com no máximo uma ocorrência de β.

Usando o exemplo do relacionamento *"Organiza"*, entre *"Evento"* e *"Trabalho"*, nota-se que a participação de *"Trabalho"* no relacionamento é total e que, ao mesmo tempo, a cardinalidade máxima de *"Evento"* é igual a *1*. Para representar esta situação no BDG, é necessário garantir que qualquer nó rotulado como *"Trabalho"* deve possuir *uma, e somente uma* aresta rotulada como *"Organiza"* em suas listas de adjacências o conectando a algum nó rotulado como *"Evento"*. Isto garante que todo trabalho é organizado obrigatoriamente por um único evento, garantindo a coerência com o modelo relacional da Figura 4.1.

Por outro lado, a entidade *"Evento"*, mantém participação parcial no relacionamento enquanto que a cardinalidade máxima de *"Trabalho"* é igual a *N*. Isto denota que um evento pode se conectar a qualquer quantidade de trabalhos (*0* ou *muitos*) e, portanto, nenhuma restrição é necessária por parte do nó *"Evento"* quanto à aresta *"Organiza"*.

Assim sendo, estabelecer os números mínimo e máximo de arestas permitidas nas listas

de adjacências de um nó pode garantir que as restrições de cardinalidade sejam atendidas pelas instâncias do BDG.

Outras regras que não são atendidas pelo *Neo4j* são as restrições definidas em processos de especialização para garantir participação total da superclasse (completude) e também para garantir a disjunção entre as subclasses. Entretanto, uma característica bastante interessante do *Neo4j* é que ele permite que os nós possam assumir mais de um rótulo nas instâncias do BDG (Robinson et al.; 2013). Isso permite que uma ocorrência de um nó pertença a mais de uma entidade simultaneamente.

Neste caso, garantir a participação total de uma superclasse em um processo de especialização significa garantir que todas as ocorrências da superclasse nas instâncias do BDG assumam pelo menos dois rótulos: o primeiro rótulo representa a própria superclasse; o segundo rótulo e os demais representam uma ou mais subclasses da especialização. Já para os casos em que seja necessário garantir a disjunção entre as subclasses, deve-se definir os rótulos destas como divergentes no grafo de esquema, a fim de que qualquer instância da superclasse que se especializar o faça em apenas uma das subclasses.

No exemplo da Figura 4.25, as pessoas *"Alex Roberto"* e *"Flavio Ribas"* representam ocorrências de *"Pessoa"* (superclasse) e de *"Orientador"* (subclasse) em um processo de especialização. Nesta mesma especialização foi definida a subclasse *"Estudante"*, cujas ocorrências são exemplificadas pelo nós *"Aline Alves"* e *"Breno Barros"*. Fechando este processo de especialização, ainda existe a subclasse *"Avaliador"*, representada por *"Andrew Bond"*. Fazendo uma análise do *script* mostrado na Figura 4.15, é possível constatar que estas pessoas foram instanciadas no BDG com dois rótulos, sendo o primeiro rótulo definido como *"Pessoa"*. Para *"Alex Roberto"* e *"Flavio Ribas"*, o segundo rótulo é definido como *"Orientador"*, denotando que estes orientadores são, sobretudo, pessoas. No caso de *"Aline Alves"* e *"Breno Barros"*, o segundo rótulo é definido como *"Estudante"*. Por fim, para a pessoa de *"Andrew Bond"*, o segundo rótulo foi definido como *"Avaliador"*. Também é perceptível que neste mesmo *script* todas as pessoas assumem apenas um rótulo de subclasse, já que a especialização de *"Pessoa"* em *"Orientador"*/*"Estudante"*/*"Avaliador"* é definida como uma disjunção.

Como este é o único processo de especialização do modelo e a superclasse não tem

participação total no processo, então seria possível instanciar um nó apenas com o rótulo de *"Pessoa"*, representando alguém no sistema que não se enquadra como orientador, estudante e nem avaliador.

Por fim, é possível constatar que SGBDs de grafos ainda precisam evoluir seu conjunto de restrições de integridade, para garantir que os dados contidos nos BDGs estejam coerentes com as regras impostas desde o modelo conceitual de dados. O SGBD *Neo4j* oferece suporte a um banco de dados de grafo nativo, que efetivamente lida com os dados em um formato de grafo (Robinson et al.; 2013), permitindo um desempenho bastante satisfatório para conjuntos massivos e altamente conectados de dados. Além disso, a linguagem *Cypher*, desenvolvida para este SGBD, possibilita consultas e manipulação de dados de forma bastante intuitiva, o que facilita o desenvolvimento e a manutenção do BDG. Entretanto, manter a integridade dos dados em um grafo ainda é uma tarefa que está além deste SGBD e recai sobre os aplicativos que possam acessar este BDG e também sobre seus desenvolvedores, o que pode gerar complexidade extra em seu desenvolvimento.

Capítulo 5

Implementação do Processo de Mapeamento

Neste capítulo serão abordados os detalhes sobre a implementação dos algoritmos de mapeamento propostos nesta dissertação em um protótipo de *software* desenvolvido em linguagem *Java* e que toma como entrada um arquivo estruturado em *XML* (*eXtensible Markup Language*), o qual representa um modelo conceitual com os elementos discutidos na Seção 2.1. Como saída, é gerado um grafo de esquema, também representado em XML.

5.1. Representação em XML

Tanto o esquema conceitual quanto o esquema lógico propostos são representados como modelos estruturados em *XML*. A linguagem *XML* é uma ferramenta simples e eficaz que pode ser utilizada para armazenar e transportar dados entre aplicações. Além disso, documentos *XML* possuem uma estrutura hierárquica que pode ser bem compreendida tanto por humanos como por máquinas (W3Schools; 2020). A escolha da linguagem *XML* para representação dos esquemas de entrada e saída para o mapeamento se deve por ser uma tecnologia consolidada, de fácil compreensão e que, portanto, facilita na geração manual do esquema conceitual a ser mapeado. Por outro lado, o modelo gerado automaticamente pelo *software* também pode ser verificado manualmente em caso de necessidade de correções tanto no próprio grafo quanto na programação.

Os dados em um arquivo *XML* são armazenados em elementos estruturais da linguagem, delimitados por *tags* e responsáveis por criar uma hierarquia bem definida dentro do documento. Para delimitar um elemento são usados uma *tag* de abertura e uma *tag* de fechamento e entre elas é definido o conteúdo do elemento, que por sua vez pode ser composto por texto simples ou por outra estrutura de elementos *XML*.

