Cláudio de Lima

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

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina para a obtenção do Grau de Mestre em Ciência 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.

Lima, Cláudio de
Projeto lógico de bancos de dados NoSQL documento a partir de esquemas conceituais Entidade-Relacionamento Estendido (EER) / Cláudio de Lima ; orientador, Ronaldo dos Santos Mello - Florianópolis, SC, 2016.
140 p.

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

Inclui referências

1. Ciência da Computação. 2. Banco de dados. 3. Projeto lógico NoSQL. 4. Modelo conceitual EER. 5. Modelo lógico NoSQL documento. I. Mello, Ronaldo dos Santos. II. Universidade Federal de Santa Catarina. Programa de Pós Graduação em Ciência da Computação. III. Título.

Cláudio de Lima

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

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

Florianópolis, 08 de Março de 2016.

_____
Prof. Carina Friedrich Dorneles, Dr.ª
Coordenadora do Curso

Banca Examinadora:

_____
Prof. Ronaldo dos Santos Mello, Dr.
Orientador
Universidade Federal de Santa Catarina

_____
Prof.ª Renata de Matos Galante, Dr.ª
Universidade Federal do Rio Grande do Sul (Videoconferência)

_____
Prof.ª Patrícia Vilain, Dr.ª
Universidade Federal de Santa Catarina

_____
Prof. Roberto Willrich, Dr.
Universidade Federal de Santa Catarina

Em memória à minha querida avó, Zilda.

AGRADECIMENTOS

Agradeço primeiramente a minha esposa, Camila, pelo seu apoio incondicional, companheirismo, amor e amizade, além da compreensão nos momentos de ausência física e espiritual. Com carinho, agradeço a minha mãe, Suzana, e a minha tia, Suely, que me deram apoio para o desenvolvimento deste trabalho.

Agradeço ao meu professor e orientador, Ronaldo, por mais uma oportunidade de orientação, e por todo apoio e auxílio no desenvolvimento deste trabalho.

Agradeço aos amigos dos tempos de SESC, pelo apoio, amizade e todo aprendizado adquirido, e também aos novos colegas e amigos da SeTIC/UFSC, em especial, ao Rodrigo, grande incentivador.

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

O movimento denominado *NoSQL* surge como tendência para solucionar os desafios inerentes às necessidades atuais de gerenciamento de dados na nuvem, como o tratamento de grandes volumes de dados, a escalabilidade horizontal e o suporte a modelos flexíveis de armazenamento de dados. A organização dos dados em BDs NoSQL requer significativas decisões de projeto, uma vez que afetam requisitos como escalabilidade, desempenho e consistência. Embora BDs NoSQL não requeiram um *esquema padrão* associado aos dados, eles são categorizados por modelos de dados. O presente trabalho está inserido nesta problemática e propõe uma abordagem para projeto lógico de BDs NoSQL que seguem o modelo de dados de documento. Este modelo é flexível quanto ao suporte a dados e apropriado para aplicações Web, e a abordagem define processos que convertem modelagens conceituais para representações lógicas adequadas e eficientes, para fins de manipulação correta, armazenamento e acesso a dados na nuvem. A proposta é constituída por regras de conversão capazes de transformar cada um dos construtores do modelo conceitual Entidade-Relacionamento Estendido (EER) em uma representação lógica para BDs NoSQL da categoria *documento*. Um processo de conversão *EER-NoSQL* automático é proposto, com a finalidade de ordenar a aplicação das regras na produção de um esquema NoSQL documento que tenta evitar a redundância de dados e, ao mesmo tempo, procura gerar uma representação bem estruturada das informações modeladas pelo projeto conceitual. A consideração de informações relativas à principal carga estimada para o BD que está sendo modelado produz esquemas NoSQL documento otimizados. O estudo de caso apresentado demonstra o ganho obtido por documentos conformados a estes esquemas, no tempo de ocupação diário do sistema, para a execução das operações mais frequentes do BD.

Palavras-chave: Banco de dados NoSQL documento. Projeto lógico de banco de dados. Esquema conceitual EER. Modelagem de dados NoSQL. Esquema lógico NoSQL documento.

ABSTRACT

The movement called *NoSQL* comes as a tendency to address the challenges related to the management of data in the cloud, like the processing of large volumes of data, the horizontal scalability and the support for flexible forms of data storage. Data organization on NoSQL databases (DBs) requires significant design decisions, since they affect requirements such as scalability, performance and consistency. Although NoSQL DBs do not require a *default schema* associated with the data, they are categorized by data models. This work addresses this problem by proposing an approach for the logical design of NoSQL document DBs that follows the document data model. This data model is flexible in terms of data support and suitable for Web applications, and the approach defines processes that convert a conceptual modeling for proper and efficient logical representations aiming at the correct handling, storage and access to data in the cloud. Out proposal consists of conversion rules that transform each one of the concepts of the Extended Entity-Relationship (EER) conceptual model to a logical representation in a NoSQL DB of the *document* category. An automatic conversion process *EER-NoSQL* is proposed with the purpose to order the execution of the rules for generating a NoSQL document schema that tries to avoid data redundancy and, at the same time, tries to generate a well-structured representation of the conceptual schema information. Additionally, our methodology considers the information workload for the DB being modeled in order to produce an optimized NoSQL document schema. A case study presented in this work shows the efficiency improvement obtained, in terms of accessing time, for documents that respects the generated logical schemas, on considering the frequent DB operations.

Keywords: NoSQL document database. Database logical design. EER conceptual schema. NoSQL data modeling. NoSQL document logical schema.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACID – Atomicidade, Consistência, Isolamento e Durabilidade
ACO – Atributo Composto
ANC – Atributo Não-Composto
BASE – Basically Available, Soft state, Eventually consistent
BD – Banco de Dados
BDR – Banco de Dados Relacional
BDOO – Banco de Dados Orientado a Objetos
BSON – Binary JSON
CAP – Consistency, Availability, Partition Tolerance
CPU – Central Processing Unit
DaaS – Database as a Service
DDD – Domain Driven Design
DDL – Data Definition Language
DTD – Document Type Definition
DHT – Distributed Hash Table
ENT – Entidade
ER – Entidade-Relacionamento
EER – Entidade-Relacionamento Estendido
ETL – Extract, Transform, Load
FAG – Frequência de Acesso Geral
FAM – Frequência de Acesso Mínima
FAT – Frequência de Acesso Total
GHI – Generalização com Ênfase na Hierarquia
GSB – Generalização com Ênfase nas Subclasses
GSP – Generalização com Ênfase na Superclasse
JSON – JavaScript Object Notation
IDEF1X – Integration DEFinition for Information Modeling
NoSQL – Not only SQL
MVCC – Multiversion Concurrency Control
NoAM – NoSQL Abstract Model
OO – Orientado a Objetos
OMT – Object Modeling Technique
RAM – Random Access Memory
RBU – Relacionamento Modelado por Bloco Único
RHI – Relacionamento Modelado por Hierarquia
RRE – Relacionamento Modelado por Referências
SGBD – Sistema de Gerenciamento de Banco de Dados
SGBDOO – Sistema de Gerenciamento de Banco de Dados OO
SOA – Service Oriented Architecture

SQL – Structured Query Language
SSD – Solid State Drive
TI – Tecnologia da Informação
UHI – União com Ênfase na Hierarquia
UML – Unified Modeling Language
USB – União com Ênfase na Subclasse
USP – União com Ênfase nas Superclasses
XML – eXtensible Markup Language
YAML – YAML Ain't Markup Language

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta a contextualização do trabalho, incluindo a motivação, o problema e a proposta para solucionar o problema. Na sequência, são apresentados os objetivos, as atividades de pesquisa e a organização do trabalho.

1.1 CONTEXTUALIZAÇÃO

Computação em nuvem é um paradigma que visa prover serviços sob demanda e com pagamento baseado no uso, considerando desde o usuário final que hospeda seus documentos pessoais na Internet até empresas que terceirizam toda infraestrutura de Tecnologia da Informação (TI). Neste contexto, a imensa quantidade de dados gerados diariamente em vários domínios de aplicação, como por exemplo, na Web e em redes sociais, traz grandes desafios na forma de manipulação, armazenamento e processamento de consultas em várias áreas da Computação, inclusive na área de Banco de Dados (BD) (VIEIRA et al., 2012).

Um ambiente de computação em nuvem é composto por importantes modelos de serviços, que definem um padrão arquitetural para as soluções. Um desses modelos é o modelo de serviço de *Sistema de Banco de Dados como um Serviço* (DaaS). Ele surge como um paradigma para a gestão de dados em ambientes corporativos. Neste paradigma, um prestador hospeda um BD e o fornece como um serviço (AGRAWAL et al., 2009). Um DaaS introduz diversos desafios de pesquisa à comunidade de BD, incluindo, por exemplo, segurança, gerenciamento de recursos compartilhados e extensibilidade.

Ambientes de computação em nuvem têm sido utilizados para o gerenciamento de dados do tipo *Big Data*. O *Big Data* pode ser definido como o processamento analítico, eficiente e escalável, de grandes volumes de dados complexos produzidos por várias aplicações (VIEIRA et al., 2012). Aplicações científicas e de engenharias, redes sociais, redes de sensores, dados médicos e biológicos, bem como transações de comércio eletrônico (*e-commerce*) são exemplos de aplicações no contexto *Big Data*.

Os tradicionais Sistemas Gerenciadores de BD (SGBDs) relacionais não são os mais adequados às necessidades atuais de gerenciamento de dados na nuvem, e às necessidades do domínio do problema do *Big Data*, como a execução de consultas com baixa latência, o tratamento de grandes volumes de dados, a escalabilidade elástica horizontal, o suporte a modelos flexíveis de armazenamento de dados, e o suporte eficiente à replicação e distribuição dos dados (VIEIRA et al., 2012). Uma tendência para solucionar os desafios inerentes a esta problemática é o movimento denominado *NoSQL*, que consiste

em SGBDs não-relacionais projetados para gerenciar grandes volumes de dados e que disponibilizam estruturas e interfaces de acesso simples (SOUSA et al., 2010). Os BDs NoSQL proporcionam um grande número de operações de leitura e escrita por segundo, característica comum em aplicações Web modernas (CATTELL, 2010).

O suporte a tipos de dados complexos, semiestruturados ou não estruturados também favorece o uso de BDs NoSQL, que atualmente são categorizados nos seguintes modelos de dados: *chave-valor*, *documento*, *colunar* e *baseado em grafos* (MCMURTRY et al., 2013; VIEIRA et al., 2012). O modelo *chave-valor* é o mais simples e consiste essencialmente em uma grande *tabela hash* onde um valor é associado a uma chave única que é utilizada para recuperar os dados no BD. O modelo de dados *colunar* organiza seus dados em linhas e colunas e a sua principal característica é a sua abordagem desnormalizada para estruturar dados esparsos. BDs colunares podem ser vistos como a exploração de dados tabulares com a divisão das colunas em grupos conhecidos como famílias de colunas. Os BDs NoSQL *baseados em grafos* têm como foco principal os relacionamentos que entidades têm umas com as outras. A principal finalidade desta categoria de BD NoSQL é permitir que uma determinada aplicação execute eficientemente consultas que atravessam uma rede de nós e arestas, e analise os relacionamentos entre as entidades.

A categoria de BDs NoSQL documento é semelhante em conceito a um BD chave-valor, exceto pelo fato que os valores armazenados são documentos. Um documento é um conjunto de campos e valores nomeados (pares chave-valor), sendo que cada valor de campo pode conter um item escalar simples ou um item composto, tal como uma lista ou um documento aninhado. Os dados nos campos de um documento podem ser codificados em uma variedade de maneiras, incluindo JSON (JSON, 2014), BSON, XML, YAML, ou mesmo armazenados como texto simples. Os BDs NoSQL orientados a documento formam uma categoria apropriada para aplicações Web, que envolve o armazenamento de dados semiestruturados e a execução de consultas dinâmicas (KAUR; RANI, 2013). Estes BDs são capazes de suportar escalabilidade horizontal, proporcionar alta disponibilidade e serem flexíveis quanto ao suporte a dados. Por estes motivos, esta categoria de BD NoSQL foi escolhida para fins deste trabalho.

A organização dos dados em BDs NoSQL requer significativas decisões de projeto, uma vez que elas afetam os requisitos principais de qualidade, incluindo escalabilidade, desempenho e consistência (BUGIOTTI et al., 2014). Para que se obtenha acesso a um imenso volume de dados disponível com maior exatidão, é necessário associar semântica aos dados. Esquemas conceituais e ontologias são fundamentais no processo de associação de

semântica aos dados. A importância de um modelo associado aos dados está no melhor entendimento sobre os dados e na demonstração de como os dados são persistidos no BD alvo.

O projeto tradicional de BDs é um processo constituído por três fases de modelagem de dados (BATINI; CERI; NAVATHE, 1992; ELMASRI; NAVATHE, 2011): conceitual, lógica e física. Inicialmente, na etapa de modelagem conceitual um esquema indicando as informações de um domínio é representado em um modelo de alto nível de abstração. Na sequência, na modelagem lógica, o esquema conceitual é transformado em um esquema de mais baixa abstração adequado ao modelo de dados alvo, ou seja, o modelo no qual o BD será fisicamente implementado. Entende-se por modelo lógico o modelo que representa a classe, ou a natureza dos dados manipulados pelo SGBD. Exemplos de modelos lógicos são os modelos *hierárquicos*, *orientados a objetos* e *relacionais*.

Embora BDs NoSQL não requeiram um *esquema padrão* associado aos dados, estes dados apresentam alguma estrutura, e muitas vezes torna-se necessário obter vantagens desta estrutura. Metodologias e ferramentas de suporte ao projeto de BDs NoSQL é um tópico ainda pouco explorado na literatura de BDs (ATZENI; BUGIOTTI; ROSSI, 2012). Atualmente, o projeto de BDs NoSQL é geralmente baseado em melhores práticas e orientações (KATSOV, 2012), sem uma metodologia sistemática. Muitos autores observaram que o desenvolvimento de metodologias de alto nível e ferramentas de suporte ao projeto de BDs NoSQL são necessárias (ATZENI et al., 2013; BADIA; LEMIRE, 2011; BUGIOTTI et al., 2014; HSIEH, 2014; KATSOV, 2012; MAGUIRE; O'KELLY, 2013; MOHAN, 2013). No contexto acadêmico, percebe-se uma carência de trabalhos que propõem metodologias de projeto para BDs NoSQL (BUGIOTTI et al., 2014; CHEBOTKO; KASHLEV; LU, 2015; JOVANOVIC; BENSON, 2013), ou seja, de processos que convertam modelagens conceituais para representações lógicas adequadas e eficientes, para fins de manipulação correta, armazenamento e acesso a dados na nuvem.

Este trabalho aborda esta problemática com uma proposta para o projeto lógico de BDs NoSQL que seguem o modelo de dados de documento, modelo esse que é flexível quanto ao suporte a dados e apropriado para aplicações Web. Esta proposta apresenta, concretamente, uma abordagem de projeto composta por processos que convertem, através de regras pré-definidas e de um processo principal que controla a execução dessas regras, modelagens conceituais para representações lógicas adequadas e eficientes para BDs NoSQL de documento, contribuindo, assim, para a área de gerência de dados na nuvem.

1.2 PERGUNTA DE PESQUISA

Esta dissertação busca responder a seguinte pergunta de pesquisa: É possível definir melhores estratégias para persistência e manipulação de dados, para BDs NoSQL da categoria documento, utilizando uma abordagem de projeto para BDs NoSQL que converte, através de processo automático, esquemas conceituais em esquemas lógicos adequados ao BD destino?

1.3 OBJETIVOS

O objetivo desta dissertação é definir uma abordagem capaz de apoiar o projeto lógico de BDs NoSQL de documento a partir de uma modelagem conceitual clássica definida no modelo Entidade-Relacionamento Estendido (EER), representando adequadamente as informações modeladas no projeto conceitual em um esquema otimizado definido por um modelo lógico NoSQL que suporta o modelo de dados de documento, modelo lógico esse capaz de suportar escalabilidade horizontal e ser flexível quanto ao suporte a dados semiestruturados.

Os objetivos específicos desta dissertação são:

- Formalizar um modelo de dados lógico para BDs NoSQL documento. A abordagem de conversão deverá produzir esquemas que respeitam um modelo lógico NoSQL documento adequado ao nível de projeto lógico de um BD NoSQL nesta categoria;

- Definir regras de conversão para os construtores do modelo EER. Um conjunto de regras de conversão deverá ser definido por este trabalho com a finalidade de prover estratégias de conversão de todos os construtores conceituais do modelo EER para um esquema lógico correspondente de um BD NoSQL documento;

- Estabelecer um processo geral de conversão *EER - NoSQL* para a aplicação das regras. Um processo de conversão deverá ser definido na forma de um algoritmo para a aplicação organizada das regras de conversão. Este algoritmo deverá produzir esquemas NoSQL documento compactos, não redundantes e bem estruturados quanto às restrições impostas pelos relacionamentos entre os conceitos do esquema conceitual;

- Estender o processo de conversão para considerar uma Análise da Carga do BD. Uma modelagem de carga de um BD tem a finalidade de gerar informações que guiam o processo de conversão na produção de esquemas NoSQL documento mais otimizados. O processo de conversão *EER–NoSQL* deverá considerar informações da carga estimada para o BD;

- Avaliar a eficácia da abordagem proposta através de estudo de caso. Os esquemas lógicos NoSQL documento produzidos pela abordagem deverão ser avaliados através de experimento.

1.4 ATIVIDADES DE PESQUISA

As etapas realizadas para alcançar os objetivos desta dissertação foram as seguintes:

- Pesquisa do *estado da arte* na área de modelagem de dados e projeto de bancos de dados para NoSQL;

- Estudo de *modelos lógicos* para BDs NoSQL da categoria documento;

- Formalização de um modelo de dados lógico para BDs NoSQL documento;

- Definição de regras de conversão para os construtores do modelo conceitual EER;

- Estabelecimento de um *processo de conversão* para a aplicação das regras, incluindo a consideração de informações da carga de dados estimada para o BD;

- Avaliação dos esquemas lógicos NoSQL documento produzidos pelo processo de conversão proposto através de estudo de caso;

- Análise dos resultados gerados a partir do estudo de caso, incluindo a *avaliação do desempenho* medindo o volume de acesso gerado por operações sobre documentos conformados a cada esquema lógico NoSQL documento gerado.

1.5 ORGANIZAÇÃO

Este trabalho está organizado em mais 5 capítulos. O capítulo 2 é dedicado à fundamentação teórica. Ele inicia com a apresentação do movimento NoSQL, incluindo características de BDs NoSQL da categoria documento, que é o modelo de dados alvo da abordagem proposta. Com a finalidade de embasar o projeto lógico de BDs NoSQL, são apresentados os princípios da metodologia para projeto de BDs tradicionais, com o foco na modelagem lógica, incluindo características do modelo conceitual EER. Também é introduzido o conceito de *agregados*, termo definido na área de *Domain-Driven Design* (DDD) (EVANS, 2003), que compõe a abordagem lógica proposta por este trabalho. Por fim, uma metodologia para a modelagem da carga de dados de um BD, que representa a carga esperada para um BD e tem por objetivo produzir informações de apoio ao processo de conversão proposto neste trabalho, é apresentada.

O capítulo 3 apresenta uma análise dos trabalhos relacionados a metodologias para o projeto de esquemas NoSQL. Esta análise visa identificar contribuições destes trabalhos para o projeto lógico de BDs NoSQL documento, bem como identificar pontos em aberto nesta área. Além disso, abordagens relevantes para a modelagem de dados NoSQL e que utilizam esquemas não-relacionais, como esquemas XML e Orientados a Objetos (OO), são revistas neste capítulo.

O capítulo 4 apresenta a proposta da abordagem para conversão de esquemas EER em esquemas lógicos NoSQL documento. Neste capítulo é definido o modelo lógico utilizado pela abordagem, as regras de conversão aplicadas sobre cada construtor do EER e o processo de conversão que define a ordem de aplicação das regras sobre os fragmentos do esquema conceitual. Ainda neste capítulo é apresentada a consideração de informações de carga estimadas para o BD NoSQL documento pelo processo de conversão. Estas informações são utilizadas pelo processo para geração de estruturas NoSQL documento otimizadas capazes de responder com mais eficiência às operações esperadas para o BD.

O capítulo 5 apresenta um estudo de caso para avaliar os esquemas lógicos NoSQL documento produzidos pela abordagem de conversão proposta por este trabalho. Um esquema EER representando o domínio de *e-commerce*, obtido através de um processo de *engenharia reversa* de uma aplicação real, foi utilizado como base para a produção dos esquemas NoSQL documento. As etapas de transformação do esquema EER em esquemas lógicos NoSQL documento são descritas detalhadamente. Na avaliação experimental, documentos JSON foram produzidos a partir destes esquemas com a finalidade

de executar operações sobre dados conformados aos respectivos esquemas lógicos NoSQL documento.

As contribuições e as conclusões referentes a esta dissertação são apresentadas pelo capítulo 6, juntamente com sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta o embasamento teórico para este trabalho. Inicialmente, são apresentados estudos a respeito de *NoSQL*, uma classe de BDs não-relacionais projetados para gerenciar grandes volumes de dados, e que atualmente são categorizados por modelos de dados (*chave-valor*, *documento*, *colunar* e *baseado em grafos*) (MCMURTRY et al., 2013; VIEIRA et al., 2012). Características de BDs NoSQL da categoria documento, modelo de dados alvo da abordagem proposta, também são apresentadas.

Com a finalidade de embasar o projeto lógico de BDs NoSQL, apresentam-se os princípios da metodologia para projeto de BDs tradicionais, com o foco na modelagem lógica, incluindo características do modelo conceitual EER. O conceito de *agregados*, termo definido na área de *Domain-Driven Design* (DDD) (EVANS, 2003), também é introduzido, uma vez que faz parte da solução proposta por este trabalho. Por fim, aborda-se a modelagem de carga de dados, que representa a carga esperada para um BD e que tem por objetivo produzir informações de apoio ao processo de conversão proposto neste trabalho.

2.1 NOSQL

O movimento NoSQL está fortemente relacionado ao paradigma de computação em nuvem. Computação em nuvem é uma tendência de tecnologia cujo objetivo é proporcionar serviços de Tecnologia da Informação (TI) sob demanda e com pagamento baseado no uso. Trata-se de uma abordagem global que visa prover serviços para as massas, que vão desde o usuário final que hospeda seus documentos pessoais na Internet até empresas que terceirizam toda a sua infraestrutura de TI. A nuvem é uma metáfora para a Internet ou infraestrutura de comunicação entre os componentes arquiteturais de um sistema informatizado, baseada em uma abstração que oculta à complexidade da infraestrutura. Cada parte desta infraestrutura é provida como um serviço e estes são normalmente alocados em centros de dados que utilizam hardware compartilhado para computação e armazenamento (BUYYA et al., 2009). Neste contexto, a imensa quantidade de dados gerados diariamente em vários domínios de aplicação, como por exemplo, na Web e em redes sociais, traz grandes desafios na forma de manipulação, armazenamento e processamento de consultas em várias áreas da Computação, inclusive na área de Banco de Dados (BD) (VIEIRA et al., 2012).

Um ambiente de computação em nuvem é composto por importantes modelos de serviços, que definem um padrão arquitetural para as soluções. Um desses modelos é o modelo de serviço de *Sistema de Banco de Dados como um*

Serviço (DaaS). Ele surge como um paradigma para a gestão de dados em ambientes corporativos. Neste paradigma, um prestador hospeda um BD e o fornece como um serviço (AGRAWAL et al., 2009). Um DaaS introduz diversos desafios de pesquisa à comunidade de BD, incluindo, por exemplo, segurança, gerenciamento de recursos compartilhados e extensibilidade. O MongoDB (MONGODB, 2014) é um exemplo de DaaS.

Ambientes de computação em nuvem têm sido utilizados para o gerenciamento de dados do tipo *Big Data*. O *Big Data* pode ser definido como o processamento analítico, eficiente e escalável, de grandes volumes de dados complexos produzidos por várias aplicações (VIEIRA et al., 2012). Aplicações científicas e de engenharias, redes sociais, redes de sensores, dados médicos e biológicos, bem como transações de comércio eletrônico (*e-commerce*) são exemplos de aplicações no contexto *Big Data*.

Os tradicionais SGBDs relacionais não são os mais adequados às necessidades atuais de gerenciamento de dados na nuvem e às necessidades do domínio do problema do *Big Data*, como a execução de consultas com baixa latência, o tratamento de grandes volumes de dados, a escalabilidade elástica horizontal, o suporte a modelos flexíveis de armazenamento de dados e o suporte simples à replicação e distribuição dos dados (VIEIRA et al., 2012). Uma tendência para lidar com toda essa problemática é o movimento denominado *NoSQL*.

NoSQL é um termo genérico para uma classe definida de SGBDs não-relacionais projetados para gerenciar grandes volumes de dados. Estes BDs proporcionam um grande número de operações de leitura e escrita simples por segundo, que são características comuns em aplicações Web (CATTELL, 2010). Além da alta taxa de geração dos dados, outro fator que influenciou a criação dos BDs NoSQL foi o suporte a tipos de dados complexos, semiestruturados ou não estruturados.

Segundo (TIWARI, 2011), *NoSQL* (ou *Not Only SQL*) é um termo genérico para todos os BDs que não seguem o popular e bem-estabelecido SGBD relacional e muitas vezes se relacionam com grandes conjuntos de dados acessados e manipulados na escala da Web. Isso significa que *NoSQL* não é um único produto ou até mesmo uma única tecnologia. *NoSQL* representa uma classe de produtos e uma coleção de diversos e, por vezes relacionados, conceitos sobre armazenamento e manipulação de dados.

A seguir são apresentadas características-chave acerca de *NoSQL* e características dos *BDs NoSQL* da categoria documento, modelo de dados alvo da abordagem proposta.

2.1.1 Características-Chave

No contexto de *NoSQL*, em se tratando do fator *consistência*, o teorema CAP[1] (*Consistency, Availability, Partition Tolerance*) (BREWER, 2000; GILBERT; LYNCH, 2002) tem sido utilizado com a justificativa de desprezo da consistência, de forma que os produtos normalmente não permitem transações que ultrapassam mais de um nó, ou seja, não há controle de réplicas. Como o teorema CAP justifica essa questão, este tratamento é substituído pela *consistência eventual*. Nessa abordagem, é garantido que todas as réplicas eventualmente convergem ao mesmo estado no momento que a conectividade for restabelecida e passam o tempo necessário para que o sincronismo seja finalizado. Portanto, a justificativa de desprezar a propriedade *consistência* é que as propriedades *disponibilidade* e *tolerância a falhas* continuam sendo garantidas (VIEIRA et al., 2012).

Com relação à manipulação de dados complexos e não estruturados, o controle de tolerância a falhas pode ser complexo. Com o objetivo de fornecer melhor desempenho e alta escalabilidade, muitos produtos *NoSQL* (em contraste com a política de controles de transação do tipo ACID[2]) utilizam a abordagem denominada BASE (*Basically Available, Soft state, Eventually consistente*), que envolve a eventual propagação de atualizações e a não garantia de consistência nas leituras (VIEIRA et al., 2012). A abordagem BASE é implementada de forma diferente em alguns produtos. Por exemplo, alguns produtos se denominam eventualmente consistentes, porém, fornecem algum tipo de consistência como a política de *controle de concorrência multiversionada* (MVCC) (CATTELL, 2010), um mecanismo eficiente que permite que vários processos acessem um determinado dado paralelamente sem corrompê-lo e sem oferecer a possibilidade de ocorrência de *deadlocks*.

