João Paulo Poffo

PROJETO LÓGICO DE BANCOS DE DADOS NOSQL COLUNARES A PARTIR DE ESQUEMAS CONCEITUAIS ENTIDADE-RELACIONAMENTO ESTENDIDO (EER)

Dissertação submetida ao Programa de Pós-Graduação em Ciências da Computação para a obtenção do Grau de Mestre em Ciências da Computação. Orientador: Prof. Dr. Ronaldo dos Santos Mello.

Florianópolis

2016

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

Poffo, João Paulo
Projeto Lógico de Bancos de Dados NoSQL Colunares a partir de Esquemas Conceituais Entidade-Relacionamento Estendido (EER) / João Paulo Poffo ; orientador, Ronaldo dos Santos Mello - Florianópolis, SC, 2016.
85 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Ciência da Computação.

Inclui referências

1. Ciência da Computação. 2. Projeto de Banco de Dados. 3. Modelagem Lógica. 4. NoSQL. 5. Banco de Dados Colunar. I. Mello, Ronaldo dos Santos. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. III. Título.

João Paulo Poffo

PROJETO LÓGICO DE BANCOS DE DADOS NOSQL COLUNARES A PARTIR DE ESQUEMAS CONCEITUAIS ENTIDADE-RELACIONAMENTO ESTENDIDO (EER)

Esta Dissertação foi julgada aprovada para a obtenção do Título de "Mestre em Ciências da Computação", e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciências da Computação.

Florianópolis, 5 de março de 2016.

Prof.ª Dr.ª Carina Friedrich Dorneles
Coordenadora
Universidade Federal de Santa Catarina

Banca Examinadora:

Ronaldo dos Santos Mello
Orientador
Universidade Federal de Santa Catarina

Helena Grazziotin Ribeiro
Universidade de Caxias do Sul

Carina Friedrich Dorneles
Universidade Federal de Santa Catarina

Raul Sidnei Wazlawick
Universidade Federal de Santa Catarina

À memória de dona Anna Yolanda Poffo,
minha mãe.

AGRADECIMENTOS

Agradeço à minha esposa por ser um porto seguro e muitas vezes o vento que soprou em minhas velas.

Se não fosse pela compreensão e apoio do meu orientador, Prof. Dr. Ronaldo, esta dissertação não poderia ter sido concluída.

Aos colegas da pós-graduação e do Grupo de Banco de Dados da UFSC (GBD-UFSC) que, mesmo sem saber, colaboraram no desenvolvimento deste trabalho.

À HBSIS, empresa que acreditou na minha capacidade e permitiu que agarrasse essa oportunidade. E à minha empresa, Gessis, na figura de meu sócio Eder, que tornou dessa dissertação "nosso" projeto e me instigou a concluí-lo, apesar das adversidades.

Por fim, agradeço a todas as pessoas que me ajudaram de forma direta ou indireta na realização deste trabalho, e a Universidade Federal de Santa Catarina pela oportunidade de trabalho e capacitação.

RESUMO

Tecnologias emergentes muitas vezes mudam paradigmas. *NoSQL* é uma delas e está ganhando terreno com a onda *Big Data*, onde o volume de dados quebrou a barreira dos petabytes e a informação contida nestes dados é fundamental para decisões estratégicas. Nesse contexto, os tradicionais bancos de dados relacionais se mostram inadequados para gerenciar com eficiência o acesso a tais dados e, consequentemente, as metodologias tradicionais de projeto, voltadas para a construção destes bancos de dados, devem ser revistas para se adequar aos novos modelos de dados que surgiram com o *NoSQL*, como é o caso do modelo de dados colunar. Nota-se, especificamente para o caso de projeto de bancos de dados para BDs colunares, que a literatura carece de metodologias de projeto lógico, metodologias estas que convertem modelagens conceituais para representações lógicas adequadas e eficientes para fins de armazenamento e acesso. Assim sendo, esta dissertação propõe uma metodologia de projeto lógico para bancos de dados colunares que contribui para preencher o hiato existente entre o clássico projeto de banco de dados e a vanguarda tecnológica com o movimento *NoSQL*, em particular, a construção de bancos de dados colunares. Experimentos preliminares demonstraram, comparativamente, que a metodologia é promissora.

Palavras-chave: Projeto de Banco de Dados. Modelagem Lógica. NoSQL. Banco de Dados Colunar. Esquema lógico colunar.

ABSTRACT

Emerging technologies often break paradigms. *NoSQL* is one of them and is gaining space with the *Big Data* ascension, where the data volume exceeded the petabyte frontier and the information within these data can be of great importance to strategic decisions. In this case, legacy relational databases show themselves inadequate to efficiently manage these data and consequently, their traditional project methodologies must be reviewed to be suitable to new models, such as columnar data model. Regarding columnar database design, the literature lacks of methodologies for logical design, i.e., processes that convert a conceptual schema to a logical schema that optimize access and storage. In this context, this work proposes an approach for logical design of columnar databases that contributes to fill the void between classic project methodologies and the technological forefront with the *NoSQL* movement, in particular, columnar databases. Preliminary experiments demonstrated that the methodology is promising, if compared with a baseline.

Keywords: Database design. Logical design. NoSQL. Columnar database. Columnar logical schema.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Com o advento de serviços de computação e armazenamento na nuvem, a oportunidade de disponibilizar Sistemas Gerenciadores de Bancos de Dados (SGDB) como serviço ganha força, como testemunhado pela Amazon RDS[1] e Microsoft SQL Azure[2] (CURINO et al., 2011). NoSQL é um desses movimentos que vem se destacando por prover BDs que possuem características como alta disponibilidade e escalabilidade, características estas fundamentais para aplicações como redes sociais, redes de sensores, repositórios de perfil, provedores de conteúdo, dentre outras (HOFF, 2010).

NoSQL é um termo que é mais comumente usado para designar uma classe de BDs não relacionais que gerenciam grandes volumes de dados. Ele é dividido em quatro grandes modelos de dados (Chave/Valor, Colunar, Documento e Grafos) e possuem diversas aplicações na atualidade (KAUR; RANI, 2013) (ver Capítulo 2 para maiores detalhes). Em *DB-Engines* (Solid IT, 2016) há um *ranking*[3] onde, dentre os dez primeiros SGBDs, sete são relacionais. Entretanto, o que chama a atenção são os BDs NoSQL: MongoDB, Cassandra e Redis alcançando a quarta, oitava e décima colocação, respectivamente. MongoDB é um BD de documento, Cassandra é um BD colunar e Redis é chave-valor.

BDs colunares são assim chamados pois o menor item de informação no seu modelo de dados é uma coluna. Segundo Sadalage e Fowler (2013), a melhor forma de imaginar um BD colunar é como uma estrutura agregada de dois níveis. Assim como BDs chave-valor, o primeiro nível possui uma chave para identificar um agregado de interesse. A diferença para o modelo colunar é que este agregado é formado de um mapa de valores mais detalhados. Este segundo nível é chamado de coluna. Além de acessar todas as colunas de uma só vez, é possível acessar cada uma delas individualmente e determinadas colunas podem ser multivaloradas ou conter outro conjunto de colunas.

[1] Disponível em: `https://aws.amazon.com/rds`.

[2] Disponível em: `https://azure.microsoft.com`.

[3] O *ranking* de Solid IT (2016) se baseia em menções em sites, interesse geral no sistema, frequência de discussões técnicas em fóruns, ofertas de emprego, presença em perfis de profissionais e relevância em redes sociais.

A metodologia padrão de projeto de um BD relacional ocorre em três etapas: modelagem conceitual, modelagem lógica e modelagem física (NAVATHE; BATINI; CERI, 1992; HEUSER, 2008). Em contraste, esta sequência, para BDs colunares, parece ter sido suprimida e, ao invés do projeto começar com a modelagem conceitual, inicia-se modelando os conjuntos de colunas com base nas principais consultas esperadas (WANG; TANG, 2012).

O modelo de dados utilizado em um projeto conceitual deve ser expressivo, simples, mínimo e formal (ELMASRI; NAVATHE, 2000). Existem vários modelos que possuem estas características, entre eles a *Unified Modelling Language* (UML), o *Object with Roles Model* (ORM) (KRISTENSEN, 1996) e o *Entidade-Relacionamento Estendido* (EER) (NAVATHE; BATINI; CERI, 1992). A motivação para este trabalho é propor uma reconciliação entre a abordagem tradicional de projeto de BD e o projeto de BDs colunares (muito pouco explorado na literatura), contribuindo com a definição de um processo que, a partir de uma modelagem conceitual definida de acordo com o modelo clássico Entidade-Relacionamento Estendido (EER), realize um projeto lógico baseado em regras de conversão para esta categoria de BD NoSQL. A intenção é aplicar boas práticas para geração de um esquema lógico para BDs colunares, esquema esse que seja eficiente e aderente aos seus principais objetivos de utilização (demonstrado experimentalmente), que é geralmente o *workload* (principais operações sobre os dados realizadas no BD) das aplicações que irão utilizá-lo. A escolha pelo modelo EER como modelo conceitual de entrada para a metodologia proposta neste trabalho se justifica por este ser um modelo conceitual consagrado para o projeto conceitual de BD, uma vez que os demais modelos enfatizam o desenvolvimento de aplicações orientadas a objetos. Além disso, o modelo EER é um dos mais ricos em termos construtores disponíveis para modelagem conceitual de dados.

Trabalhos presentes na literatura, como Schroeder e Mello (2008), que realiza o mapeamento de modelagens conceituais EER para modelos hierárquicos como XML, e Fong (1995), que propõe o projeto lógico de BDs Orientados a Objetos (BDOOs), estão relacionados a esta dissertação no sentido de propor metodologias para projeto lógico de esquemas de dados. Contudo, eles não tratam de BDs NoSQL. Bugiotti et al. (2014) apresenta um modelo NoSQL abstrato (NoAM) que objetiva englobar quaisquer modelos de dados de BDs NoSQL, mantendo escalabilidade, consistência e desempenho e, similarmente porém mais completa, Chebotko, Kashlev e Lu (2015) propõem uma abordagem direcionada ao BD colunar Cassandra. A Microsoft, em (SHARP et al., 2013) e Schram e Anderson (2012), sugerem orientações limitadas para a modelagem lógica e física de BDs colunares, não apresentando um conjunto claro e completo de regras para o mapeamento de uma modelagem conceitual. Outros trabalhos (SCHRAM; ANDERSON, 2012; WANG; TANG, 2012; KAUR; RANI, 2013) tratam do projeto lógico para BDs colunares com base em domínios específicos, mas carecem de completude em se tratando de modelagem conceitual, regras de conversão e validação da proposta. Contrastando, Meijer e Bierman (2011) apresentam um modelo matemático para BDs NoSQL e demonstram sua correlação com o modelo relacional. Porém, não é direcionado para BDs colunares e não trata a modelagem conceitual e um processo de conversão. Enfim, a literatura ainda carece de uma metodologia detalhada para o projeto lógico de BDs colunares.

1.1 OBJETIVO

Como solução para a problemática recém apresentada, esta dissertação propõe um processo, baseado em regras de mapeamento, de modelagens conceituais EER para uma modelagem lógica correspondente para BDs colunares. O processo é exemplificado por meio de uma notação de projeto lógico que introduz algumas representações visando uma compreensão mais holística e completa possível do modelo colunar, destacando características que não são comuns a BDs relacionais. Este processo utiliza um algoritmo de mapeamento que define a ordem de aplicação das regras. Experimentos são apresentados com a finalidade de avaliar o processo comparando-o com uma linha base (NoAM). Para atingir este objetivo geral, são destacados os seguintes objetivos específicos:

- Análise de produtos disponíveis no mercado e definições da comunidade científica visando a apresentação do conjunto de conceitos mais relevantes para o modelo de dados de um BDs colunar;
- Definição de uma notação de esquema lógico para BDs colunares;
- Definição de regras para conversão de construtores de um esquema conceitual para um esquema lógico de um BD colunar, bem como a definição de um processo de conversão que aplica estas regras;
- Experimentos para avaliação do processo proposto.

1.2 MÉTODO DE PESQUISA

O método de pesquisa aplicado a esta dissertação foi inspirado pelo trabalho de Schroeder e Mello (2008), que faz o mapeamento de modelagens conceituais EER para XML. O tema deste trabalho foi construído a partir da observação que o relacionamento entre modelagem conceitual e BDs colunares, ao invés de XML, era potencialmente inexplorado na literatura.

A partir disso, foi realizada uma revisão bibliográfica para identificar trabalhos correlatos verificando limitações ou incompletudes nos mesmos (SCHROEDER; MELLO, 2008; NAVATHE; BATINI; CERI, 1992; SHARP et al., 2013; SCHRAM; ANDERSON, 2012; WANG; TANG, 2012; KAUR; RANI, 2013; MEIJER; BIERMAN, 2011). Este esforço constatou que o tema escolhido, projeto lógico de BDs colunares a partir de esquemas conceituais EER, carecia de soluções detalhadas.

Os principais veículos de pesquisa de publicações foram o Google scholar, IEEE Xplorer, DBLP, ACM, SIGMOD, dentre outros. Partindo da cunhagem do termo NoSQL em 1998, mas focando em trabalhos mais recentes, dentre livros, eventos e periódicos. As palavras chave foram escolhidas com base no que queríamos ligar: Modelagem conceitual (*conceptual design, EER, ORM, database project* etc) e lógica de BDs colunares (*logical design, NoSQL, columnar, column oriented databases*). A combinação desses termos e derivações dos mesmos permitiu a identificação de diversos artigos. Além da colaboração do Grupo de Banco de Dados (GBD) da UFSC, sempre atento a novas publicações relacionadas. Foram priorizadas buscas em inglês pois em português raramente eram efetivas.

Para fundamentar teoricamente o tema proposto foi essencial buscar a contribuição de autores consagrados como Sadalage e Fowler (2013), Heuser (2008), Elmasri e Navathe (2005), identificando um formato adequado para sua formalização e avaliação experimental.

O método de pesquisa experimental foi sendo definido como prova de conceito no decorrer do levantamento bibliográfico e coincidiu com a publicação do NoAM (BUGIOTTI et al., 2014). Um trabalho recente e muito citado por trabalhos relacionados, que apresenta um teste de execução com comparação de desempenho entre ele e as abordagens anteriores. Dessa forma, visamos expandi-lo propondo uma prova de conceito similar para permitir validação.

O acompanhamento contínuo da literatura permitiu que trabalhos como Bugiotti et al. (2014), Chebotko, Kashlev e Lu (2015) sejam incorporados, enriqueçam esta pesquisa e demonstrem que a área possui interesse e investimento para evolução.

1.3 ORGANIZAÇÃO DA DISSERTAÇÃO

Esta dissertação possui mais 5 capítulos. No próximo capítulo é apresentada a fundamentação teórica do tema onde são detalhados conceitos sobre projeto de BDs e NoSQL com ênfase em BDs colunares. O capítulo 3 descreve os trabalhos correlatos, destacando seus pontos fortes e qual o enfoque deste trabalho. O capítulo 4 detalha a metodologia de conversão de uma modelagem conceitual EER para um esquema lógico de um BD colunar com base em uma notação de esquema lógico para este tipo de BD, proposta também neste trabalho. O capítulo 5 apresenta experimentos de avaliação da metodologia e o capítulo 6 é dedicado à conclusão.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo explora os temas associados a este trabalho. Inicialmente é apresentado o projeto clássico de BD, com ênfase na modelagem lógica. Em seguida, descreve-se os BDs NoSQL e suas diferenças com relação a BDs tradicionais. Por último, os BDs colunares são explanados.

2.1 PROJETO DE BANCO DE DADOS

O projeto de um BD visa definir adequadamente, de acordo com um modelo de BD alvo, os fatos do mundo real e seus relacionamentos. Para tanto, ele deve considerar uma representação dos dados que maximize o armazenamento e rapidez de acesso aos mesmos (HEUSER, 2008). No projeto de BDs, a modelagem conceitual tem seus alicerces em décadas de evolução de modelos de dados que, segundo Kim (1990), passou por cinco gerações: sistemas de arquivos, modelos hierárquicos, de redes, relacional e semântico. Com o surgimento do movimento NoSQL, talvez estejamos à margem de uma nova geração.

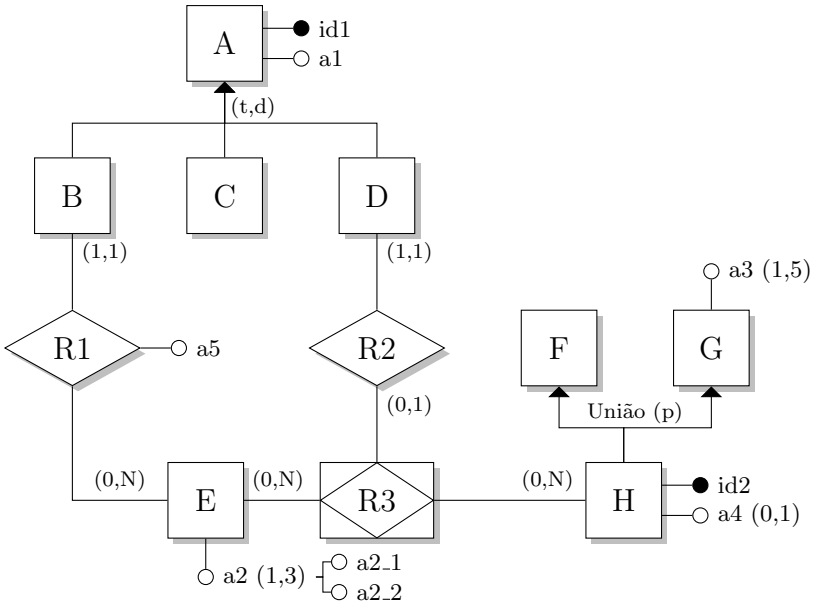
As etapas tradicionais de um projeto de BD são: levantamento de requisitos, projeto conceitual, lógico e físico (NAVATHE; BATINI; CERI, 1992). No levantamento de requisitos, um analista identifica pessoas chave para entrevistas e então coleta documentos e informações importantes com especialistas do domínio para a elaboração de um artefato formal de requisitos de dados da organização. No projeto conceitual, é elaborada uma representação de alto nível, normalmente por meio de diagramas, que representa um esquema conceitual cuja finalidade é descrever os fatos relevantes do domínio e seus relacionamentos. O projeto lógico corresponde à conversão do esquema conceitual em uma modelagem direcionada ao BD utilizado. O projeto físico define o modelo lógico em termos de instruções que podem ser interpretadas pelo SGBD para gerar o BD desejado, além de atenção com a eficiência de acesso por meio do planejamento e definição de índices, visões e outros recursos típicos de SGBDs.