O exemplo abaixo demonstra uma estrutura *XML* que define alguns elementos. O elemento raiz deste documento é o elemento *diagrama*, cujo conteúdo é dado por um elemento *entidade* contendo três elementos *atributo*. O atributo *tipo* do elemento diagrama indica o tipo do modelo conceitual utilizado. Trabalhos futuros podem explorar outros tipos de modelo conceitual como UML. O elemento *entidade* conta com os atributos *id* e *nome*. Considera-se que o *id* é único para cada elemento do documento. O terceiro *atributo* da entidade, nomeado como *Telefones*, é definido como *multivalorado* e *composto* por dois componentes: *Tipo* e *Numero*.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <diagrama tipo="EER">
3    <entidade nome="Pessoa">
4      <atributo>Nome</atributo>
5      <atributo>Email</atributo>
6      <atributo multivalorado="true" nome="Telefones">
7        <componente>Tipo</componente>
8        <componente>Numero</componente>
9      </atributo>
10   </entidade>
11 </diagrama>
```

Código 5.1. Exemplo de Estrutura XML

Pode-se perceber que os documentos XML mantém seus elementos organizados de maneira hierárquica em forma de árvore. A árvore que representa a estrutura exemplificada pelo Código 5.1 é mostrada na Figura 5.1:

Nesta figura são mostradas a hierarquia e as relações *pai/filho* entre os elementos XML definidos anteriormente.

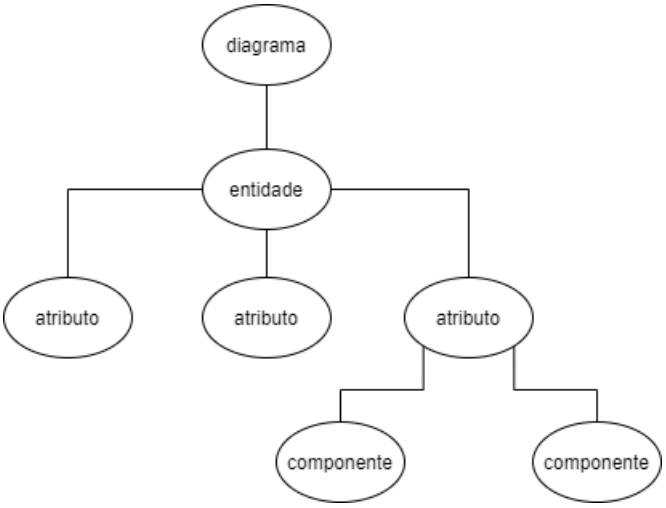

Figura 5.1. Árvore da Estrutura XML 5.1

5.2. Representação do Modelo Conceitual em XML

O modelo conceitual proposto neste trabalho pode ser entendido como um diagrama EER, que pode ser representado hierarquicamente como uma estrutura XML. Pode-se considerar que os elementos da modelagem EER são armazenados dentro de um *container* único, o qual foi definido como sendo o próprio diagrama. Portanto, o diagrama pode ser estruturado em XML como o elemento raiz do documento. Como existem vários tipos de diagrama que podem ser utilizados como modelos conceituais para bancos de dados, estabelece-se o atributo tipo para definir qual é o tipo de diagrama que deve ser considerado para o mapeamento. Embora neste momento o único tipo de diagrama utilizado pelo software seja o que representa a modelagem EER, outros tipos de modelagem conceitual podem ser implementados no futuro.

Para ajudar a montar uma estrutura que possa representar o modelo conceitual, é necessária uma forma de validação de documento XML, para que este esteja de acordo com a sintaxe e as definições propostas nesta dissertação. Portanto, os modelos conceituais devem respeitar regras definidas em um arquivo de definição de esquema XML (*XML Schema Definition - XSD*). O Código 5.2 representa o conjunto de definições XML para a representação do modelo conceitual.

As linhas *106* até *117* do Código 5.2 definem o elemento “diagrama” como uma coleção de várias *entidades*, vários *relacionamentos* e várias *especializações*. Também define que

o diagrama deve ter um atributo chamado “tipo”, que indica o tipo do diagrama representado pelo modelo. Neste caso, o atributo “tipo” deve ser preenchido com o valor “EER”.

As entidades são definidas nas linhas *12* a *17* como um conjunto de elementos denominados “atributo”. Por sua vez, nas linhas *4* a *10*, os elementos que representam os atributos podem ser estruturados como um conjunto de componentes (atributos multivalorados), ou como um elemento simples (atributos monovalorados).

Os relacionamentos são definidos nas linhas *55* até *74*. Cada relacionamento no modelo possui um par de elementos “origem” e “destino” ou um conjunto de pelo menos três elementos denominados “participante”. Estes elementos fazem referências a entidades que participam do relacionamento. Além disso, o relacionamento pode ter um conjunto de elementos “atributo”. Como atributo XML, um relacionamento deve possuir um nome (obrigatório) e um tipo, sendo que este último indica se o relacionamento é binário ou n-ário.

Os elementos que representam as especializações do modelo conceitual são tipificados nas linhas *91* a *104*. É definido que cada elemento “especialização” deve ter um, e somente um, elemento “superclasse” e pelo menos um elemento “subclasse”. Também é definido que os elementos “especialização” possuem um atributo denominado “tipo”, que indica se a especialização é uma disjunção (valor *default*) ou uma sobreposição.

```
1  <?xml version="1.0"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3
4  <xs:complexType name="tAtributo" mixed="true">
5    <xs:sequence>
6      <xs:element name="componente" minOccurs="0" maxOccurs="unbounded"
       type="xs:string"/>
7    </xs:sequence>
8    <xs:attribute name="multivalorado" type="xs:boolean" default="false"/
       >
9    <xs:attribute name="nome" type="xs:string"/>
10 </xs:complexType>
11
12 <xs:complexType name="tEntidade">
13   <xs:sequence>
```

```
14      <xs:element name="atributo" type="tAtributo" maxOccurs="unbounded"/
    >
15    </xs:sequence>
16    <xs:attribute name="nome" type="xs:string" use="required"/>
17  </xs:complexType>
18
19  <xs:complexType name="tEntidadeParticipante">
20    <xs:simpleContent>
21      <xs:extension base="xs:string">
22        <xs:attribute name="participacao">
23          <xs:simpleType>
24            <xs:restriction base="xs:string">
25              <xs:enumeration value="Total"/>
26              <xs:enumeration value="Parcial"/>
27            </xs:restriction>
28          </xs:simpleType>
29        </xs:attribute>
30        <xs:attribute name="cardinalidade">
31          <xs:simpleType>
32            <xs:restriction base="xs:string">
33              <xs:enumeration value="1"/>
34              <xs:enumeration value="N"/>
35            </xs:restriction>
36          </xs:simpleType>
37        </xs:attribute>
38      </xs:extension>
39    </xs:simpleContent>
40  </xs:complexType>
41
42  <xs:group name="origemDestino">
43    <xs:sequence>
44      <xs:element name="origem" type="tEntidadeParticipante"/>
45    <xs:element name="destino" type="tEntidadeParticipante"/>
46    </xs:sequence>
47  </xs:group>
48
49  <xs:group name="participantes">
```