Uma propriedade importante nos produtos *NoSQL* é o poder de *escalar horizontalmente* de forma não compartilhada, ou seja, replicando e particionando os dados em diferentes servidores. A escalabilidade horizontal permite que um grande volume de operações de leitura e escrita possa ser executado de forma muito eficiente. Além dos conceitos de particionamento e distribuição serem bem definidos, não existe a aplicação do conceito de BDs federados em produtos *NoSQL*. Em *NoSQL*, todo o BD é considerado um só, enquanto que em BDs federados é possível administrar e usar separadamente

[1] O teorema CAP mostra que sistemas em nuvem podem suportar apenas duas das três propriedades (consistência, disponibilidade e tolerância a falhas) ao mesmo tempo.

[2] As propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) são consideradas em transações e devem ser aplicadas pelos métodos do controle de concorrência e recuperação em SGBDs relacionais (ELMASRI; NAVATHE, 2011).

cada BD e, em alguns momentos, utilizar todos os BDs como se fossem um único. Segundo (VIEIRA et al., 2012), o conceito de escalabilidade vertical está relacionado com o uso de vários núcleos (CPUs) que compartilham memória e discos. Portanto, mais núcleos ou memórias podem ser adicionados para aumentar o desempenho do sistema, porém, essa abordagem é limitada e normalmente é cara. Já a escalabilidade horizontal está relacionada com a funcionalidade de distribuição de dados e de carga por diversos servidores, sem o compartilhamento de memória ou disco.

A escalabilidade horizontal em produtos *NoSQL*, como o Cassandra[3], é alcançada com o particionamento dos dados utilizando a técnica de DHT[4] (*Distributed Hash Table*). Nesta técnica, as entidades de dados são representadas por pares de chave-valor, sendo que a chave identifica unicamente a entidade representada pelo valor. O conjunto de entidades do domínio de dados é organizado em grupos e colocado em um nó do ambiente em questão (VIEIRA et al., 2012).

A técnica *sharding* é outra estratégia de particionamento horizontal dos dados em uma arquitetura sem compartilhamento de recursos. Diferente das técnicas de divisão dos dados por colunas (técnicas de normalização e particionamento vertical), na técnica *sharding* os dados de uma tabela são divididos por tuplas. Cada partição forma parte de um *shard*, sendo que esta pode ser recuperada a partir de um SGBD específico. Inúmeras vantagens de particionamento são obtidas usando esta técnica. Por exemplo, como as tabelas estão divididas e distribuídas em múltiplos servidores, o número total de tuplas em cada tabela de cada servidor é reduzido e, consequentemente, o tamanho dos índices também é reduzido, o que geralmente melhora o desempenho de consultas (VIEIRA et al., 2012).

Algumas soluções propõem o uso integrado de SGBD relacional e *NoSQL*. Estas soluções armazenam os dados utilizando arquiteturas *NoSQL* e utilizam o paradigma *MapReduce* para processos ETL (Extração, Transformação e Carga), por exemplo, na geração de cubos em produtos *Data Warehouse* tradicionais. As linguagens de consulta construídas sobre uma plataforma *MapReduce* possuem limitações para os programadores quando comparadas com o processamento *MapReduce* tradicional, incluindo otimizações relativas à diminuição da transferência de dados pela rede (VIEIRA et al., 2012).

---

[3] cassandra.apache.org/

[4] As DHTs fazem parte de uma classe de sistemas distribuídos descentralizados que proveem um serviço de busca similar a uma *tabela hash*. Pesquisas em DHT foram motivadas por sistemas *peer-to-peer* (P2P). As primeiras quatro DHTs - *CAN*, *Chord*, *Pastry* e *Tapestry* - foram introduzidas quase que simultaneamente em 2001.

Os BDs NoSQL não requerem um esquema associado aos dados e, por esse motivo, são considerados adequados a aplicações do tipo *data-driven*, ou seja, concentram-se no modelo de consulta e constroem o modelo de dados ao redor dele, a fim de satisfazerem suas necessidades de forma eficiente (SILVA, 2011). Entretanto, estes dados em geral apresentam alguma estrutura e obter vantagens desta estrutura torna-se muitas vezes necessário. A persistência de dados de aplicações deve ser mapeada para itens de dados modelados (elementos), como documentos e pares chave-valor, disponíveis no sistema destino, e, portanto, a organização dos dados em BDs NoSQL requer significativas decisões de projeto. Segundo (BUGIOTTI et al., 2014), estas decisões afetam os requisitos principais de qualidade, incluindo escalabilidade e desempenho, bem como a consistência.

Os BDs NoSQL estão disponíveis em uma variedade de formas e funcionalidades. Segundo (MCMURTRY et al., 2013; VIEIRA et al., 2012), atualmente estes BDs são categorizados em *chave-valor*, *documento*, *colunar* e *baseado em grafos*. Um BD NoSQL chave-valor implementa o mais simples dos mecanismos de armazenamento NoSQL, pelo menos conceitualmente. Um BD chave-valor é essencialmente uma grande *tabela hash*, sendo que cada valor é associado a uma chave única que é utilizada para determinar o local de armazenamento e recuperação do dado. Uma *tabela hash* é a estrutura de dados mais simples que pode conter um conjunto de par chave-valor. Essas estruturas de dados são extremamente populares porque proporcionam um algoritmo muito eficiente para acessar os dados, de complexidade de tempo médio *O(1)*. A chave do par chave-valor é um valor único no conjunto e pode ser facilmente encontrado para acessar os dados (TIWARI, 2011). Exemplos de BDs chave-valor são: *Voldemort*[5], *Riak*[6] e *Amazon DynamoDB*[7].

Os BDs NoSQL colunares organizam seus dados em linhas e colunas. Em sua forma mais simples eles se assemelham logicamente aos BDs relacionais. No entanto, a grande característica de um BD colunar encontra-se na sua abordagem desnormalizada para estruturar dados esparsos. É possível pensar em um BD colunar como a exploração de dados tabulares (compreendendo linhas e colunas) com a possibilidade de dividir colunas em grupos conhecidos como famílias de colunas. Cada coluna da família possui um conjunto de colunas que são logicamente relacionadas (MCMURTRY et al., 2013). Segundo (TIWARI, 2011), cada unidade de dados neste modelo pode ser considerada como um conjunto de pares de chave-valor, em que a unidade em si é identificada com a ajuda de um identificador primário,

[5] www.project-voldemort.com/voldemort/
[6] basho.com/products/riak-kv
[7] aws.amazon.com/pt/dynamodb/

frequentemente referido como a chave primária, chamada de *row-key*, sendo esta unidade armazenada de forma ordenada. As unidades de dados são classificadas e ordenadas com base na *row-key*. Deste modo, o armazenamento orientado a colunas permite que dados sejam armazenados eficazmente (TIWARI, 2011). Exemplos de BDs colunares são: *Cassandra*, *Hbase*[8] e *Hypertable*[9].

A categoria de BDs NoSQL baseado em grafos permite armazenar entidades e os relacionamentos que estas entidades têm umas com as outras. Estes BDs armazenam dois tipos de informação, os *nós*, que são instâncias de entidades, e as *arestas*, que especificam os relacionamentos entre os nós. O grande propósito do BD baseado em grafos é permitir que uma determinada aplicação execute eficientemente consultas que atravessem uma rede de nós e arestas, bem como analise os relacionamentos entre as entidades. *Neo4J*[10], *Infinite Graph*[11] e *Sones*[12] são exemplos de BDs NoSQL baseados em grafos.

A categoria de BDs NoSQL documento será detalhada a seguir.

2.1.2 Bancos de Dados NoSQL documento

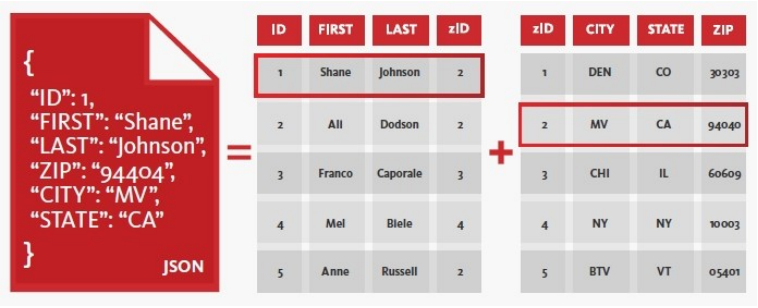
A categoria de BDs NoSQL documento, modelo alvo deste trabalho, é semelhante em conceito a um BD chave-valor, exceto pelo fato de que os valores armazenados são documentos. Neste contexto, um documento é um conjunto de campos e valores nomeados (conjunto de pares chave-valor), cada um podendo ser um *item escalar simples* ou um *elemento composto*, tal como uma lista, um conjunto ou um documento aninhado. Os dados nos campos de um documento podem ser codificados em uma variedade de maneiras, incluindo JSON, BSON, XML, YAML ou mesmo armazenadas como texto simples. Ressalta-se que o termo *documento*, neste contexto, não implica uma estrutura de texto de forma livre. Ao invés disso, documento é o nome dado a um conjunto de itens de dados relacionados que constituem uma entidade (MCMURTRY et al., 2013). Segundo (TIWARI, 2011), neste modelo de dados é possível indexar os documentos não apenas através do seu identificador primário, mas também através de suas propriedades. *MongoDB* (MONGODB, 2014), *RavenDB*[13] e *CouchDB*[14] são exemplos de BDs NoSQL orientados a documento.

[8] hbase.apache.org
[9] www.hypertable.com/
[10] neo4j.com/
[11] www.objectivity.com/products/infinitegraph/
[12] github.com/sones/sones
[13] ravendb.net/

Embora o surgimento do movimento *NoSQL* seja relativamente recente, a noção de modelo de dados baseado em documento não é nova. Em 1989 a empresa IBM lançou o *Lotus Notes*, que foi projetado como um repositório de documentos com muitos recursos associados, como a autorização em nível de documento, versionamento, indexação, fluxo de trabalho, distribuição e replicação (MOHAN, 2013). Os BDs NoSQL orientados a documentos são agrupados em forma de coleções. Comparados aos BDs relacionais, coleções correspondem às tabelas e documentos às tuplas. No entanto, existem diferenças entre os dois modelos de dados, uma vez que, em BDs relacionais, cada tupla em uma tabela tem o mesmo número de campos, enquanto que documentos em uma coleção podem ter campos diferentes. Além disso, os documentos são endereçados no BD por meio de uma chave única utilizada exclusivamente para representar esse documento.

Os BDs NoSQL orientados a documentos armazenam e agregam dados em documentos no formato JSON (formato de dados comum a dados NoSQL da categoria documento), em vez de tabelas com linhas e colunas, como no modelo relacional. Esta abordagem resulta em melhor flexibilidade do modelo de dados, maior eficiência na distribuição de documentos, assim como desempenho de leitura e escrita superiores (COUCHBASE, 2014). A Figura 1 apresenta uma comparação de um documento no formato JSON com informações de tabelas de um BD relacional. Neste exemplo, um único documento JSON contém informações contidas nas tuplas (linhas) de duas tabelas diferentes.

Figura 1 - Documento JSON comparado a tabelas de BD relacional.



Fonte: Couchbase (2014).

Segundo (KAUR; RANI, 2013), os BDs orientados a documento são uma das categorias de BDs apropriadas para aplicações Web que envolve o

[14] couchdb.apache.org/

armazenamento de dados semiestruturados e a execução de consultas dinâmicas, sendo capazes de suportar escalabilidade horizontal, proporcionar alta disponibilidade e flexibilidade quanto ao suporte a dados semiestruturados. Um BD orientado a documento é utilizado para armazenar, recuperar e gerenciar dados semiestruturados, que são armazenados na forma de documentos (KAUR; RANI, 2013).

Os principais produtos de SGBDs NoSQL documento (*MongoDB*, *Couchbase*, *CouchDB* e *RavenDB*) armazenam dados em formato JSON (JSON, 2014). Particularmente, o MongoDB, que é o BD documento de código-aberto mais popular do mundo *NoSQL*, armazena dados na notação BSON, que é uma representação binária de documentos JSON, o qual permite serialização binária nos dados. A vantagem de armazenar dados em JSON é percebida na facilidade para mapear estruturas de objetos, da maioria das linguagens de programação, diretamente nessa representação, sem a necessidade de tradutores. A Figura 2 apresenta um exemplo de documento no BD NoSQL *MongoDB* (MONGODB, 2014). Neste exemplo, o campo *Fornecimento* armazena um documento (elemento) aninhado (*embedded*). Como alternativa aos aninhamentos, os campos também podem referenciar outros documentos armazenando seus identificadores, como o campo *categoria_id*. Embora não existam esquemas (formais) no *MongoDB*, os nomes dos BDs, os nomes de suas respectivas coleções e os índices são armazenados como metadados.

Figura 2 - Documento JSON no MongoDB.

```
{
  "_id": 320001,
  "descricao": "BOLSA PRAIA MATERIAL RECICLAVEL",
  "preco": 10,
  "Fornecimento": [
      { "fornecedor_id": "ABRIL TECIDOS REUTILIZAVEIS" },
      { "fornecedor_id": "VENTURA PRODUTOS" } ],
  "Catalogo": {
      "categoria_id": 15
  }
}
```

Fonte: Desenvolvido pelo autor.

No *MongoDB* uma coleção contém um conjunto de documentos e um documento é um conjunto de pares chave-valor. Os documentos possuem um *esquema dinâmico*, ou seja, documentos na mesma coleção não precisam ter o mesmo conjunto de campos ou estrutura, e os campos comuns em documentos de uma coleção podem conter diferentes tipos de dados (MONGODB, 2014).

2.2 PROJETO LÓGICO DE BANCO DE DADOS

O projeto de BD é um processo constituído por três fases de modelagem de dados (BATINI; CERI; NAVATHE, 1992; ELMASRI; NAVATHE, 2011): os projetos conceitual, lógico e físico. Iniciando-se pelo projeto conceitual, um esquema representando as informações de um domínio é modelado através de um modelo de alto nível de abstração. Nos níveis subsequentes, o esquema conceitual é transformado em esquemas de mais baixa abstração (projeto lógico) até alcançar uma representação adequada ao modelo alvo, o modelo no qual o BD será fisicamente estabelecido.

Entende-se por modelo lógico o modelo que representa a classe, ou a natureza dos dados manipulados pelo SGBD. Exemplos de modelos lógicos são os modelos hierárquicos, orientados a objetos e relacionais. O modelo específico se refere ao modelo de armazenamento estabelecido pelo SGBD em questão. O projeto conceitual é a etapa de maior nível de abstração, pois não apresenta dependência de nenhum modelo específico ou lógico de BD. O projeto lógico apresenta uma abstração média, pois é dependente do modelo lógico, mas não do modelo específico do SGBD. Por último, o projeto de mais baixo nível é estabelecido pelo projeto físico, que é totalmente dependente do modelo específico de armazenamento de um SGBD específico.

A metodologia de modelagem em três níveis constitui um consenso para o projeto de BD tradicional. O processo *top-down* desta metodologia é considerado uma das melhores formas de realizar a modelagem de um BD, visto ser mais claro e natural para o projetista modelar as informações de uma aplicação a partir de modelos que refletem mais diretamente os fatos do mundo real e seus relacionamentos (EMBLEY, 1998; MOK; EMBLEY, 2006). Com o foco na modelagem lógica, esta seção apresenta os princípios da metodologia para projeto de BDs tradicionais com a finalidade de embasar o projeto lógico de BDs NoSQL.

2.2.1 Projeto Conceitual de BDs Tradicionais

Segundo (ELMASRI; NAVATHE, 2011), o modelo utilizado em um projeto conceitual deve ser: (i) *Expressivo*: prover estruturas de modelagem e conceitos suficientes para representar os dados de um domínio de informação; (ii) *Simples*: ser capaz de ser compreendido por usuários leigos; (iii) *Mínimo*: cada conceito do modelo deve ter um significado distinto dos demais conceitos; (iv) *Esquemático*: possuir uma notação esquemática para que a exibição da modelagem seja fácil de interpretar; e (v) *Formal*: cada conceito do modelo deve apresentar uma interpretação bem definida e única.

Um modelo conceitual deve equilibrar sua expressividade e simplicidade para que os requisitos mencionados sejam atendidos, visto que o aumento da expressividade, em geral, reduz a simplicidade do modelo. Metodologias consolidadas para a modelagem de BD assumem que o modelo conceitual deve ser independente de quaisquer modelos de implementação e que a independência entre os modelos do projeto de um BD é um requisito fundamental para garantir a portabilidade da metodologia (BATINI; CERI; NAVATHE, 1992; ELMASRI; NAVATHE, 2011). Exemplos de modelos conceituais que atendem a estes requisitos são o modelo Entidade-Relacionamento (ER) (CHEN, 1976), o modelo Entidade-Relacionamento Estendido (EER) e o diagrama de classes da UML[15].

Os construtores conceituais utilizados por um modelo conceitual podem ser divididos em três tipos de abstração (BATINI; CERI; NAVATHE, 1992): (i) Na *abstração de classificação*, um conceito é identificado como uma classe ou entidade do mundo real juntamente com suas propriedades ou atributos; (ii) Na *abstração de agregação*, os conceitos identificados são associados uns aos outros, gerando novas classes que representam estas associações ou relacionamentos; e (iii) Na *abstração de generalização* é estabelecido um relacionamento de subconjunto entre os elementos de duas ou mais classes. Neste caso, todas as propriedades de uma classe definida como genérica são herdadas pelas classes que constituem especializações da classe genérica.

Em geral, um modelo conceitual provê construtores de modelagem para os três tipos de abstração apresentados. Por ser um modelo que provê construtores adequados para estes tipos de abstração, o modelo EER é considerado um modelo expressivo para a modelagem conceitual de BDs (ELMASRI; NAVATHE, 2011). O modelo EER inclui todos os conceitos de modelagem do modelo ER (CHEN, 1976) e inclui os conceitos de *subclasse e superclasse*, *especialização e generalização*, bem como o conceito de *categoria ou tipos união*, o qual é utilizado para representar um conjunto de objetos (entidades) que é a união de objetos de diferentes tipos de entidade. Associado a estes conceitos tem-se o importante mecanismo *de herança de atributo e de relacionamento* (ELMASRI; NAVATHE, 2011).

A seguir são apresentados os construtores conceituais providos pelo modelo EER para cada um dos tipos de abstração.

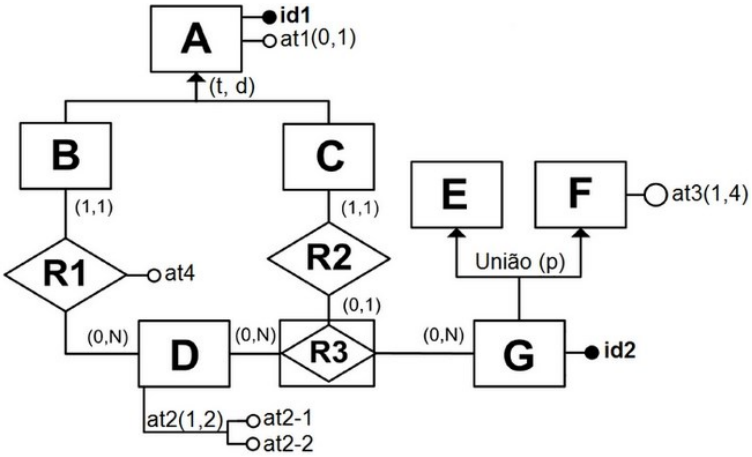
Com relação à *abstração de classificação*, os conceitos são classificados em *Tipos Entidade*, os quais são constituídos de *atributos* que representam as propriedades deste conceito. Os atributos são caracterizados por suas ocorrências mínimas e máximas e pelo modelo de conteúdo que apresentam. Quanto à *ocorrência mínima*, um atributo é caracterizado como *opcional* ou

[15] www.uml.org/

como *obrigatório*. Um atributo opcional apresenta ocorrência mínima igual a 0, o que denota que este atributo pode não ser especificado para uma instância do tipo entidade ao qual pertence. Um atributo obrigatório apresenta ocorrência mínima igual a 1, denotando que este atributo será sempre especificado para qualquer instância do tipo entidade. Quanto à *ocorrência máxima*, um atributo é caracterizado como *monovalorado* ou *multivalorado*. A ocorrência máxima de um atributo monovalorado é sempre 1 e de um multivalorado deve ser superior a 1. Quanto ao modelo de conteúdo, um atributo é caracterizado como *simples* ou *composto*. O conteúdo de um atributo simples é um valor de tipo simples, enquanto que um atributo composto constitui uma composição de outros atributos.

O atributo *at3* da Figura 3 é um exemplo de atributo *multivalorado* e *obrigatório*. O atributo *at2* é um exemplo de atributo *composto*, *obrigatório* e *multivalorado*. Além destas nomenclaturas, um atributo pode também ser identificador. Um atributo identificador atua como identificador único para as instâncias de um tipo entidade. O identificador de um tipo entidade pode ser composto por mais de um atributo identificador, o que caracteriza uma identificação composta. O atributo *id2* é um exemplo de atributo identificador, neste caso atuando como o identificador da entidade G.

Figura 3 - Exemplo de um esquema EER.



Fonte: Desenvolvido pelo autor.

Quanto à *abstração de agregação*, uma associação entre conceitos estabelece um *Tipo Relacionamento*. Um tipo relacionamento possui um *grau* que é definido pelo número de tipos entidade que estão associados pelo

relacionamento. Além disto, cada entidade participante de um relacionamento define suas ocorrências mínimas e máximas no relacionamento. A cardinalidade de um tipo relacionamento é dada pelas ocorrências máximas das entidades envolvidas nele. A participação (1,1) de uma entidade em um relacionamento é conhecida como *dependência de existência*.

Exemplos de tipos relacionamento são mostrados na Figura 3, representados por losangos nomeados (notação do modelo ER original) e conectando tipos entidade (representados por retângulos nomeados). Por exemplo, no tipo relacionamento R1 (relacionamento binário, visto que possui duas entidades participantes) a entidade B tem participação (0,N) em R1, o que denota que uma instância de B pode estar associada a nenhuma ou a N instâncias de D através do relacionamento R1. Já a entidade D tem participação (1,1) em R1, denotando que todas as instâncias de D devem estar associadas a uma e apenas uma instância de B através de R1. Neste caso, diz-se que a cardinalidade de R1 é 1:N pois a participação máxima de D é 1 e a de B é N. Um tipo relacionamento também pode possuir atributos, como o atributo *at4* em R1. Além disto, um tipo relacionamento nomeado RA pode estar sendo associado por outro tipo relacionamento, nomeado RB. Neste caso, RA é transformado em uma *entidade associativa* como ocorre para R3 que está sendo associado por R2 na Figura 3.

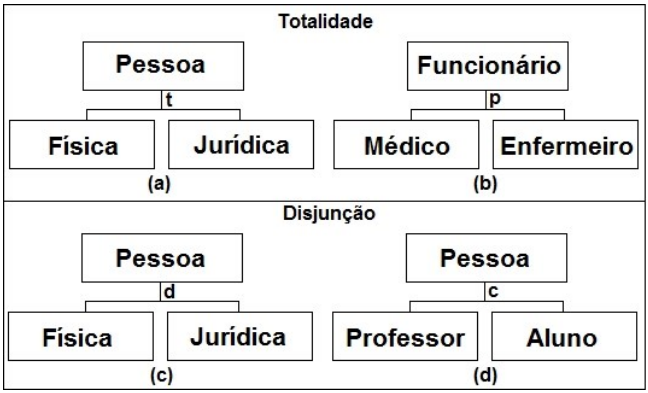
Em relação à *abstração de generalização*, *associações de herança* entre tipos entidade envolvem entidades caracterizadas como *genéricas* (superclasse) e entidades *especializadas* (subclasses). Duas composições providas por modelos conceituais com relação à abstração de generalização estão presentes no modelo EER: *hierarquias de generalização* e *tipos união* (ou categorias). Estes dois tipos de composições impõem diferentes restrições sobre uma associação de herança de tipos entidade. Em hierarquias de generalização, as restrições de totalidade e de disjunção estabelecem quatro possibilidades de restrição sobre hierarquias de generalização: total e disjunta (t, d); parcial e disjunta (p, d), total e compartilhada (t, c), parcial e compartilhada (p, c) (BATINI; CERI; NAVATHE, 1992; HEUSER, 2008).

A Figura 4 apresenta exemplos das *restrições* de *disjunção* e de *totalidade*. A restrição de *totalidade* estabelece que uma instância da entidade genérica (superclasse) deve ser especializada em pelo menos uma instância de uma das entidades especializadas (subclasses). Na Figura 4 (a), *toda* pessoa *é* física *ou* jurídica (total), e na Figura 4 (b), *nem todo* funcionário *é* médico *ou* enfermeiro (parcial). A restrição de *disjunção* estabelece que a superclasse deve ser especializada em apenas uma das subclasses em cada instância da superclasse. Na Figura 4 (c), *toda* pessoa *é apenas* física *ou* jurídica (disjunta), e na Figura 4 (d), *uma* pessoa *pode ser* professor *e* aluno (compartilhada). A hierarquia de generalização apresentada pela Figura 3 envolvendo A como

superclasse e B e C como subclasses é um exemplo de uma generalização *total* e *disjunta*, e o tipo união parcial, apresentado pela Figura 3, estabelece que nem todas as instâncias de E e F são especializadas por G.

Em alguns casos, é necessário representar um relacionamento entre *superclasse/subclasse* com mais de uma superclasse, onde as superclasses representam diferentes tipos de entidade. Nestes casos, a subclasse representa uma coleção de objetos que é um subconjunto da união de tipos de entidades distintas. Esta subclasse é chamada no modelo EER de um *tipo união* ou uma *categoria* (ELMASRI; NAVATHE, 2011). Os tipos união são um caso mais restrito de herança múltipla, sendo que a coleção de instâncias de uma subclasse é representada pela união das instâncias das suas superclasses. Entretanto, uma instância de subclasse pode ser a especialização de apenas uma das suas superclasses. Para tipos união, apenas a restrição de totalidade é imposta, o que estabelece duas possibilidades de restrição: tipos união totais e tipos união parciais.

Figura 4 - Restrições de totalidade e disjunção.



Fonte: Desenvolvido pelo autor.

Composições que constituem particularidades (*auto-relacionamentos*[16] e *entidades fracas*[17]) ou restrições de tipos relacionamento e hierarquias de generalização (hierarquias de generalização de múltiplos níveis ou com casos

[16] *Auto-relacionamento* é utilizado para representar hierarquias e composições de elementos do mundo real que são ocorrências em uma mesma entidade.

[17] *Entidade fraca* é uma entidade que não possui existência própria (depende da existência de outra entidade) ou que, para ser identificada, depende da identificação de outra entidade.

de herança múltipla) não são discutidas nesta seção, mas devem ser consideradas na modelagem lógica de um BD.

2.2.2 Projeto Lógico de BDs Tradicionais

A atividade fundamental do projeto lógico de um BD é transformar um esquema conceitual em um esquema lógico de dados. Enquanto o objetivo do projeto conceitual é produzir um esquema expressivo e capaz de representar os dados de um domínio de informação, o objetivo do projeto lógico é transformar um esquema conceitual em uma representação equivalente em um modelo lógico que se aproxima ao modelo de implementação do BD. O projeto lógico visa alcançar um esquema lógico capaz de utilizar com eficiência as construções permitidas pelo modelo lógico em questão e pode ser subdividido em duas fases ou estágios: *projeto lógico de alto nível* e *projeto lógico dependente de modelo* (BATINI; CERI; NAVATHE, 1992; ELMASRI; NAVATHE, 2011).

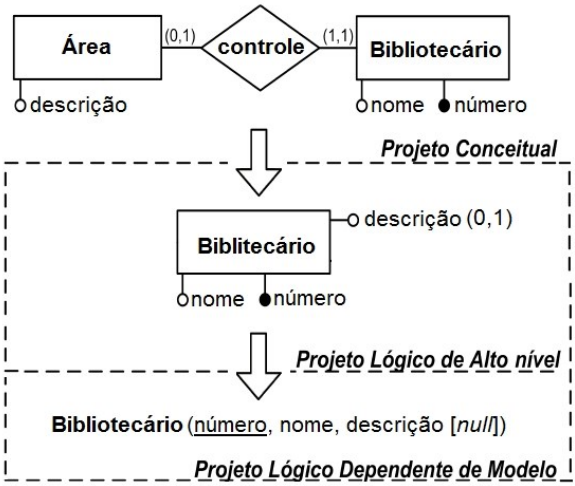
No *projeto lógico de alto nível*, o esquema conceitual é transformado em um esquema mais otimizado que é representado através de um modelo lógico independente (abstrato) de qualquer modelo de implementação existente para o tipo de BD em questão.