O modelo de dados adotado em um projeto conceitual deve ser expressivo, simples, mínimo e formal (ELMASRI; NAVATHE, 2000). Existem vários modelos que possuem estas características, entre eles a *Unified Modelling Language* (UML), *Object with Roles Model* (ORM) (KRISTENSEN, 1996) e Entidade-Relacionamento Estendido (EER) (HEUSER, 2008; NAVATHE; BATINI; CERI, 1992).

Segundo Heuser (2008), modelos de dados são usados para comunicação entre as pessoas da organização e até mesmo a comunicação entre programas. Assim, ao usar um modelo de dados, é necessário levar em consideração padrões de confecção de modelos. Existem muitas propostas de padronização como o próprio Entidade-Relacionamento (ER), Engenharia de Informações, MERISE, UML etc. Junto delas existe uma ampla variedade de ferramentas para auxiliar nessa modelagem, denominadas *Computer Aided Software Engineering* (CASE). O uso de um modelo em conjunto com uma ferramenta CASE gera um artefato que, no caso da modelagem conceitual, é denominado esquema conceitual.

Uma das formas consideradas expressivas para representar um projeto de alto nível de um BD é o ER (ELMASRI; NAVATHE, 2000; NAVATHE; BATINI; CERI, 1992), modelo este já consagrado como ferramental para a modelagem conceitual de BDs. Ele possui 3 conceitos principais: entidade, relacionamento e atributo. Entidade é a abstração de um conjunto de objetos similares do mundo real (representada por um retângulo, na notação clássica do ER). Relacionamento é uma associação semântica entre entidades (representada por um losango). Atributo é uma propriedade associada a uma entidade ou relacionamento (representada por uma linha terminada em um círculo com o nome do atributo). Para esta dissertação são considerados os conceitos e a notação do modelo EER, uma extensão do modelo ER que define construtores de modelagem para os 3 tipos de abstração de um modelo conceitual: classificação, agregação e generalização (NAVATHE; BATINI; CERI, 1992).

Figura 1: Exemplo de um esquema EER.



Fonte: Schroeder e Mello (2008).

Utilizando a Figura 1 como exemplo, podemos identificar os construtores do tipo **abstração de classificação**, cujos conceitos são denominados entidades e os atributos (propriedades) que a compõe. Esses atributos possuem ocorrência mínima, ocorrência máxima e modelo de conteúdo. Quando a ocorrência mínima é zero, diz-se que o atributo é opcional, quando é 1, é obrigatório. Se a ocorrência máxima for 1, o atributo é monovalorado, se for maior que 1 ele é multivalorado. Além disso, um atributo possui um modelo de conteúdo simples ou composto. Um atributo composto armazena um grupo de outros atributos, caso contrário, é um atributo simples. Um exemplo de atributo multivalorado composto obrigatório é o atributo `a2` do exemplo. Um atributo é denominado identificador se ele é utilizado para identificar de forma inequívoca uma das instâncias da entidade (`id1` da entidade `A`, por exemplo).

Os construtores do tipo **abstração de agregação** determinam as associações entre conceitos e são representados por relacionamentos. Eles são caracterizados pelo seu grau, ocorrências mínima e máxima de cada grau, cardinalidade e existência ou não de atributos. Quanto ao grau, ele é definido pela quantidade de entidades relacionadas. Binário no caso de duas entidades, ternário no caso de três, e assim por diante. Cada grau possui uma ocorrência mínima e máxima que seguem os mesmos preceitos das ocorrências dos atributos na abstração de classificação. Sua cardinalidade é dada pelas ocorrências máximas de seus graus. Cada relacionamento pode possuir atributos e estar associado a outros relacionamentos. Neste caso, o relacionamento deve ser promovido para uma entidade associativa (`R3` é um exemplo na Figura 1).

Os construtores do tipo **abstração de generalização** são a relação de herança entre conceitos representando um comportamento de super ou subconjunto. Basicamente, eles são restrições adicionais impostas às entidades, sendo composta pelos tipos generalização e união.

Segundo Heuser (2008), Navathe, Batini e Ceri (1992), com a abstração de generalização (ou especialização, se visto no sentido oposto) de entidades é possível atribuir propriedades particulares a um subconjunto das instâncias (*especializadas*) de uma entidade *genérica*. Esta generalização possui as restrições de integridade parcial ou total. Ela é parcial se a existência de uma instância especializada é opcional. Ela é total se a instância de pelo menos um dos conceitos especializados sempre existir, ou seja, a generalização nunca terá uma instância sem especialização. Outra restrição é se a generalização é exclusiva ou compartilhada. No primeiro caso, uma instância da entidade genérica pode pertencer a apenas uma especialização na hierarquia, enquanto que na compartilhada, ela pode pertencer a várias. No exemplo da Figura 1, a entidade `A` é genérica e as entidades `B`, `C` e `D` são especializações desta entidade. Essa generalização é total e disjunta, ou seja, se houver uma instância de `A`, ela se especializa em `B`, `C` ou `D`, sem exceções. As combinações possíveis para as restrições de integridade de uma abstração de generalização são:

- *Total e disjunta (t,d)*: instâncias da entidade genérica devem pertencer a apenas uma dentre todas as entidades especializadas;
- *Total e compartilhada (t,c)*: instâncias da entidade genérica devem pertencer a uma ou mais dentre todas as entidades especializadas;
- *Parcial e disjunta (p,d)*: instâncias da entidade genérica podem pertencer a apenas uma dentre todas as entidades especializadas;
- *Parcial e compartilhada (p,c)*: instâncias da entidade genérica podem pertencer a uma ou mais dentre todas as entidades especializadas.

A ausência de informações sobre estas restrições de integridade para generalizações em um diagrama EER denota o padrão `(t,d)` (*default*).

Outro tipo de restrição de integridade é o de *União*, que pode ser apenas disjunta total ou disjunta parcial. No caso de união total, todas as instâncias das entidades genéricas são também instâncias da entidade especializada. No caso de união parcial, nem todas as instâncias das entidades genéricas são também instâncias da entidade especializada. Este é o caso da entidade `H` no exemplo da Figura 1.

Existem outros recursos do modelo EER, como entidades fracas e auto-relacionamentos. Eles serão discutidos na proposta de processo para modelagem lógica de BD apresentada no Capítulo 4.

A Figura 2 exemplifica as etapas de um projeto clássico de BDs referentes à modelagem conceitual, além de uma possível modelagem lógica relacional e um padrão de implementação da sua modelagem física. O foco desta dissertação é a conversão da etapa 1 para a etapa 2, ou seja, o mapeamento conceitual-lógico. No exemplo tem-se a conversão de alguns conceitos constituintes do modelo EER: Entidade (Músculo e Membro), relacionamento (Move), atributo identificador (Nome), o atributo opcional (Força e Fadiga) e a cardinalidade do relacionamento.

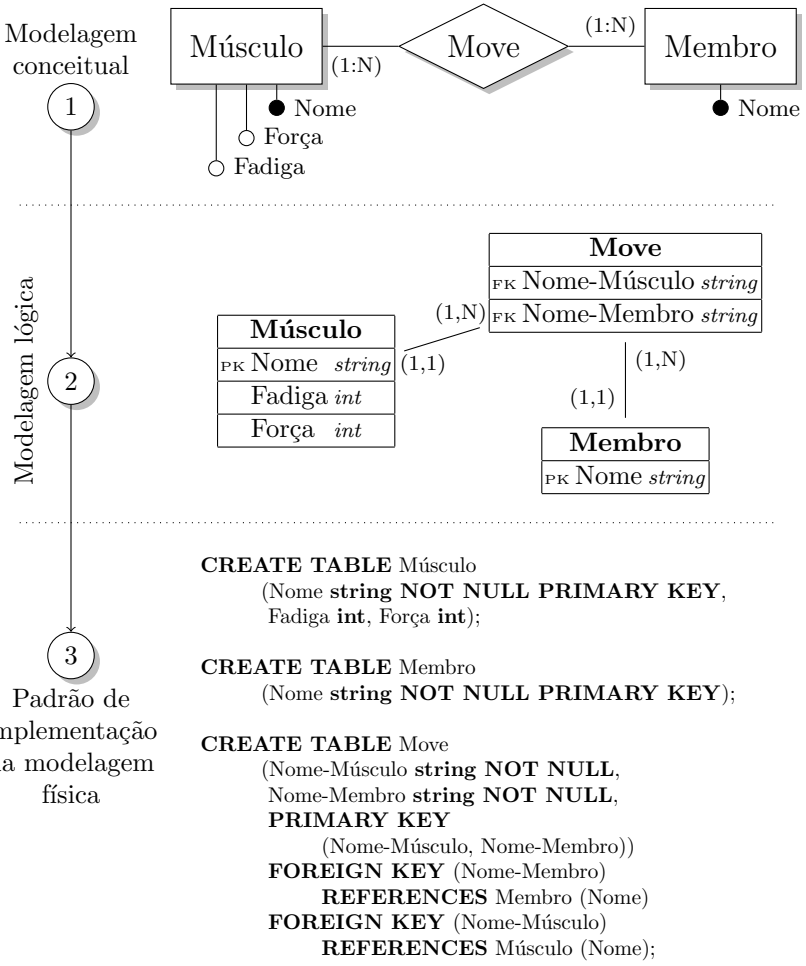
Para Heuser (2008), uma das principais regras é a conversão das entidades: cada entidade é mapeada para uma tabela e cada atributo nesta entidade é mapeado para uma coluna na tabela respectiva. Os atributos identificadores se tornam a chave primária da tabela. O exemplo da Figura 2 gera 2 tabelas e mais uma tabela para o relacionamento, uma vez que suas cardinalidades máximas são `N`. Essas e outras regras de conversão são normalmente controladas por um algoritmo de mapeamento que rege a etapa de projeto lógico. Este algoritmo segue alguns princípios, como evitar redundâncias, minimizar junções e evitar colunas opcionais. A proposta de projeto lógico desta dissertação também propõe algoritmos baseados em princípios direcionados ao modelo lógico para BDs colunares, que são detalhados no Capítulo 4.

2.2 NOSQL

Um BD obedece um modelo que define como seus dados são armazenados, recuperados ou manipulados. Este modelo é implementado por SGBDs que devem garantir as propriedades Atomicidade, Consistência, Isolamento e Durabilidade (ACID). Até recentemente este acrônimo mágico, fundamental e intocável dos BDs reinava soberano. As propriedades ACID são adequadas a SGBDs que executam em servidores potentes, porém independentes.

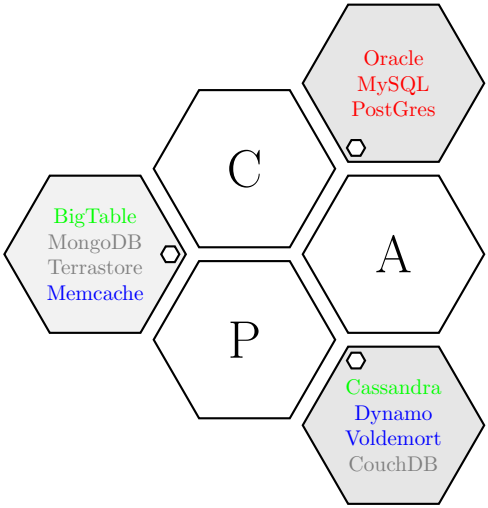
Os SGBDs, quando pressionados a escalar horizontalmente (executar em várias máquinas) ao invés de verticalmente (aumentar a potência da máquina), geralmente empregam abordagens de escalabilidade como *Sharding* (separar dados em múltiplas máquinas) ou *Optimistic Locking* (permite que várias transações concluam sem interferirem entre elas), que causam problemas semelhantes ao controle de normalização da modelagem (MILANOVIĆ; MIJAJLOVIĆ, 2012).

Figura 2: Exemplo de projeto clássico de BD demonstrando a transição entre as etapas.



Fonte: Elaborada pelo autor.

Figura 3: Teorema CAP com exemplos de BDs para cada combinação de duas características.



Fonte: Elaborada pelo autor.

O mundo digital vem crescendo muito rápido e os dados gerados por sistemas computacionais estão se tornando mais complexos de gerenciar devido a questões de aumento de *volume* (terabytes para petabytes), *variedade* (dados estruturados, não-estruturados e híbridos) e alta *velocidade* de geração (crescimento exponencial). A este fenômeno global dos 3 Vs é dado o nome de *Big Data*, que tipicamente corresponde a coleções de dados tão massivas que não podem ser gerenciadas adequadamente por ferramentas convencionais de gerência de dados, como por exemplo, SGBDs relacionais. Para contornar esse problema, surgiram propostas como NoSQL e NewSQL (esse último um híbrido entre NoSQL e SGBDs relacionais) (MONIRUZZAMAN; HOSSAIN, 2013).

O teorema CAP (também conhecido como teorema de Brewer (2000) e provado formalmente por Gilbert e Lynch (2002)) afirma que, para qualquer sistema compartilhando dados, é impossível garantir simultaneamente três propriedades: *consistência* (*Consistency*), *disponibilidade* (*Availability*) e tolerância a particionamento (*Partitioning*) (Figura 3).

Em geral, para aplicações baseadas em estratégias de escalabilidade horizontal, é necessário escolher entre consistência e disponibilidade. SGBDs tradicionais, como Oracle e MySQL, dentre outros, priorizam a consistência e disponibilidade (POKORNY, 2013). A existência do conhecido protocolo 2PC (*Two-Phase Commit*) garante consistência e atomicidade de transações. Conforme o teorema CAP, disponibilidade não pode ser garantida, e para conseguir isso é necessário relaxar a consistência. Assim, o que é escrito não será lido de forma consistente. Isso é chamado de consistência eventual ou fraca e este é o foco de vários SGBDs NoSQL como Cassandra, Dynamo e Voldemort.

Segundo o próprio Brewer (2012), não há mais uma escolha clara entre C, A ou P. O que existem são decisões em uma granularidade muito fina dentro de um sistema onde é escolhido qual o grau adequado de consistência ou disponibilidade para cada operação. Em síntese, o CAP é mais contínuo que binário.

Segundo Pritchett (2008), BDs NoSQL estão quase sempre disponíveis (*basically available*) e não estão continuamente consistentes (*soft state*), mas o sistema de armazenamento garante que, se nenhuma mudança for feita em um dado, eventualmente todas as cópias distribuídas deste dado retornarão à última informação atualizada, ou seja, se alcançará a consistência. Este novo conjunto de propriedades recebe o nome de BASE e a disponibilidade é conquistada por meio de tolerância a falhas, o que evita o comprometimento do sistema como um todo.

O movimento NoSQL cobre uma ampla gama de tecnologias, arquiteturas e prioridades. Normalmente, NoSQL diz respeito a BDs que não provem propriedades ACID e atualizações são eventualmente propagadas (seguem as propriedades BASE). Ele representa tanto uma escola de pensamento quanto uma tecnologia particular. Mesmo o nome confunde pois, para alguns, o significado é literal, ou seja, *No SQL* (Sem SQL), mas a indústria a define como *Not only SQL* (Não apenas SQL) (MILANOVIĆ; MIJAJLOVIĆ, 2012). Em suma, pode-se afirmar que o movimento NoSQL diz respeito a uma classe de BDs que não adotam o modelo relacional e não necessariamente aplicam o padrão SQL para acesso a dados, tendo como principal propósito o gerenciamento de *Big Data*.

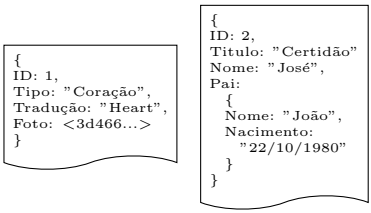
BDs NoSQL costumam ser divididos em 4 grandes categorias: *Chave/Valor*, *Colunares*, *Documentos* e *Grafos* (MONIRUZZAMAN; HOSSAIN, 2013; KAUR; RANI, 2013). Estas categorias denotam diferentes modelos de dados utilizados, conforme ilustra a Figura 4.

Figura 4: Principais modelos de dados NoSQL (Chave/Valor, Documento, Colunar e Grafo).

Chave/Valor:

Órgão	
Chave	**Valor**
1	Tipo: Coração Qtd: 1 Saúde: Boa
2	Tipo: Rim Qtd: 2

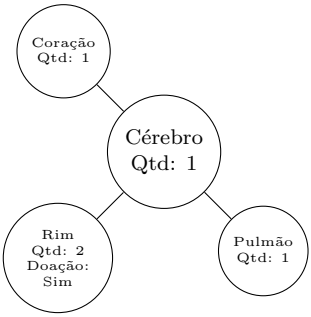
Documento:



Colunar:

Órgão			
Chave	**Colunas**		
1	Tipo: Coração	Qtd: 1	Saúde: Boa
2	Tipo: Rim	Qtd: 2	

Grafo:



Fonte: Elaborada pelo autor.

As principais características desses modelos de dados NoSQL são as seguintes (MONIRUZZAMAN; HOSSAIN, 2013):

- *Chave/valor*: armazenam itens de dados como identificadores alfanuméricos (chaves) associados a valores (noção de uma *hash table*). Os valores podem conter qualquer conteúdo (simples ou estruturado (complexo)) visto como uma caixa preta (*black box*) para o SGBD. Buscas são possíveis apenas pela indicação de um valor de chave;

- *Colunares*: aplicam uma estrutura distribuída, orientada a colunas, que acomoda múltiplos atributos por chave. Este modelo de dados é o foco deste trabalho, sendo detalhado na próxima seção;

- *Documentos*: foram inspirados pelo BD Lotus Notes (MONIRUZZAMAN; HOSSAIN, 2013), ou seja, são projetados para armazenar documentos. Estes são codificados em formatos padrão como XML, JSON (*JavaScript Object Notation* - formato mais utilizado) ou BSON (*Binary* JSON). Cada entrada possui uma chave de acesso, porém, ao contrário do modelo chave/valor, seu conteúdo (estrutura) é um conjunto de atributos com domínios simples ou complexos, sendo esta estrutura conhecida e passível de consulta e atualização pelo SGBD (*white box*);

- *Grafos*: constitui-se de um conjunto de vértices e arestas, sendo que ambos podem ter atributos. Ele é o único dos 4 modelos de dados que se preocupa com a modelagem de relacionamentos entre dados, representados por meio de arestas.