```
50   <xs:sequence>
51     <xs:element name="participante" type="tEntidadeParticipante"
     minOccurs="3" maxOccurs="unbounded"/>
52   </xs:sequence>
53 </xs:group>
54
55 <xs:complexType name="tRelacionamento">
56   <xs:sequence>
57   <xs:choice>
58     <xs:group ref="origemDestino"/>
59     <xs:group ref="participantes"/>
60   </xs:choice>
61
62     <xs:element name="atributo" type="tAtributo" minOccurs="0"
     maxOccurs="unbounded"/>
63   </xs:sequence>
64
65   <xs:attribute name="nome" type="xs:string" use="required"/>
66   <xs:attribute name="tipo" default="Binario">
67   <xs:simpleType>
68      <xs:restriction base="xs:string">
69        <xs:enumeration value="Binario"/>
70        <xs:enumeration value="Enario"/>
71      </xs:restriction>
72   </xs:simpleType>
73   </xs:attribute>
74 </xs:complexType>
75
76 <xs:complexType name="tSuperclasse">
77   <xs:simpleContent>
78     <xs:extension base="xs:string">
79       <xs:attribute name="participacao" default="Total"/>
80               <xs:simpleType>
81                  <xs:restriction base="xs:string">
82                    <xs:enumeration value="Total"/>
83                    <xs:enumeration value="Parcial"/>
84                  </xs:restriction>
```

```
85                </xs:simpleType>
86        </xs:attribute>
87      </xs:extension>
88    </xs:simpleContent>
89  </xs:complexType>
90
91  <xs:complexType name="tEspecializacao">
92    <xs:sequence>
93      <xs:element name="superclasse" type="tSuperclasse" minOccurs="1"
      maxOccurs="1"/>
94      <xs:element name="subclasse" type="xs:string" minOccurs="1"
      maxOccurs="unbounded"/>
95    </xs:sequence>
96      <xs:attribute name="tipo" default="Disjuncao">
97    <xs:simpleType>
98        <xs:restriction base="xs:string">
99          <xs:enumeration value="Disjuncao"/>
100         <xs:enumeration value="Sobreposicao"/>
101       </xs:restriction>
102   </xs:simpleType>
103     </xs:attribute>
104 </xs:complexType>
105
106 <xs:element name="diagrama">
107   <xs:complexType>
108     <xs:sequence>
109       <xs:element name="entidade" type="tEntidade" minOccurs="0"
      maxOccurs="unbounded"/>
110
111     <xs:element name="relacionamento" type="tRelacionamento" minOccurs=
      "0" maxOccurs="unbounded"/>
112
113       <xs:element name="especializacao" type="tEspecializacao"
      minOccurs="0" maxOccurs="unbounded"/>
114     </xs:sequence>
115     <xs:attribute name="tipo" type="xs:string" use="required"/>
116   </xs:complexType>
```

```
117 </dataset_ref>
118 </xs:schema>
```

Código 5.2. Definição de Esquema XML do Modelo Conceitual

O exemplo codificado em Código 5.1 mostra o elemento *diagrama*, do tipo *EER*, o qual contém uma única *entidade* nomeada como *Pessoa* e que, por sua vez, contém três atributos, sendo os dois primeiros definidos como simples e monovalorados e o terceiro como sendo composto e multivalorado. Vale ressaltar que o Código 5.1 é validado pelas definições de esquema XML do Código 5.2.

5.3. Representação do Grafo de Esquema em XML

Assim como o modelo conceitual, o modelo lógico de grafos proposto também pode ser representado com uma estrutura hierárquica em um arquivo XML. Estruturalmente, um grafo pode ser organizado como um conjunto de nós, onde cada nó mantém listas de arestas que referenciam seus nós adjacentes. O código 5.3 implementa as definições de esquema XML para o grafo de esquema.

```
1 <?xml version="1.0"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3
4 <xs:complexType name="tArestaRecebida">
5   <xs:sequence>
6   <xs:element name="origem" type="xs:string" minOccurs="1" maxOccurs="1
    "/>
7   </xs:sequence>
8   <xs:attribute name="obrigatoria" default="false" type="xs:boolean"/>
9   <xs:attribute name="unica" default="false" type="xs:boolean"/>
10  <xs:attribute name="rotulo" type="xs:string" use="required"/>
11 </xs:complexType>
12
13 <xs:complexType name="tArestaEnviada">
14  <xs:sequence>
15  <xs:element name="destino" type="xs:string" minOccurs="1" maxOccurs="
    unbounded"/>
16  </xs:sequence>
17  <xs:attribute name="obrigatoria" default="false" type="xs:boolean"/>
```

```
18   <xs:attribute name="unica" default="false" type="xs:boolean"/>
19   <xs:attribute name="rotulo" type="xs:string" use="required"/>
20 </xs:complexType>
21
22 <xs:complexType name="tListaAdjacenciaRecebidas">
23   <xs:sequence>
24     <xs:element name="aresta" type="tArestaRecebida" maxOccurs="
     unbounded"/>
25   </xs:sequence>
26 </xs:complexType>
27
28 <xs:complexType name="tListaAdjacenciaEnviadas">
29   <xs:sequence>
30     <xs:element name="aresta" type="tArestaEnviada" maxOccurs="
     unbounded"/>
31   </xs:sequence>
32 </xs:complexType>
33
34 <xs:complexType name="tNo">
35   <xs:sequence>
36     <xs:element name="propriedade" type="xs:string" minOccurs="0"
     maxOccurs="unbounded"/>
37     <xs:element name="recebidas" type="tListaAdjacenciaRecebidas"
     minOccurs="0"/>
38     <xs:element name="enviadas" type="tListaAdjacenciaEnviadas"
     minOccurs="0"/>
39   </xs:sequence>
40   <xs:attribute name="rotulo" use="required" type="xs:string"/>
41 </xs:complexType>
42
43 <xs:element name="grafo">
44   <xs:complexType>
45     <xs:sequence>
46       <xs:element name="no" type="tNo" maxOccurs="unbounded"/>
47     </xs:sequence>
48   </xs:complexType>
49 </xs:element>
```