Na fase seguinte, o *projeto lógico dependente de modelo*, o esquema lógico de alto-nível é transformado em um esquema adequado a um modelo de implementação específico. O esquema gerado pela primeira fase é transformado, então, em um esquema mais próximo ao modelo de dados no qual o BD será implementado. Alternativas de conversão também precisam ser providas pelo processo para a conversão dos construtores do modelo de alto nível para os construtores do modelo de implementação. Preferências do usuário também podem ser consideradas durante esta conversão. As otimizações a serem aplicadas nesta fase em geral são mínimas, visto que a estrutura entre os conceitos modelados já foi simplificada pela fase anterior (BATINI; CERI; NAVATHE, 1992). Simplificações nesta etapa estão relacionadas apenas a formas alternativas de representação de uma estrutura do esquema no modelo de implementação, o que justifica a baixa otimização desta etapa. Nesta fase é necessário conhecer os construtores e restrições impostas pelo modelo de implementação para que a transformação seja efetivada.

O esquema gerado é submetido ao projeto físico do BD, que é responsável pela definição da estrutura de armazenamento física dos dados bem como pelas otimizações que devem ser realizadas sobre esta estrutura com a finalidade de melhorar o desempenho do acesso e recuperação de dados. O projeto físico não é discutido neste trabalho. A Figura 5 apresenta um exemplo

da *modelagem conceitual*, *lógica de alto nível* e *lógica dependente de modelo* para um BD relacional segundo Batini, Ceri e Navathe (1992).

Figura 5 – Exemplo de modelagem lógica em dois estágios para um BD relacional.



Fonte: Adaptado de Batini, Ceri e Navathe (1992).

O esquema conceitual representado no modelo EER (Figura 5) é transformado em um esquema mais otimizado em um modelo lógico de alto nível. Os autores (BATINI; CERI; NAVATHE, 1992) acreditam que é possível inferir características lógicas do modelo relacional no modelo EER, por este motivo também utilizam o modelo EER como o modelo de alto-nível para o projeto lógico de BDs relacionais. Neste caso, apenas as construções do EER que podem ser facilmente representadas pelo modelo relacional são utilizadas durante o projeto de alto-nível. No projeto lógico dependente de modelo, o esquema EER otimizado é transformado em um esquema relacional, neste exemplo.

2.3 AGREGADOS

Esta seção apresenta o conceito de *agregados*, termo definido na área de *Domain-Driven Design* (DDD) (EVANS, 2003), que foi utilizado na abordagem de projeto lógico proposta por este trabalho. DDD descreve uma maneira de pensar e um conjunto de princípios que ajudam desenvolvedores na criação de sistemas robustos e sustentáveis. Não se trata de uma tecnologia, ou metodologia, mas sim de uma disciplinada abordagem de projeto de software,

focada no domínio e na lógica do sistema. Segundo os preceitos do DDD, existem quatro camadas essenciais que uma solução deve possuir: (i) *Infraestrutura*: age como uma biblioteca para as outras camadas, fornecendo a comunicação entre elas, persistência de objeto, entre outras funções; (ii) *Domínio*: onde estão localizadas todas as regras de negócio; (iii) *Aplicação*: onde as atividades do aplicativo são coordenadas. Ela não contém lógica do negócio e deve se restringir apenas ao estado do progresso de uma tarefa de aplicação; e (iv) *Interface*: é responsável por apresentar informações e interpretar comandos do usuário.

Neste contexto, a atuação do DDD ocorre na camada de domínio da arquitetura, a qual é responsável pelos conceitos do domínio, dos casos de uso e regras de negócio. Os estados dos objetos são armazenados nesta camada, porém a persistência de objetos e possivelmente seus estados são tarefas da camada de infraestrutura. A utilização das práticas sugeridas pelo DDD afeta diretamente a extensibilidade, usabilidade, testabilidade e manutenibilidade de um sistema.

Dentre os artefatos do DDD, os agregados são conjuntos de *Entidades* (objetos que tem significado no domínio) ou *Objetos de Valor* (não possuem identidade para o negócio) que são encapsulados em uma única classe. (EVANS, 2003) sugere que *Entidades* e *Objetos de Valor* sejam agrupados em agregados e que limites sejam determinados para cada agregado. Deve-se escolher uma entidade para ser a raiz do agregado, que controlará o acesso aos objetos dentro do seu limite através da raiz.

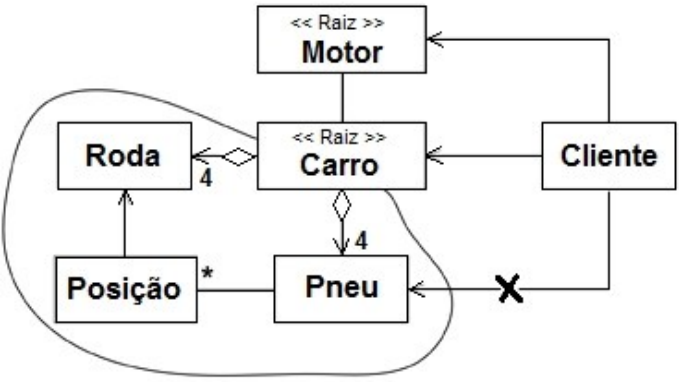
Em outras palavras, *agregado* é um grupo de objetos associados que tratamos como sendo uma unidade para fins de alterações de dados, de forma que cada agregado possua uma entidade raiz e um limite (o que está dentro do agregado). Os agregados demarcam o escopo dentro do qual as invariantes têm que ser mantidas em cada estágio do ciclo de vida. A seguir, são descritas práticas de utilização de agregados (EVANS, 2003):

- A entidade raiz deve possuir uma identificação global e é, em última instância, responsável por verificar invariâncias;
- As entidades dentro do limite possuem uma identificação local e única somente dentro do agregado;
- Nada fora do limite do agregado pode fazer referência a qualquer coisa dentro deste agregado, exceto à entidade raiz. A entidade raiz pode fornecer referência às entidades internas para outros objetos, mas esses objetos podem utilizá-las apenas de forma transitória, sem reter a referência;

- Somente raízes dos agregados podem ser obtidas diretamente através de consultas ao BD. Todos os outros objetos devem ser encontrados através de travessia de associações;
- Uma operação de exclusão deve remover tudo dentro do limite do agregado de uma só vez.

A Figura 6 apresenta um exemplo de *identidade global*, representada pelas entidades raízes *motor* e *carro*, que podem ser *acessadas* pelo objeto *cliente*. Um exemplo de *identidade local* é a entidade *pneu*, que está dentro dos limites do agregado *carro*, e por este motivo não pode ser acessado diretamente por objetos externos ao agregado *carro*, neste exemplo, o objeto *cliente*. Referências válidas podem ser observadas entre o objeto *cliente* e as entidades *carro* e *motor*. Um objeto fora do limite do agregado pode referenciar uma entidade raiz ou consultar seu conteúdo pelo identificador do respectivo agregado (EVANS, 2003).

Figura 6 - Identidade global e local de agregados.



Fonte: Adaptado de Evans (1993).

Uma questão importante com relação a *agregados*, apontada por (VERNON, 2013), é a existência de conflito de escolha em seu projeto, envolvendo a sua *granularidade*. Por um lado, cada agregado deve ser suficientemente grande para incluir todos os dados envolvidos em algumas restrições de integridade ou outras regras de negócios. Por outro lado, os agregados devem ser tão pequenos quanto possível para reduzir conflitos de concorrência e para atender os requisitos de desempenho e escalabilidade.

2.4 MODELAGEM DE CARGA

A modelagem dos dados referentes à carga esperada para um BD tem o objetivo de produzir informações de apoio ao processo de conversão proposto por este trabalho na produção de esquemas que possam responder de forma mais eficiente às principais operações do BD. A metodologia de modelagem descrita nesta seção é constituída da modelagem de carga para BDs convencionais proposta em (BATINI; CERI; NAVATHE, 1992) e estendida em (SCHROEDER; MELLO, 2008; SCHROEDER, 2008; SCHROEDER; DUARTE; MELLO, 2011).

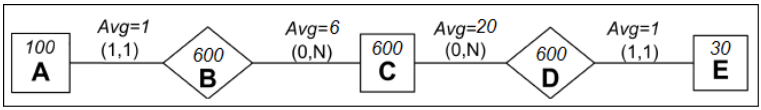
A modelagem da carga de BD proposta em (BATINI; CERI; NAVATHE, 1992) inclui duas categorias de informação: *volume de dados* e *carga da aplicação*. O *volume de dados* é mensurado pelo número médio de instâncias esperadas para cada tipo entidade e tipo relacionamento de um esquema EER. A *carga da aplicação* é estimada por uma descrição das principais operações da aplicação. Esta descrição envolve a frequência de execução das operações e as entidades e relacionamentos acessados por cada uma delas. Essas informações são definidas a seguir.

Definição 1 (*Esquema EER com Volume de Dados*). Um esquema EER com informações adicionais referentes ao volume de dados de um BD define um número médio de instâncias *N* para cada tipo entidade *N(E)* ou relacionamento *N(R)*. Para cada tipo entidade *E* participante de um tipo relacionamento *R* existe uma cardinalidade média associada *Avg(E,R)*. *Avg(E,R)* estabelece o número médio de instâncias de *R* associadas a uma instância de *E*.

Um exemplo de esquema EER com volume de dados é apresentado na Figura 7. O número médio de instâncias para tipos entidade e tipos relacionamento é indicado no interior de suas respectivas notações gráficas. Neste exemplo, o número médio de instâncias estimado para a entidade A é 100. A cardinalidade média dos relacionamentos é representada sobre a participação mínima e máxima de cada entidade envolvida. Para o relacionamento B, por exemplo a cardinalidade média para a entidade participante A é 6. O valor obtido pela multiplicação do número médio de instâncias de uma entidade com a sua respectiva cardinalidade média em um determinado relacionamento é sempre igual ao valor obtido pela multiplicação destes conceitos para a outra entidade participante do mesmo relacionamento. No relacionamento B da Figura 7, por exemplo, o valor desta multiplicação para a entidade A (100*6) é igual ao valor obtido para C (600*1), que é de 600 instâncias. Outra questão a ser observada é que o valor desta multiplicação é

assumido como o número médio de instâncias para o tipo relacionamento. Neste caso, o número médio de instâncias para o relacionamento B é 600.

Figura 7 - Esquema EER com informações sobre a estimativa de volume de dados do BD.



Fonte: Adaptado de Schroeder (2008).

Definição 2 (*Operação*). Uma operação *O* é uma interação básica com um BD que inclui ações de consulta e atualização de dados. Para cada operação existe uma frequência média *f(O)* de execução e uma lista ordenada de tipos entidade e relacionamento $\{t_1, t_2, ..., t_n\}$ acessados por *O*. A ordem da lista especifica a sequência na qual os tipos são acessados por *O*. Cada tipo t_i $(1 \leq i \leq n)$ possui um número médio de instâncias $f_O(t_i)$ que são acessadas por *O*.

A Tabela 1 apresenta algumas operações para o esquema da Figura 7. Por exemplo, O1 possui uma frequência média de 100 vezes ao dia ($f(O_1)$=100). Os tipos entidade e relacionamento A, B e C são acessados por O1 nesta sequência. O conceito inicial A possui $f_{O1}(A)$=100. Na sequência de navegação pelos tipos acessados por O1, o número médio de instâncias acessadas de B e C é obtido pela multiplicação de $f_{O1}(A)$ pelo *Avg(A,B)*, ou seja, 100 x 6 = 600. Desta forma, o valor de $f_{O1}(B)$ e $f_{O1}(C)$ é 600.

Tabela 1 – Exemplos de operações para o esquema EER da Figura 7.

O	***f(O)***	t_i	$fO(t_i)$
O1	100	A	100
		B	600
		C	600
O2	70	E	70
O3	50	E	50
		D	10000
		C	10000

Fonte: Desenvolvido pelo autor.

Três conceitos são introduzidos em (SCHROEDER; MELLO, 2008; SCHROEDER, 2008) e utilizados neste trabalho: *Frequência de Acesso Geral* de um tipo entidade ou tipo relacionamento (FAG), *Frequência de Acesso*

Total do Esquema (FAT) e *Frequência de Acesso Mínima* (FAM). A *Definição 3* determina a FAG, que denota o volume total de instâncias de um tipo entidade ou tipo relacionamento que são acessadas por todas as operações das quais este tipo participa.

Definição 3 (*Frequência de Acesso Geral - FAG*). Dado um tipo entidade ou tipo relacionamento *t* de um esquema EER com volume de dados que é acessado por operações do BD, existe uma frequência de acesso geral *f(t)*, sendo $f(t) = \sum_{i=1}^{n} foi(t)$ e *n* representa o número total de operações em que *t* participa.

A Tabela 2 apresenta as frequências de acesso geral (FAGs), de acordo com os dados fornecidos pela Tabela 1, dos conceitos do esquema EER da Figura 7.

Tabela 2 - Frequência de acesso geral (FAG) dos conceitos do esquema da Figura 7.

Conceito	***FAG***
A	100
B	600
C	10600
D	10000
E	120

Fonte: Desenvolvido pelo autor.

Valores abaixo de uma *Frequência de Acesso Mínima* (FAM) são considerados uma frequência de acesso do BD insignificante para a aplicação. Assume-se que um usuário especialista provê este valor em forma de *percentagem* no início do processo de conversão.

A *Definição 4* estabelece um valor correspondendo à *Frequência de Acesso Total* (FAT) de todas as operações consideradas. Este valor é utilizado para a obtenção da FAM, conforme a *Definição 5*.

Definição 4 (*Frequência de Acesso Total - FAT*). Dado um esquema EER com volume de dados e um conjunto de operações, a frequência de acesso total é dada por $fTOT(t) = \sum_{i=1}^{n} f(ti)$, onde *n* é o número de tipos entidade e tipos relacionamento de um esquema EER com volume de dados.

Considerando as operações da Tabela 1, obtém-se $f_{TOT}(EER)$=21420. Conforme especificado pela *Definição 4*, este valor é obtido pela soma da FAG

de todos os tipos envolvidos nas operações consideradas. De acordo com (BATINI; CERI; NAVATHE, 1992), 20% das operações mais frequentes produzem 80% da carga do BD. Com base nesta afirmação, o cálculo da FAM assume que a FAT estimada corresponde a 80% da carga do BD.

Definição 5 (*Frequência de Acesso Mínima - FAM*). Dado um esquema EER com volume de dados, uma $f_{TOT}(EER)$ correspondendo a 80% da carga da aplicação e um valor percentual $p_{min}(EER)$ representando o acesso mínimo da aplicação, existe uma frequência de acesso mínima $f_{min}(EER)$ que corresponde ao $p_{min}(EER)$ aplicado sobre $f_{TOT}(EER)$ definida por $f_{min}(EER) = (f_{TOT}(EER) * p_{min}(EER))/80$.

Considerando as operações da Tabela 1 e as FAGs da Tabela 2, dado $p_{min}(EEV)$=1% como um valor percentual denotando o valor mínimo de acesso relacionado à frequência de acesso geral ($f_{TOT}(EER)$=21420), obtém-se f_{min}=267,75 pela aplicação da fórmula $f_{min}(EER) = (f_{TOT}(EER) * p_{min}(EER))/80$.

2.5 CONSIDERAÇÕES FINAIS

Este capítulo apresentou o embasamento teórico necessário para este trabalho. O processo de conversão que envolve a etapa de projeto lógico é a principal tarefa a ser tratada por este trabalho. O modelo lógico NoSQL documento a ser utilizado por este trabalho é definido e exemplificado no capítulo 4. Uma abordagem de conversão para um modelo lógico NoSQL documento que considera todos os conceitos do EER é também proposta neste mesmo capítulo. Além de considerar todos os construtores do EER, a abordagem provê um conjunto de alternativas de conversão para cada um dos construtores conceituais.

A modelagem de carga de dados apresentada representa a carga esperada para um BD. Ela visa produzir informações de apoio ao processo de conversão proposto neste trabalho. Estimativas da carga do BD podem ser utilizadas para a escolha das alternativas de conversão mais apropriadas e também para a geração de uma estrutura NoSQL documento otimizada.

3 TRABALHOS RELACIONADOS

Metodologias e ferramentas de suporte ao projeto de BDs NoSQL é um tópico ainda pouco explorado na literatura de BD (ATZENI; BUGIOTTI; ROSSI, 2012; MOHAN, 2013). Atualmente, o projeto de BDs NoSQL é geralmente baseado em melhores práticas e orientações (KATSOV, 2012), sem uma metodologia sistemática. Muitos autores observaram que o desenvolvimento de metodologias de alto nível e ferramentas de suporte ao projeto de BDs NoSQL são necessários (ATZENI et al., 2013; BADIA; LEMIRE, 2011; BUGIOTTI et al., 2014; HSIEH, 2014; KATSOV, 2012; MAGUIRE; O'KELLY, 2013; MOHAN, 2013). No contexto acadêmico, percebe-se uma carência de trabalhos que propõem metodologias de projeto para BDs NoSQL (BUGIOTTI et al., 2014; CHEBOTKO; KASHLEV; LU, 2015; JOVANOVIC; BENSON, 2013).

Este capítulo realiza uma análise comparativa das abordagens correlatas com o intuito de apresentar um estado da arte sobre o tema. Uma revisão sistemática da literatura foi realizada com o intuito de identificar abordagens existentes acerca de metodologias de projeto para BDs NoSQL. A revisão foi realizada via mecanismos de busca: *Google Scholar*[18], *DBLP*[19], *ACM DL*[20] e *IEEE Xplore*[21]. Utilizou-se os seguintes termos de busca para identificar as abordagens: "*NoSQL design*", "*NoSQL logical design*", "*NoSQL logical approach*" e "*NoSQL data modeling*". O termo "*NoSQL logical design*" trouxe os melhores resultados.

Além disso, abordagens relevantes para a modelagem de dados NoSQL e que utilizam esquemas não-relacionais, como esquemas XML e Orientado a Objetos (OO), são revistas nesta seção visando identificar contribuições para a modelagem lógica de BDs NoSQL. Modelos XML e OO, assim como os modelos de dados NoSQL, são modelos de dados (ou objetos) complexos, cujas similaridades em termos de estratégias de mapeamento, como o tratamento de atributos multivalorados e aninhados, podem ser adaptadas para o projeto lógico de BDs NoSQL.

Para fins de análise e classificação, os trabalhos relacionados são separados em três grupos de acordo com o modelo lógico: (i) Modelo lógico NoSQL; (ii) Modelo lógico XML; e (iii) Modelo lógico OO. A partir desta revisão, uma análise comparativa entre os trabalhos é realizada com o objetivo

[18] scholar.google.com
[19] dblp.uni-trier.de/
[20] dl.acm.org/
[21] ieeexplore.ieee.org/

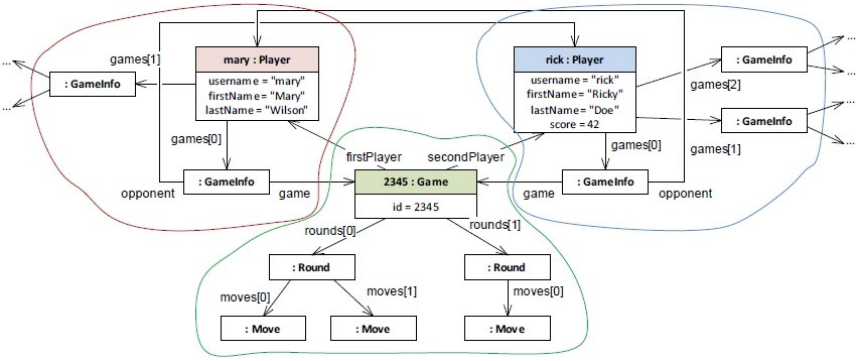
de relacionar os níveis de modelagem (conceitual, lógico e físico) atendidos por cada proposta.

3.1 MODELO LÓGICO NOSQL

O trabalho de (BUGIOTTI et al., 2014) apresenta uma abordagem lógica para o problema de projeto de BD NoSQL, chamada de *NoAM* (BUGIOTTI et al., 2013), explorando os pontos em comum de algumas categorias de BDs NoSQL. A proposta é baseada em um modelo de dados abstrato intermediário, em nível lógico, utilizado para representar os dados de aplicações como coleções de objetos agregados. Ela demonstra como a representação intermediária *NoAM* pode ser implementada em alguns BDs NoSQL, levando em conta as suas características específicas. O modelo lógico NoSQL proposto, assim como neste trabalho, é baseado no conceito de *agregados* (SADALAGE; FOWLER, 2013). A abordagem sugere um suporte eficaz para escalabilidade, consistência e desempenho e baseia-se em quatro fases principais.

A primeira fase trata do *projeto de agregados*, que visa identificar as várias classes de objetos agregados necessários em uma aplicação. Essa atividade é conduzida por casos de uso e requisitos funcionais. A Figura 8 apresenta alguns objetos de uma aplicação que deve gerenciar vários tipos de objetos, incluindo jogadores (*player*), jogos (*game*) e rodadas (*round*). Neste exemplo, as caixas denotam objetos e as setas denotam ligações entre eles. Os círculos em torno dos objetos sugerem a maneira de como eles poderiam ser agrupados em *agregados* (o círculo em torno dos objetos denota o limite do agregado).

Figura 8 - Exemplo de objetos de uma aplicação.

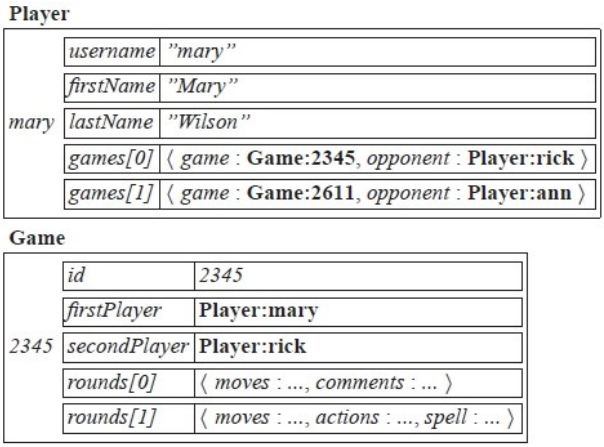


Fonte: Bugiotti et al. (2014).

A fase seguinte trata do *particionamento de agregados*. Nesta fase os agregados são divididos em elementos de dados menores. Essa atividade é conduzida por casos de uso e requisitos de desempenho. Na sequência, na fase de *projeto de BD NoSQL de alto nível*, os agregados são mapeados para o modelo de dados intermediário *NoAM*, de acordo com as partições identificadas. A Figura 9 apresenta um exemplo de modelagem de dados no modelo de dados abstrato *NoAM*, onde *caixas internas* denotam *entradas* enquanto *caixas externas* denotam *blocos*. A última fase trata da *implementação*, que converte a representação intermediária de dados para os elementos de modelagem específicos do BD NoSQL destino. A fase de implementação é a única dependente do BD destino.

O trabalho de (JOVANOVIC; BENSON, 2013) utiliza IDEF1X[22] (*Integration DEFinition for Information Modeling*), que é uma linguagem de modelagem de dados para o desenvolvimento de modelos de dados semânticos, nas etapas de modelagem conceitual e lógica. Na modelagem conceitual, IDEF1X é utilizado para representar o domínio de uma aplicação, e na etapa de modelagem lógica para representar o modelo lógico para NoSQL baseado em *agregados* (DDD). O modelo lógico é obtido através de um processo de conversão entre os modelos.

Figura 9 - Exemplo de modelagem de dados no modelo abstrato NoAM.



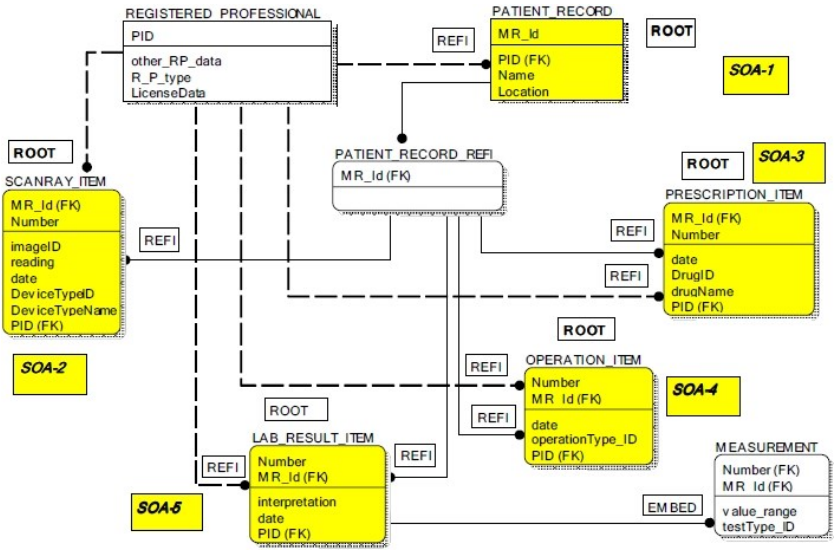
Fonte: Bugiotti et al. (2014).

[22] www.idef.com/idef1x.htm

Segundo (JOVANOVIC; BENSON, 2013), o modelo lógico proposto pode ser utilizado em BDs NoSQL das categorias *chave-valor*, *documento* e *colunar*. A proposta fornece suporte para análise de diferentes formas de modelagem, como por exemplo, o particionamento do esquema de dados em agregados menores e independentes para o contexto do SOA (*Service Oriented Architecture*). Em SOA, aplicações são organizadas em pequenos serviços, reduzindo a modelagem de dados para peças pequenas e independentes em torno de um objetivo.

A Figura 10 apresenta um exemplo de modelo lógico, na notação IDEF1X, representado por cinco agregados (entidades raízes) destacados, no domínio de um prontuário médico, para o contexto do SOA. Na Figura 10, as *entidades raízes* possuem *caixas* com a notação *ROOT*, as entidades que deverão ser aninhadas a outras entidades possuem *caixas* com a notação *EMBED*, e por fim, as entidades que possuem valores de referência para outras entidades recebem *caixas* com a notação *REFI*.

Figura 10 - Modelo lógico IDEF1X representado por cinco agregados.



Fonte: Jovanovic e Benson (2013).

O trabalho de (CHEBOTKO; KASHLEV; LU, 2015) propõe uma metodologia de modelagem de dados, dirigida por consultas, para o BD NoSQL *colunar* Cassandra. O trabalho apresenta princípios de modelagem e regras e padrões de mapeamento para guiar a modelagem de dados lógica para

o BD Cassandra. Diagramas visuais para a modelo de dados lógico e físico do BD Cassandra são introduzidos e chamados de *diagramas Chebotko*. (CHEBOTKO; KASHLEV; LU, 2015) também demonstra uma ferramenta que automatiza o processo de modelagem de dados.

O modelo ER é utilizado por (CHEBOTKO; KASHLEV; LU, 2015) para a modelagem de dados em nível conceitual, pois, segundo os autores, a notação do ER é independente de tecnologia. Diagramas de *fluxo de trabalho da aplicação* que definem *padrões de acesso a dados* para as tarefas da aplicação também são utilizados no início do processo proposto.

O modelo de dados lógico corresponde ao esquema do BD Cassandra, com *esquemas tabelas* definindo colunas, chaves primárias, de partição e de *cluster*. O mapeamento dirigido por consultas conceitual-lógico é definido por princípios, regras de mapeamento e padrões de projetos. Detalhes da metodologia, e da ferramenta que automatiza a metodologia, são encontrados nos *sites*[23] [24] dos fabricantes.