O foco deste trabalho é o projeto lógico de BDs colunares. Esta categoria de BD NoSQL é explicada a seguir.

2.3 BDS COLUNARES

BDs colunares (cujos termos sinônimos na literatura são *Extensible Record Stores*, *Wide-Column, Column-Family, Column-Oriented, Columnar Datastore* ou *BigTable-Implementation*) são assim chamados pois o menor item de informação é uma coluna. Cada coluna possui três propriedades: nome, valor e uma marca de tempo (*timestamp*), i.e., as três propriedades se repetem a cada registro e, devido a característica flexível de BDs colunares, todas essas propriedades são controladas pela aplicação, sendo a marca de tempo fundamental para o problema de concorrência, ou seja, em caso de exclusão mútua, a informação mais recente é persistida, usando a marca de tempo como referência. A marca de tempo será omitida daqui em diante por se tratar mais de um aspecto de projeto físico que lógico de BDs colunares.

Segundo Sadalage e Fowler (2013), uma boa forma de imaginar um BD colunar é como uma estrutura agregada de 2 níveis. Assim como os BDs chave-valor, o primeiro nível possui uma chave para identificar um agregado de dados de interesse. A diferença para BDs colunares é que este agregado é formado de um mapa de valores (conjunto de colunas) mais detalhados. Este segundo nível é chamado de coluna. Além de acessar todas as colunas de uma só vez, é possível acessar cada uma delas individualmente.

Esta pesquisa foca no hiato entre a modelagem conceitual e o projeto lógico de um BD colunar. Contudo, é importante entender a estrutura de dados de um BD colunar para que seja possível definir um processo adequado de projeto para ele. É preciso compreender os conceitos do modelo de dados colunar, que leva em conta um projeto distribuído de dados em vários nodos, similar a um BD distribuído tradicional.

Mathies (2015) define que uma *família de colunas* é um container de colunas ordenadas de forma léxica. É possível que uma linha possua colunas que não estejam definidas na família, tornando sua estrutura flexível e, dessa forma, frequentemente esparsa. Ainda, valores de colunas podem ser monovalorados ou multivalorados (lista de valores).

Figura 5: Representação do modelo de dados de um BD colunar.

Família de colunas								
Chave	Coluna 1		Coluna 2		...	Coluna N		
	Dado	Tempo	Dado	Tempo		Dado	Tempo	
Chave 2	Coluna 1'		Coluna 2'			Coluna N'		
	Dado	Tempo	Dado	Tempo		Dado	Tempo	
⋮					⋱			
Chave M	Coluna 1M		Coluna 2M			Coluna NM		
	Dado	Tempo	Dado	Tempo		Dado	Tempo	

Fonte: Elaborada pelo autor.

Figura 6: Representação esparsa dos dados de um BD colunar.

Pessoas						
1	Nome		Idade			
	João	12:30	25	23:00		
2	Nome				Endereço	
	Maria	09:00			Rua S/N	09:00

Fonte: Elaborada pelo autor.

Uma *chave* identifica unicamente uma linha na família de colunas. Esta linha pode determinar, de acordo com a estratégia de particionamento dos dados, em qual nó do *cluster* o dado é gravado. A mesma chave pode ser utilizada em famílias de colunas diferentes. A Figura 5 mostra a representação básica do modelo e dos dados de um BD colunar. A flexibilidade do modelo faz com que seja necessária a definição das colunas linha a linha em complemento à definição global da família.

A característica esparsa do modelo de dados de um BD colunar é evidenciada na Figura 6, visualizando os espaços vazios na representação dos dados nas colunas cuja linha não possui informação (coluna *Endereço* na linha da chave 1 e coluna *Idade* na linha da chave 2). Contudo, o comportamento observado na prática, na representação física dos dados (Figura 7), é que cada coluna de cada linha é gravada sequencialmente em ordem alfabética.

Figura 7: Representação física dos dados de um BD colunar.

Pessoas				
1	Idade		Nome	
	25	23:00	João	12:30
2	Endereço		Nome	
	Rua S/N	09:00	Maria	09:00

Fonte: Elaborada pelo autor.

Figura 8: Modelo de dados de um BD colunar que considera o conceito de super coluna.

Família de colunas									
Chave Binária	Coluna 1		**Super Coluna 1**					Coluna N	
	Dado	Tempo	Coluna C1		...	Coluna CN		Dado	Tempo
			Dado	Tempo		Dado	Tempo		

Fonte: Elaborada pelo autor.

Cassandra, um dos BDs colunares mais populares, originalmente criado pelo Facebook e agora mantido pela Fundação Apache (2009), possui algumas funcionalidades especiais, como o tipo coluna chamado de Super Coluna (*Super Columns*) e chaves compostas (*Composite Keys*). As super colunas se comportam como uma família de colunas aninhada em outra família de colunas. A diferença é que estas super colunas não são acessíveis diretamente, ou seja, é necessário recuperar a linha da família de colunas para posteriormente recuperar uma linha específica da super coluna. Um exemplo de modelo de dados com a noção de super coluna é mostrado na Figura 8. As chaves compostas são um modo de adicionar dimensões para a chave em uma família de colunas (mais de uma coluna identificando unicamente a linha), permitindo que seja desenvolvido um modo mais complexo de manipulação, persistência e gerência do modelo de dados.

A Figura 9 apresenta um exemplo onde as chaves são compostas por uma primeira (*1* e *2*) e uma segunda dimensão (*1-A*, *1-B* e *2-B*). Esta estrutura recebe o nome de *wide-rows* pois, as linhas de uma chave podem crescer indeterminadamente. Supondo uma família de colunas de `Tweets` onde a chave primária pode ser composta por `usuário` e `data/hora` do `Tweet`. Esta é uma séria candidata a receber o recurso de chave composta. A primeira dimensão seria `usuário` e a segunda `data/hora`, ou seja, uma chave dentro de uma chave.

Figura 9: Modelo de dados de um BD colunar que considera o conceito de chave composta.

Chaves compostas									
1	A				B				
	Idade		Nome		Endereço		Idade		
	25	23:00	João	12:30	Rua Nova	19:20	78	11:55	
2	B								
	Endereço		Nome						
	Rua S/N	09:00	Maria	09:00					

Fonte: Elaborada pelo autor.

Tabela 1: Conceitos de um modelo de dados colunar presentes em SGBDs colunares.

Recurso	BD Colunar	Cassandra	Riak	HBase	DynamoDB	Accumulo	Teradata	SybaseIQ
Coluna tipo coleção		●	●	●	●	◐	●	●
Estrutura flexível		●	●	●	●	●	●	○
Chave composta		●	○	○	●	○	●	●
Super coluna		●	○	○	○	○	○	○

Fonte: Elaborada pelo autor.

Em suma, ressalta-se que não há norma ou padrão, além da arquitetura de armazenamento, de quais conceitos um modelo de dados de um BD colunar deve possuir. Isso é comprovado por meio da Tabela 1, que mostra quais conceitos são considerados pelos principais SGBDs colunares. Esta tabela indica 3 tipos de informação: possui o conceito ●, não possui o conceito ○ e não possui o conceito, mas ele pode ser adaptado ou possui o conceito de forma limitada ◐.

BDs colunares podem ser particionados tanto vertical quanto horizontalmente (CATTELL, 2010). Devido a essa facilidade, ao seu uso crescente no mercado e também pelo seu modelo permitir simples acesso aos dados por meio das chaves e possibilitar, mesmo que de forma limitada, a modelagem de objetos complexos, ele foi escolhido como foco deste trabalho. O processo de projeto lógico proposto considera todos esses possíveis conceitos de um modelo de dados colunar na conversão de uma modelagem conceitual EER.

Segundo Sadalage e Fowler (2013), boas aplicações para BDs colunares são armazenamento de eventos (*logs*), sistemas de gerenciamento de conteúdo, páginas pessoais, blogs etc. Os BDs colunares não são boas alternativas quando o escopo de um sistema não está claro e mais propenso a ajustes frequentes na modelagem. Isto ocorre devido ao alto custo associado a mudanças estruturais no esquema, que podem afetar um grande volume de dados já presente no BD. Apesar do esquema ser extremamente flexível, mudanças tendem a denegrir o desempenho do sistema. Esse fato reforça a necessidade de um projeto conceitual e lógico bem definido, com um processo de conversão formalizado e claro.

3 TRABALHOS RELACIONADOS

Este capítulo apresenta os pontos principais de alguns trabalhos relacionados com o objetivo deste estudo, bem como seu diferencial e potenciais contribuições.

Além da metodologia clássica para projeto de BDs relacionais (HEUSER, 2008; NAVATHE; BATINI; CERI, 1992), algumas propostas são encontradas para o mapeamento de modelagens conceituais para lógicas de BDs não-convencionais, como é o caso de BDs XML (SCHROEDER; MELLO, 2008), BDs orientados a objeto (FONG, 1995) e BDs NoSQL (BUGIOTTI et al., 2014; CHEBOTKO; KASHLEV; LU, 2015). Existem diversas orientações de como modelar diretamente algumas estruturas lógicas em BDs colunares (SCHRAM; ANDERSON, 2012; WANG; TANG, 2012; CABIBBO, 2013; KAUR; RANI, 2013; SHARP et al., 2013). O que se vê nesse cenário é a falta de uma abordagem clara ou completa que transforme uma modelagem conceitual, como é o caso do modelo EER, em uma modelagem lógica direcionada a BDs colunares. Essa justificativa evidencia a originalidade da proposta desta pesquisa.

Schroeder e Mello (2008) propõe uma abordagem para mapeamento de uma modelagem conceitual EER para uma lógica equivalente no formato XML. Neste processo são aplicadas regras para conversão de todos os conceitos do EER: entidades, relacionamentos, atributos simples, atributos especiais, agregação, especialização/generalização, dentre outros. Além disso, o processo de conversão considera estimativas de acesso do BD (*workload*) para gerar estruturas XML otimizadas. Apesar de não ser um trabalho fortemente relacionado, pois o modelo de BD é diferente, a metodologia utilizada pode servir como guia para o desenvolvimento deste trabalho, pois ambos os modelos (XML e colunar) podem ser empregados para representar objetos complexos. O mesmo vale para a abordagem de (FONG, 1995), cujo foco é a modelagem de BDs orientados a objetos e a utilização do EER no mapeamento conceitual-lógico. Contudo, a abordagem não considera informações de carga para apoiar este processo.

Bugiotti et al. (2014) propõe uma solução de projeto de BDs para qualquer modelo de dados NoSQL. A base da sua solução é o que eles chamam de modelo de dados abstrato, ou *NoAM*. O processo deles flui por meio da modelagem conceitual, um esquema de objetos agregados em NoAM e sua implementação. Mesmo a proposta atingindo as 3 fases de projeto de BD, eles não enfatizam BDs colunares, não consideram todos os construtores conceituais nem formalizam a conversão do modelo conceitual para uma representação lógica no NoAM. Chebotko, Kashlev e Lu (2015) apresentam uma abordagem similar e mais completa, que abrange desde a modelagem conceitual ER (sem as extensões presentes no EER) até a modelagem física, propondo, inclusive, um esquema lógico baseado no modelo do BD colunar Cassandra. A sua modelagem lógica é dirigida por consultas, que é um item fundamental em seus mantras: *conheça seus dados, conheça suas consultas, utilize agregados e duplique dados*. Eles não executam experimentos de avaliação, mas afirmam que a abordagem já é amplamente utilizada em ambientes de produção. A abordagem desta proposta difere da deles por não ser orientada a agregados e por apresentar experimentos que demonstram a sua viabilidade.

Com relação à indústria, onde o movimento NoSQL surgiu, a Microsoft apresenta instruções detalhadas para a criação de um BD colunar enfatizando otimizações para gravação e leitura, mostrando exemplos de estruturas e situações análogas à relacional e suas desvantagens (SHARP et al., 2013). Estas orientações são complementadas com o conceito de *wide-column*, que é o equivalente a transpor uma tabela, ou seja, ela acaba por possuir mais colunas que linhas. Orientações para indexação e para maximizar escalabilidade, disponibilidade e consistência também são dadas. Entretanto, o foco é uma modelagem lógica baseada nas principais consultas da aplicação e não em estruturas conceituais.

O trabalho de Schram e Anderson (2012) apresenta um estudo de caso sobre um sistema no qual o crescimento no volume de acessos e armazenamento de dados tornou a abordagem relacional original (no caso, um SGBD MySQL) incapaz de atender a demanda, principalmente em termos de disponibilidade. Assim sendo, optou-se pela utilização de um BD NoSQL colunar, no caso, o Cassandra. Todo o passo a passo desse estudo, construção, modelagem e desafios são discutidos. O projeto adapta um sistema de coleta e extração de informações sobre crises chamado EPIC. Informações do Twitter são coletadas desde 2009, sendo apresentada a transição relacional para NoSQL e os resultados obtidos. O domínio do problema é baseado no Twitter. Ele possui basicamente as entidades *Eventos*, *Tweets* e *Usuários*, sendo modelado de forma que um usuário possui *tweets* que pertencem a um ou mais eventos, ou seja, um relacionamento `N:M` entre usuários e eventos, cujas associações são *tweets*. Para a conversão, faz-se uso da desnormalização na família de colunas *Tweets* com o objetivo de agrupar informações importantes do usuário e associá-lo a seus eventos, pois as informações não são alteradas com frequência. Como estudo de caso, seu processo mais prático serve como referência, mas difere desta abordagem por partir da modelagem lógica, não possuir um conjunto claro de regras de conversão nem resultados que possam validar a proposta.

De forma similar, mas apenas com a intenção de apresentar o contraste entre o relacional e o não-relacional, o trabalho de Wang e Tang (2012) também apresenta princípios simples de modelagem conceitual utilizando UML e a conversão destas modelagens conceituais, sem grande detalhamento, para o modelo de dados colunar do Cassandra. Ele difere da nossa proposta por usar uma modelagem conceitual e lógica muito restrita sem apresentar regras de conversão claras nem uma validação da proposta. Porém, é direcionado ao BD colunar Cassandra.

Kaur e Rani (2013) apresentam uma modelagem conceitual (ER) para um estudo de caso de comentários em *posts* de *blogs*. Para este estudo de caso é realizada uma conversão para uma modelagem lógica direcionada para o MongoDB (segue o modelo de dados de documento) e outra para o Neo4j (segue o modelo de dados em grafo). O foco não é a conversão em si, mas a otimização deste modelo para consulta, demonstrando parcialmente seus resultados. Além disso, ele difere da nossa proposta por não possuir completude no processo de conversão conceitual-lógico e não ser direcionado a BD colunar.

Analogamente, o trabalho de Meijer e Bierman (2011) apresenta um modelo matemático para BDs NoSQL Chave/Valor e demonstra sua correlação com o modelo relacional (chaves primárias e estrangeiras), chamado de *CoSQL*. Ele também apresenta uma simples generalização da álgebra relacional sobre conjuntos que define uma linguagem comum de consulta para BDs relacionais e não relacionais com o objetivo de criar uma camada lógica sobre BDs não-relacionais sendo que, para isso, precisa de construtores para representar o relacionamento em BDs Chave/Valor. A sua proposta para criar uma camada lógica serviu de inspiração para justificar nosso processo de conversão, demonstrando por meio de álgebra relacional a correlação entre o modelo conceitual e o lógico de BDs colunares. Difere da nossa proposta por não apresentar uma conversão, mas sim uma correlação entre as estruturas dessas camadas.

A Tabela 2 apresenta um comparativo dos trabalhos relacionados encontrados. Ela não tem por objetivo expor suas deficiências, mas principalmente indicar as contribuições desta pesquisa em relação a eles, uma vez que o tema é recente e carece ainda de soluções efetivas. Ela exibe 5 características: *Modelagem conceitual* indica a consideração de pelo menos um modelo conceitual no projeto dos dados. *Projeto lógico* indica que há algum tipo de representação lógica presente no trabalho. *Regras de conversão* indica se são definidas orientações claras de como transformar a modelagem conceitual na modelagem lógica. *BD colunar* serve para identificar se o projeto lógico é voltado para BDs colunares e, por fim, se houve *avaliação experimental da proposta*. A cada uma destas categorias foi atribuído três níveis: (i) demonstra completamente o conceito ●; (ii) não demonstra o conceito ○ e; (iii) demonstra parcialmente o conceito ou apresenta conceito equivalente ◐.

Conforme destacado na Tabela 2, não existe até o momento um trabalho que contemple conjuntamente o projeto lógico de um BD colunar desde a análise detalhada dos conceitos de uma modelagem conceitual, a formalização e aplicação de um conjunto de regras de conversão enriquecida por informações de acesso, a geração de um esquema lógico de dados direcionado ao modelo colunar e uma avaliação experimental que ateste a sua viabilidade. A formalização de regras de conversão para todos os construtores do modelo EER, a definição de um processo de conversão detalhado que aplica essas regras e sua avaliação são as principais contribuições deste trabalho. Esta proposta é apresentada no próximo capítulo.

Tabela 2: Comparativo dos trabalhos relacionados

Característica	Schroeder e Mello (2008)	Navathe, Batini e Ceri (1992)	Sharp et al. (2013)	Schram e Anderson (2012)	Wang e Tang (2012)	Kaur e Rani (2013)	Meijer e Bierman (2011)	Bugiotti et al. (2014)	Chebotko, Kashlev e Lu (2015)	Esta dissertação
Modelagem conceitual	●	●	○	○	◐	◐	◐	●	◐	●
Projeto lógico	●	●	●	●	◐	◐	◐	●	●	●
Regras de conversão	●	●	○	○	○	◐	○	●	●	●
BD colunar	○	○	●	●	●	○	○	◐	●	●
Avaliação da proposta	●	●	○	◐	○	◐	●	●	◐	●

Fonte: Elaborada pelo autor.

4 PROPOSTA

Este capítulo detalha uma abordagem para modelagem lógica de BDs colunares por meio da proposta de um processo de conversão de modelagens conceituais EER para o modelo lógico de um BD colunar que considera todos os conceitos apresentados na seção 2.3. A escolha por esta categoria de BD NoSQL se justifica por questões de limitação de escopo da dissertação e também pelo fato de não existir na literatura uma metodologia formal e detalhada para a conversão de uma modelagem conceitual para um esquema lógico neste modelo de BD.

Primeiramente é introduzida uma notação para a modelagem lógica de um BD colunar. Em seguida, um processo completo de conversão de um esquema conceitual EER para o esquema lógico de um BD colunar é proposto e detalhado por meio de algoritmos de alto nível, inspirado por Elmasri e Navathe (2005).