```
50
51 </xs:schema>
```

Código 5.3. Definição de Esquema XML do Modelo de Grafos

Segundo as definições do Código 5.3, um nó admite elementos *propriedade*, os quais representam os atributos mapeados a partir do modelo conceitual. Também admitem duas listas de adjacências, onde a primeira organiza todas as arestas que chegam até o nó (arestas *recebidas*), enquanto que a segunda mantém registros de todas a arestas que partem do nó (arestas *enviadas*).

Todas as arestas contêm informações sobre cardinalidade e participação relativas ao nó em que são referenciadas. Assim sendo, caso um nó α represente uma entidade e que tem participação total em um relacionamento r, então é a aresta θ, que representa r, deve ser referenciada por α de tal forma que θ seja *obrigatória* em qualquer ocorrência de α no BDG. Caso a cardinalidade máxima de e no relacionamento r seja igual a *1*, então a aresta θ deverá ser marcada como única em α.

A Figura 5.2 representa o mapeamento para o grafo de esquema do modelo conceitual mostrado na Figura 2.5. Neste mapeamento, as entidades "*Funcionário*" é "*Dependente*" são mapeadas como nós conectados com uma aresta rotulada como "*Tem*". O nó "*Telefone*" representa o atributo multivalorado "*Telefones*" de "*Funcionário*".

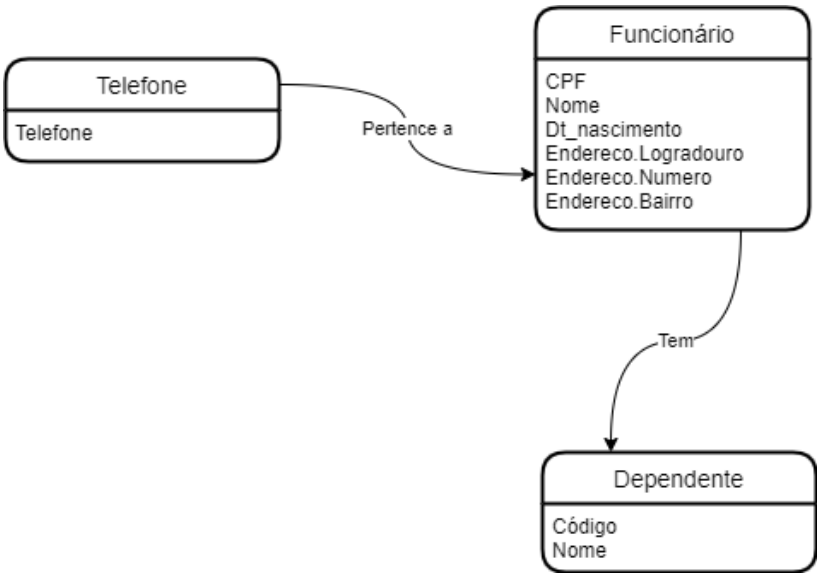

Figura 5.2. Grafo de Esquema do Modelo da Figura 2.5

O Código 5.4 representa o grafo da Figura 5.2 em XML. O grafo de esquema é delimitado na estrutura XML pelo elemento *grafo*, que serve de container para os três nós do grafo da Figura 5.2. O trecho de código que vai da *linha 3* até a *linha 20* representa a estrutura XML do nó rotulado como "*Funcionário*", que possui um conjunto de seis elementos *propriedade* e duas listas de adjacências.

A primeira lista de adjacências de "*Funcionário*" está delimitada pelo elemento *recebidas* e contém uma referência a aresta "*Pertence a*" que se origina no nó "*Telefone*". Neste caso, qualquer aresta cujo destino seja o nó "*Funcionário*" deve ser estruturada dentro do elemento *recebidas* deste nó. Para estas arestas deve ser marcado no nome do nó de *origem* da aresta.

A segunda lista de adjacências de "*Funcionário*" referencia todas as arestas que partem de "*Funcionário*" em direção a algum outro nó de destino. No exemplo do Código 5.4, a única aresta que parte de "*Funcionário*" é a aresta "*Tem*", que segue em direção ao nó rotulado como "*Dependente*". Neste caso, cada aresta deve manter uma referência ao seu nó de *destino*.

No nó "*Telefone*", definido entre as linha *22* e *31*, a aresta "*Pertence a*" é referenciada dentro da lista de adjacências de arestas *enviadas* deste nó. Esta aresta é marcada como *obrigatória* e *única* já que cada ocorrência de "*Telefone*" deve pertencer a uma única ocorrência de "*Funcionário*". O mesmo ocorre com o nó "*Dependente*" nas linhas *33* a *43*.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <grafo>
3    <no rotulo="Funcionario">
4      <propriedade>CPF</propriedade>
5      <propriedade>Nome</propriedade>
6      <propriedade>Dt_nascimento</propriedade>
7      <propriedade>Endereco.Logradouro</propriedade>
8      <propriedade>Endereco.Numero</propriedade>
9      <propriedade>Endereco.Bairro</propriedade>
10     <recebidas>
11         <aresta rotulo="Pertence a">
12             <origem>Telefone</origem>
```

```
13          </aresta>
14      </recebidas>
15      <enviadas>
16          <aresta rotulo="Tem">
17              <destino>Dependente</destino>
18          </aresta>
19      </enviadas>
20    </no>
21
22    <no rotulo="Telefone">
23      <propriedade>Telefone</propriedade>
24      <enviadas>
25          <aresta rotulo="Pertence a" obrigatoria="true" unica="true">
26              <destino>
27                Funcionario
28              </destino>
29          </aresta>
30      </enviadas>
31    </no>
32
33    <no rotulo="Dependente">
34      <propriedade>Codigo</propriedade>
35      <propriedade>Nome</propriedade>
36      <recebidas>
37          <aresta rotulo="Tem" obrigatoria="true" unica="true">
38              <origem>
39                Funcionario
40              </origem>
41          </aresta>
42      </recebidas>
43    </no>
44  </grafo>
```

Código 5.4. Estrutura XML do grafo da Figura 5.2

5.3.1. Restrições de Integridade em Listas de Adjacências

As listas de adjacências do grafo de esquema devem armazenar as restrições de participação e de cardinalidade máxima de seu nó proprietário em cada aresta listada.

No exemplo do Código 5.4, o nó rotulado como "*Dependente*" está conectado a "*Funcionário*" com uma aresta "*Tem*". Esta aresta é registrada na lista de *enviadas* de "*Funcionário*" e na lista de *recebidas* de "*Dependente*". O modelo conceitual da Figura 2.5 indica que:

- Um "*Funcionário*" pode ser responsável por *zero* ou *vários* dependentes;
- Cada "*Dependente*" está sob a responsabilidade de *um, e somente um* funcionário.

Para este caso a aresta "*Tem*" deve registrar restrições de participação e cardinalidade máxima em sua referência dentro de "*Dependente*" a fim de garantir a validade da segunda regra listada acima. Percebe-se também que, por parte de "*Funcionário*" não é necessária nenhuma restrição, uma vez que funcionários podem se relacionar (ou não) livremente com quantos dependentes forem necessários.

Para cada instância de "*Dependente*" é *obrigatória* a conexão com algum "*Funcionário*", a fim de respeitar a participação *total* de "*Dependente*" no relacionamento "*Tem*" com "*Funcionário*". Também é possível concluir que esta conexão é *única*, uma vez que a cardinalidade máxima de "*Dependente*" neste mesmo relacionamento é igual a *1*.