3.2 MODELO LÓGICO XML

A literatura apresenta diversos trabalhos relacionados a metodologias de projeto para BDs XML (SCHROEDER; MELLO, 2008; CHOI; LIM; JOO, 2003; FONG et al., 2006; ELMASRI et al., 2002; BIRD; GOODCHILD; HALPIN, 2000).

Uma abordagem de projeto de geração de esquemas XML a partir de esquemas conceituais (EER) é apresentada por (SCHROEDER; MELLO, 2008), considerando a carga de trabalho esperada de aplicações XML. O esquema conceitual é traduzido em um esquema lógico XML no nível de projeto lógico. O processo de conversão aplicado por este nível é definido por um conjunto de regras para a conversão de cada construtor conceitual EER para uma representação equivalente no modelo lógico XML. Este modelo lógico é um modelo abstrato para representar os diferentes modelos de implementação XML. No nível de projeto de implementação, um esquema lógico XML é traduzido em um esquema definido por um modelo de implementação específico que pode ser, por exemplo, uma especificação *XML Schema* ou *DTD*. A consideração da carga de trabalho visa para produzir esquemas XML que minimizam o impacto das relações de referência sobre o desempenho das principais operações de acesso.

Metodologias para produzir esquemas XML em *DTD* a partir de esquemas EER são apresentadas em (CHOI; LIM; JOO, 2003; FONG et al.,

[23] kdm.dataview.org/
[24] datastax.com/what-we-offer/products-services/training/

2006). A metodologia proposta por (FONG et al., 2006) considera *grande parte* dos construtores conceituais do EER e suas restrições para gerar um esquema XML em DTD. Ela endereça o problema de aplicações que desejam manter seus dados em BDs relacionais e precisam trabalhar com estes dados no formato XML no nível da aplicação. Por este motivo, uma abordagem de engenharia reversa é utilizada para transformar um esquema relacional em um esquema EER que posteriormente é convertido em uma DTD. O trabalho de (CHOI; LIM; JOO, 2003) propõe uma metodologia de projeto para XML no qual um esquema EER é transformado diretamente em uma especificação em DTD que, por sua vez, é transformada em um esquema físico.

O trabalho de (ELMASRI et al., 2002) apresenta uma metodologia para a conversão de esquemas EER em estruturas hierárquicas. Um algoritmo é proposto para a geração de visões hierárquicas customizadas a partir de uma modelagem EER. Inicialmente, eventuais ciclos de um esquema EER são removidos e, na sequência, *o usuário informa* uma entidade de partida para a geração de uma estrutura hierárquica. Esta entidade de partida se transforma no elemento raiz da estrutura hierárquica, sendo que todos os caminhos no esquema EER que têm relacionamentos com a entidade de partida formarão a hierarquia de elementos da estrutura a ser gerada. A estrutura hierárquica é dita customizada visto que diferentes visões podem ser obtidas mediante a escolha da entidade de partida pelo usuário. (ELMASRI et al., 2002) também apresenta um algoritmo para a geração de esquemas em *XML Schema* a partir de estruturas hierárquicas.

A abordagem de (BIRD; GOODCHILD; HALPIN, 2000) visa reduzir a redundância de dados e aumentar a conectividade das instâncias XML resultantes. Ela propõe um projeto de BD em três níveis, sendo que no nível conceitual são utilizados elementos da UML. No nível lógico, diagramas de classe são estendidos com estereótipos, especificando conceitos específicos do modelo de dados XML. A modelagem explícita de conceitos XML através de diagramas UML estendidos permite definir regras de conversão para o nível de implementação, utilizando o *XML Schema*.

3.3 MODELO LÓGICO OO

Alguns trabalhos apresentam ainda processos de conversão de modelagens conceituais para representações lógicas OO (BISKUP; MENZEL; POLLE, 1995; ELMASRI; JAMES; KOURAMAJIAN, 1993; FONG, 1995; NACHOUKI; CHASTANG; BRIAND, 1991; NARASIMHAN; NAVATHE; JAYARAMAN, 1993).

Na abordagem de (NACHOUKI; CHASTANG; BRIAND, 1991), o processo inicia com a criação de um esquema OO inicial a partir de um

diagrama ER. Na sequência, caminhos de acesso lógico de operações sobre o esquema inicial são determinados e, para cada caminho, constrói-se uma árvore de consulta. Os caminhos de acesso são então fundidos e resultam em um esquema acíclico. Finalmente, as classes do esquema são mescladas com os caminhos e as classes resultantes são codificadas na linguagem do SGBDOO O2. O trabalho de (BISKUP; MENZEL; POLLE, 1995) utiliza *F-Logic*, um formalismo baseado em lógica para especificar o raciocínio sobre conceitos OO, complementado por uma noção de restrições semânticas para a sua transformação. Um esquema ER é primeiro transformado num esquema OO abstrato em *F-Logic*, o qual é refinado através da decomposição e mescla de classes. O esquema resultante é então mapeado para um esquema OO concreto utilizando o SGBDOO ONTOS.

A abordagem de (ELMASRI; JAMES; KOURAMAJIAN, 1993) utiliza o modelo EER como partida e um modelo OO virtual como alvo, enfatizando a transformação de diferentes tipos de construções de generalização e especialização. Além do mapeamento de estruturas EER, o trabalho aborda a geração automática e a integração de métodos que impõem restrições de integridade no esquema EER.

O trabalho de (NARASIMHAN; NAVATHE; JAYARAMAN, 1993) enfatiza a integração das restrições definidas por um esquema ER em um esquema OO, além da transformação estrutural. Ela sugere a criação de uma classe especial de restrição com uma subclasse para cada classe no esquema OO, obtendo-se, assim, duas hierarquias de classes, uma para a estrutura, e a outra para os métodos de restrição, ambas com correspondência 1:1. No mapeamento estrutural, os relacionamentos binários são sempre representados como atributos objeto-valorados.

O trabalho de (FONG, 1995) apresenta uma metodologia de reengenharia de um modelo EER existente para o modelo OMT (*Object Modeling Technique*), utilizando um conjunto de regras de tradução do modelo EER para um modelo genérico OO em OMT. Argumenta-se que, do ponto de vista do usuário, o modelo EER é mais compreensível que o modelo OMT devido a sua simplicidade. Desta forma, adota-se o método tradicional de projeto de BDs iniciando pela modelagem EER e sua posterior conversão para um esquema OMT como parte do projeto de um BDOO.

3.4 COMPARATIVO

O quadro comparativo apresentado na Tabela 3 situa cada uma das abordagens apresentadas nas etapas de projeto conceitual, lógico e de implementação de um BD, descrevendo os respectivos modelos utilizados em cada uma delas.

Quanto aos trabalhos relacionados a metodologias de projeto para BDs NoSQL, observa-se que eles propõem esquemas lógicos utilizando o conceito de agregado no nível lógico. Esta tendência vai ao encontro do que é comentado no trabalho de (SADALAGE; FOWLER, 2013), que afirma que os modelos de dados NoSQL das categorias *chave-valor*, *documento*, e *colunar* são considerados BDs orientados a agregados.

Tabela 3 - Comparativo dos trabalhos relacionados.

Abordagem	**Projeto de Banco de Dados**		
	Conceitual	Lógico	Implementação
Modelo Lógico NoSQL			
(BUGIOTTI et al., 2014)	UML	NoAM baseado em agregados	Elementos de modelos de BDs NoSQL
(JOVANOVIC; BENSON, 2013)	IDEF1X	IDEF1X baseado em agregados	-
(CHEBOTKO; KASHLEV; LU, 2015)	ER	Diagramas Chebotko	Esquema CQL Cassandra
Modelo Lógico XML			
(SCHROEDER; MELLO, 2008)	EER	XML Lógico	DTD / XML Schema
(CHOI; LIM; JOO, 2003)	EER	-	DTD
(FONG et al., 2006)	EER	-	DTD
(ELMASRI et al., 2002)	EER	Estruturas hierárquicas	XML Schema
(BIRD; GOODCHILD; HALPIN, 2000)	UML	UML+ estereótipos	XML Schema
Modelo Lógico OO			
(NACHOUKI; CHASTANG; BRIAND, 1991)	ER	Esquema OO Lógico	DDL O_2
(BISKUP; MENZEL; POLLE, 1995)	ER	F-Logic	DDL ONTOS
(ELMASRI; JAMES; KOURAMAJIAN, 1993)	EER	Esquema OO	-
(NARASIMHAN; NAVATHE; JAYARAMAN, 1993)	ER	Esquema OO	-
(FONG, 1995)	EER	OMT	-

Fonte: Adaptado de Lima e Mello (2015b).

O trabalho de (BUGIOTTI et al., 2014) explora os pontos em comum de algumas categorias de BDs NoSQL e, embora atue nas três etapas de projeto,

não explora a utilização completa de construtores conceituais para a modelagem de um domínio de aplicação *nem formaliza* processos de conversão entre modelagens conceituais e representações lógicas no modelo *NoAM*. As duas primeiras fases da abordagem de (BUGIOTTI et al., 2014), *projeto* e *particionamento* de agregados, respectivamente, *são apresentadas de maneira superficial*, dificultando comparações com a abordagem proposta nesse trabalho. O modelo NoAM é abstrato e genérico pois abrange BDs NoSQL das categorias chave-valor, documento e colunar. Entretanto, sugere três estratégias de modelagem, com diferentes níveis de complexidade para "*chaves* e *valores*", e cada estratégia *adequa-se* a uma das três categorias suportadas.

A proposta de (JOVANOVIC; BENSON, 2013) apresenta a conversão de um modelo conceitual em um modelo lógico através da utilização da linguagem IDEF1X. O trabalho apresenta uma descrição *muito superficial* da sua proposta, uma vez que *não formaliza* um processo de conversão entre modelagens conceituais e representações lógicas, e também não aborda a etapa de modelagem física. Entretanto, um dos autores obteve continuidade na indústria com uma proposta que parte de uma modelagem relacional na linguagem IDEF1X (em nível lógico), utiliza elementos de modelagem XML, e então gera uma modelagem para BDs NoSQL (em nível de implementação) (BENSON, 2014).

O trabalho de (CHEBOTKO; KASHLEV; LU, 2015) atua nas três etapas de projeto e utiliza *consultas* para realizar/apoiar o mapeamento conceitual-lógico da metodologia. A proposta é bastante específica, pois trabalha com *um produto específico* do modelo *colunar* de BDs NoSQL. Os autores disponibilizam *alguns detalhes* da metodologia e da ferramenta, sem custo, nos sites dos fabricantes. O modelo ER, utilizado na proposta de (CHEBOTKO; KASHLEV; LU, 2015), não suporta alguns conceitos semânticos necessários para modelar bases de dados mais recentes (superclasse/subclasse, especialização e generalização, herança de atributos e relacionamento e tipos união/categorias). A proposta não apresenta experimentos de avaliação, mas os autores (CHEBOTKO; KASHLEV; LU, 2015) afirmam que a abordagem é amplamente adotada em ambientes de produção.

Em relação aos trabalhos relacionados de metodologias de projeto para BDs XML, observa-se que a proposta de (SCHROEDER; MELLO, 2008) atua nas três etapas de projeto e considera todos os construtores do modelo conceitual EER para a geração de uma estrutura XML equivalente. Informações referentes à carga estimada para o BD são utilizadas para efetuar otimizações na estrutura XML durante o processo de transformação. No trabalho de (ELMASRI et al., 2002), os esquemas hierárquicos e respectivos esquemas XML gerados limitam-se a representar visões sobre um esquema

EER, não considerando todas as relações semânticas concebidas na modelagem conceitual. Além disso, o modelo hierárquico utilizado é bastante simplificado, definindo apenas uma estrutura de grafo sem considerar restrições específicas do modelo XML. Já o trabalho de (BIRD; GOODCHILD; HALPIN, 2000) propõe uma metodologia de projeto baseada na UML, identificando quais seriam os principais passos de um algoritmo de mapeamento do nível conceitual para o lógico, mas sem propô-lo concretamente. Quanto aos modelos conceituais utilizados, observa-se o grande emprego do ER e do EER (LIMA; MELLO, 2015b).

Por fim, com relação às abordagens que apresentam processos de conversão de modelagens conceituais para representações OO, observa-se que há muitas semelhanças na transformação de construtores do ER para OO. Pequenas diferenças existem na maneira pela qual os tipos de relacionamento binários são representados e também a respeito de estruturas de generalização consideradas. O trabalho de (NACHOUKI; CHASTANG; BRIAND, 1991) considera o processamento de requisitos durante a transformação. A qualidade de um esquema OO é discutida nos trabalhos de (NACHOUKI; CHASTANG; BRIAND, 1991) e (BISKUP; MENZEL; POLLE, 1995). O primeiro concentra-se na identificação de classes, enquanto o segundo investiga como um esquema OO resultante de uma transformação inicial pode ser melhorado através da fusão e decomposição de classes. Uma diferença entre as abordagens OO está no tratamento de restrições de integridade, o qual geralmente é superficial. Observa-se também que as abordagens de (BISKUP; MENZEL; POLLE, 1995; ELMASRI; JAMES; KOURAMAJIAN, 1993; FONG, 1995; NARASIMHAN; NAVATHE; JAYARAMAN, 1993) apresentam metodologias independentes de modelos OO específicos (LIMA; MELLO, 2015b).

3.5 CONSIDERAÇÕES FINAIS

Este capítulo revisou trabalhos que apresentam propostas para a modelagem lógica de documentos NoSQL, incluindo abordagens relevantes para a modelagem de dados NoSQL e que utilizam esquemas não-relacionais, como esquemas XML e Orientado a Objetos (OO). Os trabalhos relacionados foram comparados quanto aos modelos utilizados para cada etapa de projeto.

Percebe-se, na literatura, uma carência de trabalhos que propõem metodologias sistemáticas de projeto para BDs NoSQL, com processos que convertam modelagens conceituais para representações lógicas adequadas e eficientes. Muitos autores observaram que o desenvolvimento de metodologias de suporte ao projeto de BDs NoSQL são necessárias (ATZENI et al., 2013;

BADIA; LEMIRE, 2011; BUGIOTTI et al., 2014; HSIEH, 2014; KATSOV, 2012; MAGUIRE; O'KELLY, 2013; MOHAN, 2013).

Os trabalhos relacionados a projeto lógico para BDs NoSQL não exploram a utilização completa de construtores conceituais para a modelagem de um domínio de aplicação, nem formalizam e detalham processos de conversão entre modelagens conceituais e representações lógicas (BUGIOTTI et al., 2014; CHEBOTKO; KASHLEV; LU, 2015; JOVANOVIC; BENSON, 2013). Com exceção do trabalho de (CHEBOTKO; KASHLEV; LU, 2015), metodologia específica para o BD NoSQL *colunar* Cassandra e que apresenta alguns detalhes de regras de mapeamento, a *superficialidade* dos demais trabalhos, com relação ao detalhamento e definição de regras e de um processo de conversão, dificulta, inclusive, a realização de experimentos comparativos.

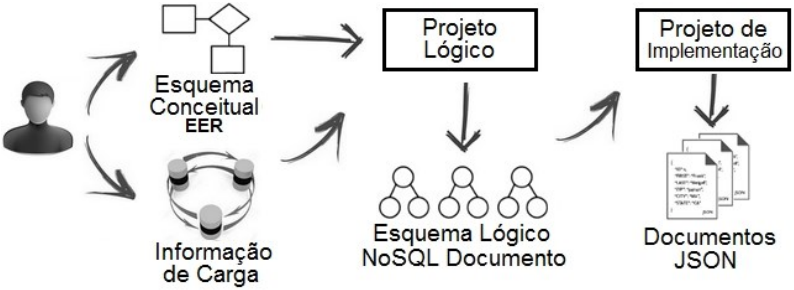
Uma abordagem para o projeto lógico de BD NoSQL documento é proposta no próximo capítulo deste trabalho. O objetivo de tal abordagem é propor soluções para a conversão de esquemas conceituais de alta abstração em esquemas lógicos NoSQL do modelo documento. A conversão entre os modelos provê estratégias de conversão que consideram *todos os construtores do modelo EER*. Com a finalidade de *otimizar* as estruturas lógicas NoSQL documento, o processo de conversão pode considerar a carga de dados e operações esperadas para os documentos do BD. Estas otimizações na estrutura são obtidas através de decisões tomadas no mapeamento de estruturas conceituais para estruturas NoSQL documento, no sentido de *não permitir redundância de dados* e ao mesmo tempo apresentar uma *estrutura adequada* para representar conceitos frequentemente acessados pelas operações do BD.

4 PROPOSTA PARA PROJETO LÓGICO DE BANCOS DE DADOS NOSQL DOCUMENTO

Este capítulo apresenta a abordagem desenvolvida para apoiar o projeto lógico de BDs NoSQL documento. A abordagem enfatiza a conversão de esquemas conceituais definidos pelo modelo EER em esquemas lógicos definidos por um modelo lógico NoSQL da categoria documento. A abordagem proposta por este trabalho é fundamentada na metodologia de projeto lógico em dois níveis descrita em (BATINI; CERI; NAVATHE, 1992) com algumas variações. A Figura 11 ilustra a abordagem proposta, que tem como entrada um esquema conceitual EER e informações de carga de dados, ambos fornecidos por um usuário especialista.

Metodologias consolidadas para a modelagem de BDs assumem que o modelo conceitual deve ser independente de quaisquer modelos de implementação e que a independência entre os modelos do projeto de um BD é um requisito fundamental para garantir a portabilidade da metodologia (BATINI; CERI; NAVATHE, 1992; ELMASRI; NAVATHE, 2011). O modelo EER é um modelo clássico, simples e considerado adequado à representação de dados de um domínio de aplicação em um alto grau de abstração (ELMASRI; NAVATHE, 2011). Desta forma, uma abordagem de projeto lógico para BDs NoSQL baseada no modelo EER pode ser considerada flexível, pois pode ser facilmente adaptada para outros modelos conceituais como o diagrama de classes da UML, por exemplo. Informações de carga são utilizadas para a geração de uma estrutura NoSQL que possa responder de forma eficiente as principais e mais custosas operações do BD NoSQL documento.

Figura 11 - Projeto lógico para BDs NoSQL documento.



Fonte: Adaptado de Lima e Mello (2015a).

Na fase de projeto lógico, o esquema conceitual é transformado em um esquema lógico NoSQL documento através de um processo baseado em regras de conversão. O modelo lógico NoSQL documento adotado é um modelo abstrato definido neste trabalho para representar o modelo de implementação NoSQL. O formato de intercâmbio de dados JSON (JSON, 2014) é o principal modelo de implementação para os BDs NoSQL da categoria documento. Embora o projeto de implementação seja considerado por esta abordagem, *este trabalho enfatiza a modelagem lógica* de BDs NoSQL (e não detalha o projeto de implementação), visto que o modelo lógico NoSQL documento empregado por esta abordagem pode ser *facilmente* mapeado para o modelo de implementação de BDs NoSQL documento.

Este capítulo está dividido em 4 seções. A primeira seção destina-se a apresentar o modelo lógico NoSQL documento proposto. Na segunda seção, as regras de conversão dos construtores do EER para os respectivos construtores do modelo lógico são apresentadas. O algoritmo de conversão, o qual determina o processo de conversão através da aplicação ordenada das regras consideradas, é apresentado na terceira seção, que também considera informações de carga estimadas para o BD NoSQL documento. Informações de carga são utilizadas pelo processo para a geração de estruturas NoSQL documento otimizadas capazes de responder com mais eficiência às operações esperadas para o BD. A última seção apresenta as considerações finais deste capítulo.

4.1 MODELO LÓGICO NOSQL DOCUMENTO

O modelo lógico para BDs NoSQL documento proposto é uma adaptação da abordagem de agregados (EVANS, 2003) acrescido com os construtores e as restrições do formato de dados comum a dados NoSQL da categoria documento, JSON. A escolha por uma representação lógica baseada em agregados justifica-se pelo fato que eles apoiam os requisitos típicos dos BDs NoSQL, ou seja, são unidades de distribuição (fornecem suporte à escalabilidade) e consistência (na medida em que for necessário), bem como podem ser divididos em pequenos elementos de dados (por questões de desempenho) (BUGIOTTI et al., 2014; HELLAND, 2007; JOVANOVIC; BENSON, 2013; SADALAGE; FOWLER, 2013). Desta forma, recomenda-se que um BD NoSQL documento seja projetado considerando a noção de agregados, pois estes BDs gravam e recuperam documentos (JSON, BSON ou XML). Estes documentos, por sua vez, são, concretamente, estruturas de dados hierárquicas que podem consistir em coleções de dados aninhados, além de valores escalares (MCMURTRY et al., 2013; SADALAGE; FOWLER, 2013).

Contribuições dos trabalhos relacionados, como do modelo lógico genérico NoAM (BUGIOTTI et al., 2014), e relacionados ao modelo lógico XML através das características hierárquicas do XML, também são considerados no modelo lógico proposto neste trabalho.

Os conceitos do modelo lógico NoSQL documento proposto são *coleções* (noção de agregado), *blocos* e *atributos*. Coleções, blocos e atributos constituem fatos que fazem sentido no domínio de uma aplicação. Um esquema NoSQL documento possui uma ou mais coleções. Uma coleção, por sua vez, é composta por um bloco raiz, sendo que todas as atualizações na coleção passam por este bloco, garantindo, assim, as regras de negócio da aplicação. O bloco raiz é o único bloco acessível de fora do limite da coleção. A formalização dos conceitos do modelo lógico proposto é apresentada a seguir.

Definição 6 (*Esquema Lógico NoSQL documento*). Um esquema lógico NoSQL documento END_i é um conjunto de coleções C_i, onde cada coleção possui um *nome único* no esquema.

Definição 7 (*Coleção*). Uma coleção $c_j \in C_i$ é um conjunto não vazio de blocos B_j, possuindo c_j um bloco raiz $br_j \in B_j$.

Definição 8 (*Bloco Raiz*). Um bloco raiz br_k é composto por um atributo a_k que identifica unicamente o bloco raiz br_k em uma coleção c_k, e um conjunto não vazio de atributos A_k e/ou blocos B_k.

Definição 9 (*Bloco*). Um bloco b_x é composto por um conjunto de atributos A_x e/ou um conjunto de blocos aninhados B_x, sendo possível a existência de uma ou mais restrições de *disjunção* para dois ou mais blocos aninhados $\{b_1, ..., b_n\} \in B_x$.

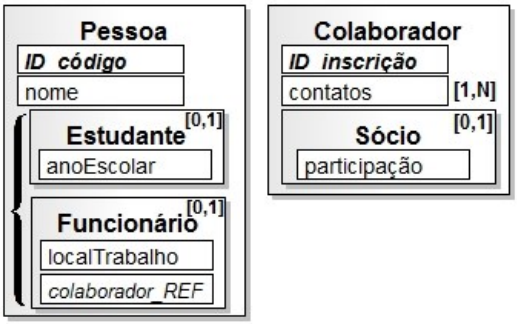
Definição 10 (*Atributo*). Um atributo a_y de um bloco b_y é uma tupla (c_y, v_y), onde c_y identifica unicamente a_y em b_y, e v_y é o valor de a_y.

Definição 11 (*Relacionamento Hierárquico*). Um relacionamento hierárquico é um relacionamento definido entre dois blocos bo_m e bd_m, sendo bo_m o bloco agregador (bloco origem) e bd_m o bloco embutido (bloco agregado ou destino).

Definição 12 (*Relacionamento de Referência*). Um relacionamento de referência é representado por um atributo de referência $ar_n \in bo_n$ ou um conjunto de atributos de referência $AR_n \in bo_n$, onde bo_n identifica um bloco origem. O atributo de referência ar_n refere-se ao atributo identificador a_o do bloco raiz da coleção destino br_o.

Com o objetivo de apresentar a notação gráfica de cada conceito definido para o modelo lógico proposto, a Figura 12 apresenta um exemplo de esquema lógico NoSQL documento. Os atributos constituem propriedades de um bloco e podem ser do tipo *normal*, *identificador* ou de *referência*. Um atributo do tipo normal não impõe nenhuma restrição ao bloco que pertence, sendo representado graficamente por um retângulo contendo o nome do atributo. O atributo *nome* na Figura 12 é um exemplo de atributo normal da coleção *Pessoa*.

Figura 12 - Exemplo de esquema representado no modelo lógico para um BD NoSQL documento proposto neste trabalho.



Fonte: Adaptado de Lima e Mello (2015a).

Um atributo do tipo identificador faz parte do conjunto de atributos cujos valores identificam unicamente uma instância do bloco ao qual pertencem. Este atributo é a chave do bloco agregador quando nos referimos ao bloco raiz. Sua representação gráfica é semelhante à de atributos normais, porém o nome deste atributo é formado pela *tag ID_* acrescido do nome do atributo. Este é o caso do atributo *ID_inscrição*, que representa o identificador único da coleção *Colaborador*. Atributos de referência são utilizados para referenciar outras coleções, sendo que estes atributos mantêm valores de atributos identificadores. Sua representação gráfica é semelhante à de atributos normais, porém o nome deste atributo é formado pelo nome da coleção acrescido da *tag _REF*. O atributo *colaborador_REF* da coleção *Pessoa* é um atributo de referência que contém valores que referenciam instâncias da coleção *Colaborador*.

Conforme definido anteriormente, o modelo lógico proposto suporta dois tipos de relacionamentos: *hierárquicos* ou de *referência*. Relacionamentos hierárquicos são representados por blocos aninhados (sub-blocos) de um bloco (origem). Um relacionamento hierárquico define as ocorrências mínima e

máxima do conceito destino presente no conceito origem. O relacionamento entre *Pessoa* e seu bloco aninhado *Estudante* é um exemplo: *Pessoa* pode ter *zero* ou *um Estudante* ([0,1]) e um *Estudante* é um bloco aninhado de *Pessoa*. A ocorrência máxima e mínima padrão para os conceitos destino é 1. A restrição de disjunção entre blocos aninhados presentes em um mesmo nível hierárquico é representada pelo símbolo de chave ("{"), graficamente alinhado à esquerda destes blocos. A restrição de disjunção definida para os blocos aninhados *Estudante* e *Funcionário* é um exemplo.

Relacionamentos de referência são representados por um atributo de referência (ou um conjunto de atributos de referência) descrito com o *nome da coleção* referenciada acrescido da *tag* *_REF*. Este atributo indica o atributo identificador da coleção, como é o caso do relacionamento entre o atributo *colaborador_REF* e a coleção *Colaborador*.

A próxima seção apresenta as regras de conversão dos construtores do modelo EER para os respectivos construtores do modelo lógico NoSQL documento recém definido.

4.2 REGRAS DE CONVERSÃO EER – MODELO LÓGICO NOSQL DOCUMENTO

Um conjunto de regras de conversão é definido nesta seção com a finalidade de prover estratégias de conversão para todos os construtores conceituais do modelo EER. Estas regras tratam a conversão de uma estrutura de dados em grafo para estruturas de dados em árvore, considerando que o modelo lógico NoSQL documento proposto é composto por estruturas de dados na forma de *árvores hierárquicas*. Contribuições dos trabalhos relacionados, incluindo estratégias de conversão, são utilizadas e adaptadas por este trabalho para considerar as construções do modelo NoSQL documento. As regras de conversão são agrupadas em quatro grupos que fornecem alternativas para conversão de conceitos do EER: (i) *Entidades e Atributos*; (ii) *Tipos Generalização*; (iii) *Tipos União* (Categorias); e (iv) *Tipos Relacionamento*.

4.2.1 Conversão de Entidades e Atributos

Entidades e atributos do modelo EER são, de modo geral, convertidos em *blocos* e *atributos* no modelo lógico NoSQL documento, respectivamente. Entretanto, algumas variações a esta regra geral são aplicadas aos atributos EER multivalorados e compostos. A seguir, as Regras ENT, ANC e ACO são definidas para converter tipos entidade, atributos não compostos e atributos compostos, respectivamente. As Regras ANC e ACO, destinadas à conversão

de atributos, são acionadas pela Regra ENT durante a conversão de um tipo entidade.