A estratégia geral do processo de conversão é prover um esquema lógico com conjuntos de dados que suportem famílias de colunas aninhadas e associações bidirecionais. A primeira é atingida através do uso de *chaves compartilhadas*, que provêm melhor eficiência de acesso já que dados com uma associação que pode ser definida através de uma hierarquia (relacionamentos `1:1` e `1:L` - explicados na sequência) podem ser armazenados próximos um dos outros. A segunda é alcançada quando a primeira não pode ser aplicada, através de *relações artificiais*. Ambos os conceitos são detalhados adiante. A decisão de não utilizar a abordagem tradicional de agregados para BDs NoSQL, seguida por vários trabalhos relacionados, ocorreu em virtude da possibilidade de que a abordagem de agregados pode se tornar ineficiente quando vários níveis de aninhamentos estão presentes, tanto em termos de operações de leitura quanto de escrita, para certos domínios de aplicação. Isso é ilustrado experimentalmente no capítulo 5.

4.1 NOTAÇÃO LÓGICA PARA BDS COLUNARES

Segundo Heuser (2008), uma modelagem conceitual indica quais dados do domínio devem ser considerados no BD, mas não como eles estão armazenados. Por outro lado, quando se deseja um nível de abstração na visão do usuário de um SGBD, temos o modelo lógico. Denomina-se *projeto lógico* a geração de um esquema adequado ao modelo de dados de um BD alvo (modelo lógico) que pode se basear em uma modelagem conceitual de alto nível.

A maximização do desempenho de acesso para as principais consultas e a minimização do volume de armazenamento são os principais objetivos do projeto lógico de um BD. Para atender o primeiro requisito, é necessário deixar os dados fortemente relacionados próximos uns dos outros e, para atender o segundo objetivo, é necessário reduzir redundâncias e estruturas subutilizadas. Isso normalmente gera um dilema, que consiste em um desafio a superar.

Uma notação clara para projeto lógico pode ajudar na resolução desse dilema, através de um outro aspecto relevante, que é a apresentação do esquema ao analista, de forma a destacar a estrutura lógica dos dados. Desta forma, o analista pode visualizar com antecedência possíveis fatores que denegririam o desempenho ou dificultariam a implementação do BD. Afinal, um BD colunar não é como um BD relacional em que as colunas se agrupam em uma linha, nem como um BD orientado a objetos em que, abstraindo detalhes de organização, os atributos de uma instância estão próximos uns dos outros.

Durante um processo de modelagem lógica, é importante identificar potenciais ofensores de desempenho, tanto através do processo de conversão quanto pela experiência do analista (apresentando um esquema representativo de como os dados estarão ligados e serão lidos/gravados). Isso evita más decisões de modelagem pois, diferentemente dos BDs relacionais, não é tão fácil mudar o padrão de consulta com o sistema já em produção. Nestes, a adição, por exemplo, de um índice permite alterar drasticamente o padrão de acesso a uma tabela e melhorar ou piorar consideravelmente seu desempenho de acordo com o plano de execução escolhido pelo SGBD. Em BDs colunares isso ainda não é possível e não será eficiente, por exemplo, filtrar dados através de uma coluna que não possua índice sem que haja uma implementação por parte do desenvolvedor.

A conversão de esquemas conceituais em esquemas lógicos é uma transformação entre modelos de dados com diferentes níveis de abstração. Como não há um padrão para BDs colunares, este trabalho define uma notação lógica com esse objetivo, bem como os conceitos utilizados nela e no processo de conversão. Essas definições são dadas a seguir.

Definição 1 (Coluna)**.** *Uma coluna* `C` *é uma tupla* $C =< n, v >$*, onde* n *é o nome da coluna e* v *é o valor da coluna.*

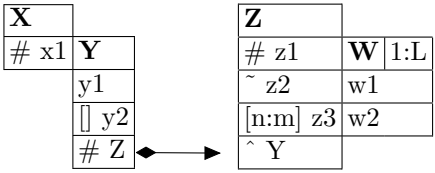
Definição 2 (Família de colunas)**.** *Uma família de colunas é uma função de mapeamento* $f : A \mapsto B$ *onde* A *é um conjunto de chaves únicas e* B *é um conjunto de colunas tal que, para cada* $\alpha \in$ `A`*, há um único subconjunto* `B'` $\subseteq$ `B`*.*

Definição 3 (Chave compartilhada)**.** *Considere um esquema lógico* `E`*, sendo* $F \in E$ *uma família de colunas e* $F' \in E$ *outra família de colunas. Uma chave é dita compartilhada quando um mesmo valor de chave é utilizado como identificador em mais que uma família de colunas, ou seja,* $chave(F) = chave(F')$*.*

Definição 4 (Associação artificial)**.** *Considere um esquema lógico* `E`*, sendo* $F \in E$ *uma família de colunas e* $F' \in E$ *outra família de colunas. Uma associação artifical existe se algum atributo em uma família de colunas corresponde à chave de outra família de colunas, ou seja, dado um* $c_i \in F$*,* $c_i.v = chave(F')$*.*

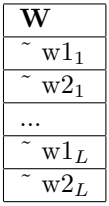
A Figura 10 apresenta a notação lógica de um esquema de um BD colunar proposta por este trabalho. A família de colunas é representada na forma de um retângulo com o nome da família no topo e, ao seu lado, opcionalmente, uma cardinalidade. Essa restrição de cardinalidade identifica que as colunas se repetirão dinamicamente dentro da família conforme a cardinalidade (um exemplo é a família de colunas `W`). As colunas (ou atributos) são representados por linhas dentro do retângulo de uma família de colunas. Elas podem iniciar por um identificador de chave (cerquilha: #), obrigatoriedade (til: ˜), associação artificial (circunflexo: ˆ) ou coluna do tipo coleção (abre e fecha colchetes: []) com uma cardinalidade opcional (se oculto, assume-se [0:n]). Tipos de dados não são descritos nessa versão da notação lógica, pois BDs NoSQL suportam virtualmente qualquer tipo de dado.

Figura 10: Visão geral da notação diagramática proposta para representação de um esquema de um BD colunar.



Fonte: Elaborada pelo autor.

Figura 11: Exemplo de uma ocorrência de família de colunas com cardinalidade máxima maior que zero.



Fonte: Elaborada pelo autor.

Na cardinalidade da família, caso exista, as colunas podem variar de N a M. Se a cardinalidade mínima for zero, pode não existir registro nesta família. Para evitar se ater apenas a uma tecnologia (como as super colunas do BD Cassandra), esta é uma estrutura dinâmica e artificial (não é nativa de algum BD colunar) de criação de colunas que depende da flexibilidade do esquema presente no BD colunar alvo. É possível visualizar uma representação do esquema, na Figura 11, que exemplifica uma Família de colunas chamada `W` com cardinalidade 1:L.

O relacionamento do tipo `1:L`, também introduzido neste trabalho, é similar a um relacionamento com cardinalidade `1:N` no que se trata de limite superior, ou seja, o limite superior não é conhecido, contudo, subentende-se que esse limite não é alto. Normalmente, o lado `L` é associado a entidades fracas em uma modelagem EER, como dependentes de funcionários, lista de contatos para uma pessoa, entre outros. Este relacionamento é definido através de chaves compartilhadas, como é o caso da família `W` aninhada à família `Z`. O objetivo deste conceito é aninhar dados fortemente relacionados permitindo a utilização do conceito de chave compartilhada (com `1:N`, isso não é possível) e otimização do esquema.

A chave compartilhada é o reuso de uma chave de outra família de colunas. Ela é representada pelo aninhamento de duas famílias de colunas, de forma a demonstrar esse compartilhamento de chave entre elas, ou seja, a família de colunas aninhada não possui chave e compartilha a chave da família de colunas de nível hierarquicamente superior. Um exemplo ocorre entre as famílias de colunas `X` e `Y`, ou seja, a família `Y` compartilha (está aninhada) a chave da família `X`.

Já uma associação artificial denota um relacionamento de referência entre famílias de colunas. Um exemplo é a chave de `Z` presente na família `Y`, conforme mostra a Figura 10. A associação artificial entre elas é representada por uma linha cujas pontas são setas e losangos. As setas representam a existência de um campo com associação artificial (^) e o losango denota agregação de uma família na outra através da referência. Considera-se, neste trabalho, que uma associação artificial sempre é bidirecional devido a limitações, em estruturas de indexação por exemplo, que inviabilizam rastreamento eficiente dessa informação em BDs colunares.

4.2 PROCESSO DE CONVERSÃO

A representação lógica proposta na seção anterior é usada com o objetivo de identificar um cenário aplicável de alternativas de mapeamento para as construções existentes no EER para BDs colunares. Uma modelagem conceitual simples, como um relacionamento `1:1`, pode gerar diversas alternativas de representação em uma modelagem lógica. Portanto, é necessário avaliar e descartar alternativas ineficientes de mapeamento com relação a aspectos de desempenho de acesso e de armazenamento.

É importante salientar que, devido à natureza flexível de representação de dados e o controle de consistência fraco presente em BDs colunares, alguns tipos conhecidos de restrições estão ausentes. A mais evidente é a inexistência de integridade referencial. Tratamentos desse tipo devem ser controlados pela aplicação. Schwartz (2015) argumenta que a ausência de um esquema em um BD é uma falácia. Um esquema sempre existe, se não é reforçado pelo BD, a aplicação assume o controle da integridade dos dados, ou seja, a consistência do esquema fica a cargo da aplicação. Neste contexto, conceitos introduzidos anteriormente, como chave compartilhada, são utilizados para lidar com a ausência de integridade referencial em BDs colunares. Esta estratégia reduz o número de referências físicas entre famílias de colunas (uma vez que ocorrências de uma família estão aninhadas a ocorrências de outra família, compartilhando a chave da família de nível superior) e, como consequência, a sobrecarga de gerência para manter o BD íntegro.

Esta seção apresenta os algoritmos de alto nível para a conversão de construtores do modelo EER para a representação lógica de BDs colunares definida na seção 4.1. Estes algoritmos são baseados em uma noção de paternidade entre entidades definida a seguir.

Definição 5 (Paternidade entre entidades)**.** *Dadas duas entidades E_P e E_F, define-se que E_P é pai de E_F (ou, em outras palavras, E_F é filho de E_P), se: (a) E_F é uma entidade especializada e E_P é a entidade genérica em um relacionamento de generalização; (b) E_P é a entidade que une duas ou mais entidades em um relacionamento do tipo união, sendo E_F uma das entidades unidas; (c) E_P é a entidade do lado obrigatório e com cardinalidade máxima igual a 1 em um relacionamento do tipo* `1:1`*,* `1:N` *ou N-ário.*

Em resumo, o processo proposto percorre de forma recursiva todas as entidades e, para cada uma, verifica se ela atua como filha em um relacionamento, segundo esta noção de paternidade recém-definida. Se existir, a entidade pai é priorizada, ou seja, é convertida primeiro. Esta verificação e priorização é repetida até que não haja mais essa noção de paternidade. Quando não houver mais relacionamento paterno a percorrer, inversamente, as entidades são convertidas em famílias de colunas e suas chaves e colunas são definidas. Este processo é detalhado formalmente na sequência.

A recursividade do processo rege a decisão da entidade inicial (varredura em profundidade) e, a partir dela, todos os relacionamentos filhos são processados (varredura em largura). Esta última pode gerar um problema de grafo cíclico (supondo uma modelagem EER vista como um grafo, onde as entidades são nodos e os relacionamentos são arestas), que é resolvido utilizando técnicas de coloração de grafos, ou seja, marca-se as entidades já processadas para evitar que sejam analisadas novamente.

Chaves compartilhadas são elementos fundamentais à nossa abordagem. O mecanismo de priorização da conversão da entidade pai garante aninhamentos mais adequados de famílias de colunas, uma vez que nunca uma entidade filha será criada sem que seja ao menos considerado se é possível compartilhar a chave com sua entidade pai. Nesta primeira versão do processo proposto, a ordem de processamento das filhas não é relevante, pois cada relacionamento é avaliado independentemente, podendo gerar novas chaves compartilhadas em cada etapa da conversão. Cabe ressaltar que essa não é a única alternativa possível para geração do esquema lógico. Um dos trabalhos futuros imediatos é considerar informações de carga para otimizar mais ainda o esquema lógico gerado.

O algoritmo geral que implementa o processo de conversão, bem como outros algoritmos importantes chamados a partir dele, são detalhados a seguir.

4.2.1 Algoritmo geral de mapeamento do esquema conceitual para o esquema lógico

O processo de mapeamento geral é regido pelo Algoritmo 1. Sua entrada é um esquema conceitual representado no modelo EER. Inicialmente, todas as entidades do esquema EER da entrada são percorridas (linha 2) e, para cada uma, o Algoritmo 2 (`criarFamilia`) é acionado (linha 3), sendo sua saída adicionada ao conjunto de famílias de colunas que compõem o esquema lógico para BDs colunares, que é a saída do algoritmo (linha 5).

Algoritmo 1: Conversão de um esquema conceitual EER para um esquema lógico de um BD Colunar

Entrada: Esquema EER (α)
Saída: Esquema lógico para um BD colunar (α')

```
1 α' ← ∅
2 para cada ϵ ∈ α|ϵ é uma entidade faça
3 |  α' ← α' ∪ criarFamilia(ϵ)
4 fim
5 retorna α'
```

Quando uma entidade é visitada pelo laço, se há relacionamento de paternidade, a entidade pai é convertida antes, de forma recursiva (ver Algoritmo 2). Cada entidade convertida é marcada para evitar duplicidade de conversão. O laço neste algoritmo objetiva alcançar todas as entidades do esquema EER, mesmo aquelas que são parte de grupos disjuntos de entidades relacionadas. Um exemplo geral é dado a seguir, ilustrando a conversão de cada construtor.

Exemplo 1 (Conversão de esquema lógico)**.** *A conversão do esquema na Figura 1 gera o esquema lógico colunar na Figura 16*[1] *por meio da aplicação do Algoritmo 1.*

4.2.2 Geração de famílias de colunas

Como regra geral do mapeamento de entidades, para cada entidade em um esquema EER é criada uma família de colunas (Algoritmo 2), não havendo diferenciação entre entidades fracas e fortes. Em Heuser (2008), entidades fracas são entidades dependentes e só existem quando relacionadas a outra entidade. Segundo ele, autores recentes preferem não utilizar esse conceito, pois, dependendo da realidade modelada, a entidade pode ser fraca e ao mesmo tempo central ao modelo. Assim, é admitido que se ela está conceitualmente separada, é importante que existam duas representações lógicas, ou seja, duas famílias de colunas, uma para ela e outra para a entidade forte da qual ela depende.

[1] A Figura 16 está posicionada no final da seção seguinte, onde este processo completo é simulado, sendo ela referenciada diversas vezes ao longo do texto.

A entrada do Algoritmo 2 pode ser uma entidade ou um relacionamento e sua saída é um conjunto de famílias de colunas. Para cada entidade analisada, este algoritmo provê a conversão de todos os seus construtores conceituais (relacionamentos, hierarquias de generalização ou união e atributos). O mesmo acontece se a entrada for um relacionamento.

Primeiramente, é inicializado o conjunto de famílias de colunas da saída (linha 1). Se a entidade ou relacionamento não foram visitados (linha 2), então uma lista de conceitos EER é criada com as generalizações, uniões e relacionamentos promovidos à entidades associativas onde a entrada é filha (linha 3). Se a entrada é um relacionamento, é assumido que ele não possui outros relacionamentos, permanecendo essa lista vazia. Então, para cada conceito (linha 4), o mesmo algoritmo é acionado recursivamente, tendo a entidade pai como entrada (linha 5). Em seguida, uma família de colunas é criada (linha 7) com um nome (linha 8) e uma chave (ver Algoritmo 3). Na sequência, para cada atributo não identificador da entrada (linha 10), Algoritmo 5 é acionado (linha 11) para definir uma coluna (ou família de colunas agregada) para a família de colunas recém criada. A nova família de colunas é adicionada à saída (linha 13) e, em seguida, é percorrida a lista criada na linha 3, bem como relacionamentos reflexivos ou `N:M` da entrada (linha 14). Então, para cada relacionamento é acionado o Algoritmo 6 que recebe o relacionamento e a nova família de colunas como entrada. Enfim, o resultado dessa conversão é adicionado à saída (linha 15) e o então a mesma é retornada ao algoritmo acionador (linha 18).

Cabe ressaltar que relacionamentos são percorridos duas vezes pelo algoritmo: uma primeira vez para identificar e priorizar a paternidade (linha 4) e uma segunda vez para criar as estruturas lógicas de associação necessárias (linha 14). Relacionamentos reflexivos são tratados apenas na segunda passada pois, na primeira passada, a entidade associada é a mesma sendo processada.