Para explicitar este regramento no grafo de esquema, na aresta *"Tem*", definida no código XML nas linhas *37* a *41*, também são definidos os atributos XML "*obrigatoria*" e "*unica*", indicando que para cada instância de "*Dependente*" é obrigatório que uma única ocorrência da aresta "*Tem*" o conecte a uma instância de "*Funcionário*".

Portanto, quando um nó representa uma entidade com *participação total* em um relacionamento, a aresta que representa este relacionamento dentro do nó deve ser marcada com o atributo "*obrigatoria*" com o valor "*true*". Em casos de participação parcial, não é necessário definir este atributo na estrutura da aresta.

Já os nós que representam entidades com cardinalidade máxima igual a *1* em um determinado relacionamento devem acrescentar o atributo "*unica*" com valor "*true*" na definição da aresta que representa o relacionamento, a fim de garantir que as instâncias do nó no

BDG se conectem a no máximo uma instância do nó parceiro no relacionamento. Para os casos em que a cardinalidade máxima é definida como *N* no modelo relacional, este atributo não é definido na aresta em XML, ou pode ser definido com o valor "*false*".

5.3.2. Representação de Especializações em XML

Ao considerar o modelo conceitual EER faz-se necessário o tratamento dos processos de especialização que possam ser definidos da modelagem. Para o caso de especializações simples, em que é definida uma subclasse para representar um subconjunto dos elementos da superclasse, basta definir uma aresta rotulada como "*É um*" na estrutura XML da subclasse em sua lista de arestas enviadas. Da mesma maneira, cria-se uma referência na superclasse, em sua lista de arestas recebidas. Neste caso, nenhuma restrição se faz necessária por parte das entidades envolvidas.

Entretanto, para os casos de especializações que envolvam duas ou mais subclasses é necessário representar estas especializações como *hiperarestas*. A representação de uma *hiperaresta* na estrutura XML do grafo de esquema é bastante semelhante a representação de uma aresta binária comum, a qual é referenciada duas vezes, sendo uma pelo nó de origem em sua lista de arestas *enviadas* e outra pelo nó de destino em sua lista de arestas *recebidas*. *Hiperarestas* representantes das especializações são referenciadas pela superclasse em sua lista de arestas enviadas, já que a superclasse está na *cauda* da *hiperaresta* de especialização. De forma semelhante, ela é referenciada por cada uma das subclasses envolvidas, em suas respectivas listas de arestas *recebidas*, já que as subclasses estão na *cabeça* da *hiperaresta*.

O Código 5.5 mostra os exemplos de estrutura XML dos grafos da Figura 3.7 e 3.8, que exemplificam respectivamente uma especialização simples de "*Funcionário*" em "*Gerente*" e uma disjunção total de "*Funcionário*" em "*Horista/Mensal*". A aresta que indica que um *gerente é um funcionário* foi referenciada por ambos os nós "*Funcionário*" e "*Gerente*".

Já a especialização de "*Funcionário*" em "*Horista*" e "*Mensal*" foi feita com uma *hiperaresta* definida na lista de arestas *enviadas* a partir de "*Funcionário*". Percebe-se que esta aresta possui mais de um destino, evidenciando se tratar de uma *hiperaresta*.

```
1 <?xml version="1.0" encoding="utf-8"?>
```

```
<grafo>
  <no rotulo="Funcionario">
    <propriedade>CPF</propriedade>
    <propriedade>Nome</propriedade>
    <propriedade>Dt_nascimento</propriedade>
    <propriedade>Endereco.Logradouro</propriedade>
    <propriedade>Endereco.Numero</propriedade>
    <propriedade>Endereco.Bairro</propriedade>
    <recebidas>
        <aresta rotulo="E um">
            <origem>Gerente</origem>
        </aresta>
    </recebidas>
    <enviadas>
        <aresta rotulo="Disjuncao Total" participacao="Total" tipo="
    Disjuncao">
            <destino>Mensal</destino>
            <destino>Horista</destino>
        </aresta>
    </enviadas>
  </no>

  <no rotulo="Gerente">
    <propriedade>Dt_inicio</propriedade>
    <enviadas>
        <aresta rotulo="E um">
            <destino>Funcionario</destino>
        </aresta>
    </enviadas>
  </no>

  <no rotulo="Mensal">
    <propriedade>Salario</propriedade>
    <recebidas>
        <aresta rotulo="Disjuncao Total" participacao="Total" tipo="
    Disjuncao">
            <origem>Funcionario</origem>
```

```
37          </aresta>
38      </recebidas>
39    </no>
40
41    <no rotulo="Horista">
42      <propriedade>Escala_pagto</propriedade>
43      <recebidas>
44          <aresta rotulo="Disjuncao Total" participacao="Total" tipo="
      Disjuncao">
45              <origem>Funcionario</origem>
46          </aresta>
47      </recebidas>
48    </no>
49  </grafo>
```

Código 5.5. Exemplo de Especializações

Para o caso de *hiperarestas* devem ser definidos os atributos "*participacao*" e "*tipo*" na aresta que representa a especialização na superclasse. O atributo "*participacao*" pode assumir os valores "*Total*" ou "*Parcial*" para indicar a participação da superclasse na especialização. Já o atributo "*tipo*" assume os valores "*Disjuncao*" ou "*Sobreposicao*", indicando a relação semântica entre as subclasses da especialização.

5.4. Protótipo de Software

Os algoritmos de mapeamento do modelo conceitual EER para o modelo lógico de multigrafos de propriedades foram implementados em linguagem *Java*, que foi escolhida por ser uma linguagem orientada a objetos bastante consolidada, com um alto grau de portabilidade e um grande número de ferramentas de suporte. Além disso, oferece suporte ao tratamento e manipulação de dados em arquivos XML.

A implementação do protótipo de *software* foi dividida em três pacotes que tratam de etapas distintas do processo de mapeamento. O primeiro pacote, denominado *conceitual*, foi implementado para fazer a leitura do modelo conceitual a partir de um arquivo XML e instanciar um conjunto de objetos que representem os elementos definidos na modelagem conceitual. O pacote denominado como *lógico* trata dos objetos que representam o modelo lógico de grafos proposto nesta dissertação. Por fim, o pacote *mapeamento*

foi implementado para lidar com a interface do software para coordenar e dar início ao processo de mapeamento.

5.4.1. Implementação do Modelo Conceitual

O pacote *conceitual* mantém a implementação das classes que representam os elementos do modelo conceitual EER. Dentro deste pacote foi implementada a classe *Conceitual*, cujos objetos são instanciados para representar esquemas conceituais. Um objeto da classe *Conceitual* possui listas de referências para objetos das classes *Entidade*, *Relacionamento* e *Especializacao*, que representam os elementos da modelagem EER considerada neste trabalho.