A aplicação das regras para conversão de entidades e atributos EER são exemplificadas através da Figura 13. A entidade *Estudante* (Figura 13 (a)) gera o bloco *Estudante* no esquema lógico NoSQL documento (Figura 13 (b)). Os atributos *matrícula* e *nome* são transformados em atributos (monovalorados) no esquema NoSQL documento. O atributo *matrícula* é marcado como um atributo identificador (recebendo a *tag ID_*), visto que atua como identificador da entidade *Estudante*. O atributo multivalorado *telefone* gera um atributo opcional pela aplicação da Regra ANC. A ocorrência do atributo *telefone* no bloco *Estudante* é definida como [0,N].

Regra ENT (*Entidade*). A conversão de um tipo entidade ***E*** procede da seguinte forma:

1. Gerar um bloco ***b*** com nome ***E***;

2. Dado o conjunto **A** = $\{a_1, a_2, ..., a_n\}$ de atributos de ***E***, para cada atributo $a_i \in$ **A** fazer:

Se (a_i não é um atributo composto) **Então** aplicar a ***Regra ANC***.

Senão aplicar a ***Regra ACO***.

Regra ANC (*Atributo Não-Composto*). A conversão de um atributo não composto ***A*** de um tipo entidade ***E*** convertida em um bloco ***b*** procede da seguinte forma:

Se (***A*** é um atributo monovalorado) **Então** gerar um atributo ***a*** como um atributo do bloco ***b***. A ocorrência de ***a*** em ***b*** é definida como **[0-1, 1]**, dependendo se ***A*** é opcional ou obrigatório. Se ***A*** é um atributo identificador de ***E***, o atributo ***a*** deve ser marcado como um atributo identificador.

Senão gerar um atributo ***a*** como um atributo do bloco ***b***. A ocorrência de ***a*** em ***b*** é definida como **[0-1, N]**, dependendo se ***A*** é opcional ou obrigatório.

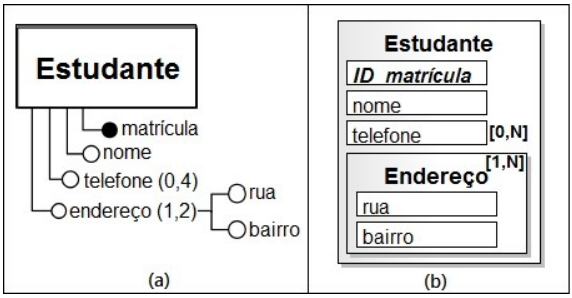
Regra ACO (*Atributo Composto*). A conversão de um atributo composto ***C*** de um tipo entidade ***E*** convertida em um bloco ***b*** procede da seguinte forma:

1. Gerar um bloco ***bc*** e um relacionamento hierárquico de ***b*** para ***bc*** onde a ocorrência de ***bc*** em **b** é definida como **[0-1, 1-N]**, dependendo se ***C*** é opcional ou obrigatório, e se ***C*** é um atributo monovalorado ou multivalorado;

2. Dado o conjunto **A** = $\{a_1, a_2, ..., a_n\}$ de atributos componentes de ***C***, para cada atributo $a_i \in$ **A** aplicar a ***Regra ANC***.

Como exemplo da aplicação da Regra ACO, o atributo composto *endereço* é transformado em um bloco aninhado ao bloco *Estudante*, conforme ilustra a Figura 13. O modelo de conteúdo do bloco *endereço* é definido pelos seus dois atributos componentes: *rua* e *bairro*.

Figura 13 - Aplicação das regras de conversão de entidades e atributos.



Fonte: Desenvolvido pelo autor.

4.2.2 Conversão de Tipos Generalização

Três alternativas são propostas para a conversão de hierarquias de generalização do modelo EER para composições equivalentes no modelo lógico NoSQL documento. As alternativas diferenciam-se, principalmente, pelo tamanho da porção de esquema NoSQL documento gerado e pela forma de representação de uma hierarquia de generalização e suas respectivas restrições (total e disjunta; total e compartilhada; parcial e disjunta; e parcial e compartilhada). Estas alternativas são baseadas em estratégias para a conversão de generalizações do EER para modelos de BDs convencionais, definidas em (BATINI; CERI; NAVATHE, 1992; HEUSER, 2008) e adaptadas por este trabalho para o modelo NoSQL documento. As alternativas são definidas a seguir.

A aplicação da Regra GSP para conversão de tipos generalização é exemplificada através da Figura 14. A Figura 14 (b) apresenta um fragmento de esquema lógico NoSQL documento gerado pela aplicação da Regra GSP. O atributo *tipo* gerado tem a função de identificar as instâncias de quais subclasses está-se representando. A ocorrência deste atributo é determinada de acordo com as restrições de totalidade que podem ser impostas sobre hierarquias de generalização compartilhadas: [1,N] para generalizações totais e compartilhadas, e [0,N] para generalizações parciais e compartilhadas. Para generalizações disjuntas, esta definição cria um atributo para atuar como

discriminador, que apresenta ocorrência [0,1] para generalizações parciais e disjuntas, e [1,1] para generalizações totais e disjuntas.

Regra GSP (*Generalização com Ênfase na Superclasse***)**. A conversão de um tipo generalização ***G*** procede da seguinte forma:

1. Dada uma entidade e_{sp} definida como a superclasse de ***G***, aplicar a ***Regra ENT*** para conversão de e_{sp} gerando um bloco b_{sp};

2. Dado um conjunto $E_{sb} = \{e_{sb1}, e_{sb2}, ..., e_{sbn}\}$ de subclasses de e_{sp}, para cada $e_{sbi} \in E_{sb}$ fazer:

Se (e_{sbi} não foi convertido) **Então** definir atributos opcionais em b_{sp} para representar os atributos de e_{sbi}.

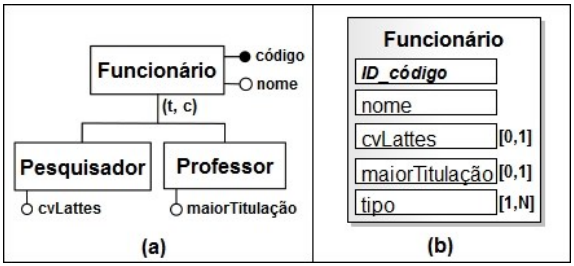
Senão dado o bloco gerado b_{sbi}, gerar relacionamento hierárquico de b_{sp} para b_{sbi} onde a ocorrência de b_{sbi} em b_{sp} é definida como **[0,1]**.

Criar um atributo normal ***tipo*** como um atributo de b_{sp} onde a ocorrência de ***tipo*** em b_{sp} é definida como **[0-1,1-N],** dependendo se ***G*** é *total* ou *parcial*, e é *disjunta* ou *compartilhada*.

Os atributos das subclasses são definidos como atributos opcionais do bloco gerado. Isto ocorre porque, do contrário, uma dependência incorreta estaria sendo gerada mesmo para generalizações totais e compartilhadas, onde *nem sempre* as instâncias das subclasses são representadas ao *mesmo tempo* (compartilhadas).

Em casos de conversão de hierarquias de generalização com múltiplos níveis hierárquicos, pode surgir uma exceção para a definição dos atributos das subclasses no bloco da superclasse quando um bloco já houver sido gerado para representar uma subclasse. Neste caso, o bloco previamente criado deve ser definido como um bloco aninhado opcional do bloco gerado para representar a superclasse.

Figura 14 - (a) Hierarquia de generalização total e compartilhada; (b) Aplicação da Regra GSP.



Fonte: Desenvolvido pelo autor.

A principal restrição para a aplicação da Regra GSP ocorre quando uma das subclasses estiver definida como *bloco referenciado*. Assume-se como bloco referenciado uma superclasse que anteriormente foi processada por outra hierarquia de generalização e que foi referenciada por um bloco externo. Esta restrição garante que o bloco referenciado se tornará um bloco raiz, evitando que este bloco referenciado se torne um bloco aninhado em uma conversão futura. Esta restrição é típica do modelo lógico proposto, baseado em agregados.

A Regra GSP assume que a distinção explícita entre as subclasses é *irrelevante* para a maioria das instâncias da superclasse. Assim sendo, a existência de relacionamentos com as subclasses é uma restrição para a aplicação da Regra GSP, visto que, conceitualmente, pode significar que a distinção entre as subclasses é relevante para a aplicação. Esta alternativa de conversão gera a menor porção de esquema NoSQL documento se comparada às demais alternativas para conversão de hierarquias de generalização. A regra correspondente à segunda alternativa de conversão é definida a seguir.

Regra GSB (*Generalização com Ênfase nas Subclasses***)**. A conversão de um tipo generalização ***G*** procede da seguinte forma:

1. Dada uma entidade e_{sp} definida como a superclasse de ***G*** e o conjunto $E_{sb} = \{e_{sb1}, e_{sb2}, ..., e_{sbn}\}$ de subclasses de e_{sp}, fazer:

Gerar um bloco b_{sbi} para cada subclasse $e_{sbi} \in E_{sb}$ através da aplicação da ***Regra ENT*** e definir os atributos de e_{sp} como atributos obrigatórios em cada b_{sbi}.

A aplicação da Regra GSB para conversão de tipos generalização é exemplificada através da Figura 15.

Figura 15 - (a) Hierarquia de generalização total e disjunta; (b) Aplicação da Regra GSB.

Fonte: Desenvolvido pelo autor.

A Figura 15 (b) apresenta um fragmento de esquema lógico NoSQL documento gerado pela aplicação da Regra GSB. Neste exemplo, as subclasses são transformadas em blocos criados para representá-las, sendo os atributos da superclasse replicados para ambos os blocos de subclasse. Esta regra não deve ser aplicada para uma generalização parcial, pois, neste caso, instâncias de subclasses nem sempre existem.

Esta alternativa de conversão não é recomendada para generalizações compartilhadas, visto que, se uma instância de superclasse é especializada em mais de uma instância de subclasse, ocorre redundância de dados quanto aos valores de atributos que representam a superclasse nos blocos das subclasses. A aplicação da Regra GSB também não é recomendada nos casos em que os relacionamentos associados com a superclasse deverão ser convertidos em relacionamentos com cada uma das subclasses.

A definição da terceira alternativa de conversão é apresentada a seguir.

Regra GHI (*Generalização com Ênfase na Hierarquia***)**. A conversão de um tipo generalização ***G*** procede da seguinte forma:

1. Dada uma entidade e_{sp} definida como a superclasse de ***G*** e o conjunto $E_{sb} = \{e_{sb1}, e_{sb2}, ..., e_{sbn}\}$ de subclasses de e_{sp}, gerar um bloco b_{sp} para e_{sp} através da aplicação da ***Regra ENT***, tornar os atributos de e_{sp} atributos em b_{sp} e fazer:

Se (*nenhuma subclasse de* ***G*** *foi marcada como convertida*) **e** (***G*** *é disjunta*) **Então** representar a restrição de disjunção;

2. Para cada subclasse $e_{sbi} \in E_{sb}$, fazer:

Se (e_{sbi} *não é uma subclasse marcada como convertida*) **Então** gerar um bloco b_{sbi} através da aplicação da ***Regra ENT*** e fazer:

Gerar um relacionamento hierárquico de b_{sp} para b_{sbi}. A ocorrência de b_{sbi} em b_{sp} é definida como **[1,1]** se ***G*** é *total* e *disjunta* e se *nenhuma subclasse de* ***G*** *foi marcada como convertida*. Para os demais casos, definir a ocorrência de b_{sbi} em b_{sp} como **[0,1]**.

Senão deixar bc_{sbi} ser um bloco previamente criado para representar e_{sbi}, gerar atributo(s) de referência em bc_{sbi} referenciando-se ao identificador de b_{sp} e definir b_{sp} como *bloco referenciado***.**

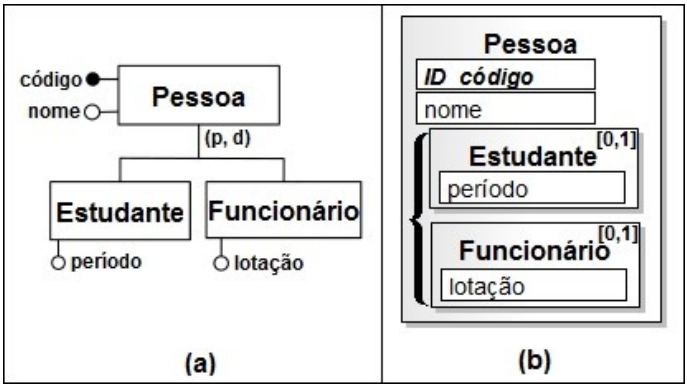
Nesta alternativa, a superclasse e suas subclasses são explicitamente representadas por blocos, sendo que cada bloco mantém seus próprios atributos. Relacionamentos hierárquicos são estabelecidos entre o bloco da superclasse e os blocos das subclasses, correspondendo aos relacionamentos de especialização no nível conceitual. Entretanto, quando alguma subclasse de uma generalização já tiver sido convertida (subclasse marcada), o

relacionamento com o bloco da superclasse é estabelecido por um relacionamento de referência entre o bloco da superclasse e o bloco previamente criado para representar a subclasse convertida. Neste caso, cria-se um atributo na subclasse que referencia o bloco da superclasse, que é definido como *bloco referenciado*. Uma subclasse já convertida é uma subclasse que foi processada anteriormente por outra hierarquia de generalização e se transformou em um bloco aninhado de algum outro bloco ou em atributos de um bloco que não está representando apenas a subclasse em questão.

A Regra GHI realiza verificações antes de representar a restrição de disjunção. No caso em que pelo menos uma das subclasses já houver sido marcada como convertida, a restrição de disjunção não é representada, pois *nem todas* as subclasses serão representadas como blocos aninhados da superclasse para representar a disjunção corretamente. Nestes casos, a ocorrência dos blocos aninhados na superclasse (relacionamento hierárquico) é definida como [0,1].

A Figura 16 (b) apresenta um fragmento de esquema lógico NoSQL documento gerado pela aplicação da Regra GHI sobre as hierarquias de generalização da Figura 16 (a).

Figura 16 - (a) Hierarquia de generalização parcial e disjunta; (b) Aplicação da Regra GHI.



Fonte: Desenvolvido pelo autor.

A aplicação da regra GHI, conforme apresentado na Figura 16, gera o bloco *Pessoa* e os blocos aninhados *Estudante* e *Funcionário*. Por tratar-se de uma generalização parcial, a restrição de parcialidade é representada pela cardinalidade [0,1] nos blocos aninhados (*Estudante* e *Funcionário*) do bloco *Pessoa*. Já a restrição de disjunção é representada no bloco *Pessoa* através do

símbolo de chave ("{"), graficamente alinhado à esquerda dos blocos aninhados destino (*Estudante* e *Funcionário*).

A alternativa provida pela Regra GHI gera uma maior porção de esquema NoSQL documento, mas torna a representação dos blocos envolvidos mais flexível, principalmente quando existem tipos relacionamento associados à superclasse e às subclasses. As restrições sobre as hierarquias de generalização são explicitamente representadas por esta regra.

4.2.3 Conversão de Tipos União

Categorias ou tipos união podem ser considerados casos mais restritos de herança múltipla. Desta forma, as estratégias de conversão são análogas às estratégias para transformação de hierarquias de generalização. Entretanto, considerando principalmente os casos de restrição possíveis (totais e parciais), alguns ajustes devem ser considerados na conversão destes tipos. Um tipo união com restrição total pode ser convertido similarmente como um tipo generalização total e disjunto com algumas variações (BATINI; CERI; NAVATHE, 1992). Desta forma, as Regras USB, USP e UHI são apresentadas na sequência e constituem adaptações para conversão de tipos união nas Regras GSP, GSB e GHI, respectivamente.

Regra USB (*União com Ênfase na Subclasse***)**. A conversão de um tipo união ***U*** procede da seguinte forma:

1. Dada uma entidade e_{sb} definida como a subclasse de ***U***, aplicar a ***Regra ENT*** para conversão de e_{sb}, gerando um bloco b_{sb};

2. Dado o conjunto $E_{sp} = \{e_{sp1}, e_{sp2}, ..., e_{spn}\}$ de superclasses de e_{sb}, para cada $e_{spi} \in E_{sp}$, definir atributos opcionais em b_{sb} para representar os atributos de b_{spi}. No caso de um bloco b_{spi} já tiver sido gerado para representar uma superclasse e_{spi}, definir este bloco como bloco aninhado de b_{sb} com ocorrência **[0,1]**;

3. Se (e_{sb} não possui atributo identificador) **Então** criar um atributo identificador a_{id} para atuar como identificador de b_{sb};

4. Criar um atributo normal a_{tipo} em b_{sb}, onde a ocorrência de a_{tipo} em b_{sb} é definida como **[1,1]**.

Além de inverter o tratamento da superclasse de uma generalização para a subclasse de um tipo união, a principal adaptação da Regra USB com relação à Regra GSP é a criação de um atributo (a_{id}) para atuar como identificador no bloco gerado para representar a subclasse, quando a subclasse não possuir atributo identificador. A restrição de parcialidade estabelece que as superclasses *nem sempre* devam ser especializadas pela subclasse, por este

motivo esta regra só pode ser aplicada para tipos união total. O atributo discriminador (a_{tipo}) gerado pela Regra USB é sempre um atributo normal com ocorrência [1,1], pois estará representando uma união total (com ocorrência mínima 1) onde existe uma disjunção entre as superclasses representadas pela subclasse (com ocorrência máxima 1).

A principal restrição para a aplicação da Regra USB ocorre quando uma das superclasses estiver definida como *bloco referenciado*. Assume-se como bloco referenciado uma entidade que anteriormente foi processada por outra hierarquia e que foi referenciada por um bloco externo. Esta restrição garante que o bloco referenciado se tornará um bloco raiz, evitando que este bloco referenciado se torne um bloco aninhado em uma conversão futura. Esta restrição é típica do modelo lógico proposto, baseado em agregados. A existência de relacionamentos com as superclasses também é uma restrição importante para a aplicação da Regra USB, visto que, conceitualmente, pode significar que a distinção entre as superclasses é *relevante* para a aplicação.

Regra USP (*União com Ênfase nas Superclasses***)**. A conversão de um tipo união ***U*** procede da seguinte forma:

1. Dada uma entidade $\boldsymbol{e_{sb}}$ definida como a subclasse de ***U*** e o conjunto $\boldsymbol{E_{sp}}=\{\boldsymbol{e_{sp1}}, \boldsymbol{e_{sp2}}, ..., \boldsymbol{e_{spn}}\}$ de superclasses de $\boldsymbol{E_{sb}}$, fazer:

Gerar um bloco $\boldsymbol{b_{spi}}$ para cada superclasse $\boldsymbol{e_{spi}} \in \boldsymbol{E_{sp}}$ através da aplicação da ***Regra ENT*** e definir os atributos de $\boldsymbol{e_{sb}}$ como *atributos obrigatórios* em cada $\boldsymbol{b_{spi}}$ se ***U*** é *total*, e como *atributos opcionais* se ***U*** é *parcial*.

A alteração estabelecida pela Regra USP sobre a Regra GSB refere-se à definição dos atributos da subclasse como opcionais quando se tratar de um tipo união parcial. Isto ocorre porque em um tipo união parcial as superclasses não são sempre especializadas pela subclasse.

Já para a Regra UHI, as principais adaptações com relação à Regra GHI referem-se à geração de um atributo identificador para a entidade que representa a subclasse e ao tratamento diferenciado para tipos união total e parcial. Em uma união parcial as superclasses nem sempre são especializadas pela subclasse. Portanto, somente um tipo união total pode definir as superclasses como blocos aninhados do bloco da subclasse. Isto ocorre porque estabelecer as superclasses como bloco aninhado do bloco da subclasse tornaria a existência das superclasses dependente de uma instância da subclasse e, quando o tipo união é parcial, tal dependência não é válida.

Regra UHI (*União com Ênfase na Hierarquia***)**. A conversão de um tipo união ***U*** procede da seguinte forma:

1. Dada uma entidade e_{sb} definida como a subclasse de ***U*** e o conjunto $E_{sp} = \{e_{sp1}, ..., e_{spn}\}$ de superclasses de e_{sb}, gerar um bloco b_{sb} através da aplicação da ***Regra ENT*** para representar e_{sb} e fazer:

Se (*nenhuma superclasse de **U** foi marcada como convertida*) **e** (***U*** é *total*) **Então** representar a restrição de disjunção;

2. Para cada $e_{spi} \in E_{sp}$, fazer:

Se (e_{spi} *não é uma superclasse marcada como convertida*) **Então** gerar um bloco b_{spi} através da aplicação da ***Regra ENT*** e fazer:

Se (***U*** é *total*) **e** (e_{spi} *não é uma superclasse marcada como convertida*) **Então** fazer b_{spi} ser um bloco aninhado de b_{sb}. A ocorrência de b_{spi} em b_{sb} é definida como **[1,1]**.

Senão Se (e_{sb} *não está marcada como convertida*) **Então** fazer b_{sb} ser um bloco aninhado de cada b_{spi}. A ocorrência de b_{sb} em cada b_{spi} é definida como **[0,1]** quando ***U*** é *parcial* e como **[1,1]** quando ***U*** é *total*.

Senão gerar atributo(s) de referência em b_{spi} referenciando-se ao identificador de b_{sb}, definir a ocorrência dos atributos como *opcionais* se ***U*** é *parcial* e como *obrigatórios* se ***U*** é *total*, e definir b_{sp} como *bloco referenciado***.**

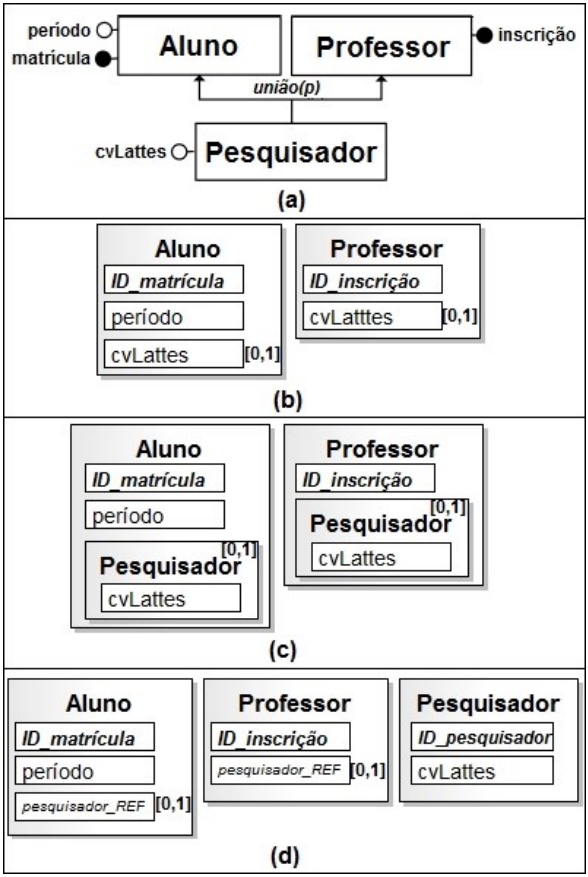
3. Se (e_{sb} não possui atributo identificador) **e** (b_{sb} não estiver sido definido como bloco aninhado dos blocos gerados para representar as superclasses) **Então** criar um atributo identificador a_{id} para atuar como identificador de b_{sb};

Para tipos união parcial ou para tipos união total cujas superclasses estão marcadas como convertidas, um bloco representando a subclasse é definido como bloco aninhado de cada bloco de superclasse. A *restrição* para a aplicação desta conversão ocorre quando a subclasse estiver marcada como convertida. Neste caso, como sempre existirá uma disjunção entre as superclasses de um tipo união, *a redundância é evitada*, pois uma instância de subclasse nunca estará atuando como um bloco aninhado de mais de uma instância de superclasse.

Na última opção de conversão provida pela Regra UHI, os blocos das superclasses são associados ao bloco da subclasse através de relacionamentos de referência. Neste caso, o bloco da subclasse é definido como *bloco referenciado*. A Figura 17 (a) apresenta um exemplo de tipo união parcial e os fragmentos do esquema lógico NoSQL documento gerado pela aplicação das regras USP e (duas possíveis conversões da regra) UHI, respectivamente.

Considerando o exemplo da Figura 17 (a), a Regra USB não pode ser aplicada para a conversão de tipos união com restrição parcial. A aplicação da Regra USP gera apenas blocos para representar as superclasses e os atributos da subclasse devem ser definidos como atributos opcionais nos blocos criados, devido à restrição de parcialidade, conforme apresentado na Figura 17 (b).

Figura 17 - (a) Tipo união parcial; (b) Blocos gerados no modelo lógico NoSQL documento através da aplicação da Regra USP; (c) e (d) Blocos gerados através da aplicação da Regra UHI.



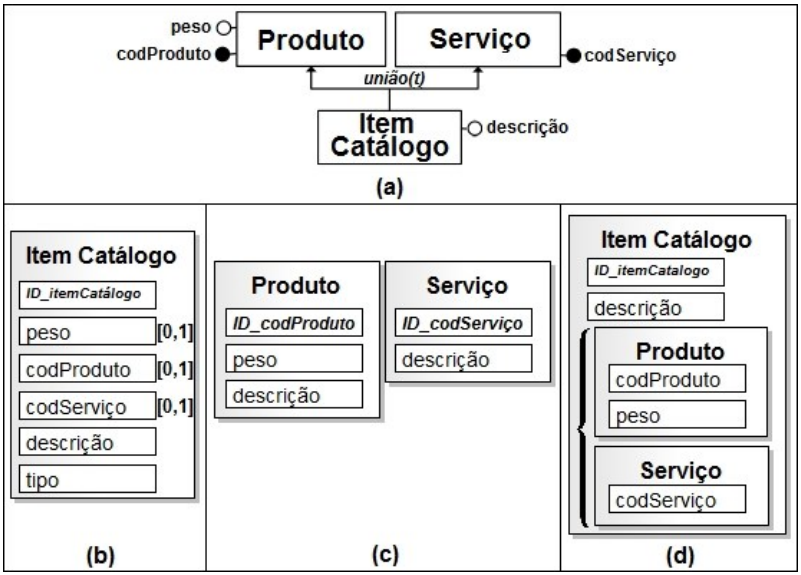
Fonte: Desenvolvido pelo autor.

A restrição parcial também requer que os relacionamentos de especialização sejam estabelecidos através de relacionamentos hierárquicos a

partir das superclasses ou através de relacionamentos por referência na aplicação da Regra UHI sobre um tipo união parcial. Na Figura 17 (c) relacionamentos hierárquicos são estabelecidos a partir de cada superclasse com a subclasse. Já na Figura 17 (d) relacionamentos de referência são utilizados para a representação da união parcial.

A Figura 18 (a) apresenta um exemplo de tipo união total e os fragmentos de esquema lógico NoSQL documento gerados pela aplicação das Regras USB, USP e UHI, respectivamente.

Figura 18 - (a) Tipo união total; (b) (c) e (d) Blocos gerados através da aplicação das Regras USB, USP e UHI, respectivamente.



Fonte: Desenvolvido pelo autor.

Seguindo as restrições de tipos união, uma instância de bloco *Item Catálogo* não pode representar ao mesmo tempo uma instância de *Produto* e *Serviço*, por isso não faz sentido eleger um dos identificadores das superclasses como o atributo identificador de *Item Catálogo*. Neste caso, um novo atributo identificador (*ID_itemCatálogo*) foi criado para este fim. Uma situação semelhante ocorre na aplicação da Regra UHI na Figura 18 (d). Nesta aplicação da Regra UHI, um atributo *ID_itemCatálogo* é criado para atuar como o identificador do bloco que representa a subclasse *Item Catálogo*.

4.2.4 Conversão de Relacionamentos

Nesta seção são apresentadas três regras para conversão de relacionamentos do modelo EER para composições equivalentes no modelo lógico NoSQL documento. As restrições de *grau* e de *cardinalidades mínima e máxima* são consideradas para a escolha da regra de conversão adequada. A primeira regra (RBU) é definida a seguir.