Algoritmo 2: *criarFamilia*
CONVERSÃO DE ENTIDADE OU RELACIONAMENTO PARA FAMÍLIA DE COLUNAS

Entrada: Entidade ou relacionamento (ϵ)
Saída: Conjunto de famílias de colunas (ω)

$\omega \leftarrow$ Família de colunas previamente gerada por ϵ ou conjunto vazio
se *VerificarEMarcarSePrimeiraVisitaA(ϵ)* **então**
  $\pi_P \leftarrow$ generalizações, uniões, relacionamentos pais N-ários ou binários de ϵ
  para cada $\pi \in \pi_P$ **faça**
    $\omega \leftarrow \omega \cup criarFamilia$(Entidade pai em π)
  fim
  $\epsilon' \leftarrow$ Nova família de colunas
  $\epsilon'.Nome \leftarrow \epsilon.Nome$
  $\epsilon'.Chave \leftarrow definirChave(\epsilon)$
  para cada $\delta \in \epsilon | \delta$ *é atributo não identificador* **faça**
    $\omega \leftarrow \omega \cup converterAtributo(\epsilon', \delta)$
  fim
  $\omega \leftarrow \omega \cup \epsilon'$
  para cada $\pi \in \epsilon | \pi \in \pi_P \vee \pi.Tipo \in \{reflexivo, N:M\}$ **faça**
    $\omega \leftarrow \omega \cup converterRelacionamento(\pi, \epsilon')$
  fim
fim
retorna ω

Exemplo 2 (Geração de família de colunas)**.** *Considerando o esquema EER da Figura 1, a conversão da entidade* `A` *gera uma família de colunas homônima com colunas equivalentes. Em seguida, na conversão da entidade* `B`*, na busca por entidades pais, é identificada uma generalização total e disjunta. Como a entidade pai* `A` *já está convertida, é criada uma família de colunas* `B` *e uma chave compartilhada para associá-los. O mesmo ocorre com as entidades* `C` *e* `D`*.* `R1` *e* `R2` *são tratados pela conversão de relacionamentos, detalhado mais adiante, quando a conversão dos pais das entidades* `E` *e* `H` *é alcançada. A Seção 4.3 apresenta como a recursão ocorre (iniciando no Algoritmo 1 e atravessando os Algoritmos 2 e 6) para definir corretamente a ordem de conversão para as entidades e relacionamentos. Um bom exemplo de como a ordem de priorização foi* ajustada *pela recursividade é a entidade* `F` *que, alfabeticamente, seria alcançada antes de* `H`*, porém, pela definição de paternidade,* `H` *é priorizada.*

4.2.3 Geração de chaves compartilhadas

O Algoritmo 3 é responsável por gerar chaves compartilhadas. Ele recebe como entrada uma entidade ou relacionamento e retorna um conjunto de atributos que compõem a chave. Inicialmente, a saída recebe a chave da entrada (linha 1). Se não existe uma chave (linha 2), então, se a entrada é uma entidade (linha 3), é construída uma lista de generalizações e relacionamentos mandatórios `1:L` ou `1:1`, nesta ordem (linha 4), sendo o primeiro item da lista usado na variável que corresponde à saída (linha 5). O primeiro item é escolhido com o intuito de utilizar o relacionamento com maior potencial de cardinalidade, para que o conceito de chave compartilhada seja melhor empregado. Se a entrada for um relacionamento (linha 6), a chave é o lado do relacionamento com cardinalidade máxima e mínima igual a 1 (linha 7). Finalmente, se a variável que corresponde à saída do algoritmo estiver vazia (linha 9), uma coluna identificadora é criada e utilizada como chave (linha 10).

Cabe ressaltar que a chave retornada por esse algoritmo é aplicada à família de colunas sendo processada. Se a família já possuir um identificador único, o mesmo é utilizado como chave. No caso de chave compartilhada (linhas 4-5), a chave corresponde à chave da família pai escolhida pela primeira ocorrência dentre generalizações, uniões com totalidade, relacionamentos `1:L` e `1:1` com paternidade obrigatória com a entidade sendo convertida, nesta ordem. Se a entrada do algoritmo for um relacionamento, somente haverá chave compartilhada se houver um lado pai com cardinalidade `1:1`. Caso não seja encontrada nenhuma possibilidade de chave compartilhada nem a existência de um identificador único, define-se uma chave.

Algoritmo 3: *definirChave*
DEFINIÇÃO DA CHAVE

Entrada: Entidade ou relacionamento (ϵ)
Saída: Atributos que compõem a chave (ω)

```
1  ω ← ε.Key
2  se ω = ∅ então
3  |  se ε é entidade então
4  |  |  π ← Generalizações, uniões, relacionamentos 1 : L e
   |  |    1 : 1 com paternidade obrigatória de ε,
   |  |    respectivamente
5  |  |  ω ← π1.Key
6  |  senão
7  |  |  ω ← chave do lado (1,1) de ε, se existir
8  |  fim
9  |  se ω = ∅ então
10 |  |  ω ← {ID}
11 |  fim
12 fim
13 retorna ω
```

Exemplo 3 (Geração de chave compartilhada)**.** *Considere como início a conversão da entidade* `A` *na Figura 1. Como ela possui um atributo identificador (*`id1`*), o mesmo é definido como chave da família de colunas correspondente. Como* `A` *é uma entidade pai de* `B`*,* `C`*,* `D` *e* `E` *através de* `R1` *e* `R3`*, todos eles compartilham a mesma chave, com exceção de* `E`*, como demonstrado na Figura 16. Outros exemplos são* `E` *com* `E-a2`*,* `F` *com* `F-H` *e* `G` *com* `G-H`*.*

4.2.4 Geração de associações artificiais

Quando a geração de uma chave compartilhada não é possível para estabelecer um relacionamento entre famílias de colunas, como por exemplo, para a conversão de relacionamentos `N:M` e generalizações parciais, uma associação artificial é definida. O termo artificial indica que esta é uma referência cuja integridade precisa ser gerenciada pela aplicação.

O Algoritmo 4 gera associações artificais. Ele é acionado pelo Algoritmo 6, porém, é declarado próximo à definição de chave compartilhada para facilitar a compreensão do processo. Ele é responsável por definir um relacionamento artificial entre famílias de colunas em um esquema lógico de um BD colunar através da geração de famílias de colunas adicionais. Estas famílias de colunas são definidas como *Auxiliares* (quando compartilham a chave com seu pai) ou *Intermediárias* (quando são famílias de colunas independentes referenciadas por uma família de colunas auxiliar).

A entrada desse algoritmo é composta de duas famílias de colunas (primeira e segunda) e o relacionamento entre elas. Tendo como saída as famílias de colunas adicionais necessárias para a definição da associação artificial. O algoritmo é dividido em duas partes: *(i)* a definição da família de colunas origem e, *(ii)* a criação da associação artificial.

Para definir a família de colunas origem é verificado se a primeira família foi criada a partir de um relacionamento (linha 2). Se sim, significa que ela já foi convertida e deve ser usada como família auxiliar (ver exemplo 4), sendo ela definida como a família origem (linha 3). Se não (linha 4), o algoritmo procura por uma família de colunas que foi criada para ligar as duas famílias de colunas (ver exemplo 10) e usa ela como a família origem (linha 5). Depois disso, se não foi encontrada uma família origem (linha 7), uma entidade temporária é criada (linha 8) com um relacionamento `1:1` com a entidade que gerou a primeira família de colunas da entrada e o resultado da conversão da entidade temporária é definida como a família origem (ver exemplo 5).

Algoritmo 4: *criarAssociacaoArtificial*
Criação de associação artificial

Entrada: Duas famílias de colunas $(\epsilon_1; \epsilon_2)$ e o relacionamento (π)
Saída: Famílias de colunas adicionais (ω)

1 $\omega \leftarrow \emptyset$
2 **se** ϵ_1 *foi criada para um relacionamento* **então**
3 | $\epsilon' \leftarrow \epsilon_1$
4 **senão**
5 | $\epsilon' \leftarrow$ Família de colunas pré-existente entre ϵ_1 e ϵ_2
6 **fim**
7 **se** $\epsilon' = \emptyset$ **então**
8 | $\epsilon_T \leftarrow$ Entidade temporária com nome composto pelos nomes da entrada associada com ϵ_1 através de relacionamento $1:1$
9 | $\epsilon' \leftarrow criarFamilia(\epsilon_T)$
10 | $\omega \leftarrow \epsilon'$
11 **fim**
12 $\delta_1 \leftarrow$ Nova coluna chave
13 $\delta_1.Nome \leftarrow \epsilon_2.Nome$
14 $\epsilon' \leftarrow \epsilon' \cup \delta_1$
15 **se** $\pi.Tipo \neq (N:M) \vee \pi$ *promovido para entidade associativa* **então**
16 | $\delta_2 \leftarrow$ Nova coluna que representa uma associação artificial
17 | $\delta_2.Nome \leftarrow \epsilon_2.Nome$
18 | $\epsilon_2 \leftarrow \epsilon_2 \cup \delta_2$
19 **fim**
20 **retorna** ω

A definição de família de colunas origem é importante pois uma associação artificial nunca ocorre diretamente entre as duas famílias, ou seja, sempre haverá uma família de colunas auxiliar e talvez uma intermediária entre ambas. Inicialmente, o algoritmo procura (linhas 2-6) por uma família de colunas auxiliar pré-existente (por exemplo, criada na iteração sobre o primeiro relacionamento quando duas entidades possuem mais de um relacionamento entre si, ou relacionamentos N:M) e, se não encontrar, cria uma família auxiliar para esse objetivo (linhas 8-10).

Exemplo 4 (Associação artificial em relacionamentos N-ários)**.** *Um relacionamento* `N:M` *promovido à entidade associativa é convertido exatamente como um relacionamento N-ário. Durante a conversão da entidade* `E` *da Figura 1, por exemplo, é detectado que ela tem um relacionamento* `N:M` *(*`R3`*) com* `H` *que foi promovido à entidade associativa. Então, é criada uma família de colunas* `R3`*. A definição da chave detecta que há relacionamento* `1:1` *com* `D` *através de* `R2` *e é realizado o compartilhamento da chave. Na sequência, uma associação artificial é definida de* `E` *para* `H` *através de* `R3` *(coluna* `E`*). Quando* `H` *for convertida, a associação artificial para a outra direção é criada (coluna* `H` *em* `R3`*).*

Exemplo 5 (Associação artificial para relacionamentos *1:N*)**.** *Quando o processo de conversão analisa a entidade* `E` *da Figura 1, por exemplo, é detectado que ela tem uma entidade pai* `B` *através de* `R1`*. Então,* `B` *é convertida primeiro. Em seguida,* `E` *é criada e seus relacionamentos são convertidos, neste caso,* `R1`*. Durante a conversão de* `R1` *(realizada pelo Algoritmo 6, a ser explicado adiante), é detectado que é um relacionamento binário* `1:N` *com atributos. Portanto, uma família de colunas é criada para o relacionamento. Quando isto ocorre, o algoritmo detecta que pode compartilhar a chave com o lado obrigatório. Então, a família de colunas* `R1` *é criada e recebe a chave de* `E` *como uma chave secundária. Deste modo,* `B` *pode alcançar qualquer ocorrência de* `E` *relacionada a ele. Também* `E` *recebe uma coluna referenciando* `R1`*, para que a associação seja bidirecional.*

Após a definição da família origem, uma associação artificial é estabelecida através da definição de uma nova coluna chave (linha 12), nomeando-a (linha 13) e adicionando-a à família origem (linha 14). Apenas se o relacionamento não for `N:M` ou um relacionamento promovido à entidade associativa (linha 14) (ver exemplo 6), o outro lado da associação artificial é definido como uma coluna (linha 16). Essa coluna é nomeada (linha 17) e adicionada à segunda família da entrada (linha 18). Ao final, o conjunto de famílias de colunas adicionais é retornado (linha 20). Este resultado pode ser vazio, pois as famílias de colunas adicionais podem ter sido criadas anteriormente.

Um último exemplo de geração de associação artificial, para o caso de relacionamentos `N:M`, é apresentado a seguir.

Exemplo 6 (Associação artificial para relacionamentos `N:M`)**.** *Considere duas entidades `X` e `Y` associadas por um relacionamento `N:M` sem atributos que não tenha sido promovido a entidade associativa. Suponha que a entidade `X` seja analisada primeiro. Uma família de colunas `X` é criada e o Algoritmo 6 é acionado para converter o relacionamento. Este, por sua vez, cria uma família de colunas auxiliar com chave compartilhada com `X` e outra chave apontada para `Y` (ainda não analisada). O mesmo acontece com a entidade `Y` quando ela for analisada. Assim sendo, as duas famílias de colunas se referenciam mutuamente.*

4.2.5 Geração de colunas

O Algoritmo 5 é responsável por converter um atributo de uma entidade ou relacionamento. Ele recebe como entrada a família de colunas destino e o atributo a ser convertido. A saída é um conjunto de famílias de colunas adicionais. A primeira parte verifica se o atributo é composto (linha 2). Se sim, uma nova família de colunas é criada (linha 3), nomeada (linha 4), sua chave é definida (linha 5), é dada a mesma cardinalidade do atributo (linha 6) e então, para cada atributo participante da composição (linha 7), o algoritmo se aciona recursivamente (linha 8) e o resultado é adicionado à saída do algoritmo. Em seguida, a família de colunas criada é adicionada à saída (linha 10). Se o atributo não for composto (linha 11), uma nova coluna é criada (linha 12), nomeada (linha 13), a cardinalidade do atributo é copiada (linha 14) e a coluna é adicionada à família da entrada (linha 15). Ao final, a saída é retornada (linha 17).

Algoritmo 5: *converterAtributo*
CONVERSÃO DE ATRIBUTO PARA COLUNA

Entrada: Família de colunas (ϵ') e atributo (δ) para conversão
Saída: Famílias de colunas adicionais (ω)

1 $\omega \leftarrow \emptyset$
2 **se** δ *é composto* **então**
3 $\quad \epsilon'' \leftarrow$ Nova família de colunas vazia
4 $\quad \epsilon''.Nome \leftarrow \epsilon'.Nome + \delta.Nome$
5 $\quad \epsilon''.Chave \leftarrow \epsilon'.Chave$
6 $\quad \epsilon''.Cardinalidade \leftarrow \delta.Cardinalidade$
7 $\quad$ **para cada** $\delta' \in \delta | \delta'$ *é um atributo que compõe* **faça**
8 $\quad\quad \omega \leftarrow \omega \cup converterAtributo(\epsilon'', \delta')$
9 $\quad$ **fim**
10 $\quad \omega \leftarrow \omega \cup \epsilon''$
11 **senão**
12 $\quad \delta' \leftarrow$ Nova coluna
13 $\quad \delta'.Nome \leftarrow \delta.Nome$
14 $\quad \delta'.Cardinalidade \leftarrow \delta.Cardinalidade$
15 $\quad \epsilon' \leftarrow \epsilon' \cup \delta'$
16 **fim**
17 **retorna** ω

Conforme mostrado no Algoritmo 5, uma recursão ocorre no caso do atributo ser composto, de modo a converter todos os atributos componentes. Além disso, as restrições de cardinalidade de cada atributo são preservadas no esquema lógico. Os casos de conversão de cada tipo de atributo presente no modelo EER podem ser sumarizados conforme segue:

- **Identificador:** gera uma chave de uma família de colunas através do Algoritmo 3;
- **Obrigatório e opcional:** gera uma coluna em uma família de colunas. Para BDs colunares, colunas obrigatórias e opcionais são similares pois não há controle desse tipo de restrição por parte do SGBD colunar;
- **Multivalorado:** gera uma coluna do tipo coleção em uma família de colunas. A cardinalidade não é imposta pelo BD colunar. As restrições são referentes à limitações globais[2];
- **Composto:** como não é um recurso nativo em BDs colunares. Este tipo de atributo é representado como uma nova família de colunas com chave compartilhada. Dessa forma, uma organização hierárquica similar é gerada;
- **Composto e multivalorado:** é convertido da mesma forma que um atributo composto com a definição adicional, no esquema lógico, de uma cardinalidade para a família de colunas gerada.

Exemplo 7 (Geração de uma coluna)**.** *O atributo* `a1` *da entidade* `A` *Figura 1 é um exemplo de um atributo monovalorado e obrigatório. Ele gera uma coluna simples na família de colunas* `A`*. A família de colunas* `E-a2` *é um exemplo de como um atributo composto é convertido. Ele se torna uma família de colunas* `E-a2` *com chave compartilhada com* `E` *e seus atributos se tornam colunas.*

[2]Por exemplo, no BD colunar Cassandra, há um limite máximo de 2GB para qualquer valor de coluna e de aproximadamente 64KB por item em colunas do tipo coleção (Fundação Apache, 2009)

4.2.6 Conversão de relacionamentos

Esta seção detalha a conversão de relacionamentos de diversos tipos, presentes em uma modelagem EER, para um esquema lógico de um BD colunar. Primeiramente, algumas considerações sobre as estratégias de mapeamento adotadas neste trabalho para relacionamentos são feitas.

Cabe salientar que, em projetos tradicionais de BDs, relacionamentos de associação do tipo `1:1` tipicamente têm as entidades envolvidas fundidas em um único construtor no esquema lógico (SCHROEDER; MELLO, 2008; FONG, 1995). Diferentemente, nossa abordagem cria ao menos duas famílias de colunas (uma terceira pode ainda ser criada, caso o relacionamento possua atributos) e, em geral, o conceito de chave compartilhada é utilizado. A justificativa para isso é que, didaticamente, existem poucos atributos em entidades e relacionamentos, para evitar a poluição e dificuldade de compreensão da modelagem conceitual. Entretanto, em cenários do mundo real, estes fatos normalmente possuem muito mais informações associadas. No caso de BDs colunares, temos a vantagem que, em geral, todos os dados de uma família de colunas são trazidos para a memória quando um acesso é realizado. Assim sendo, a união de entidades potencializa sua subutilização. Na abordagem proposta por este trabalho, como a noção de chave compartilhada coloca os dados próximos por meio de fragmentação (*sharding*), a multiplicidade de famílias potencializa o paralelismo e a evolução e especialização do esquema é facilitada, pois a mudança de requisitos de cardinalidade não implicará em alterações drásticas na modelagem lógica.

No caso de relacionamentos `1:L`, o lado `L` representa um limite superior conhecido ou notavelmente restrito de registros ou instâncias associados ao lado 1, conforme explicado anteriormente. Este tipo de cardinalidade mais restrita é levado em conta visando uma maior otimização para vários BDs, inclusive colunares. No caso deste trabalho, utiliza-se a noção de agregação para que seja possível a extração em bloco dos dados. A principal diferença entre o caso `1:L` e o caso `1:1` é que haverá uma cardinalidade máxima na família do lado `L`. Ela é definida na criação da família de colunas. Portanto, o mapeamento deste relacionamento é tratado da mesma forma que um relacionamento do tipo `1:1`, conforme apresenta o Algoritmo 6, com esta pequena diferença.

Devido à flexibilidade da estrutura de colunas de um registro em BDs colunares, o comportamento de relacionamentos `1:L` é similar ao de um atributo composto, ou seja, as colunas são criadas dinamicamente, sempre respeitando a restrição de cardinalidade máxima.

Diferentemente, relacionamentos `1:N` sempre geram uma associação artificial, pois a quantidade de instâncias associadas do lado "N"pode ser elevada e a estratégia de agregação pode gerar instâncias muito grandes. Além disso, é definida uma família de colunas auxiliar associada com o lado pai do relacionamento, por meio de uma chave compartilhada, que mantém as associações existentes e os atributos do relacionamento, se existirem. Esta separação permite que a família de colunas auxiliar seja acessada pela aplicação somente se informações do relacionamento forem desejadas.

Para relacionamentos `N:M`, define-se uma família de colunas auxiliar em cada uma das famílias de colunas correspondentes às entidades envolvidas no relacionamento, e elas se referenciam de forma a manter a navegabilidade bidirecional e separação de responsabilidade quanto ao relacionamento. Um dos benefícios é a manutenção dos dados do relacionamento próximos das entidades lógicas, facilitando o acesso quando necessário. Uma nova família de colunas é criada para manter o relacionamento quando este possuir atributos. Isso se faz necessário para evitar redundância destes atributos nas famílias de colunas das entidades.