A classe *Entidade* possui um *Nome* e uma lista de objetos da classe *Atributos*. Por sua vez, cada objeto da classe *Atributo* possui um *Nome*, um *booleano* que indica se o atributo é *Multivalorado* e uma lista de *Componentes* que é instanciada somente em casos de atributos compostos. Cada entidade definida no arquivo XML de entrada gera um objeto da classe *Entidade*, que deve ser adicionado na lista de entidades do objeto *Conceitual* que representa o modelo.

Os relacionamentos são tratados pela classe *Relacionamento*, que mantém um *Nome*, um *Tipo* (binário ou n-ario), uma lista de *Atributos* e uma lista de *Entidades Participantes*, que contém informações sobre participação, cardinalidade máxima e papéis de cada entidade que participa do relacionamento. No caso de relacionamentos binários, a lista de entidades que participam do relacionamento contém referências a apenas duas entidades, sendo considerado que a primeira entidade da lista é a "origem" do relacionamento, enquanto que a segunda entidade listada é considerada como "destino". Em caso de relacionamentos *n-ários*, a lista de entidades participantes do relacionamento deve conter mais de duas entidades referenciadas.

A classe *Especializacao* é utilizada para manipular os elementos das especializações definidas na modelagem conceitual. Nesta classe são mantidas uma referência para o objeto *Entidade* que representa a *Superclasse* e uma lista de referências aos objetos *Entidade* que representam as *Subclasses*. Além disso, um objeto desta classe possui dados sobre a *Participação* da superclasse no processo e o *Tipo* de relação semântica entre as subclasses (disjunção ou sobreposição), quando for o caso.

O processo de mapeamento começa com a leitura do arquivo XML que representa o modelo conceitual EER e com a instanciação de um objeto da classe *Conceitual*. Ao instanciar um objeto da classe *Conceitual* é possível manipular suas listas de elementos, adicionando entidades, relacionamentos e especializações de acordo com a leitura do arquivo XML que contém os dados do modelo conceitual.

Assim que a leitura é concluída, o objeto *conceitual* que representa o modelo EER dentro do *software* estará povoado com o conjunto de entidades, relacionamentos e especializações definidos no arquivo XML de entrada de dados. O próximo passo é partir para a conversão do modelo conceitual para o modelo lógico de grafos.

5.4.2. Implementação do Modelo de Grafos

O modelo lógico de multigrafos de propriedade é implementado no pacote *logico*. Este pacote tem uma classe denominada *Logico*, a qual representa o próprio esquema lógico do BDG e agrega a lista de nós que compõem o grafo de esquema. O processo de mapeamento, iniciado com a leitura do modelo conceitual, produz uma instância da classe *Lógico* e, a medida que os elementos do modelo conceitual vão sendo mapeados, novos elementos são adicionados ao grafo de esquema.

A classe *Logico* possui apenas uma lista de objetos da classe *No*. As arestas do grafo de esquema são geradas como referências internas nos nós mapeados. Cada objeto da classe *No* mantém um rótulo, uma lista de *Propriedades*, uma lista de *Arestas Enviadas*, as quais referenciam as arestas cuja origem é o próprio nó, e, por fim, uma lista de *Arestas Recebidas* que referenciam as arestas que tem o nó como destino.

A classe *Aresta* mantém um *Rótulo* e uma lista de *Propriedades*. Todas as arestas devem ser referenciadas nas listas de adjacências dos nós em que incidem. Arestas que representam relacionamentos, por exemplo, devem constar tanto na listas de arestas enviadas de seu nó de origem quanto na lista de arestas recebidas de seu nó de destino. Arestas que representam especializações simples devem ser referenciadas como uma das arestas enviadas da subclasse e como uma das arestas recebidas pela superclasse. No caso de hiperarestas de especialização, que envolvem uma superclasse e várias subclasses, a aresta deve ser referenciada como enviada pela superclasse e recebida por cada um dos nós referentes às subclasses.

O objeto *Logico* é gerado automaticamente pelo processo de mapeamento do software e representa o grafo de esquema mapeado a partir do do modelo conceitual de entrada. Nós e arestas são instanciados a partir deste mapeamento e inseridos no grafo de esquema. Uma vez que o mapeamento do grafo de esquema está completo, ele gera o arquivo XML que o descreve a partir de seu conjunto de nós e arestas.

5.4.3. Implementação do Processo de Mapeamento

No pacote *mapeamento* é implementada a classe *MainWindow*, que coordena a leitura do modelo conceitual de entrada, faz o mapeamento para o grafo de esquema e produz o arquivo XML de saída. Esta classe contém o atributo *eer*, que é um objeto da classe *Conceitual*, e o atributo *grafo*, que é um objeto da classe *Logico*. A classe *MainWindow* implementa a *interface* do *software* e organiza cada passo do processo de mapeamento.

O primeiro passo do processo é a leitura do modelo conceitual de entrada. Isto é feito pelo método *povoaConceitual*, que lê o arquivo XML de entrada e instancia o objeto *eer*. O método *povoaEntidades* faz uma varredura por todas as entidades no arquivo de entrada inserindo cada uma delas no objeto *eer*. O mesmo ocorre com os relacionamentos e as especializações ao serem executados os métodos *povoaRelacionamentos* e *povoaEspecializacoes*.

Após o fim da leitura do arquivo XML, o objeto *eer* mantém referências a todos os elementos da modelagem EER representada pelo arquivo de entrada. O próximo passo, portanto, é processar o mapeamento do modelo EER para o grafo de esquema. Isto é realizado pelo método *mapeiaLogico*, que faz com que cada entidade, relacionamento e especialização seja mapeada para algum elemento do grafo de esquema e inserido no objeto *grafo*.

Após o mapeamento do último elemento da modelagem EER, o arquivo XML de saída é gerado pelo objeto *grafo* e o *software* encerra sua execução após esta etapa, concluindo o processo de mapeamento.

Capítulo 6

Conclusões e Trabalhos Futuros

Ao longo deste projeto de pesquisa foi realizado um estudo sobre projetos de bancos de dados *NoSQL*, em especial os bancos de dados de grafos. Conforme abordado em vários dos trabalhos citados nesta dissertação, são raras as propostas de se aplicar técnicas tradicionais de projetos de bancos de dados a bancos de dados *NoSQL*, embora seja reconhecida a sua importância.

Ao se desenvolver projetos de bancos de dados de grafos, bem como nas demais categorias de bancos de dados *NoSQL*, são consideradas regras baseadas em boas práticas de desenvolvimento, abrindo mão de um processo mais formal de *design*. Neste contexto, esta abordagem traz uma proposta de aplicação de regras tradicionais de projetos de bancos de dados no desenvolvimento e projeto de bancos de dados de grafos.