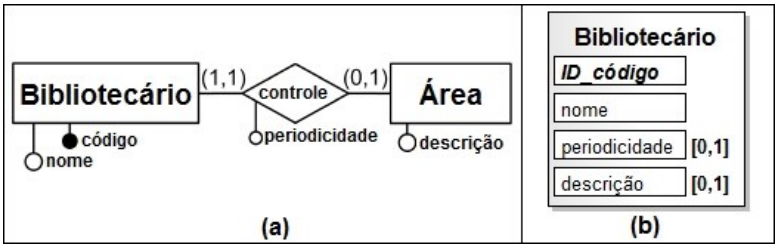
Regra RBU (*Relacionamento Modelado por Bloco Único***).** A conversão de um tipo relacionamento binário ***R*** procede da seguinte forma:

> **1.** Dado um tipo relacionamento ***R*** o qual relaciona as entidades e_1 e e_2, gerar um bloco b_{e1} através da aplicação da ***Regra ENT*** para e_1 e definir os atributos de e_2 e ***R*** como atributos em b_{e1}.

A Regra RBU gera um único bloco para representar o relacionamento e é aplicável *apenas* a tipos relacionamento com cardinalidade (1,1). Para a correta aplicação desta regra é necessário que *pelo menos* uma das entidades apresente a participação (1,1) (a outra entidade, neste caso, pode ter participação (0,1)). Os atributos desta entidade com participação (1,1) são então representados como atributos da outra entidade da relação. Eventuais atributos identificadores da entidade com participação (1,1) são tratados como atributos normais no bloco gerado, pois a existência de dois identificadores para o bloco gerado não faria sentido.

A Figura 19 apresenta um exemplo de aplicação desta regra, sendo que os atributos da entidade *Área* são adicionados ao modelo de conteúdo do único bloco criado (*Bibliotecário*) devido à participação (1,1) de *Área* no relacionamento. Os atributos do relacionamento *controle* e da entidade *Área* são definidos como opcionais devido à participação (0,1) de *Bibliotecário* neste relacionamento.

Figura 19 - (a) Relacionamento 1:1; (b) Aplicação da Regra RBU.



Fonte: Desenvolvido pelo autor.

Observa-se que a conversão de *entidades fracas* é diretamente suportada pela regra RBU, uma vez que a entidade fraca é representada no modelo de conteúdo do bloco criado para representar a entidade forte, tendo como identificador o atributo identificador da entidade forte.

A segunda regra é definida a seguir.

Regra RHI (*Relacionamento Modelado por Hierarquia***).** A conversão de um tipo relacionamento binário ***R*** procede da seguinte forma:

1. Dado um tipo relacionamento ***R*** que relaciona as entidades e_1 e e_2, gerar um bloco b_{e1} através da aplicação da ***Regra ENT*** para e_1;

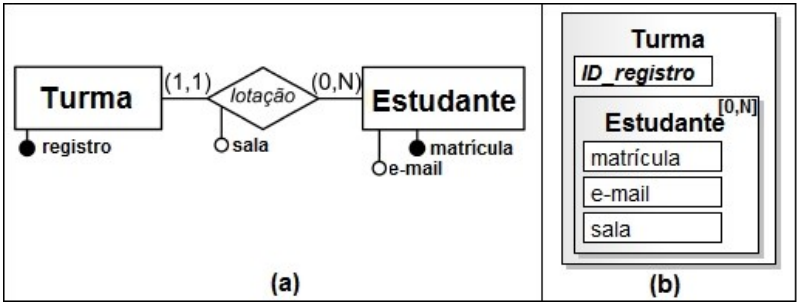
2. Gerar um bloco b_{e2} através da aplicação da ***Regra ENT*** como um bloco aninhado de b_{e1} para representar a entidade e_2. A ocorrência de b_{e2} em b_{e1} é definida de acordo com a participação de e_1 em ***R*** (opcional ou obrigatória);

3. Definir os atributos de ***R*** como atributos em b_{e2}.

A Regra RHI pode ser aplicada a tipos relacionamento com cardinalidade (1:1) ou (1:N). Para a correta aplicação desta regra também é necessário que *pelo menos* uma das entidades apresente participação (1,1) no relacionamento. Nesta estratégia de conversão, a entidade que representa a entidade de participação (1,1) é representada como um bloco aninhado do bloco que representa a outra entidade do relacionamento. Os atributos do relacionamento são representados como atributos no modelo de conteúdo do bloco aninhado.

A Figura 20 apresenta um exemplo de aplicação desta regra.

Figura 20 - (a) Relacionamento 1:N; (b) Aplicação da Regra RHI.



Fonte: Desenvolvido pelo autor.

A Regra RHI não deve ser aplicada para relacionamentos em que a entidade de participação (1,1) é definida com *bloco referenciado*. Assume-se como bloco referenciado uma entidade que anteriormente foi processada e que foi referenciada por um bloco externo. Esta restrição garante que o bloco referenciado se tornará um bloco raiz, evitando que este bloco referenciado se torne um bloco aninhado em uma conversão futura. Esta restrição é típica do modelo lógico proposto, baseado em agregados, e evita que atributos (identificadores) de blocos aninhados sejam referenciados por outro bloco externo do esquema NoSQL documento.

A terceira regra é definida a seguir.

Regra RRE (*Relacionamento Modelado por Referências***).** A conversão de um tipo relacionamento $\boldsymbol{R}$ procede da seguinte forma:

1. Dado um tipo relacionamento $\boldsymbol{R}$ e o conjunto $\boldsymbol{E} = \{\boldsymbol{e_1}, \boldsymbol{e_2}, ..., \boldsymbol{e_n}\}$ de entidades relacionadas por $\boldsymbol{R}$, para cada $\boldsymbol{e_i} \in \boldsymbol{E}$ gerar um bloco $\boldsymbol{b_{ei}}$ através da aplicação da ***Regra ENT***;

2. Se (***R** é um relacionamento binário sem atributos*) **e** (a participação de uma das entidades em $\boldsymbol{R}$ chamada $\boldsymbol{e_1}$ é definida como **([0-1],1)**), **Então** gerar atributo(s) de referência a partir de $\boldsymbol{b_{e1}}$ referenciando-se ao identificador de $\boldsymbol{b_{e2}}$ e definir $\boldsymbol{b_{e2}}$ como *bloco referenciado*;

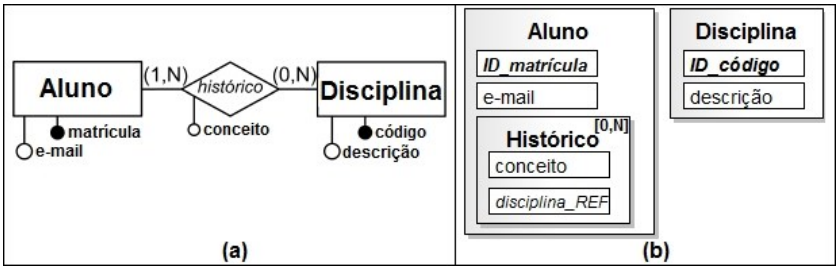
Senão gerar um bloco $\boldsymbol{b_R}$ como um bloco aninhado de $\boldsymbol{b_{e1}}$ e definir os atributos de $\boldsymbol{R}$ como atributos em $\boldsymbol{b_R}$. Para cada $\boldsymbol{e_i} \in \boldsymbol{E}$ gerar atributo(s) de referência em $\boldsymbol{b_R}$ referenciando-se ao identificador de $\boldsymbol{b_{ei}}$ e definir $\boldsymbol{b_{ei}}$ como *bloco referenciado*.

A Regra RRE é aplicada em relacionamentos (N:N) ou relacionamentos com grau *n*, onde $n > 2$. Esta regra cria um bloco para representar o relacionamento e o torna bloco aninhado de um bloco que representa uma das entidades envolvidas no tipo relacionamento. Relacionamentos de referência com os blocos das demais entidades são estabelecidos a partir do bloco criado para representar o relacionamento conceitual, e os blocos das demais entidades são definidos como blocos referenciados. A Figura 21 apresenta um exemplo de aplicação desta regra sobre um tipo relacionamento (N:N). Neste caso, o bloco do relacionamento *Histórico* é representado como um bloco aninhado de *Aluno* e um relacionamento de referência com *Disciplina* é estabelecido a partir do bloco *Histórico*.

Quando um bloco para representar o relacionamento é criado como um bloco aninhado do bloco correspondente a uma das entidades do relacionamento, *evita-se a redundância de dados* na conversão de relacionamentos (N:N). O relacionamento com as demais entidades

participantes é estabelecido através de relacionamentos de referência entre o bloco criado para representar o relacionamento e os blocos das demais entidades. Isto ocorre porque representar uma das entidades como bloco aninhado do bloco da outra entidade tornaria possível que os dados relativos a uma instância do bloco aninhado aparecessem repetidas vezes para as instâncias do bloco pai, dada uma participação máxima N de ambas as entidades.

Figura 21 - (a) Relacionamento N:N; (b) Aplicação da Regra RRE.



Fonte: Desenvolvido pelo autor.

A Regra RRE também é acionada em casos nos quais as Regras RBU e RHI não podem ser aplicadas a relacionamentos (1:1) e (1:N). Nestes casos, um relacionamento de referência é estabelecido diretamente da entidade com participação máxima 1 para o bloco que representa a outra entidade, *sem criar um bloco para representar o relacionamento*, minimizando o número de blocos gerados pela aplicação da regra. Observa-se que este tratamento não pode ser assumido quando existir *atributos no relacionamento*, necessitando, nestes casos, a criação de um bloco para representar o relacionamento e encapsular seus atributos.

Ainda, algumas verificações precisam ser executadas pelo processo de conversão para a aplicação das regras. Um exemplo poderia ser a conversão de um relacionamento (1:1) onde ambas as entidades têm participação opcional no relacionamento. Neste caso, a Regra RRE deve ser aplicada, visto que as Regras RBU e RHI necessitam da existência de uma entidade com participação (1,1) para a sua aplicação. Um exemplo de decisão que deve ser tomada pelo processo é a escolha da entidade que deverá representar o bloco pai do bloco criado para representar o relacionamento na aplicação da Regra RRE. A próxima seção apresenta o algoritmo de conversão, o qual determina o processo de conversão através da aplicação ordenada das regras apresentadas.

4.3 PROCESSO DE CONVERSÃO

Um processo *automático* para conversão de esquemas EER em esquemas lógicos NoSQL documento é proposto nesta seção. Contribuições dos trabalhos relacionados são utilizadas e adaptadas por este trabalho para considerar as construções e restrições do modelo NoSQL documento. O processo automático estabelece a ordem de conversão dos fragmentos de um esquema EER, assim como a regra de conversão que deve ser aplicada sobre cada fragmento considerado. O processo tem por finalidade representar integralmente um esquema EER em um esquema NoSQL documento e produzir um esquema lógico compacto e livre de redundância de dados. Um esquema lógico NoSQL documento compacto é obtido pela aplicação, sempre que possível, das regras de conversão que geram a menor porção de esquema NoSQL documento. A redundância de dados pode ser evitada, por exemplo, não permitindo que um conceito seja representado por mais de um bloco do esquema NoSQL documento.

Apesar de o processo ter sido proposto para a conversão automática de esquemas EER, nada impede que seja utilizado no contexto de um ambiente mais amplo de projeto de BDs NoSQL documento onde um usuário especialista seja responsável por revisar o esquema lógico gerado e proceder eventuais alterações.

Alguns relacionamentos entre conceitos de um esquema conceitual não podem ser mapeados para relacionamentos hierárquicos em um esquema NoSQL documento, conforme apresentado nas regras de conversão na Seção 4.2. Isto ocorre porque esquemas conceituais, como o EER ou o diagrama de classes da UML, podem definir grafos (inclusive com ciclos), o que impossibilita a representação em uma estrutura em árvores como denotam os esquemas NoSQL documento. Nestes casos, relacionamentos de referência são necessários para representar relacionamentos entre conceitos de um esquema conceitual em um esquema NoSQL documento. Embora em muitos casos necessários, relacionamentos de referência geram esquemas fragmentados que muitas vezes prejudicam o desempenho de consultas sobre documentos conformados a estes esquemas.

Quando uma consulta necessita recuperar dados que estão associados por um relacionamento de referência em um documento, a execução de *junções por valor* é necessária. Em uma *junção por valor*, valores de atributos de referência de um bloco são comparados com valores de atributos identificadores de outro bloco. Diferentemente de *junções por estrutura*, onde se recupera os blocos aninhados de um determinado bloco, *junções por valor* tornam a execução de consultas mais custosa na grande maioria dos casos em que são aplicadas.

O processo de conversão privilegia, sempre que possível, a geração de relacionamentos hierárquicos. Na conversão de relacionamentos, com exceção de relacionamentos N:N e N>2, as regras RBU e RHI, que não geram relacionamentos de referência, tem preferência de aplicação em relação a regra RRE, que gera relacionamentos de referência. Já na conversão de hierarquias, as regras GHI e UHI, que podem gerar relacionamentos de referência, são as últimas opções de escolha para a conversão de tipos generalização e tipos união, respectivamente.

Com a finalidade de *minimizar o número de comparações* que podem ser geradas pela execução de consultas sobre conceitos relacionados por referências, o processo *pode ser dirigido* por informações de carga do BD estimadas sobre um esquema conceitual. Considera-se como informações de carga o volume de instâncias esperadas para os conceitos do esquema EER e dados referentes às consultas e suas respectivas frequências de execução sobre o BD. A *modelagem de carga*, apresentada na Seção 2.4, é considerada neste trabalho com o objetivo de produzir informações de apoio ao processo de conversão que utiliza as informações de carga do BD. Embora seja difícil coletar informações sobre a carga de dados, estas informações são consideradas essenciais na tomada de decisões durante a modelagem lógica e física de um BD. Segundo (BATINI; CERI; NAVATHE, 1992), 20% das operações mais frequentes sobre os dados produzem 80% da carga do BD. Desta forma, é suficiente analisar 20% das operações mais frequentes do BD. O desempenho do BD só pode ser definitivamente determinado após a modelagem física, entretanto, a estrutura NoSQL documento produzida pela modelagem lógica pode ser considerada um fator determinante para o bom desempenho de aplicações baseadas em documentos NoSQL.

O detalhamento do processo de conversão é fornecido pelo *Algoritmo 1* (*EER - NoSQL*), que é composto pelas funções *converterHierarquias* e *converterRelacionamentos*. Tipos generalização e união são convertidos primeiramente devido à maior possibilidade de aplicar otimizações em estruturas NoSQL documento geradas para representar hierarquias. Em muitos casos é possível, por exemplo, representar uma hierarquia de generalização através de um único bloco. Os fragmentos gerados pela conversão de tipos generalização e união são então considerados juntamente com o esquema conceitual para a conversão de tipos relacionamento, indicado na *linha 2* do *Algoritmo 1*. Considera-se, como entrada deste algoritmo, um esquema EER e (opcionalmente) informações de carga que contêm o volume de dados do BD, um valor correspondente à *Frequência de Acesso Mínima* (FAM) e uma lista de valores que estabelecem a *Frequência de Acesso Geral* (FAG) para cada tipo entidade e tipo relacionamento do esquema EER.

Quando o processo de conversão considera informações de carga, os valores de FAM e FAG são utilizados pelas funções *converterHierarquias* e *converterRelacionamentos* (*linhas 1 e 2*). Na sequência do processo, atributos identificadores são criados para blocos raízes que, eventualmente, ainda não possuem atributos identificadores (*linha 4*).

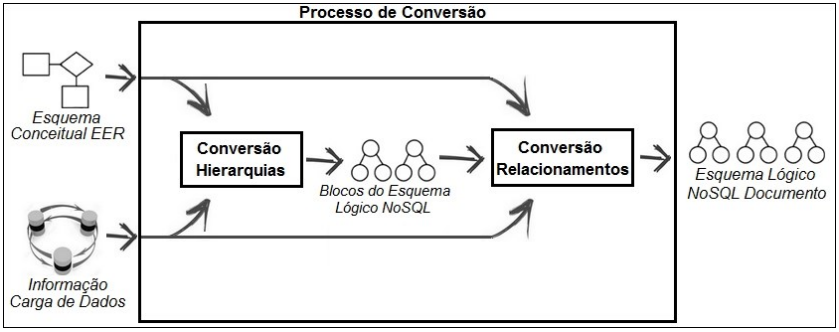
Algoritmo 1: *EER-NoSQL*

Entrada: Um esquema conceitual EER, *com volume de dados* – **EC**
Um valor de frequência de acesso mínima f_{min}*(EC)* - ***FAM***
Uma lista de frequência de acesso geral - ***FAG***
Saída: Um esquema Lógico NoSQL documento – **EL**

1 **H** ← ***converterHierarquias***(**EC,** *FAM***,** *FAG*);
2 **R** ← ***converterRelacionamentos***(**EC***,* **H,** *FAM***,** *FAG*);
3 **EL** ← lista de coleções (**H**, **R**);
4 Criar atributos identificadores para blocos raízes das coleções (**EL**).

Após a conversão dos tipos relacionamentos, os blocos raízes gerados pelo processo são finalmente definidos como *coleções* do esquema lógico (*linha 3*). No final do processo, a lista de coleções é retornada e o esquema lógico NoSQL documento final é obtido. A Figura 22 apresenta o processo de conversão EER-NoSQL documento.

Figura 22 - Processo de conversão EER-NoSQL documento.



Fonte: Desenvolvido pelo autor.

Com o objetivo de facilitar a compreensão, a apresentação das funções de conversão presentes no *Algoritmo 1* e dos exemplos de aplicação do processo estão agrupados de acordo com a utilização ou não de informações de carga: (i) *funções convencionais* (não consideram informações de carga) e (ii) *funções otimizadas* (consideram informações de carga).

4.3.1 Funções Convencionais

A conversão de tipos generalização e união é definida pela *Função 1* a seguir (*converterHierarquias*). A *linha 2* desta função é responsável por ordenar os tipos generalização e união que ocorrem em uma hierarquia de múltiplos níveis, caso exista. O objetivo desta ordenação é fazer com que a conversão de cada tipo generalização e união inicie com os tipos no nível mais inferior da hierarquia e prossiga até os tipos no topo da hierarquia (sentido *bottom-up*). Na abordagem *botton-up* os relacionamentos em níveis mais inferiores de uma hierarquia são convertidos primeiro, fazendo com que a conversão de um nível superior receba os níveis inferiores já tratados. Esta abordagem também trata melhor os casos de herança múltipla, se comparada à abordagem *top-down* (mais complexa). Observa-se que na ausência de hierarquias de múltiplos níveis, a ordem de conversão dos tipos generalização e união presentes no esquema EER é *indiferente*, e nos casos de herança múltipla com hierarquias no mesmo nível hierárquico, a ordem de conversão entre elas é *irrelevante* para a *Função 1*.

Após a ordenação, cada tipo generalização e união são convertidos de acordo com a ordem estabelecida. Na conversão de cada tipo, uma regra de conversão é escolhida para ser aplicada. O critério de escolha das regras é pela regra que gera *a menor porção de esquema* NoSQL documento e que é capaz de representar as restrições do fragmento EER que está sendo analisado. Desta forma, as Regras GSP, GSB e GHI são avaliadas nesta sequência para a conversão de um tipo generalização e as Regras USB, USP e UHI são avaliadas nesta sequência para a conversão de um tipo união.

As Regras GSP e GSB possuem algumas restrições para a sua aplicação, conforme ilustra a *Função 1*. A primeira restrição considerada para a aplicação da Regra GSP é a exigência de que nenhuma das subclasses esteja *marcada como convertida*. Uma entidade é dita marcada como convertida pela função quando ela passa a ser representada no modelo de conteúdo de um bloco que não está somente representando a entidade em questão. Essa situação ocorre justamente para as subclasses de uma hierarquia de generalização que foi processada pela Regra GSP (*linha 7* da *Função 1*), onde as subclasses passam a ser representadas como atributos em um bloco criado para representar a superclasse e todas as suas subclasses. Uma entidade também é marcada como convertida quando a entidade que a representa passa a ser representada como um bloco aninhado de algum outro bloco do esquema. Esta situação ocorre para tipos generalização convertidos pela Regra GHI (*linha 13* da *Função 1*), onde as subclasses são transformadas em blocos aninhados do bloco que representa a superclasse.

Função 1: *converterHierarquias*

Entrada: Um esquema conceitual EER – **EC**
Saída: Um conjunto de blocos do modelo lógico NoSQL documento – **HC**

1 **H** ← uma lista {gu_1, ..., gu_n} de tipos generalização e união de **EC**;
2 **H'** ← a lista ordenada de **H** (*ordenar **H** de forma que os primeiros tipos da lista sejam os tipos generalização e união presentes nos níveis mais inferiores de uma hierarquia de múltiplos níveis*);
3 **Para cada** $gu_i \in$ **H' faça**
4 **Se** (gu_i é um tipo generalização) **então**
5 **Se** (Não há subclasse marcada como convertida em gu_i) **e** (Não há tipos relacionamento associados às subclasses de gu_i) **e** (Não há subclasses com mais de uma superclasse) **e** (Não há subclasse definida como *bloco referenciado*) **então**
6 Aplicar a **Regra GSP**;
7 Marcar todas as subclasses de gu_i como convertidas;
8 **Senão**
9 **Se** (gu_i é total e disjunta) **e** (Não há subclasse marcada como convertida em gu_i) **e** (Não há tipos relacionamento associados à superclasse de gu_i) **então**
10 Aplicar a **Regra GSB**;
11 **Senão**
12 Aplicar a **Regra GHI**;
13 Marcar todas as subclasses de gu_i como convertidas;
14 **Fim**
15 **Fim**
16 **Senão**
17 **Se** (gu_i é total) **e** (Não há superclasse marcada como convertida em gu_i) **e** (Não há tipos relacionamento associados às superclasses de gu_i) **e** (Não há superclasses com mais de uma subclasse) **e** (*Não há superclasse definida como bloco referenciado*) **então**
18 Aplicar a **Regra USB**;
19 Marcar todas as superclasses de gu_i como convertidas;
20 **Senão**
21 **Se** (Não há superclasse marcada como convertidas em gu_i) **e** (Não há tipos relacionamento associados à subclasse de gu_i)
22 Aplicar a **Regra USP**;
23 **Senão**
24 Aplicar a **Regra UHI**;
25 Marcar todas as superclasses de gu_i como convertidas;
26 **fim**
27 **fim**
28 **fim**
29 **fim**
29 **Retorna H'**

Para a aplicação da Regra GSP é exigido que não existam tipos relacionamento associados às subclasses. Esta restrição é definida porque (assume-se que) a existência destes relacionamentos indica que a distinção entre superclasse é subclasses é *relevante* para a aplicação. A Regra GSP não pode ser aplicada quando alguma das subclasses possuir mais que uma superclasse. Isso ocorre porque, em casos de herança múltipla, a subclasse que possui mais que um pai deve gerar um relacionamento de referência para representar a especialização com pelo menos uma das suas superclasses. Neste caso, se a aplicação da Regra GSP fosse permitida, o relacionamento de referência seria estabelecido a partir de um bloco que representa não apenas a subclasse, mas também uma das suas superclasses e as demais subclasses desta superclasse.

A Regra GSP também não pode ser aplicada quando uma das subclasses estiver definida como *bloco referenciado*. Assume-se como bloco referenciado uma superclasse que anteriormente foi processada por outra hierarquia de generalização e que foi referenciada por um bloco externo. Esta restrição garante que o bloco referenciado se tornará um bloco raiz, evitando que este bloco referenciado se torne um bloco aninhado em uma conversão futura. Esta restrição é típica do modelo lógico proposto, baseado em agregados.

A Regra GSB só pode ser aplicada para generalizações totais e disjuntas, conforme justificado na Seção 4.2.2. Esta regra também não é aplicada quando existem subclasses marcadas como convertidas. Além disto, tipos relacionamento com a superclasse não são permitidos. Finalmente, quando as Regras GSP e GSB não podem ser aplicadas, a última opção de conversão é provida pela regra que gera a maior porção de esquema, ou seja, a Regra GHI. Observa-se que apenas a Regra GHI lida com entidades marcadas como convertidas. Isto ocorre porque entidades marcadas já estão sendo representadas no modelo de conteúdo de um bloco do esquema NoSQL documento, o que impede de representá-las novamente no modelo de conteúdo de um outro bloco pela aplicação das Regras GSP e GSB.

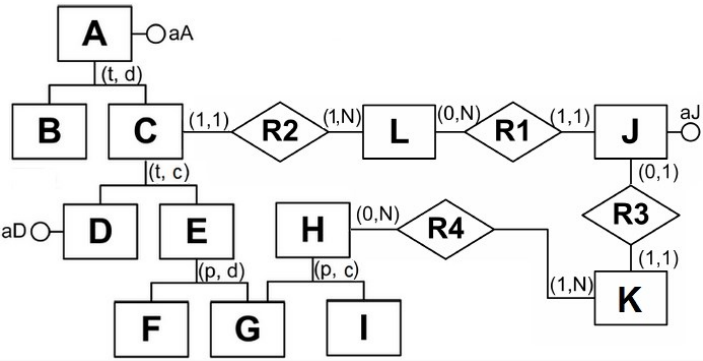
As linhas 16-28 da *Função 1* são destinadas à conversão de tipos união. As mesmas restrições aplicadas para as Regras GSP, GSB e GHI são semelhantemente aplicadas às Regras USB, USP e UHI. Porém, algumas variações ocorrem na aplicação da Regra USB e da Regra USP. Conforme justificado pela Seção 4.2.3, apenas uniões totais são convertidas pela Regra USB. Na Regra GSP, devido à restrição imposta pelos fragmentos gerados, verifica-se se o tipo generalização é total e disjunto. Diferentemente, na Regra USP, esta verificação não é realizada visto que, a princípio, tal definição pode ser aplicada tanto para uniões totais quanto para parciais.

Após a conversão de tipos generalização e união, a conversão de tipos relacionamento é executada. A função *converterRelacionamentos* (*Função 2*)

inicialmente ordena os tipos entidades do esquema conceitual. Esta ordenação tem por finalidade guiar a conversão de tipos relacionamento através de caminhos no esquema EER determinados pelos *fechamentos funcionais completos* (MOK; EMBLEY, 2006) das entidades do esquema.

Um fechamento funcional completo é determinado pela lista de entidades que podem ser alcançadas a partir de uma entidade de partida através de caminhos determinados pela participação (1,1) das entidades do caminho. Como exemplo, considere o esquema conceitual da Figura 23. O fechamento funcional completo da entidade L é determinado pelas entidades L, C, J e K visto que L tem associação (1,1) com a entidade C e com a entidade J, e a entidade J tem associação (1,1) com a entidade K.

Figura 23 - Um exemplo de esquema EER com hierarquias de múltiplos níveis.



Fonte: Adaptado de Schroeder e Mello (2008).

Os fechamentos funcionais completos de todas as entidades da Figura 23 são mostrados na Tabela 4. Após a identificação dos fechamentos funcionais completos das entidades do esquema conceitual, é gerada uma lista de entidades (LE) composta por todas as entidades que participam em mais de um fechamento funcional completo. Estas entidades devem estar ordenadas em LE de forma que as entidades que participam em um maior número de fechamentos funcionais completos apareçam primeiro. De acordo com a Tabela 4, LE = {K, J, C} visto que K participa em 3 fechamentos e J e C em 2 fechamentos. As entidades remanescentes são então adicionadas ao fim da lista LE, de forma que as entidades que possuem um maior número de relacionamentos aparecem primeiro. Seguindo o exemplo, a nova lista é obtida: LE = {K, J, C, L, H, A, B, C, D, E, F, G, I}.