Esta estratégia é expandida para relacionamentos N-ários, onde a principal diferença é a adição de uma ou mais dimensões, tornado necessária a criação de uma família de colunas intermediária para abrigar todas as referências a dimensões e suas diferentes cardinalidades. Essa estratégia é herdada do projeto tradicional de BDs relacionais Heuser (2008).

Exemplo 8 (Conversão de um relacionamento binário)**.** *Conforme ilustra a Figura 1, o relacionamento* `R1` *é do tipo* `1:N`*. Neste caso, uma associação artificial é criada, tendo uma família de colunas pai* `B` *e uma família de colunas filha* `E`*. A família pai recebe uma família de colunas auxiliar* `R1` *com chave compartilhada para abrigar a segunda chave (composta) que aponta para a família filha em conjunto com os atributos do relacionamento convertidos em colunas. A família de colunas filha recebe uma coluna referenciando a família de colunas pai. Desta forma,* `B` *pode navegar para todos os seus filhos por meio de* `R1` *e* `E` *pode acessar o pai através dessa nova coluna.*

Relacionamentos reflexivos são um caso especial de relacionamento binário que referencia a mesma entidade em ambos os lados. Esse relacionamento pode possuir qualquer cardinalidade e, para atender requisitos de navegabilidade, é necessário existir uma família de colunas auxiliar para representar a auto-referência, de forma similar ao tratamento de relacionamentos `1:N` que, neste caso, permite que todas as cardinalidades sejam atendidas. Portanto, independentemente da cardinalidade (`1:1`, `1:L`, `1:N` ou `N:M`) são sempre tratados como `1:N`. O único caso que permitiria uma estrutura diferente seria o caso `1:1`, sendo possível uma coluna adicional. Porém, essa estrutura fere o princípio de separação aplicado no decorrer deste trabalho, na qual a criação da família auxiliar não fere o desempenho e permite mudança de cardinalidade sem elevado custo de implementação.

A conversão de todos os tipos de relacionamento do modelo EER é realizada pelo Algoritmo 6. Sua entrada é um relacionamento e a entidade origem e sua saída é um conjunto de famílias de colunas. Na primeira parte do algoritmo é inicializada a saída com um conjunto vazio (linha 1) que é procedido pela análise e tratamento de cada tipo de relacionamento (linha 2), conforme segue:

- **Binário ou reflexivo:** este caso não considera relacionamentos promovidos a entidades associativas nem relacionamentos `N:M` (linha 3). Se o relacionamento tiver atributos (linha 4), é criada uma família de colunas intermediária para recebê-los (linha 5) e esta é definida como pai do relacionamento (linha 6). Caso ele não possua atributos, não é criada uma família intermediária. Na sequência, se o pai e a família de colunas origem não compartilham a mesma chave ou o relacionamento é reflexivo (linha 8), a criação da associação artificial é acionada (linha 9) e a saída adicionada à saída do algoritmo;

- **N-ário:** este caso inclui relacionamentos promovidos à entidade associativa ou relacionamentos `N:M` (linha 12). Uma família de colunas para o relacionamento (linha 13) e um relacionamento binário temporário são criados. Este relacionamento temporário, com mesma cardinalidade no lado da família de colunas origem e cardinalidade máxima 1 no lado da família de colunas recém-criada (linha 14) é, então, convertido recursivamente. Cabe salientar aqui que, como já foi criada a família de colunas intermediária (linha 13), esse relacionamento binário temporário visa a criação de uma família auxiliar e a geração de uma associação artificial entre ambas. Sua saída é adicionada à saída do algoritmo (linha 15);
- **Generalização ou união:** a conversão destes casos inicialmente verifica se o relacionamento é parcial ou se a entidade não compartilha chave com seu lado pai (linha 18). Se sim, uma associação artificial invertida é criada entre eles, usando a família origem como pai e a entidade pai no relacionamento como filha (linha 19). O resultado é adicionado à saída do algoritmo. Senão, em caso de totalidade na generalização ou união, ou se as famílias compartilham chave, não há necessidade de nenhuma operação já que a estrutura da chave compartilhada já foi aplicada.

Os relacionamentos temporários são uma forma de quebrar associações complexas da entrada, como por exemplo, relacionamentos n-ários e generalizações, em partes menores (relacionamentos binários), sendo estes tratados de forma padrão pelos algoritmos disponíveis na abordagem proposta. Ele deixa de existir após a sua utilização dentro do escopo do algoritmo.

Exemplo 9 (Conversão de generalização)**.** *Conforme mostra a Figura 1, a entidade* `A` *é pai de um relacionamento de generalização, sendo convertida antes de suas especializações. Para a entidade* `B`*, o algoritmo de conversão verifica que a entidade pai (*`A`*) já está convertida e procede então a sua conversão. Desta forma, a família* `B` *é criada e como a especialização é do tipo total, ela compartilha a chave do pai do relacionamento. Neste caso, nenhuma família de colunas adicional é criada, pois o relacionamento já é representado através da chave compartilhada. O mesmo raciocínio se aplica às entidades* `C` *e* `D`*.*

Um relacionamento de especialização permite a herança do atributo identificador da entidade genérica (NAVATHE; BATINI; CERI, 1992). Tal definição torna desnecessários os atributos identificadores nas entidades especializadas (SCHROEDER; MELLO, 2008).

Algoritmo 6: *converterRelacionamento*
CONVERSÃO DE RELACIONAMENTOS

Entrada: Relacionamento (π) e família de colunas origem (ϵ)
Saída: Conjunto de famílias de colunas (ω)

$\omega \leftarrow \emptyset$
selecione $\pi.Type$ **faça**
  caso *(Binário* $\vee$ *Reflexivo)* $\wedge\neg$ *(N:M* $\wedge$ *promovido à entidade associativa)* **faça**
    se $\pi.Atributos \neq \emptyset$ **então**
      $\omega \leftarrow criarFamilia(\pi)$
      $\pi.Pai \leftarrow \omega$
    fim
    se $\epsilon.Chave \neq \pi.Pai.Chave \vee \pi.Tipo = Reflexivo$ **então**
      $\omega \leftarrow \omega \cup criarAssociacaoArtificial(\pi.Pai, \epsilon, \pi)$
    fim
  fim
  caso *N-ario* $\vee$ *(N:M* $\wedge$ *promovido à entidade associativa)* **faça**
    $\epsilon_R \leftarrow criarFamilia(\pi)$
    $\pi_T \leftarrow$ Relacionamento binário temporário com mesma cardinalidade em ϵ e máxima igual a 1 em ϵ_R
    $\omega \leftarrow \omega \cup \epsilon_R \cup converterRelacionamento(\pi_T, \epsilon)$
  fim
  caso *Generalização* $\vee$ *União* **faça**
    se π *é parcial ou alguma entidade em* π *não compartilham chave* **então**
      $\omega \leftarrow \omega \cup criarAssociacaoArtificial(\epsilon, \pi.Pai, \pi)$
    fim
  fim
fim
retorna ω

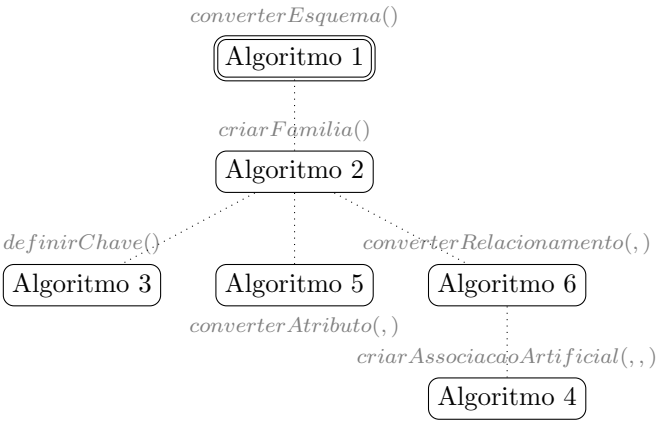
Uniões totais são similares à generalizações totais e disjuntas, pois todas as instâncias têm uma única herança. Dessa forma, a mesma lógica de conversão de generalizações pode ser aplicada. Em (SCHROEDER; MELLO, 2008), é definido um atributo identificador da herança. Contudo, é possível obter tal informação a qualquer momento consultando famílias de colunas correlacionadas usando sua chave compartilhada. Assim sendo, esse artifício não foi considerado neste processo de conversão, mas poderá ser utilizado, para fins de otimização, como trabalho futuro. Uniões parciais são mais complexas e o processo de conversão trata este caso como um relacionamento N-ário (ver exemplo 10).

Exemplo 10 (Conversão de um tipo união)**.** *A entidade `H` é uma união parcial de `F` e `G`. Supondo que a primeira entidade a ser convertida é `F`, o processo de conversão inicialmente verifica se os pais de `F` já foram convertidos. Neste caso, `H` ainda não foi tratado e, portanto, ela é convertida antes. A família de colunas `H` então é criada e, como esta não possui pai, o processo retorna a `F`, que é então convertida. Neste momento é verificado que a união é parcial e uma família de colunas artificial é concebida (`F-H`). O mesmo ocorre com a entidade `G`. Se fosse o caso de uma união total, as famílias `F` e `G` teriam chave compartilhada diretamente com `H` e as famílias auxiliares não existiriam, visto que obrigatoriamente as suas instâncias se especializariam em instâncias de `H`.*

4.3 EXEMPLO DE APLICAÇÃO DO PROCESSO DE CONVERSÃO

Esta seção demonstra a aplicação do processo de conversão proposto para o esquema conceitual exemplo mostrado na Figura 1 com o objetivo de torná-lo mais claro. A Figura 12 demonstra a relação resumida entre os algoritmos e serve como referência visual (um mapa) para sua execução aqui descrita. Os exemplos no decorrer desta seção ilustram a aplicação pontual de cada algoritmo do processo de conversão, mas, primeiramente, uma revisão do processo geral de conversão é apresentada para sumarizar o que foi apresentado anteriormente.

Figura 12: Representação visual simplificada do relacionamento entre os algoritmos que compõem o processo proposto.



Fonte: Elaborada pelo autor.

4.3.1 Revisão do processo geral de conversão

Conforme explicado anteriormente, o processo de conversão inicia pelo Algoritmo 1, que recebe um esquema conceitual de entrada e gera um conjunto de famílias de colunas que representa um esquema lógico colunar. Seu objetivo é iterar todas as entidades do modelo conceitual e acionar as suas conversões para famílias de coluna através do Algoritmo 2. Este, por sua vez, é dividido em quatro partes: verificação, recursão, conversão da entidade e dos relacionamentos. Na primeira parte, é verificado se a entidade/relacionamento de entrada já foi convertida(o) em uma família de colunas. Em caso negativo, na sua segunda parte, o algoritmo invoca a si mesmo para entidades que possui dependência (possuem pais). A terceira parte converte a entidade, definindo a sua chave através do Algoritmo 3 e invocando o Algoritmo 5 para converter cada atributo da entidade em um conjunto de famílias e/ou colunas. A quarta e última parte é a conversão dos relacionamentos através do Algoritmo 6.

A definição da chave e a criação dos atributos é fundamental para cada família de colunas. Após definida a chave, para cada atributo da entidade é acionado o Algoritmo 5. Se o atributo não for composto é criada uma coluna de mesmo nome, tipo e cardinalidade na família de colunas. Caso seja composto, é criada uma nova família de colunas com o nome do atributo composto concatenado ao nome da entidade, tendo como chave a mesma da família de colunas referente à entidade sendo convertida (chave compartilhada). A cardinalidade da família é a mesma do atributo, ou seja, se ele for monovalorado é uma família de colunas simples e se ele for multivalorado, uma cardinalidade é atribuída à família. Para cada atributo filho da composição é novamente acionado o Algoritmo 5 de forma a gerar uma recursão que define uma família de colunas nova para cada atributo composto.

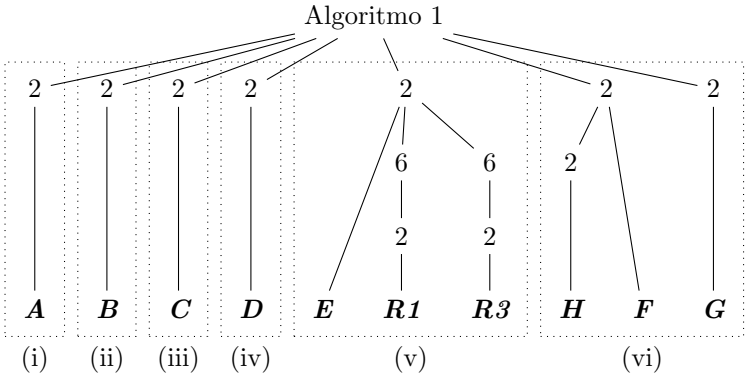
Finalmente, a última parte do processo é a conversão dos relacionamentos das entidades. Para cada relacionamento de dependência da entidade é acionado o Algoritmo 6. Neste momento é verificado o tipo do relacionamento e aplicada a estratégia de conversão adequada. Considerando o esquema conceitual EER da Figura 1, todo esse processo gera o esquema lógico representado na Figura 16. A construção sistemática deste esquema lógico é detalhado a seguir.

4.3.2 Simulação da aplicação do processo geral de conversão

Uma exemplificação do uso do processo de conversão é agora apresentado. Simulando o início do processamento na entidade `A`, em ordenação alfabética, temos a árvore de derivação na Figura 13, e de forma aleatória (para representação: `E`,`C`,`G`,`B`,`D`,`H`,`A`,`F`) na Figura 14. Duas árvores de derivação são apresentadas com o intuito de demonstrar que a ordem de processamento é automaticamente ajustada para que o esquema lógico não se altere, independentemente da entidade inicial escolhida pelo Algoritmo geral. Isso é possível através do conceito de paternidade, que prioriza sempre a conversão das entidades pais. Nestes exemplos, em ambos os casos, a entidade `A` é convertida primeiro.

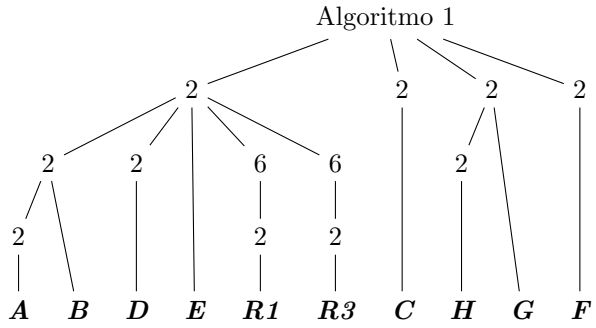
Ambas as figuras apresentam o número do algoritmo derivado nos nós (informação resumida para melhor visualização). Já nas folhas é mostrada a família de colunas gerada. Cada passo da derivação é evidenciado por um bloco pontilhado identificado com um numeral romano.

Figura 13: Árvore de derivação com iterações para conversão (ordem alfabética) das entidades presentes no exemplo da Figura 1.



Fonte: Elaborada pelo autor.

Figura 14: Árvore de derivação para conversão (aleatória) das entidades presentes no exemplo da Figura 1.



Fonte: Elaborada pelo autor.

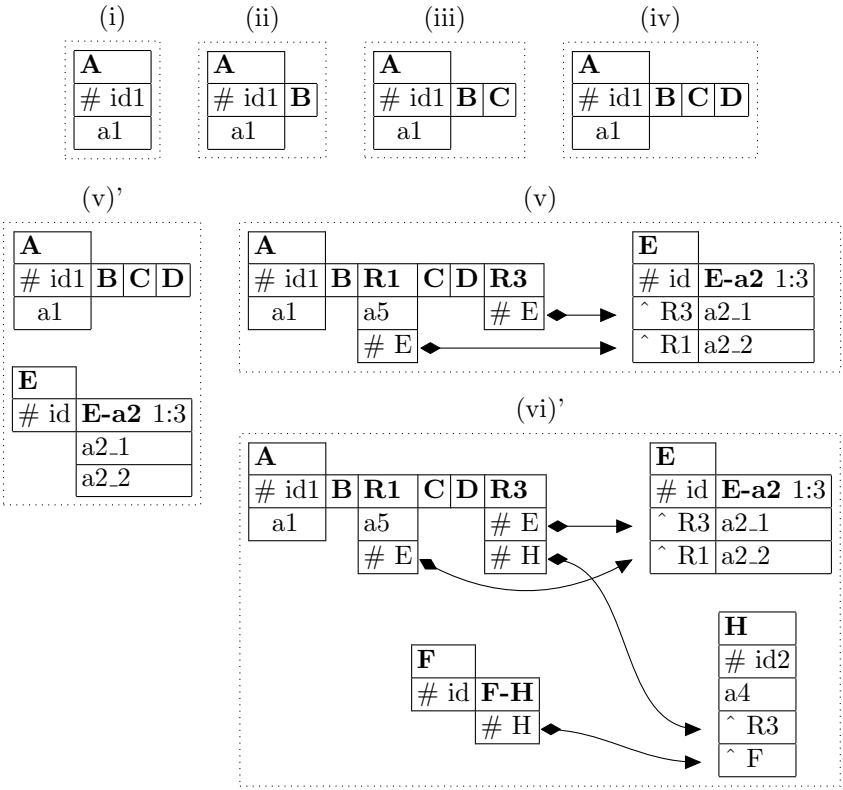
Os Algoritmos 3, 4 e 5 foram omitidos das Figuras 13 e 14, sendo apresentados aqueles que efetivamente convertem as entidades. Por exemplo, ao processar a entidade `B` pelo Algoritmo 2, é encontrada uma paternidade com `A`, que aciona recursivamente o próprio algoritmo. Porém, como a entidade já foi marcada como visitada, o processo retorna sem nenhuma operação. Na sequência, são percorridos os relacionamentos com paternidade, sendo invocada a criação de uma associação artificial para a generalização (Algoritmos 6 e 4). Entretanto, como generalização é total e disjunta, nada é criado e retorna-se ao algoritmo acionado, procedendo-se a conversão da próxima entidade (`C`). Estas operações são considerados caminhos vazios e são omitidos nas árvores de derivação.

A seguir é exemplificado o processo de conversão com base na sequência representada na Figura 13. Ele inicia no Algoritmo 1 e percorre cada entidade, sendo cada iteração principal representada através de quadros pontilhados e identificados com numeração romana. Cada uma dessas iterações é detalhada a seguir e a evolução do esquema lógico gerado a partir de cada iteração é apresentada na Figura 15.