Um processo de mapeamento do modelo conceitual EER para um modelo lógico baseado em multigrafos de propriedades é tratado em detalhes neste projeto de pesquisa e, embora já existam trabalhos com a mesma temática, a exemplo (Daniel et al.; 2016), esta proposta busca atender de forma mais abrangente as restrições de participação e de cardinalidade em relacionamentos, bem como restrições semânticas em especializações, considerando a participação das superclasses e as relações entre as subclasses, permitindo ou não a sobreposição de elementos.

Para fechar este projeto de pesquisa, foi implementado em linguagem *Java* um protótipo de software que executa o processo de mapeamento tomando como entrada um diagrama

conceitual expresso em XML e produzindo um, aqui denominado, grafo de esquema, também representado em XML.

6.1. Contribuições

Embora atualmente o uso de bancos de dados de grafos esteja em crescimento, ainda são raras as discussões e projetos de pesquisa que envolvam todos os aspectos de um projeto de bancos de dados de grafos, principalmente quanto a definição de restrições de integridade. Neste contexto, esta dissertação traz uma proposta de tratamento de restrições de cardinalidade em relacionamentos e de participação em superclasses perante suas subclasses como uma de suas primeiras contribuições.

O processo de mapeamento do modelo conceitual para o modelo lógico proposto neste trabalho é definido de forma que possa ser implementado e testado de maneira eficiente a fim de que se torne uma ferramenta nas etapas de projeto de bancos de dados de grafos e como uma maneira de verificar a consistência dos dados nas instâncias do BDG. Para isso, o formato XML proposto para representação do grafo de esquema pode eventualmente ser considerado por algum SGBDG a fim de realizar uma verificação mais detalhada de suas bases de dados, verificando não só os padrões de conexão no grafo, mas também algumas restrições importantes e que geralmente não são consideradas em sistemas gerenciadores de grafos.

6.2. Trabalhos Futuros

O software que implementa o processo de mapeamento proposto nesta dissertação produz um grafo de esquema em formato XML, entretanto uma outra forma de representação que pode ser considerada é o *JSON* (*JavaScript Object Notation*), que, assim como a linguagem XML, possui uma estrutura legível para usuários e desenvolvedores e, ao mesmo tempo, fácil de ser gerada e processada por aplicativos (Nurseitov et al.; 2009). Sendo assim, o software desenvolvido para processar o mapeamento conceitual/lógico pode ser adaptado para dar ao usuário a opção de gerar o grafo de esquema tanto em XML quanto em *JSON*.

O grafo de esquema proposto nesta abordagem é uma ferramenta que representa o projeto lógico do banco de dados sem considerar um SGBDG de destino. Uma forma interessante

de dar continuidade ao trabalho é fazer com que este esquema lógico possa ser utilizado efetivamente por um SGBDG para validar as operações no banco de dados. Desta maneira, pode-se aprimorar os conceitos do grafo de esquema além de ser possível analisar o impacto no desempenho de um banco de dados de grafo que executa um conjunto mais abrangente de verificação de integridade.

Por fim, uma modificação interessante seria a implementação de uma *interface* para possibilitar fluxos alternativos para o processo de mapeamento considerando SGBDGs alvo com seus recursos e limitações impostos. Desta forma, serão possíveis grafos de esquema que levam em consideração as características próprias dos SGBDs, permitindo que o processo de mapeamento seja mais preciso e natural para o modelo de grafos implementado fisicamente. Isso permitiria, por exemplo, que para sistemas gerenciadores que dão suporte a atributos multivalorados seja gerado um grafo de esquema diferente daqueles sistemas que não dão este tipo de suporte.

Referências Bibliográficas

Abramova, V., Bernardino, J. & Furtado, P. (2014). Experimental evaluation of nosql databases, *International Journal of Database Management Systems* **Vol. 6**(Nº 3): p. 1.

Alexandre, J. & Cavique, L. (2013). Nosql no suporte à análise de grande volume de dados, *Revista de Ciências da Computação* **Vol. 8**(Nº 8): p. 37–48.

Alvarez, G. M., Ceci, F. & Gonçalves, A. L. (2015). Análise Comparativa dos Bancos Orientados a Grafos de Primeira e Segunda Geração–Uma Aplicação na Análise Social, *CEP* **88040**: 900.

ArangoDB (2020). Open-Source Multi-Model Database For Graph, Document and Search. Acessado em 24/08/2020.
URL: *https://www.arangodb.com/*

Atzeni, P., Bugiotti, F., Cabibbo, L. & Torlone, R. (2016). Data Modeling in the NoSQL World, *Computer Standards & Interfaces* .
URL: *http://www.sciencedirect.com/science/article/pii/S0920548916301180*

Binani, S., Gutti, A. & Upadhyay, S. (2016). Sql vs. nosql vs. newsql-a comparative study, *Communications on Applied Electronics (CAE)* **6**(1): 1–4.

Catarino, M. H. (2017). *Integrando Banco de Dados Relacional e Orientado a Grafos para Otimizar Consultas com Alto Grau de Indireção*, PhD thesis, Universidade de São Paulo.

Chen, P. P.-S. (1976). The entity-relationship model—toward a unified view of data, **1**(1): 9–36.
URL: *https://doi.org/10.1145/320434.320440*

Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks, *Commun. ACM* **13**(6): 377–387.
URL: *http://doi.acm.org/10.1145/362384.362685*

Comyn-Wattiau, I. & Akoka, J. (2017). Model Driven Reverse Engineering of NoSQL Property Graph Databases: The case of Neo4j, *2017 IEEE International Conference on Big Data (Big Data)* pp. 453–458.

Costa, P. P. d. (2011). *Teoria dos grafos e suas aplicações*, Universidade Estadual Paulista (UNESP).

Daniel, G., Sunyé, G. & Cabot, J. (2016). Umltographdb: mapping conceptual schemas to graph databases, *International Conference on Conceptual Modeling*, Springer, pp. 430–444.

Date, C. J. (2004). *Introdução a sistemas de bancos de dados*, Elsevier Brasil.

DB-Engines (2019). DB-Engines Ranking - popularity ranking of database management systems. Acessado em 01/10/2019.
URL: *https://db-engines.com/en/ranking*

de Lima, C. & dos Santos Mello, R. (2015). A workload-driven logical design approach for nosql document databases, *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services*, ACM, p. 73.

De Virgilio, R., Maccioni, A. & Torlone, R. (2014). Model-driven design of graph databases, *International Conference on Conceptual Modeling*, Springer, pp. 172–185.

Elfaki, A., Aljaedi, A. & Duan, Y. (2019). Mapping ERD to Knowledge Graph, *2019 IEEE World Congress on Services (SERVICES)*, Vol. 2642, IEEE, pp. 110–114.

Elmasri, R., Navathe, S. B., Vieira, D. et al. (2011). *Sistemas de banco de dados*, Pearson Addison Wesley São Paulo.