O conceito de fechamento funcional completo foi definido por (MOK; EMBLEY, 2006). A abordagem de ordenação de entidades e conversão de

relacionamentos proposta pela *Função 2* constitui uma variação da metodologia para a geração de documentos compactos proposta por (MOK; EMBLEY, 2006). As variações incluem o uso de definições de conversão e relacionamentos de referência para evitar a redundância de dados nos blocos conformados ao esquema que está sendo produzido, bem como a consideração da participação mínima das entidades em um relacionamento.

Tabela 4 - Fechamentos funcionais completos das entidades do esquema EER da Figura 23.

Fechamentos Funcionais Completos	
A = { A }	G = { G }
B = { B }	H = { H }
C = { C }	I = { I }
D = { D }	J = { J, K }
E = { E }	L = { L, C, J, K }
F = { F }	K = { K }

Fonte: Desenvolvido pelo autor.

Na *Função 2*, a repetição iniciada na *linha 3* é executada até que todo tipo relacionamento do esquema EER tenha sido convertido. Cada laço desta estrutura de repetição gera uma porção do esquema lógico. A *entidade de partida* que gera o bloco de primeiro nível (da hierarquia) é selecionada nas *linhas 5* ou *7*. A primeira entidade é removida da lista de entidades ordenadas LE para que ela inicie a conversão de seus relacionamentos. Caso a lista LE esteja vazia, alguma das entidades de um relacionamento não marcado como convertido é escolhida.

A primeira entidade escolhida, a *entidade de partida*, é marcada como uma *entidade de continuação*. Algumas entidades de continuação são marcadas durante o processo para que a função percorra corretamente os fechamentos funcionais da *entidade de partida*. O laço de repetição iniciado na *linha 9* converte todos os relacionamentos envolvidos no fechamento da entidade de partida. Cada relacionamento é avaliado dentro de cada laço, e uma regra de conversão é escolhida para execução.

Antes da aplicação de alguma das regras, a função verifica se o tipo relacionamento selecionado é associado a alguma *entidade associativa* cujo relacionamento interno ainda não foi processado. Caso esta condição seja verdadeira (*linha 11* da *Função 2*), a função interrompe o laço e passa a processar o próximo tipo relacionamento. Este tratamento garante que a *entidade associativa* seja processada primeiramente, sendo os relacionamentos associados a ela processados em algum passo futuro.

Função 2: *converterRelacionamentos*

Entrada: Um esquema conceitual EER – **EC**
Blocos gerados pela função *converterHierarquias* - **HC**
Saída: Conjunto de blocos raízes gerados no modelo lógico NoSQL documento – **R**

1 **R** ← uma lista $\{r_1, ... r_n\}$ de tipos relacionamento de **EC**
2 **LE** ← a lista ordenada de entidades de **EC** (*ordenar através de caminhos determinados pelos fechamentos funcionais completos das entidades do esquema EER*);
3 **Para cada** $r_i \in$ **R** não convertido **faça**
4 **Se** (**LE** não está vazia) **então**
5 Remover a primeira entidade da lista **LE** e defini-la como uma *entidade de continuação*;
6 **Senão**
7 Definir como uma *entidade de continuação* uma entidade participante de um tipo relacionamento não convertido de ***R***;
8 **Fim**
9 **Enquanto** existir um $r_i \in$ ***R*** desmarcado envolvendo uma *entidade de continuação* **faça**
10 e_1 ← uma *entidade de continuação* envolvida em r_i;
11 **Se** (r_i está associado a uma *entidade associativa* cujo relacionamento interno ainda não foi processado) **então**
12 Ir ao próximo laço;
13 **Fim**
14 **Senão Se** (A máxima cardinalidade de r_i é 1:1) **e** (A participação de e_2 em r_i é (1,1)) **e** (e_2 não foi convertida) **então**
15 Aplicar a **Regra RBU**;
16 Marcar e_2 como convertida;
17 Definir e_2 como uma *entidade de continuação*;
18 **Fim**
19 **Senão Se** (r_i é binário) **e** (A participação de e_2 em r_i é (1,1)) **e** (e_2 não foi convertida) **e** (e_1 != e_2) **e** (e_2 não é bloco referenciado) **então**
20 Aplicar a **Regra RHI**;
21 Marcar e_2 como convertida;
22 Definir e_2 como uma *entidade de continuação*;
23 **Fim**
24 **Senão**
25 Aplicar a **Regra RRE**;
26 **Fim**
27 Marcar r_i como convertido;
28 **Fim**
29 **Fim**
30 **Retorna R**

A Regra RBU é aplicada para relacionamentos 1:1 quando a *outra entidade* e_2 possuir participação (1,1) e não estiver marcada como convertida. A Regra RHI é aplicada para relacionamentos binários quando e_2 possuir participação (1,1) e não estiver marcada como convertida. Ambas as regras, quando aplicadas, definem e_2 como uma *entidade de continuação* para dar sequência à navegação do fechamento funcional da *entidade de partida*.

Evita-se a geração de redundância de dados no esquema NoSQL documento marcando e_2 como convertida, impedindo que outro relacionamento defina-a como bloco aninhado ou atributo de outro bloco. A Regra RRE é aplicada para os casos não atendidos pelas Regras RBU e RHI.

Para a aplicação da Regra RHI é verificada a existência de um *auto-relacionamento*, verificando se as entidades envolvidas no relacionamento binário não são iguais. Ocorrendo este tipo de relacionamento, a aplicação da Regra RHI é descartada porque a aplicação desta regra geraria redundância ao representar um bloco como um bloco aninhado de um bloco que representa o próprio bloco. Para casos de *auto-relacionamento*, as Regras RBU e RRE podem ser aplicadas. A Regra RHI não pode ser aplicada para tipos relacionamento em que a entidade de participação (1,1) é um *bloco referenciado*. Assume-se como bloco referenciado uma entidade que anteriormente foi processada e que foi referenciada por um bloco externo. Esta restrição garante que o bloco referenciado se tornará um bloco raiz, evitando que este bloco referenciado se torne um bloco aninhado em uma conversão futura. Esta restrição é típica do modelo lógico proposto, baseado em agregados.

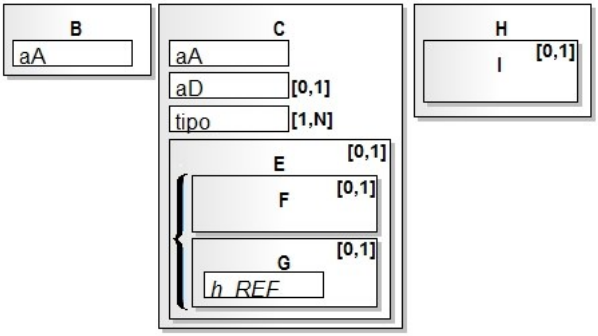
4.3.2 Exemplo de Aplicação do Processo Convencional

Para fins de exemplificação da aplicação da *Função 1*, considere o esquema conceitual da Figura 23 e os blocos gerados pela função *converterHierarquias* no modelo lógico NoSQL documento apresentados na Figura 24.

Em virtude da existência de uma hierarquia de múltiplos níveis, a conversão é iniciada pelo nível mais inferior da hierarquia. Visto que as generalizações envolvendo *E* e *H* estão no inferior da hierarquia, e ambas no mesmo nível hierárquico, a ordem de conversão entre elas é *irrelevante* para a função. Neste exemplo, escolheu-se começar pela generalização envolvendo *E* como superclasse. Por tratar-se de um caso de herança múltipla, a Regra GSP não pode ser aplicada, assim como a Regra GSB, pois se trata de uma generalização parcial e disjunta. Sendo assim, a Regra GHI é então aplicada e em seguida reaplicada para a conversão da generalização envolvendo *H*. Neste

último passo, um relacionamento de referência é gerado entre o bloco aninhado *G* e o bloco *H*. O bloco *H* é definido como *bloco referenciado*.

Figura 24 - Blocos gerados pela conversão dos tipos generalização do esquema da Figura 23.



Fonte: Desenvolvido pelo autor.

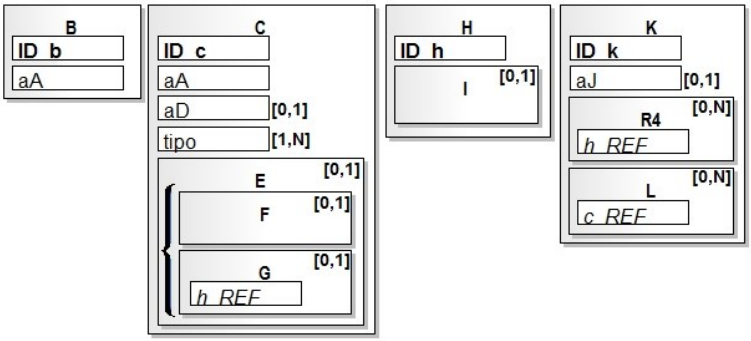
Os demais tipos generalização são processados seguindo a ordenação da hierarquia de múltiplos níveis. A generalização envolvendo a superclasse *C* é convertida pela Regra GSP. Neste caso, o atributo da subclasse *D* é representado como atributo no bloco *C* e a entidade *E* é transformada em um bloco aninhado de *C*, visto que um bloco para representar a entidade *E* já havia sido previamente gerado. A generalização envolvendo *A* no topo da hierarquia constitui a última conversão a ser processada. A Regra GSB é aplicada neste caso porque o relacionamento R2 associado à subclasse *C* impede a aplicação da Regra GSP.

Considerando o exemplo de esquema conceitual da Figura 23 e a lista LE obtida na Seção 4.3.1 (LE = {K, J, C, L, H, A, B, C, D, E, F, G, I}), a função *converterRelacionamentos* (*Função 2*) inicia definindo *K* como uma *entidade de continuação* na *linha 5*. Na sequência, todos os relacionamentos de *K* (R3, R4) são convertidos dentro do laço de repetição da *linha 9*. R3 é convertido pela Regra RBU e R4 pela Regra RRE (*H* é definido como *bloco referenciado*). Como todos os relacionamentos da entidade de continuação *K* foram processados e como a entidade *J* foi definida como *entidade de continuação* durante a conversão de R3, a função segue a conversão pelo relacionamento R1 de *J*. Neste caso a Regra RHI é aplicada e *L* é definido como um bloco aninhado de K, pois J passou a ser representado pelo modelo de conteúdo do bloco *K* após a conversão de R3. A entidade *L* é também marcada como *entidade de continuação*, e o processo prossegue com a

conversão do relacionamento R2. A Regra RRE é aplicada neste caso, e o relacionamento R2 é representado através do relacionamento de referência estabelecido a partir do bloco *L* para o bloco *C*. O bloco *C* é definido como *bloco referenciado*. Nesta última conversão, a criação de um relacionamento de referência foi necessária visto que *L* não poderia tornar-se um bloco aninhado de *C*, pois este bloco foi previamente marcado como convertido durante a conversão de R1.

A Figura 25 apresenta as coleções geradas, no modelo lógico NoSQL documento, ao término do processo de conversão do esquema conceitual EER da Figura 25.

Figura 25 - Coleções geradas pela conversão do esquema da Figura 23.



Fonte: Desenvolvido pelo autor.

4.3.3 Funções Otimizadas

A *Função 3* a seguir contém modificações (*linhas 2*, *5*, *9*, *17* e *21*) em relação à *Função 1* (*converterHierarquias*), para considerar as *informações de carga* na conversão de tipos generalização e união. Na *linha 2*, os tipos generalização e união continuam sendo ordenados para que o processo execute a conversão no sentido *bottom-up*, em casos de hierarquias de múltiplos níveis. Entretanto, quando há um caso de herança múltipla, os tipos generalização que aparecem primeiro na lista são aqueles que apresentam um *valor maior* para a *Frequência de Acesso Geral* (FAG) de suas superclasses. Isto significa que, em um caso de herança múltipla, a superclasse que é mais frequentemente acessada pela aplicação é transformada no bloco pai do bloco que representa uma subclasse que constitui uma especialização de mais de uma superclasse. Neste caso, as especializações restantes devem ser representadas por relacionamentos por referência (conforme definido na Regra GHI).

Função 3: *converterHierarquias*

Entrada: Um esquema conceitual EER com volume de dados – **EC**
Um valor de frequência de acesso mínima f_{min}(EC) - **FAM**
Uma lista de frequência de acesso geral - **FAG**
Saída: Um conjunto de blocos do modelo lógico NoSQL documento - **HC**

1 **H** ← uma lista {gu_1, ..., gu_n} de tipos generalização e união de **EC**
2 **H'** ← a lista ordenada de **H (***Ordenar **H** de forma que os primeiros tipos da lista sejam os tipos generalização e união presentes nos níveis mais inferiores de uma hierarquia de múltiplos níveis; Em casos de herança múltipla, os tipos cujas superclasses possuem uma FAG mais alta aparecem primeiro*);
3 **Para cada** $gu_i \in$ **H' faça**
4 **Se** (gu_i é um tipo generalização) **então**
5 **Se** (Não há subclasse marcada como convertida em gu_i) **e** (Não há subclasses com mais de uma superclasse) **e** (Não há subclasse definida como *bloco referenciado*) **e** (Não há subclasse s em gu_i com FAG(s) > FAM) **então**
6 Aplicar a **Regra GSP**;
7 Marcar todas as subclasses de gu_i como convertidas;
8 **Senão**
9 **Se** (gui é total e disjunta) **e** (Não há subclasse marcada em gu_i) **e** (FAG(superclasse)<FAM) **então**
10 Aplicar a **Regra GSB**;
11 **Senão**
12 Aplicar a **Regra GHI**;
13 Marcar todas as subclasses de gu_i como convertidas;
14 **fim**
15 **fim**
16 **Senão**
17 **Se** (gu_i é total) **e** (Não há superclasse marcada em gu_i) **e** (Não há superclasses com mais de uma subclasse) **e** (Não há superclasse definida como bloco referenciado) **e** (Não há superclasse s em gu_i com FAG(s) > FAM) **então**
18 Aplicar a **Regra USB**;
19 Marcar todas as superclasses de gu_i como convertidas;
20 **Senão**
21 **Se** (Não há superclasse marcada em gu_i) **e** (FAG(subclasse) < FAM**)**
22 Aplicar a **Regra USP**;
23 **Senão**
24 Aplicar a **Regra UHI**;
25 Marcar todas as superclasses de gu_i como convertidas;
26 **fim**
27 **fim**
28 **fim**
29 **fim**
30 **Retorna H'**

As outras duas alterações realizadas na *Função 3* estão relacionadas à aplicação das Regras GSP e GSB (*linhas 5* e *9*). Na aplicação da Regra GSP, ao invés de verificar se não existem tipos relacionamentos associados às subclasses, a *Função 3* verifica se a FAG das subclasses não é *maior* que a *Frequência de Acesso Mínima* (FAM).

No caso em que a FAG é superior à FAM para uma subclasse, a função assume que esta subclasse participa em operações frequentes e que, consequentemente, a distinção entre superclasse e subclasses é *relevante* e deve ser preservada, e, portanto, a Regra GSP não é utilizada. Na aplicação da Regra GSB, verifica-se se a FAG da superclasse é inferior à FAM. Caso a superclasse apresente uma FAG superior à FAM, a função assume que esta deve ser representada como um bloco e que, portanto, a aplicação da Regra GSB não é conveniente. Estas duas modificações repetem-se nas *linhas 17* e *21* (*Função 3*) para tipos união.

A *Função 4* contém modificações, em relação à *Função 2* (*converterRelacionamentos*), para considerar as *informações de carga* na conversão de tipos relacionamento. Ao invés de percorrer os fechamentos funcionais completos das entidades do esquema EER, esta função processa os tipos relacionamento segundo uma ordenação que garante que os relacionamentos com *maior* FAG são convertidos primeiro (*linha 2* da *Função 4*), e também garante que tipos relacionamentos envolvendo *entidades associativas* estejam dispostos após os tipos relacionamento internos das *entidades associativas*. Esta ordenação é estabelecida para dar prioridade para a conversão de tipos relacionamento que representam o maior impacto para a carga da aplicação.

Para a conversão de cada tipo relacionamento, a função determina qual das entidades do relacionamento gera o bloco que deve ser colocado no topo da hierarquia NoSQL documento gerada pela conversão de um relacionamento. Em se tratando de um relacionamento binário R no qual uma das entidades apresenta participação (1,1), a entidade escolhida deve ser *a outra entidade* envolvida em R (*linha 7* da *Função 4*). Para os demais casos, a entidade que possuir a maior FAG em R é definida como a entidade que gera o bloco no topo desta hierarquia que está sendo gerada (*linha 9* da *Função 4*). Para este último caso, assume-se que R é mais frequentemente acessado através da entidade que apresenta a maior FAG em R. Um exemplo da aplicação das funções otimizadas, que consideram *informações de carga de dados*, é apresentado na próxima seção.

Função 4: *converterRelacionamentos*

Entrada: Um esquema conceitual EER com volume de dados – **EC**
Blocos gerados pela função *converterHierarquias* – **HC**
Um valor de frequência de acesso mínima f_{min}(EC) - **FAM**
Uma lista de frequência de acesso geral - **FAG**
Saída: Conjunto de blocos raízes gerados no modelo lógico NoSQL documento - **R**

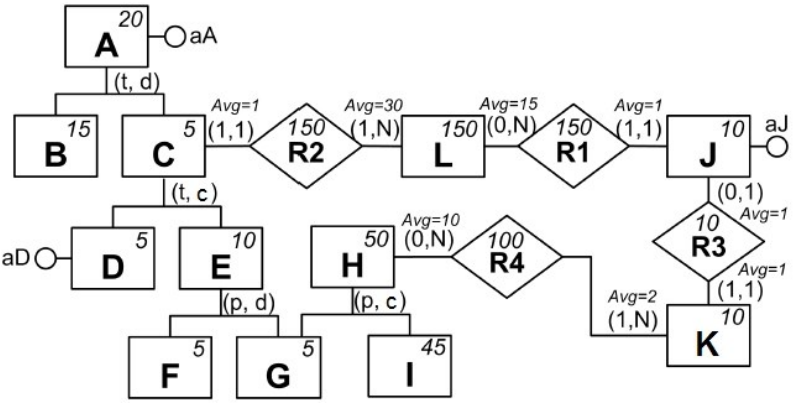
1 **R** ← uma lista $\{r_1, ... r_n\}$ de tipos relacionamento de **EC**
2 **R'** ← a lista ordenada de relacionamentos de **R** (*ordenar **R** de forma que os primeiros tipos da lista são os tipos relacionamento que possuem as maiores FAG. Garantir que tipos relacionamentos envolvendo entidades associativas estejam dispostos na lista após os tipos relacionamento internos das entidades associativas*);
3 **Para cada** $r_i \in$ **R'** não convertido **faça**
4 r_i ← o primeiro r_i não marcado como convertido de **R'**;
5 E ← o conjunto das entidades $\{e_1, ..., e_n\}$ relacionadas por r_i;
6 **Se** (r_i é binário) **e** (Há uma entidade $\mathbf{e_i} \in$ E com participação (1,1) em r_i) **então**
7 (***e₂*** é ***eᵢ***) **e** (***e₁*** é a outra entidade de r_i);
8 **Senão**
9 ***e₁*** é a entidade que possui a maior FAG em r_i;
10 **fim**
11 **Se** (A máxima cardinalidade de r_i é 1:1) **e** (A participação de ***e₂*** em r_i é (1,1)) **e** (***e₂*** não está marcada como convertida) **então**
12 Aplicar a **Regra RBU**;
13 Marcar ***e₂*** como convertida;
14 **Senão**
15 **Se** (r_i é binário) **e** (A participação de ***e₂*** em r_i é (1,1)) **e** (***e₂*** não está marcada como convertida) **e** (***e₂*** não é *bloco referenciado*) **e** (***e₁*** != ***e₂***) **então**
16 Aplicar a **Regra RHI**;
17 Marcar ***e₂*** como convertida;
18 **Senão**
19 Aplicar a **Regra RRE**;
20 **fim**
21 **fim**
22 Marcar r_i como convertido;
23 **fim**
24 **Retorna R'**

4.3.4 Exemplo de Aplicação do Processo Otimizado

A Figura 26 apresenta um esquema EER com informações sobre volume de dados. Assume-se que o volume de dados e as operações do BD que são representadas por este esquema foram providos por um usuário especialista. As operações são apresentadas pela Tabela 5 e a FAG de cada conceito do

esquema EER é dada pela Tabela 6. A FAT (soma das FAG de todos os conceitos do esquema) é 3825. Supõe-se que a FAM é dada sobre 0.15% da FAT, que é 7,18.

Figura 26 - Um esquema EER com informações sobre volume de dados.



Fonte: Adaptado de Schroeder e Mello (2008).

Tabela 5 - Operações do esquema EER da Figura 26.

O	***f(O)***	t_i	$fO(t_i)$
O1	100	J	100
		R3	100
		K	100
		R4	1000
		H	1000
O2	50	J	50
		R3	50
		K	50
O3	20	C	20
		R2	600
		L	600
O4	5	J	5
		R1	75
		L	75

Fonte: Desenvolvido pelo autor.

Tabela 6 - Frequência de acesso geral (FAG) dos conceitos do esquema EER da Figura 26.

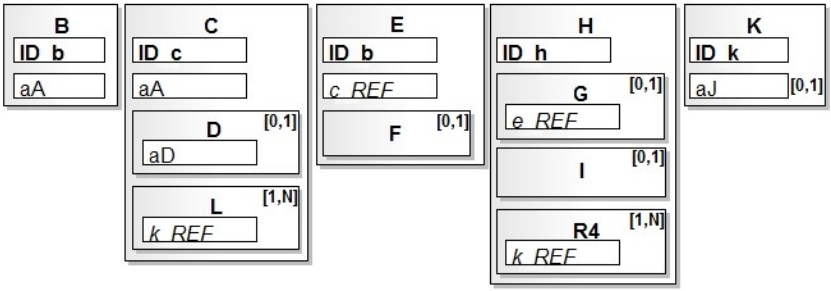
Conceito	*FAG*	*Conceito*	*FAG*
H	1000	E	0
L	675	F	0
J	155	G	0
K	150	I	0
A	0	R4	1000
B	0	R2	600
C	20	R3	150
D	0	R1	75

Fonte: Desenvolvido pelo autor.

É possível perceber que a FAG (Tabela 6) de alguns dos conceitos possui valor zero, pois os respectivos tipos entidade ou relacionamento não são acessados pelas operações consideradas (Tabela 5). Estes conceitos não estão envolvidos nas principais operações do BD, no entanto, visto que são representados no esquema conceitual, não podem ser desconsiderados pelo processo de conversão.

A conversão definida pelo *Algoritmo 1* (*EER-NoSQL*) considerando *informações de carga de dados* é exemplificada a seguir. O esquema lógico NoSQL documento gerado é apresentado na Figura 27.

Figura 27 - Coleções geradas pela conversão do esquema da Figura 26.



Fonte: Desenvolvido pelo autor.

Iniciando pela conversão de tipos generalização, a função continua a converter a hierarquia de múltiplos níveis no sentido *bottom-up*. Porém, para a herança múltipla envolvendo as hierarquias iniciadas por *E* e *H*, a função executa a conversão a partir da hierarquia cuja superclasse apresentar o *valor mais alto* para a medida de FAG. Neste caso, a conversão inicia pela hierarquia

da superclasse *H* visto que sua FAG (1000) é maior do que a FAG de *E* (0). Por tratar-se de um caso de herança múltipla, a Regra GSP não pode ser aplicada, assim como a Regra GSB, pois se trata de uma generalização parcial. A subclasse *G* é transformada em um bloco aninhado do bloco que representa a superclasse *H* através da aplicação da Regra GHI. A Regra GHI, em seguida, é reaplicada para a conversão da generalização envolvendo *E*. Neste último passo, um relacionamento de referência é gerado entre o bloco aninhado *G* e o bloco *E*. O bloco *E* é definido como *bloco referenciado*.

A definição da *Função 1* (que *não considera* informações de carga de dados) estabelece que a ordem de conversão de hierarquias envolvidas em casos de herança múltipla é *indiferente*. Neste caso, se *G* fosse transformada em um bloco aninhado de *E*, a operação *O1* deveria executar *junções por valor* caso desejasse obter valores de *G* associados pelo relacionamento de referência que deveria ser estabelecido entre *G* e *H*. Esta situação é evitada pelo esquema gerado pela *Função 3* (que *considera* informações de carga de dados) ao processar *G* como um bloco aninhado de *H*. Desta forma, um número menor de comparações é exigido para alcançar *G*, sendo ele um bloco aninhado de *H*.

A execução do *Algoritmo 1* prossegue na conversão dos demais tipos generalização do esquema EER. A generalização estabelecida pela superclasse *C* não pode ser convertida pela Regra GSP, mesmo que as subclasses *D* e *E* possuam uma FAG (0) inferior à FAM (7,18), pois há bloco representando uma das subclasses definido como bloco referenciado, neste caso, o bloco *E*. A Regra GSB também não pode ser aplicada, pois se trata de uma generalização parcial. Portanto, aplica-se a Regra GHI. Neste caso, *D* é transformado em um bloco aninhado de *C*, e um relacionamento de referência é gerado entre o bloco *E* e o bloco *C*. O bloco *C* é definido como bloco referenciado. Na conversão do tipo generalização estabelecido pela superclasse *A*, a Regra GSP não pode ser aplicada, visto que a subclasse *C* apresenta uma FAG (20) superior a FAM (7,18). Neste caso, a Regra GSB é aplicada, visto que nenhuma restrição é imposta a ela pelas informações de carga de dados consideradas.

A *Função 4* (que *considera* informações de carga de dados) estabelece uma ordem de conversão para tipos relacionamento em que são convertidos os relacionamentos que apresentam os *mais altos valores* para a FAG primeiro. Considerando as frequências de acesso geral dos conceitos apresentados pela Tabela 6, os tipos relacionamento R4, R2, R3 e R1 são convertidos nesta sequência. Na conversão de R4, a Regra RRE é aplicada e um relacionamento de referência é gerado a partir de *H*, visto que se trata de um relacionamento N:N. A entidade *H* é escolhida para estabelecer o relacionamento (*linha 9* da *Função 4*), visto que a FAG de *H* (1000) é superior à FAG de *K* (150).

Nos passos subsequentes, os relacionamentos R2, R3 e R1 são convertidos pelas Regras RHI, RBU e RRE, respectivamente. A prioridade

dada pelo algoritmo na conversão de R2 antes de R1 faz com que *L* se torne um bloco aninhado de *C* e, consequentemente, o relacionamento R1 seja estabelecido por referência. Esta escolha é realizada visto que a operação *O3*, que envolve R2, gera um maior número de acessos do que *O4* que envolve R1. Consequentemente, a representação de R2 através de referências resultaria em um número mais elevado de comparações na execução de *O3* do que na execução de *O4*. Por isso, R1 é representado por referência. Em outras palavras, *prioriza-se a representação hierárquica* no esquema NoSQL documento de relacionamentos conceituais que podem gerar uma maior frequência de acesso de acordo com a descrição das operações.

4.4 CONSIDERAÇÕES FINAIS

Este capítulo apresentou um modelo lógico, um conjunto de regras e um processo de conversão adotados pela abordagem de projeto proposta neste trabalho para a transformação de esquemas conceituais em esquemas lógicos NoSQL documento. O algoritmo proposto estabelece um processo de conversão automático que analisa as restrições sobre o esquema conceitual para determinar a regra de conversão a ser aplicada sobre cada fragmento. O algoritmo tem por finalidade gerar um esquema lógico NoSQL documento livre de redundâncias, com o menor volume de blocos possível e que representa integralmente um esquema conceitual EER. O processo de conversão automático considera todos os construtores do EER, incluindo *tipos união*, *entidades associativas*, *auto-relacionamentos* e *entidades fracas*.

Uma característica típica do modelo lógico proposto, baseado em agregados, é a definição de *blocos referenciados*. Assume-se como bloco referenciado uma entidade que anteriormente foi processada e que foi referenciada por um bloco externo. Nas regras GSP, USB e RHI são restrições que garantem que o respectivo bloco referenciado se tornará um bloco raiz, evitando que o bloco referenciado se torne um bloco aninhado em outra conversão posterior. As regras GHI, UHI e RRE definem blocos referenciados, visto que essas regras criam relacionamentos por referência.