No passo *(i)*, a conversão da entidade `A` inicia pelo acionamento do Algoritmo 2, que verifica que a entidade não foi visitada e que não tem pais. Assim sendo, gera-se uma família de colunas para ela, com seus respectivos atributos (Algoritmo 5), todos simples e monovalorados, e a entidade é marcada como visitada. A chave é definida, através do Algoritmo 3, como `id1`, levando em consideração que a entidade possui um atributo identificador. A entidade `A` não possui relacionamentos pais nem `N:M` ou reflexivos. Logo, a conversão de relacionamentos (Algoritmo 6) não é acionada;

No passo *(ii)* ocorre a conversão da entidade `B`. Para todas as entidades a serem convertidas, a primeira verificação é se esta entidade já foi visitada. Neste caso, ela ainda não foi e passa-se, então, para a próxima tarefa que é para buscar uma lista de pais, sendo retornado apenas a entidade `A`. Para cada pai o algoritmo é chamado recursivamente passando este pai como entrada. Como a entidade `A` já foi visitada, retorna-se ao fluxo de conversão da entidade `B`.

Figura 15: Evolução do esquema lógico com a aplicação do processo de conversão sobre o esquema conceitual da Figura 1.



Fonte: Elaborada pelo autor.

A próxima tarefa, então, é criar a família de colunas `B` e definir sua chave. O Algoritmo 3 identifica que a entidade não possui um identificador único, mas que existe uma generalização total como pai em `A`, podendo assim compartilhar a sua chave. Em seguida, os seus atributos são convertidos (nesse caso, não há nenhum) e é acionada a conversão dos relacionamentos pelo Algoritmo 6, pois existe um relacionamento pai. O mesmo é acionado para a conversão da generalização e, como neste caso há uma chave compartilhada, nada é feito (decisão tomada na linha 18 do algoritmo) e a conversão de `B` é concluída;

No passo *(iii)*, o processamento de conversão para a entidade `C` é análogo ao realizado para a entidade `B`. O mesmo ocorre para a conversão da entidade `C` no passo *(iv)*.

No passo *(v)* ocorre a conversão da entidade `E`. Neste momento, percebe-se que existe `R1` como relacionamento pai binário e, após a verificação de visita em `E`, o Algoritmo 2 é acionado recebendo o como entrada o lado pai em `R1`: a entidade `B`. O algoritmo então identifica que o mesmo já foi visitado e o processamento retorna para a entidade `E` para criar uma família de colunas para ela. Como `E` não possui atributo identificador nem participa de generalizações e relacionamentos `1:L` ou `1:1` com paternidade obrigatória, é criada uma coluna chave `id`.

Em seguida são convertidos os atributos de `E` pelo Algoritmo 5. Neste caso, existe apenas um atributo composto e sua conversão gera uma nova família de colunas auxiliar `E-a2` com chave compartilhada com `E`. Esta família recebe uma cardinalidade por se tratar de um atributo multivalorado. Recursivamente, o mesmo algoritmo é acionado tendo a família auxiliar e cada atributo que faz parte da composição como entrada, sendo processado um de cada vez. Estes atributos simples são convertidos em colunas de `E-a2`. O algoritmo de criação de colunas conclui e retorna ao fluxo de conversão para a entidade `E`. O item *(v)'* na Figura 15 representa o resultado do processo de conversão até esse momento.

O próximo passo é o processamento de relacionamentos pai e `N:M` ou reflexivos, que, neste caso, são `R1` e `R3`. Cada relacionamento é processado pelo Algoritmo 6. No caso de `R1`, um relacionamento binário com atributos, é criada uma família auxiliar para o relacionamento, que é, neste contexto, considerada pai no relacionamento.

Para a criação da família auxiliar R1, primeiramente verifica-se se o mesmo já foi visitado e, posteriormente, a conversão de relacionamentos pais não executa nenhuma tarefa pois R1 é um relacionamento binário e não possui outros relacionamentos. Em seguida, é criada a família de colunas de mesmo nome e a definição da chave identifica que a entrada é um relacionamento e que possui um lado obrigatório em B, compartilhando sua chave, que por sua vez, já a compartilha com a entidade A. Na sequência, seus atributos são convertidos e, por não possuir outros relacionamentos, o algoritmo retorna ao fluxo anterior.

Em seguida, é verificado que a chave da família auxiliar R1 é diferente da chave de E e que o relacionamento não é reflexivo, acionando, assim, o Algoritmo 4 para criação de uma associação artificial entre ambas. Identifica-se que a família auxiliar R1 foi criada a partir de um relacionamento e é adicionada uma coluna chave que mantém a chave da outra família E, gerando, assim, uma chave composta em R1. Na sequência, verifica-se que o relacionamento não é N:M nem foi promovido à entidade associativa. Desta forma, gera-se uma coluna do tipo associação artificial com nome R1 em E e o processamento retorna à conversão do relacionamento e, por sua vez, à conversão da entidade E. Isso é demonstrado através da linha que liga a coluna (# E) em R1 com a coluna (^ R1) em E, no item *(v)* da Figura 15.

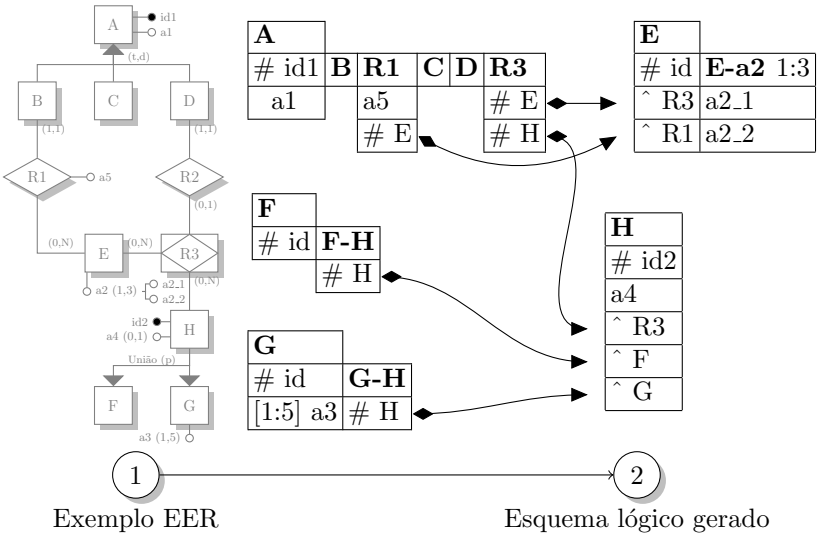
O próximo relacionamento a ser tratado, R3, é do tipo N:M promovido à entidade associativa. O Algoritmo 6 é então acionado, tendo como entrada o próprio relacionamento e a entidade E. Por causa das características do relacionamento (linha 12), é acionada a criação da família de colunas a partir de R3. Neste momento, verifica-se que R3 ainda não foi visitado e, como ele possui um relacionamento pai com D (R2), a conversão de D é acionada, que, por já ter sido visitada, retorna ao fluxo de conversão de R3. A família de colunas R3 é então criada e a definição da chave identifica que existe um relacionamento binário mandatório em D através de R2, permitindo compartilhar sua chave, que já é compartilhada com A. Esta estrutura não possui atributos e o processo passa para a conversão dos relacionamentos pai (linha 14) de R3 que, no caso, é apenas o relacionamento R2. Identifica-se que R2 é um relacionamento binário, sem atributos e com chave compartilhada com D. Assim sendo, nada é feito, concluindo-se o processo de criação da família auxiliar R3.

Na sequência, é criado um relacionamento binário temporário com cardinalidade máxima igual a 1 no lado ligado à família auxiliar `R3` e é copiada a cardinalidade entre `R3` e `E` no lado de `E`, que é usado como entrada para acionamento recursivo do Algoritmo 6. Assim, é identificado que o relacionamento é binário e não possui atributos e que a chave de `E` é diferente de `R3`, sendo, desta forma, acionada a criação da associação artificial de forma similar à `R1` e concluindo a conversão de `E`.

No passo *(vi)* ocorre a conversão da entidade `F`. Inicialmente, verifica-se que a entidade não foi visitada e, em seguida, encontra-se `H` como entidade pai dela através da união parcial e procede-se a criação da família referente à `H`. Como ela ainda não foi visitada e não tem relacionamentos pai, é criada a família de colunas `H`, sendo a sua chave definida com base no atributo identificador da entidade `id2` e as suas colunas são então convertidas. Como `H` possui um relacionamento `R3` (com cardinalidade `N:M`), é acionada a conversão do mesmo. Como a família de colunas `R3` foi criada anteriormente, nada é feito e retorna-se ao tratamento da entidade `H`. Na sequência do processamento, aciona-se o Algoritmo 6 para converter um relacionamento temporário binário que associa `H` a `R3`. Como ele não possui atributos e a chave de ambos é diferente, é acionado o Algoritmo 4 para criação de uma associação artificial. Neste caso, como a entrada foi criada para um relacionamento, uma nova coluna chave que aponta para `H` é adicionada em `R3` e a família `H` recebe uma coluna de associação artificial referente à `R3`. Esta estrutura é ilustrada no item *(vi)'* na Figura 15.

O fluxo retorna à conversão da entidade `F`, cuja família de colunas é então criada e a sua chave é definida como uma coluna nova, pois não se encontra candidato à chave compartilhada. Em seguida, passa-se à conversão dos atributos que, neste caso, não existem. O próximo passo é a conversão dos relacionamentos pais, sendo o Algoritmo 6 acionado passando a união e a família recém-criada como entradas. Por se tratar de uma união parcial, a conversão do relacionamento aciona diretamente a criação da associação artificial entre `H` e `F`. Neste momento, identifica-se que a entrada não foi criada a partir de um relacionamento nem existe uma família auxiliar pré-existente. Assim sendo, gera-se uma família de colunas auxiliar composta pelo nome de ambas as entradas (`F-H`) compartilhando chave com `F`. Ainda, define-se uma coluna chave nesta família apontando para `H` e uma coluna referente à associação artificial em `H` apontando para `F-H`.

Figura 16: Exemplo de esquema lógico gerado pelo Algoritmo 1 utilizando o esquema conceitual da Figura 1 como entrada.



Fonte: Elaborada pelo autor.

O fluxo então retorna ao Algoritmo 2 que aciona a conversão da entidade `G`, a qual tem o processo similar à `F`. O resultado final é o esquema lógico ilustrado na Figura 16, que demonstra o esquema conceitual original da Figura 1 à esquerda e o seu esquema lógico final à direita.

Por fim, vale ressaltar que as árvores de derivação demonstram que as famílias de colunas que possuem relação de paternidade são convertidas antes de suas filhas. Este é o caso das entidades `A`, `D` e `H`, que são criadas obedecendo a mesma ordem independentemente da ordenação das entidades processadas na entrada. Ainda, diferentemente de `R2`, os relacionamentos `R1` e `R3` são apresentados na árvore pois geram famílias de colunas já na primeira parte do algoritmo de criação de famílias de colunas (`R1` por ter atributos e `R3` por ser um relacionamento promovido à entidade).

5 AVALIAÇÃO EXPERIMENTAL

A estratégia de agregação de famílias de colunas proposta neste trabalho para a modelagem lógica de BDs colunares tenta valorizar o acesso a dados de modo geral, minimizando operações de leitura e escrita em dados que poderiam estar armazenados em estruturas físicas diferentes, caso essa estratégia não fosse seguida. Este raciocínio é o mesmo adotado pelo *NoAM* (BUGIOTTI et al., 2014), um trabalho fortemente relacionado a este (uma das poucas propostas na literatura que também lida com o projeto lógico de BDs NoSQL).

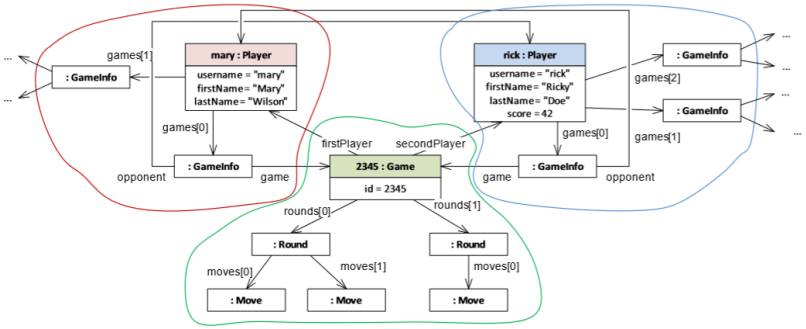
Nas próximas sessões são apresentadas a motivação e objetivos, preparação do ambiente de teste, concluindo com métricas e medição.

5.1 MOTIVAÇÃO E OBJETIVOS

É necessário avaliar o desempenho da nossa proposta de projeto lógico para BDs colunares e isso é alcançado através da comparação desta dissertação com *NoAM* (BUGIOTTI et al., 2014). Esta avaliação experimental considera o mesmo domínio de aplicação apresentado em (BUGIOTTI et al., 2014), que é um jogo composto basicamente pelas entidades *Player* e *Game*, conforme ilustra o diagrama de classes UML apresentado na Figura 17. Esta modelagem conceitual UML gerou, através da aplicação da proposta *NoAM*, dois agregados como estrutura lógica, mostrados na Figura 18, `Player` e `Game`. O agregado `Game` contém pelo menos três atributos: `firstPlayer` e `secondPlayer` - que mantém `username` de `Player`, e `Rounds` - que mantém uma coleção de etapas.

Salienta-se aqui que essa estrutura projetada é suficiente para cenários onde os níveis hierárquicos gerados pelos relacionamentos entre dados são baixos ou não crescem demasiadamente. Entretanto, se consideramos jogos modernos, que simplesmente nunca terminam e onde as etapas de um jogo tendem ao infinito, dificuldades surgem ao utilizar essa estrutura. Outras aplicações similares, como o *Twitter*, mensagens instantâneas ou redes sociais possuem a mesma tendência.

Figura 17: Diagrama de classes UML do domínio exemplo.



Fonte: Bugiotti et al. (2014).

Figura 18: Modelagem lógica gerada pela abordagem NoAM no domínio em questão.

Player

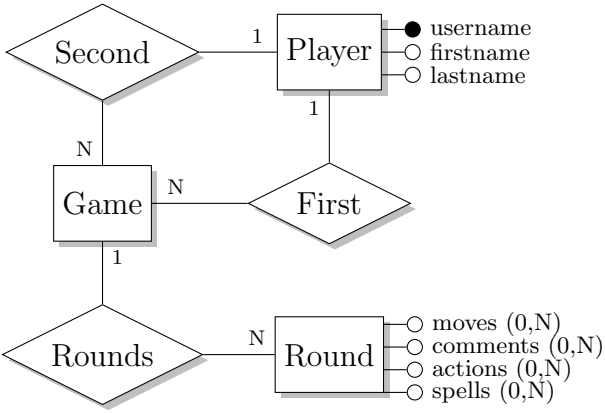
mary	*username*	*"mary"*
	firstName	*"Mary"*
	lastName	*"Wilson"*
	games[0]	⟨ *game* : **Game:2345**, *opponent* : **Player:rick** ⟩
	games[1]	⟨ *game* : **Game:2611**, *opponent* : **Player:ann** ⟩

Game

2345	*id*	*2345*
	firstPlayer	**Player:mary**
	secondPlayer	**Player:rick**
	rounds[0]	⟨ *moves* : ..., *comments* : ... ⟩
	rounds[1]	⟨ *moves* : ..., *actions* : ..., *spell* : ... ⟩

Fonte: Bugiotti et al. (2014).

Figura 19: Esquema EER gerado através de engenharia reversa do diagrama de objetos da Figura 17.



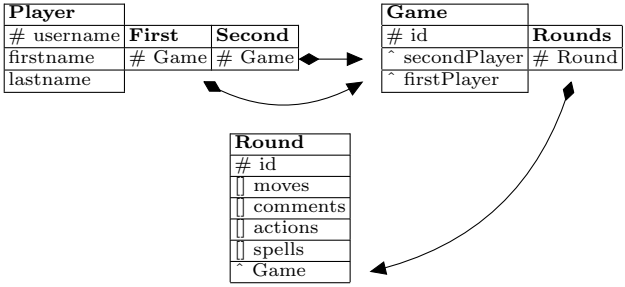
Fonte: Elaborada pelo autor.

5.2 PREPARAÇÃO E CONFIGURAÇÃO DO AMBIENTE

Para fins de comparação da nossa proposta com o mesmo experimento realizado pelo *NoAM*, foi gerado um esquema EER (mostrado na Figura 19) através de um processo de engenharia reversa (HEUSER, 2008) do diagrama de classes UML mostrado na Figura 17.

NoAM também considera o projeto físico de BDs NoSQL através da aplicação de duas estratégias: Entrada por Agregado (*Entry per Aggregate Object* - EAO) e Entrada por Atributo (*Entry per Top Field* - ETF). EAO é a estratégia que armazena toda a hierarquia de um agregado em uma única chave (`/Game/game_id`). ETF é quando a chave é composta pelos atributos de primeiro nível do agregado, sendo possível acessar diretamente o subnível através de chaves específicas, ou seja, para o domínio exemplo, é possível acessar cada instância do agregado `Game` pelas chaves `/Game/game_id`, `/Game/game_id/firstPlayer`, `/Game/game_id/secondPlayer` e `/Game/game_id/rounds`. A proposta de *NoAM* é comparada com este trabalho, no nível físico, utilizando a estratégia EAO, uma vez que a estratégia ETF não permite expansão para os próximos níveis hierárquicos. Em outra palavras, os valores no subnível *Round* (incluindo *comments*, *moves* etc) não tirariam vantagem dessa estratégia se fossem projetadas como coleções compostas.

Figura 20: Esquema lógico para BDs colunares no domínio de jogos gerado por este trabalho.

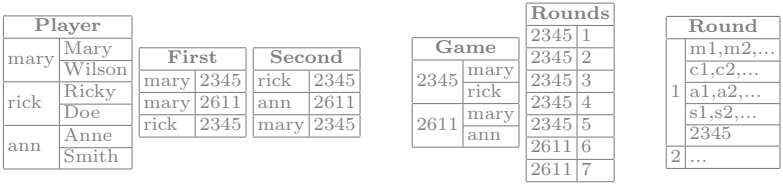


Fonte: Elaborada pelo autor.