Feofiloff, P., Kohayakawa, Y. & Wakabayashi, Y. (2011). Uma introdução sucinta à teoria dos grafos, *Universidade de São Paulo, Instituto de Matemática e Estatística, Departamento de Ciência da Computação* .

Fernandes, D. & Bernardino, J. (2018). Graph databases comparison: Allegrograph, arangodb, infinitegraph, neo4j, and orientdb., *DATA*, pp. 373–380.

Gallo, G., Longo, G., Pallotino, S. & Nguyen, S. (1993). Directed hypergraphs and applications, *Discrete applied mathematics* **42**(2-3): 177–201.

Grolinger, K., Higashino, W. A., Tiwari, A. & Capretz, M. A. (2013). Data management in cloud environments: Nosql and newsql data stores, *Journal of Cloud Computing: advances, systems and applications* **2**(1): 22.

Heuser, C. A. (2009). *Projeto de Bancos de Dados*, Bookman.

Kuderu, N. & Kumari, V. (2016). Relational database to nosql conversion by schema migration and mapping, *Int. J. Comput. Eng. Res. Trends* **3**: 506–513.

Kusu, K. & Hatano, K. (2018). Combining Two Types of Database System for Managing Property Graph Data, *2018 IEEE International Conference on Big Data (Big Data)*, IEEE, pp. 5366–5368.

Liptchinsky, V., Satzger, B., Zabolotnyi, R. & Dustdar, S. (2013). Expressive Languages for Selecting Groups from Graph-structured Data, *Proceedings of the 22nd international conference on World Wide Web*, pp. 761–770.

Liyanaarachchi, G., Kasun, L., Nimesha, M., Lahiru, K. & Karunasena, A. (2016). Migdb-relational to nosql mapper, *2016 IEEE International Conference on Information and Automation for Sustainability (ICIAfS)*, IEEE, pp. 1–6.

Microsoft (2020). Azure Cosmos DB — Microsoft Docs. Acessado em 28/08/2020.
URL: *https://docs.microsoft.com/pt-br/azure/cosmos-db/*

Neo4j (2020a). Neo4j Graph Platform – The Leader in Graph Databases. Acessado em 29/08/2020.
URL: *https://neo4j.com/*

Neo4j (2020b). Relational Databases vs. Graph Databases: A Comparison. Acessado em 23/05/2020.
URL: *https://neo4j.com/developer/graph-db-vs-rdbms/*

Neo4j (2020c). The Neo4j Cypher Manual v4.1. Acessado em 12/06/2020.
URL: *https://neo4j.com/docs/cypher-manual/current/*

Nurseitov, N., Paulson, M., Reynolds, R. & Izurieta, C. (2009). Comparison of json and xml data interchange formats: a case study., *Caine* **9**: 157–162.

OrientDB (2020). Home · OrientDB Manual. Acessado em 23/08/2020.
URL: *https://www.orientdb.org/docs/3.0.x/*

Penteado, R. R., Schroeder, R., Hoss, D., Nande, J., Maeda, R. M., Couto, W. O. & Hara, C. S. (2014). Um Estudo Sobre Bancos de Dados em Grafos Nativos, *X ERBD-Escola Regional de Banco de Dados* .

Poffo, J. P. (2016). *Projeto lógico de bancos de dados NoSQL colunares a partir de esquemas conceituais entidade-relacionamento estendido (EER)*, Master's thesis, Universidade Federal de Santa Catarina (UFSC).

Pokorny, J. (2013). Nosql databases: a step to database scalability in web environment, *International Journal of Web Information Systems* **9**(1): 69–82.

Pokorny, J. (2016). Conceptual and database modelling of graph databases, *Proceedings of the 20th International Database Engineering & Applications Symposium* pp. 370–377.

URL: *http://doi.acm.org/10.1145/2938503.2938547*

Pokorny, J., Valenta, M. & Kovacic, J. (2017). Integrity constraints in graph databases, *The 7th International Symposium on Frontiers in Ambient and Mobile Systems (FAMS 2017)* pp. 975–981.

Prestes, E. (2016). Introdução à teoria dos grafos, *Universidade Federal do Rio Grande do Sul, Instituto de Informática, Departamento de Informática Teórica, Tech. Rep* .

Robinson, I., Webber, J. & Eifrem, E. (2013). *Graph databases*, "O'Reilly Media, Inc.".

Roy-Hubara, N., Rokach, L., Shapira, B. & Shoval, P. (2017). Modeling Graph Database Schema, *IT Professional* **19**(6): 34–43.

Schreiner, G. A., Duarte, D. & dos Santos Mello, R. (2020). Bringing sql databases to key-based nosql databases: a canonical approach, *Computing* **102**(1): 221–246.

Šestak, M., Rabuzin, K. & Novak, M. (2016). Integrity constraints in graph databases-implementation challenges, *Central European Conference on Information and Intelligent Systems*.

Silberschatz, A., Korth, H. F., Sudarshan, S. & Vieira, D. (2012). *Sistema de Banco de Dados*, Elsevier.

Sousa, G. d. C. P. & Pereira, J. L. (2015). Document-based databases: estudo exploratório no âmbito das bases de dados nosql, *Atas da Conferência da Associação Portuguesa de Sistemas de Informação*, Vol. 15, Associação Portuguesa de Sistemas de Informação (APSI), pp. 191–207.

Sousa, V. M. d. (2018). *Projeto lógico de banco de dados nosql de grafos a partir de um modelo conceitual baseado no modelo entidade-relacionamento*, Master's thesis, Centro Universitário Campo Limpo Paulista (FACCAMP).

Sousa, V. M. d. & Cura, L. M. d. V. (2018). Logical design of graph databases from an entity-relationship conceptual model, *iiWAS2018: Proceedings of the 20th International Conference on Information Integration and Web-based Applications & Services*,

Association for Computing Machinery, New York, NY, USA, p. 183–189.
URL: *https://doi.org/10.1145/3282373.3282375*

Stanescu, L., Brezovan, M. & Burdescu, D. D. (2016). Automatic mapping of mysql databases to nosql mongodb, *2016 Federated Conference on Computer Science and Information Systems (FedCSIS)*, IEEE, pp. 837–840.

Teorey, T., Lightstone, S., Nadeau, T., Jagadish, H. & Vieira, D. (2014). *Projeto e Modelagem de Banco de Dados*, Elsevier.

Unal, Y. & Oguztuzun, H. (2018). Migration of Data from Relational Database to Graph Database, *Proceedings of the 8th International Conference on Information Systems and Technologies* pp. 1–5.
URL: *https://doi.org/10.1145/3200842.3200852*

Vágner, A. (2018). Store and Visualize EER in Neo4j, *Proceedings of the 2nd International Symposium on Computer Science and Intelligent Control*, pp. 1–6.

W3Schools (2020). XML Tutorial. Acessado em 23/10/2020.
URL: *https://www.w3schools.com/xml/default.asp*