O esquema EER obtido pelo processo de engenharia reversa foi convertido para o esquema lógico de um BD colunar apresentado na Figura 20 através da aplicação do nosso processo de conversão. Um exemplo da representação física dos dados, de acordo com a modelagem lógica gerada pela nossa proposta, é apresentado na Figura 21. A representação física dos mesmos dados gerada pelo *NoAM* é mostrado na Figura 22 e ajuda a contrastar as modelagens obtidas por ambas as propostas. A principal diferença entre ambas as modelagens é que a proposta deste trabalho expande o número de famílias de colunas de duas para seis. Destas seis, uma delas é a entidade `Round` que foi desaninhada do agregado `Game` e três delas estão conectadas por chaves compartilhadas (`First`, `Second` e `Rounds`). Desta forma, diversas famílias foram modeladas de forma a permanecerem próximas fisicamente, facilitando o acesso. Além disso, existem três associações artificiais, uma a menos que o número de referências definido pela modelagem *NoAM* (`game` e `opponent` na família `Player`, assim como `firstPlayer` e `secondPlayer` em `Game`). Todos os atributos em ambas as modelagens são similares, com exceção de `opponent`, que é ocultado pela arguição de que todos os jogadores são oponentes um do outro.

Para fins de avaliação, o esquema lógico foi implementado no BD colunar Cassandra e foram comparados tempos de leitura e escrita de acordo com um cenário comum.

Figura 21: Exemplo de esquema físico para o domínio de jogos gerado por este trabalho.



Player	
mary	Mary
	Wilson
rick	Ricky
	Doe
ann	Anne
	Smith

First	
mary	2345
mary	2611
rick	2345

Second	
rick	2345
ann	2611
mary	2345

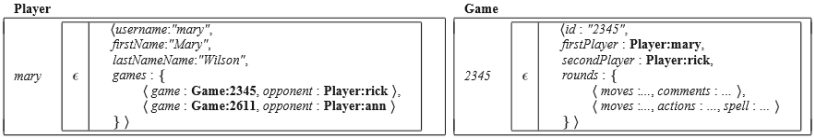
Game	
2345	mary
	rick
2611	mary
	ann

Rounds	
2345	1
2345	2
2345	3
2345	4
2345	5
2611	6
2611	7

Round	
1	m1,m2,...
	c1,c2,...
	a1,a2,...
	s1,s2,...
	2345
2	...

Fonte: Elaborada pelo autor.

Figura 22: Exemplo de esquema físico gerado pela abordagem NoAM para o esquema lógico da Figura 18.



Fonte: Bugiotti et al. (2014).

Para executar os experimentos, um *cluster* remoto do Cassandra com três nós foi utilizado, tendo cada nó 5 Gb de armazenamento SSD, 2 Gb RAM e 1 núcleo de CPU. A replicação e fator de Quorum foram definidos para três. Então, cada operação garante que ao menos dois nós concordem com os dados lidos ou escritos. É dessa forma que o Cassandra garante consistência imediata e a eventualidade é somente para o nó além do Quorum.

5.3 MEDIÇÃO E MÉTRICAS

O experimento realizado se baseou no Algoritmo 7[1], que foi implementado e executado alternando ambas as propostas. Inicialmente foi criado um jogo com dois jogadores e, em seguida, a lista de jogos para ambos os jogadores é atualizada para depois adicionar uma etapa ao jogo. Quando um jogo atinge o limite de etapas, é o momento de criar um novo jogo até o limite de jogos. O limite de etapas, que cresce conforme o número de jogos, é importante para avaliar quanto o tamanho do agregado impacta nos tempos de operação, e o limite de jogos é 4 pois foi suficiente para verificar que a modelagem de NoAM apresentou desempenho inferior à nossa proposta. A linha 2 faz uma escrita, a linha 3 faz duas leituras e duas escritas, a linha 5 faz uma leitura e uma escrita. Um ciclo completo precisa de $5+2*(\text{número de jogos}*100)$ leituras e escritas. Então, ao fim do quarto jogo, o experimento alcançou 2.020 operações (leituras e escritas).

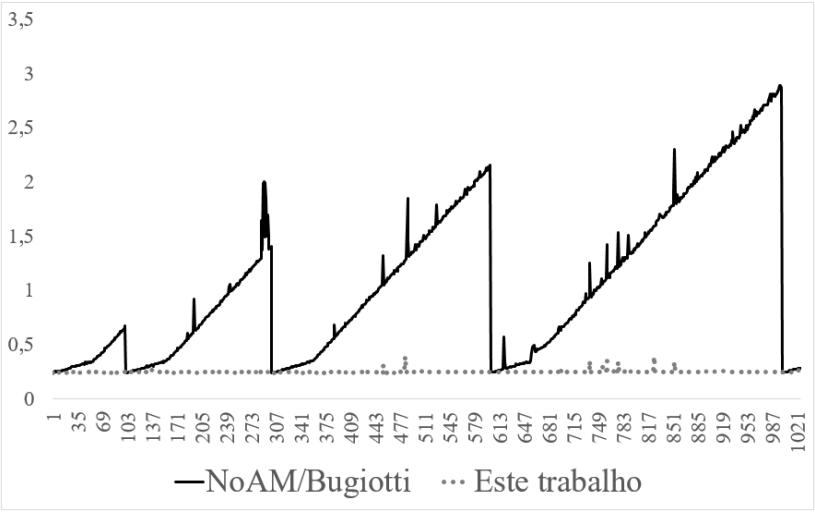
Algoritmo 7: Implementação do experimento

```
1 enquanto número de jogos <= 4 faça
2 |   Criar um jogo com dois jogadores
3 |   Atualizar lista de jogos dos jogadores
4 |   enquanto número de etapas <= número de jogos * 100
  |    faça
5 |   |   Adicionar uma etapa ao jogo criado
6 |   fim
7 fim
```

As Figuras 23 e 24 apresentam os gráficos com os resultados dos experimentos. Todos os gráficos comparam o desempenho da proposta NoAM (linha sólida) com a proposta deste trabalho (linha pontilhada). O eixo X representa o número de iterações do algoritmo e o eixo Y é a média de tempo gasto em segundos. A Figura 23 mostra apenas o desempenho das operações de escrita dos quatro jogos e suas etapas, enquanto que a Figura 24 mostra apenas o desempenho das operações de leitura. As quedas drásticas nestes gráficos (perto das iterações 100, 300, 600 e 1000) acontecem no início de um novo jogo (etapas zeradas). A Figura 25 aproxima a criação do primeiro jogo com suas primeiras 100 etapas, misturando leituras e escritas.

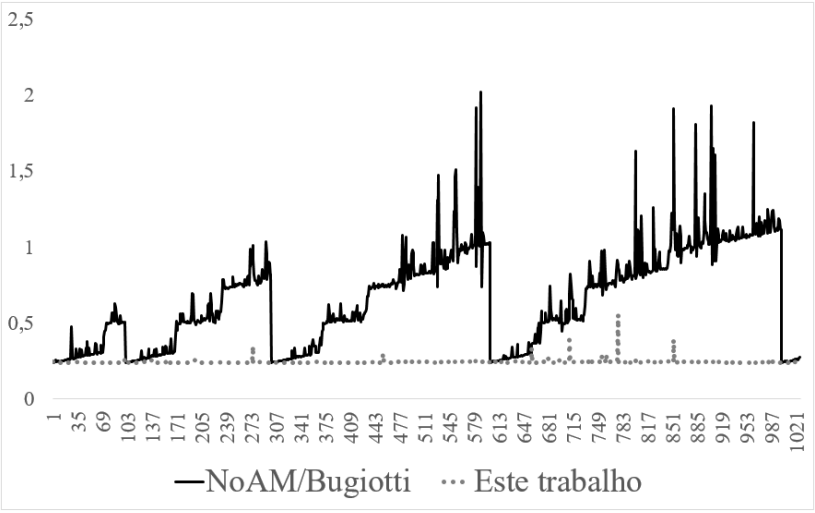
[1] A implementação deste experimento está disponível em `https://github.com/jopapo/thesisExperiment`.

Figura 23: Média de tempo de escrita para mil iterações.



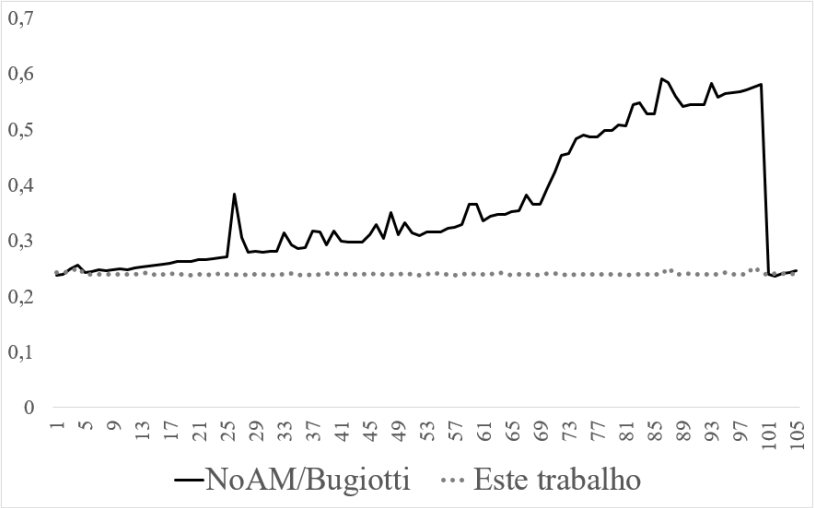
Fonte: Elaborada pelo autor.

Figura 24: Média de tempo de leitura para mil iterações.



Fonte: Elaborada pelo autor.

Figura 25: Média de tempo das primeiras cem iterações (escritas e leituras).



Fonte: Elaborada pelo autor.

O experimento demonstra que, enquanto os blocos de dados agregados crescem, na abordagem em *NoAM*, o tempo médio também cresce. Para agregados de tamanho pequeno, o tempo médio é similar em ambas as abordagens. Contudo, quando um único jogo alcança a marca de 70 etapas, a abordagem de *NoAM* cai 50% em termos de desempenho, comparado com a abordagem proposta por este trabalho. Com 100 etapas, NoAM é 2,6 vezes mais lento. Por consequência, o aumento do tamanho do agregado impacta quase linearmente à queda de desempenho deles. Este fato não se repete na proposta deste trabalho, pois são manipuladas porções significativamente menores de dados para cada operação e isto ocorre independentemente da profundidade da hierarquia. Assim, a proposta deste trabalho continua escalável mesmo com o crescimento dos dados e das relações hierárquicas entre eles. O único inconveniente é que, para obter todos os dados da hierarquia, esta proposta necessita de várias consultas. Porém, estas consultas têm um impacto minimizado em desempenho, principalmente devido a recentes avanços tecnológicos em termos de *Application Programming Interfaces* (API), que permitem que várias consultas sejam executadas em uma única requisição ao servidor.

Os resultados obtidos com este experimento foram considerados satisfatórios, uma vez que obtiveram desempenho superior ao principal trabalho relacionado (*baseline*) disponível na literatura ($NoAM$), para a mesma avaliação experimental realizada por este *baseline*.

6 CONCLUSÃO

O principal objetivo desta dissertação é estabelecer uma conexão entre o projeto clássico de BDs e BDs colunares. Os métodos, técnicas e tecnologias presentes na literatura se mostram alheios à modelagem conceitual associada a BDs colunares. Este objetivo foi alcançado através da proposta de uma abordagem eficiente para conversão de um esquema conceitual EER em um esquema lógico para BDs colunares. Dentre as contribuições deste trabalho estão uma notação lógica para BDs colunares, um conjunto de algoritmos de conversão que geram um esquema lógico nesta notação, uma abordagem diferente da de agregados, bem como uma avaliação experimental que compara esta abordagem com um trabalho relacionado similar (abordagem genérica de agregados abstrata em NoAM), apresentando resultados promissores[1].

A notação proposta define um conjunto mínimo de conceitos necessários para alcançar uma estrutura adequada a ser implementada em BDs colunares. A avaliação experimental apresenta uma modelagem de dados para BDs colunares que se revela impraticável para a abordagem NoAM (BUGIOTTI et al., 2014), porém viável nesta abordagem. Isso evidencia que uma das abordagens da literatura para projeto de BDs NoSQL (baseada em agregados) pode não ser a melhor solução para todos os casos, e apresenta a nossa solução como uma alternativa que apresenta resultados melhores. Com base em tecnologias recentes é possível atingir resultados ainda mais notáveis usando técnicas conhecidas enquanto elas vão sendo disponibilizadas com a evolução de BDs colunares (como chaves compostas, comandos múltiplos por requisição, linguagem de consulta gerais etc).

Dados massivos e escaláveis não são somente utilizados para fins de mineração de dados. Este trabalho progride em uma área que urge em fazer do NoSQL uma alternativa confiável ao projeto clássico de BDs. Domínios onde dados tendem a crescer muito rápido requerem estratégias de modelagem eficientes, como a proposta nesta dissertação.

[1] Um artigo denominado *A Logical Design Process for Columnar Databases*, proveniente desta dissertação, foi aceito e publicado na conferência ICIW 2016 (*International Conference on Internet and Web Applications and Services*).

Trabalhos futuros incluem experimentos em outros domínios de *Big Data*, como redes sociais, e com um volume maior de dados, bem como considerar informações sobre a carga de trabalho da aplicação (principais consultas e atualizações) para fins de otimização do processo proposto. Informações de carga são importantes como guia para definir estruturas lógicas mais adequadas às operações mais frequentes que utilizarão o BD. Considera-se ainda, como atividade futura, a automatização deste processo de conversão através do desenvolvimento de uma ferramenta de modelagem conceitual EER capaz de gerar o esquema lógico proposto nesta dissertação.

REFERÊNCIAS

BREWER, E. Cap twelve years later: How the rules have changed. **Computer**, IEEE, v. 45, n. 2, p. 23–29, 2012.

BREWER, E. A. Towards robust distributed systems. In: **PODC**. [S.l.: s.n.], 2000. v. 7.

BUGIOTTI, F. et al. Database design for nosql systems. In: **Conceptual Modeling**. [S.l.]: Springer, 2014. p. 223–231.

CABIBBO, L. Ondm: An object-nosql datastore mapper. **Faculty of Engineering, Roma Tre University. Retrieved June 15th**, 2013.

CATTELL, R. Scalable sql and nosql data stores. **SIGMOD Record**, v. 39, n. 4, p. 13, 2010.

CHEBOTKO, A.; KASHLEV, A.; LU, S. A big data modeling methodology for apache cassandra. In: IEEE. **Big Data (BigData Congress), 2015 IEEE International Congress**. [S.l.], 2015. p. 238–245.

CURINO, C. et al. Relational cloud: A database service for the cloud. In: **5th Biennial Conference on Innovative Data Systems Research**. Asilomar, CA: [s.n.], 2011.

ELMASRI, R.; NAVATHE, S. B. Fundamentals of database systems. Addison-Wesley, 2000.

ELMASRI, R.; NAVATHE, S. B. **Sistemas de banco de dados**. [S.l.]: Pearson, 2005. ISBN 8588639173.

FONG, J. **Mapping Extended Entity-Relationship Model to Object Modeling Technique**. 3. ed. [S.l.]: SIGMOD Record, 1995.

Fundação Apache. **Cassandra Wiki**. 2009. Disponível em: <http://wiki.apache.org/cassandra>. Acesso em: 29.10.2015.

GILBERT, S.; LYNCH, N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. **ACM SIGACT News**, ACM, v. 33, n. 2, p. 51–59, 2002.

HEUSER, C. A. **Projeto de Banco de Dados**. 6. ed. [S.l.]: Instituto de Informática da UFRGS, 2008.

HOFF, T. **What The Heck Are You Actually Using NoSQL For?** 2010. Disponível em: <http://highscalability.com/blog/2010/12/6/what-the-heck-are-you-actually-using-nosql-for.html>. Acesso em: 01.02.2015.

KAUR, K.; RANI, R. Modeling and querying data in nosql databases. In: **Big Data, 2013 IEEE International Conference on**. [S.l.: s.n.], 2013. p. 1–7.

KIM, W. **Introduction to object-oriented databases**. [S.l.]: MIT press Cambridge, MA, 1990.

KRISTENSEN, B. B. **Object-oriented modeling with roles**. [S.l.]: Springer, 1996.

MATHIES, R. **Cassandra Data Model**. 2015. Disponível em: <http://wiki.apache.org/cassandra/DataModelv2>. Acesso em: 04.02.2015.

MEIJER, E.; BIERMAN, G. A co–relational model of data for large shared data banks. **Communications of the ACM**, ACM, v. 54, n. 4, p. 49–58, 2011.

MILANOVIĆ, A.; MIJAJLOVIĆ, M. A survey of post-relational data management and nosql movement. **Faculty of Mathematics University of Belgrade, Serbia**, 2012.

MONIRUZZAMAN, A. B. M.; HOSSAIN, S. A. Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. **International Journal of Database Theory and Application**, 2013.

NAVATHE, S. B.; BATINI, C.; CERI, S. Conceptual database design: an entity-relationship approach. **Redwood City: Benjamin Cummings**, 1992.

POKORNY, J. Nosql databases: a step to database scalability in web environment. **International Journal of Web Information Systems**, Emerald Group Publishing Limited, v. 9, n. 1, p. 69–82, 2013.

PRITCHETT, D. Base: An acid alternative. **Queue**, ACM, v. 6, n. 3, p. 48–55, 2008.

SADALAGE, P. J.; FOWLER, M. Nosql distilled. Addison-Wesley, 2013.

SCHRAM, A.; ANDERSON, K. M. Mysql to nosql: data modeling challenges in supporting scalability. In: ACM. **Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity**. [S.l.], 2012. p. 191–202.

SCHROEDER, R.; MELLO, R. d. S. Uma abordagem para projeto lógico de banco de dados xml baseada em informações de carga de dados. Dissertação de mestrado. Universidade Federal de Santa Catarina, 2008.

SCHWARTZ, B. **Schemaless Databases don't exist**. 2015. Disponível em: <https://vividcortex.com/blog/2015/02/24/schemaless-databases-dont-exist/>. Acesso em: 26.02.2015.

SHARP, J. et al. Data access for highly-scalable solutions: Using sql, nosql, and polyglot persistence. Microsoft patterns & practices, 2013.

Solid IT. **DB–Engines Ranking**. 2016. Disponível em: <http://db-engines.com/en/ranking>. Acesso em: 26.01.2016.

WANG, G.; TANG, J. The nosql principles and basic application of cassandra model. In: **Computer Science Service System (CSSS), 2012 on International Conference**. [S.l.: s.n.], 2012. p. 1332–1335.