A redundância de dados é evitada na geração de esquemas NoSQL documento em dois pontos principais da abordagem de conversão. Primeiro, não permitindo que um conceito seja representado por mais de um bloco do esquema NoSQL documento, utilizando, como controle nas funções, o conceito de *entidades marcadas como convertidas*. A Regra GSB, onde a superclasse é representada em todos os blocos que representam suas subclasses, não habilita a redundância de dados, pois, conforme estabelecido pela *Função 1*, esta regra só pode ser aplicada quando tratar-se de um tipo generalização total e disjunto. Isto ocorre porque a disjunção entre as

subclasses especifica que uma instância da superclasse só será representada por uma das subclasses.

O segundo tratamento aplicado pela abordagem para evitar a redundância está relacionado à representação de tipos relacionamento com cardinalidade máxima (N:N). Nestes casos, o processo de conversão cria um bloco para representar o relacionamento como um bloco aninhado de um dos blocos do relacionamento, e estabelece relacionamentos de referência entre o bloco do relacionamento com os demais blocos participantes. Isto ocorre porque representar uma das entidades como bloco aninhado do bloco que representa a outra entidade tornaria possível que os dados relativos a uma instância do bloco aninhado apareçam repetidas vezes para as instâncias do bloco pai, dada a participação máxima N de ambas as entidades.

O processo de conversão também considera *informações da carga de dados* estimada para o BD NoSQL que está sendo projetado. A carga do BD é estimada sobre um esquema EER e compreende informações relacionadas ao volume de dados esperado para os conceitos modelados, bem como as principais operações do BD. Uma metodologia para a modelagem destas informações é aplicada para gerar informações que são utilizadas pelo processo de conversão. O objetivo principal da utilização de informações da carga é permitir que os relacionamentos entre conceitos que serão frequentemente acessados pelas operações do BD sejam representados por relacionamentos hierárquicos no esquema NoSQL documento sempre que possível. Além disto, esta estratégia faz com que o número de comparações gerado por operações envolvendo referências seja minimizado em termos do número de acesso a blocos em um documento.

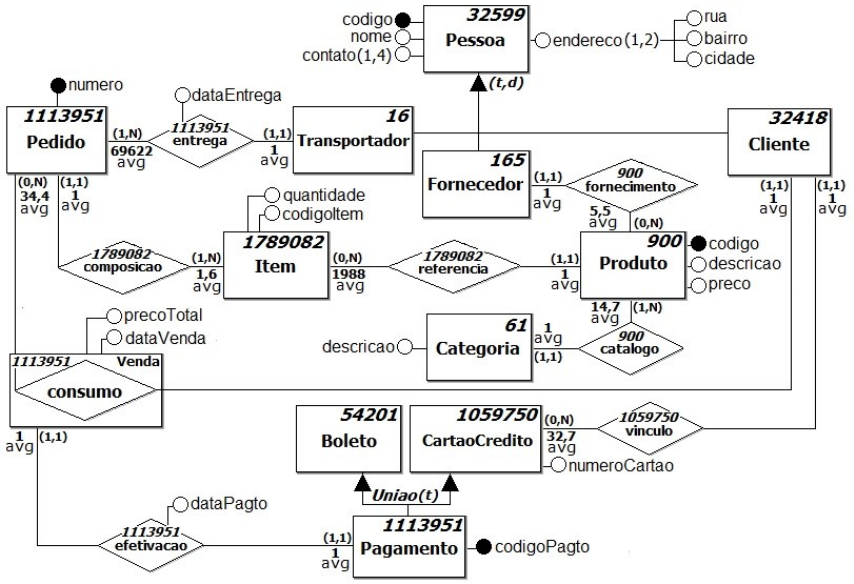
No próximo capítulo, um estudo de caso é apresentado para avaliar a abordagem proposta neste trabalho.

5 ESTUDO DE CASO

Um estudo de caso é apresentado para avaliar os esquemas lógicos NoSQL documento produzidos pela abordagem de conversão proposta por este trabalho. Um esquema EER representando o domínio de *e-commerce* foi utilizado como base para a produção dos esquemas NoSQL documento. As etapas de transformação do esquema EER em esquemas lógicos NoSQL documento são descritas detalhadamente neste capítulo, considerando as duas possibilidades de aplicação do processo de conversão da abordagem proposta, conforme apresentado no Capítulo 4: (i) conversão EER-NoSQL documento convencional; e (ii) conversão EER-NoSQL documento otimizada (baseada em análise da carga do BD).

O esquema EER utilizado para este estudo de caso é apresentado na Figura 28. Este esquema foi produzido através de um processo de *engenharia reversa* de uma aplicação real, com a finalidade de representar os principais processos de negócio relacionados ao setor de *e-commerce*, que são as transações relacionadas a pedidos de clientes. Alguns atributos foram omitidos para fins de simplificação do esquema EER.

Figura 28 - Esquema EER com volume de dados representando o domínio de e-commerce.



Fonte: Desenvolvido pelo autor.

Este capítulo está organizado em três seções. Na primeira seção é discutida a conversão do esquema EER em um esquema lógico NoSQL documento através do processo de conversão convencional (*não considera* informações de carga). Na seção seguinte, o processo de conversão baseado em análise de carga do BD é aplicado. Para a aplicação deste processo, uma previsão para o volume de dados e para as operações de um BD é considerada para o esquema EER fornecido. Na terceira seção, os esquemas lógicos NoSQL documento produzidos por ambos cenários são avaliados com relação à carga ocasionada pela execução das operações previstas sobre documentos NoSQL adequados a estes esquemas. Desta forma, a avaliação considera o projeto de implementação ao mapear os modelos lógicos para o modelo de implementação de um BD NoSQL documento, com o objetivo de avaliar o desempenho das operações sobre documentos adequados aos esquemas.

5.1 APLICAÇÃO DO PROCESSO CONVENCIONAL

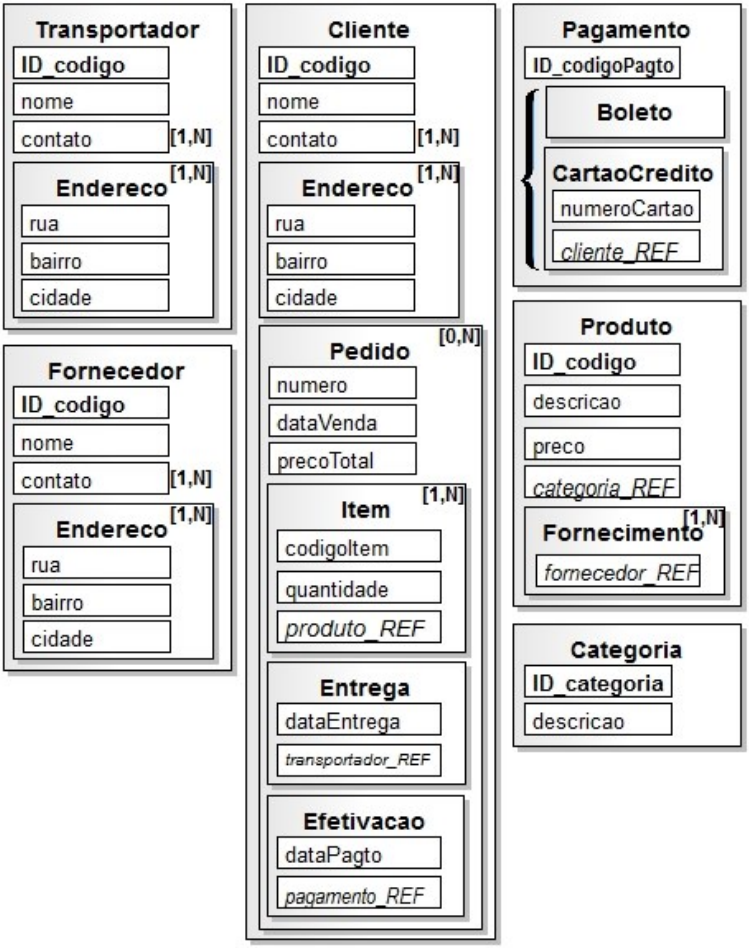
O *Algoritmo 1* (Seção 4.3) é aplicado ao esquema da Figura 28 e constata-se que não existem hierarquias de múltiplos níveis envolvendo as generalizações e o tipo união presente no esquema EER considerado. Consequentemente, a tarefa de ordenação executada pela *Função 1* não afeta a ordem de disposição dos tipos generalização e união. Optou-se por iniciar a conversão pelo tipo generalização envolvendo a superclasse *Pessoa* e as subclasses *Transportador*, *Fornecedor* e *Cliente*. O tipo relacionamento *entrega* envolvendo a subclasse *Transportador* impede que esta generalização seja convertida pela Regra GSP. Neste caso, a Regra GSB é adequada, e são criados blocos para representar as subclasses. A Figura 29 apresenta o esquema lógico NoSQL documento gerado pela conversão convencional.

O tipo união total envolvendo as superclasses *Boleto* e *CartaoCredito*, bem como a subclasse *Pagamento*, é convertido pela Regra UHI, dada a existência de tipos relacionamento envolvendo separadamente a superclasse *CartaoCredito* (*vinculo*) e a subclasse *Pagamento* (*efetivacao*). Desta forma, *Boleto* e *CartaoCredito* se tornam blocos aninhados do bloco criado para representar a subclasse *Pagamento*.

A conversão dos tipos relacionamento é iniciada pela ordenação das entidades do esquema EER. Conforme estabelecido pela *Função 2*, as entidades devem ser ordenadas de forma que apareçam primeiro aquelas que participam no maior número de *fechamentos funcionais completos*, sendo que este número deve ser maior que 1. A Tabela 7 apresenta os fechamentos funcionais completos computados para cada tipo entidade do esquema conceitual do estudo de caso. Segundo os dados desta tabela, a lista obtida pela ordenação proposta é: LE = {Cliente, Categoria, Pedido, Produto,

Transportador}. Na sequência, as entidades que foram descartadas (entidades que participam em apenas um fechamento funcional completo) são adicionadas à lista LE, de forma que as entidades associadas ao maior número de tipos relacionamento apareçam primeiro. Desta forma, a nova lista de entidades ordenadas é: LE = {Cliente, Categoria, Pedido, Produto, Transportador, Item, Fornecedor, CartaoCredito, Pagamento, Boleto, Pessoa}.

Figura 29 - Esquema Lógico NoSQL documento obtido pela aplicação do processo de conversão convencional.



Fonte: Desenvolvido pelo autor.

Obtendo-se a primeira entidade de LE, marca-se a entidade *Cliente* como uma *entidade de continuação*. Uma vez marcada como entidade de continuação, são convertidos todos os relacionamentos em que a entidade *Cliente* está envolvida. O relacionamento *consumo* (1:N) é então processado e a Regra RHI é aplicada, criando blocos para *Cliente* e *Pedido*, tornando *Pedido* um bloco aninhado de *Cliente* (em outras palavras, cria-se um relacionamento hierárquico). Após a aplicação da regra, a entidade *Pedido* é marcada como entidade de continuação. Na sequência, o relacionamento *vinculo* (1:N) é processado pela aplicação da Regra RRE, e um relacionamento de referencia é criado do bloco *CartaoCredito* para o bloco *Cliente*. O bloco *Cliente* é definido como bloco referenciado. Neste caso, a restrição para a aplicação da Regra RHI é justificada pelo fato de que a entidade com participação (1,1), *CartaoCredito*, já estava marcada. Como a entidade *Cliente* não possui mais nenhum relacionamento não processado, a conversão prossegue a partir da próxima entidade marcada como de continuação (*Pedido*).

Tabela 7 - Fechamentos funcionais completos das entidades do esquema EER da Figura 28.

Fechamentos Funcionais Completos
Boleto = {Boleto}
Cliente = {Cliente}
Categoria = {Categoria}
CartaoCredito = {CartaoCredito, Cliente}
Fornecedor = {Fornecedor}
Item = {Item, Produto, Categoria, Pedido}
Pagamento = {Pagamento}
Pedido = {Pedido, Cliente, Transportador}
Pessoa = {Pessoa}
Produto = {Produto, Categoria}
Transportador = {Transportador}

Fonte: Desenvolvido pelo autor.

O relacionamento *entrega* (1:N) é processado pela aplicação da Regra RRE, tornando o bloco criado para *entrega* um bloco aninhado do bloco *Pedido* e criando um relacionamento de referência do bloco *entrega* para o bloco *Transportador*. O bloco *Transportador* é definido como bloco referenciado. Neste caso, a restrição para a aplicação da Regra RHI é justificada pelo fato de que a entidade com participação (1,1), *Pedido*, já estava marcada como convertida (linha 19, *Função 2*). Na sequência, o

relacionamento *composicao* (1:N) é então processado e a Regra RHI é aplicada, criando um bloco para *Item* e o tornando bloco aninhado do bloco *Pedido*. Após a aplicação da regra, a entidade *Item* é marcada como entidade de continuação. Como a entidade *Pedido* não possui mais nenhum relacionamento não processado, a conversão prossegue a partir da próxima entidade marcada como de continuação (*Item*).

O relacionamento *referencia* (1:N) é processado pela aplicação da Regra RRE, criando um bloco para a entidade *Produto* e um relacionamento de referência do bloco *Item* para o bloco *Produto*. O bloco *Produto* é definido como bloco referenciado. Neste caso, a restrição para a aplicação da Regra RHI é justificada pelo fato de que a entidade com participação (1,1), *Item*, já estava marcada.

Neste ponto do processamento, não restam relacionamentos desmarcados envolvendo alguma das entidades marcadas como de continuação. Por este motivo, uma entidade que possui relacionamentos desmarcados é obtida do início da lista de entidades ordenadas (LE).

A primeira das entidades da lista que apresenta relacionamentos desmarcados é a entidade *Categoria*. O relacionamento processado é *catalogo* (1:N), através da aplicação da Regra RRE, criando um bloco para a entidade *Categoria* e o relacionamento de referência do bloco *Produto* para o bloco *Categoria*. O bloco *Categoria* é definido como bloco referenciado. Neste caso, a restrição para a aplicação da Regra RHI é justificada pelo fato de que o bloco *Produto* está definido como bloco referenciado.

Neste ponto do processamento, não restam relacionamentos desmarcados envolvendo alguma das entidades marcadas como de continuação. Por este motivo, uma entidade que possui relacionamentos desmarcados é obtida do início da lista de entidades ordenadas (LE). A primeira das entidades da lista que apresenta relacionamentos desmarcados é a entidade *Produto*. O relacionamento *fornecimento* (N:N) é processado através da aplicação da Regra RRE, criando o bloco *fornecimento* e o definindo como um bloco aninhado do bloco *Produto*, além da criação do relacionamento de referência do bloco *fornecimento* para o bloco *Fornecedor*. O bloco *Fornecedor* é definido como bloco referenciado. Neste caso, a restrição para a aplicação da Regra RHI é justificada pelo fato de que o bloco *Produto* está definido como bloco referenciado.

Neste ponto do processamento, não restam relacionamentos desmarcados envolvendo alguma das entidades marcadas como de continuação. Por este motivo, uma entidade que possui relacionamentos desmarcados é obtida do início da lista de entidades ordenadas (LE). A primeira das entidades da lista que apresenta relacionamentos desmarcados é a entidade *Pagamento*. O relacionamento *efetivacao* (1:1) é processado através

da aplicação da Regra RRE, criando o bloco *efetivacao* e o definindo como bloco aninhado do bloco *Pedido*, bem como o relacionamento de referência de *efetivacao* para o bloco *Pagamento*. O bloco *Pagamento* é definido como bloco referenciado. Neste caso, a restrição para a aplicação da Regra RBU é justificada pelo fato de que a entidade *Pagamento* foi definida pela *Função 2*, na linha 5, como entidade de continuação (e_1) e a entidade *Pedido* (e_2, neste caso) já se encontra marcada como convertida.

Conforme o detalhamento do processo fornecido nesta seção, todas as informações do esquema EER considerado foram representadas no esquema lógico NoSQL documento produzido. Além disto, o esquema lógico gerado não permite redundância de dados e ao mesmo tempo apresenta uma estrutura onde os blocos são relacionados por composições hierárquicas sempre que possível. Diante da variedade de construções do EER apresentadas pelo esquema conceitual da Figura 28, a cobertura provida pelas regras de conversão do *Algoritmo 1* (EER-NoSQL) forneceu soluções para o mapeamento de todos os construtores do EER presentes no esquema.

A seção seguinte apresenta a transformação do esquema EER através da abordagem de conversão baseada em análise da carga do BD (otimizada). Nesta próxima seção, mostra-se como a sequência de aplicação das regras de conversão pode ser alterada de forma a gerar uma estrutura NoSQL documento mais adequada à carga considerada para o BD. Na Seção 5.3, os esquemas produzidos pela abordagem de conversão (convencional e otimizada) são comparados quanto ao desempenho que proporcionam às operações mais frequentes do BD.

5.2 APLICAÇÃO DO PROCESSO OTIMIZADO

Um conjunto de informações referente à carga esperada para o BD que está sendo modelado neste estudo de caso foi fornecido para a aplicação do *Algoritmo 1*. O esquema EER provido pela Figura 28 apresenta o volume de dados esperado para o BD em termos do número de instâncias estimadas para cada conceito do esquema e da média de cardinalidade de cada conceito participante de um relacionamento. A Tabela 8 apresenta 20% das operações consideradas como as mais frequentes do BD. Considera-se que o conjunto destas 6 operações são responsáveis por gerar 80% da carga do BD, segundo a regra 80-20 (BATINI; CERI; NAVATHE, 1992). A frequência de acesso de cada conceito em uma operação denota a frequência de acesso diária àquele conceito pela operação considerada.

Tabela 8 - Operações do esquema EER da Figura 28.

O	***f(O)***	t_i	$fO(t_i)$
O1	1500	Cliente	1.500
		consumo	51.600
		Pedido	51.600
		composicao	82.560
		Item	82.560
		referencia	82.560
		Produto	82.560
		Subtotal:	434.940
O2	900	Pedido	900
		consumo	900
		Cliente	900
		efetivacao	900
		Pagamento	900
		Subtotal:	4.500
O3	450	Cliente	450
		consumo	15.480
		Pedido	15.480
		entrega	15.480
		Transportador	15.480
		Subtotal:	62.370
O4	300	Cliente	300
		consumo	10.320
		Pedido	10.320
		efetivacao	10.320
		Pagamento	10.320
		Subtotal:	41.580
O5	100	Produto	100
		referencia	198.800
		Item	198.800
		composicao	198.800
		Pedido	198.800
		Subtotal:	795.300
O6	100	Fornecedor	100
		fornecimento	550
		Produto	550
		catalogo	550
		Categoria	550
		Subtotal:	2.300
		Total:	**1.340.990**

Fonte: Desenvolvido pelo autor.

A *Frequência de Acesso Geral* (FAG) dos conceitos do esquema foi calculada e encontra-se listada na Tabela 9. A *Frequência de Acesso Total* (FAT) do esquema é obtida pela soma da FAG de todos os conceitos, chegando-se ao valor de 1.340.990 de acessos diários. Considerou-se que a *Frequência de Acesso Mínima* (FAM) deveria representar 0,8% da carga total do BD. Este valor percentual foi aplicado sobre a FAT do esquema que corresponde a 80% da carga, obtendo-se o valor 13.410 para a FAM do esquema.

Tabela 9 - Frequência de acesso geral (FAG) dos conceitos do esquema EER da Figura 28.

Conceito	FAG	Conceito	FAG
Item	281.360	composicao	281.360
Pedido	277.100	referencia	281.360
Produto	83.210	consumo	78.300
Transportador	15.480	entrega	15.480
Pagamento	11.220	efetivacao	11.220
Cliente	3.150	catalogo	550
Categoria	550	fornecimento	550
Fornecedor	100	vinculo	0
CartaoCredito	0		
Boleto	0		

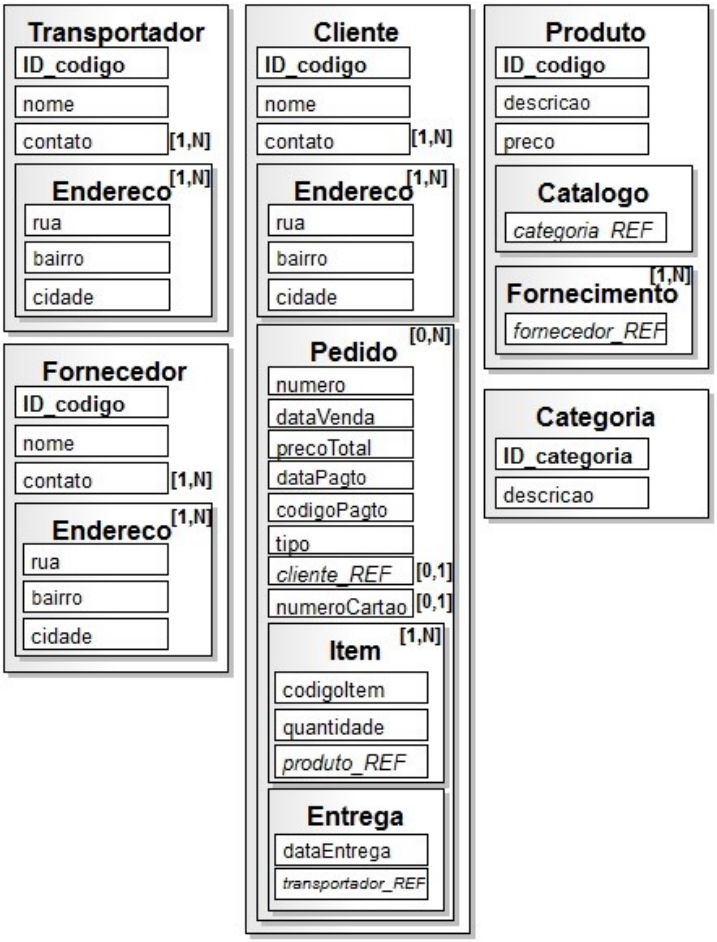
Fonte: Desenvolvido pelo autor.

Uma vez levantados os dados da carga prevista para o BD que está sendo modelado, inicia-se a aplicação da *Função 3*. Na ausência de hierarquias de múltiplos níveis, a ordem de conversão dos tipos generalização e união presentes no esquema EER é indiferente.

Iniciando-se pela generalização envolvendo a superclasse *Pessoa*, verifica-se que os requisitos para a aplicação da Regra GSP não são satisfeitos. Comparando-se com a aplicação da *Função 1*, que impediu a aplicação desta regra devido à existência de relacionamentos com as subclasses, a *Função 3* assume que a distinção entre as subclasses é *relevante* pois a FAG de *Transportador* (15.480) é superior à FAM (13.410) considerada. Quando a FAG das subclasses é inferior à FAM, assume-se que as subclasses são acessadas individualmente por operações que produzem uma frequência de acesso *irrelevante* (mínima) sobre estas subclasses. Portanto, a decisão de representar um tipo generalização em apenas um bloco mostra-se uma solução inadequada. Neste caso, aplica-se a Regra GSB.

A Figura 30 apresenta o esquema lógico NoSQL documento gerado pela aplicação do *Algoritmo 1* baseado em informações de carga (otimizado). O tipo união *Pagamento* é convertido pela Regra USB. A *Função 3* assume que a distinção entre as superclasses é *irrelevante* pois a FAG de *Boleto* e de *CartaoCredito* são inferiores à FAM (13.410) considerada.

Figura 30 - Esquema Lógico NoSQL documento obtido pelo processo baseado em análise de carga.



Fonte: Desenvolvido pelo autor.

Em geral, observa-se uma redução de blocos gerados para representar os tipos generalização e união em virtude da constatação baseada na análise da carga do BD de que a distinção entre superclasses e subclasses é desnecessária para os tipos analisados. Por outro lado, aumenta a quantidade de atributos opcionais nos blocos do esquema. No entanto, a ocorrência de atributos opcionais não caracteriza um problema para os BDs NoSQL documento, visto que esta estrutura parcial é uma característica já assumida para dados semiestruturados.

Iniciando a conversão de tipos relacionamento pela *Função 4*, uma lista de relacionamentos é obtida pela ordenação destes em ordem decrescente pelos valores da FAG de cada tipo relacionamento. Considerando os dados da Tabela 9, a lista de relacionamentos é dada por R' = {composicao, referencia, consumo, entrega, efetivacao, catalogo, fornecimento, vinculo}. É importante destacar que os tipos relacionamento que não estão presentes na tabela devem ser considerados nesta lista como os últimos relacionamentos a serem convertidos. A ordem estabelecida pela lista garante que os relacionamentos que geram o maior número de acessos diários são convertidos preferencialmente.

O primeiro relacionamento da lista, *composicao*, é convertido através da aplicação da Regra RHI. Nesta conversão, cria-se o bloco *Item* que é transformado em um bloco aninhado do bloco que representa a entidade *Pedido*. Na sequência, o relacionamento *referencia* é processado pela Regra RRE, e cria-se o bloco *Produto* e um relacionamento de referência do bloco *Item* para o bloco *Produto*. O bloco *Produto* é definido como bloco referenciado. Neste caso, a Regra RHI não pode ser aplicada, pois o bloco *Item* está marcado como convertido. O relacionamento *consumo* é o próximo da lista de conversão e é processado pela Regra RHI, tornando o bloco *Pedido* bloco aninhado do bloco *Cliente*.

Dando sequência à conversão dos relacionamentos da lista, o mapeamento do relacionamento *entrega* faz com que seja criado um relacionamento de referência a partir do bloco *Pedido*, visto que o bloco *Pedido* não pode ser transformado em um bloco aninhado do bloco *Transportador*, pois anteriormente foi processado como filho do bloco *Cliente*. Neste caso, aplica-se a Regra RRE. O bloco *Transportador* é marcado como bloco referenciado.

O relacionamento (1:1) envolvendo as entidades *Pagamento* e *Pedido* (*Venda*), *efetivação*, é convertido pela aplicação da Regra RBU, sendo os atributos da entidade *Pagamento* representados como atributos do bloco *Pedido*. Neste caso, como a entidade *Pedido* (277.100) possui maior FAG que *Pagamento* (11.220), a entidade *Pagamento* é definida como "e_2" pela *Função 4*, possibilitando a aplicação da Regra RBU. Na sequência, o relacionamento

catalogo é processado pela Regra RRE e o bloco *Categoria* é marcado como bloco referenciado. O relacionamento *fornecimento* é processado pela Regra RRE, criando um relacionamento de referência do bloco *Produto* para o bloco *Fornecedor*. O bloco *Fornecedor* é marcado como bloco referenciado. Neste relacionamento (N:N), a decisão de referenciar o bloco *Fornecedor* a partir do bloco *Produto* ocorreu pela verificação da maior FAG (*linha 9* da *Função 4*). O bloco *Fornecedor* é marcado como bloco referenciado. O último relacionamento a ser considerado é *vinculo*, sendo processado pela aplicação da Regra RRE. Neste caso, cria-se um relacionamento de referência do bloco *Pedido* (a entidade *Pagamento* foi fundida no bloco *Pedido*) para o bloco *Cliente*. O bloco *Cliente* é definido como bloco referenciado.

As principais diferenças entre os esquemas lógicos gerados pelas aplicações do *Algoritmo 1* são a quantidade de blocos e relacionamentos de referência presentes no esquema, além da estrutura de aninhamento dos blocos. A *Função 4* assume que relacionamentos com os mais altos valores para a FAG devem ter a preferência de conversão, justamente para que relacionamentos hierárquicos sejam gerados para representar os relacionamentos conceituais de maior impacto para a carga do BD. As consequências desta estratégia de conversão são avaliadas na próxima seção, mediante a execução das operações sobre documentos NoSQL adequados a estes esquemas lógicos NoSQL documento produzidos.

5.3 AVALIAÇÃO EXPERIMENTAL

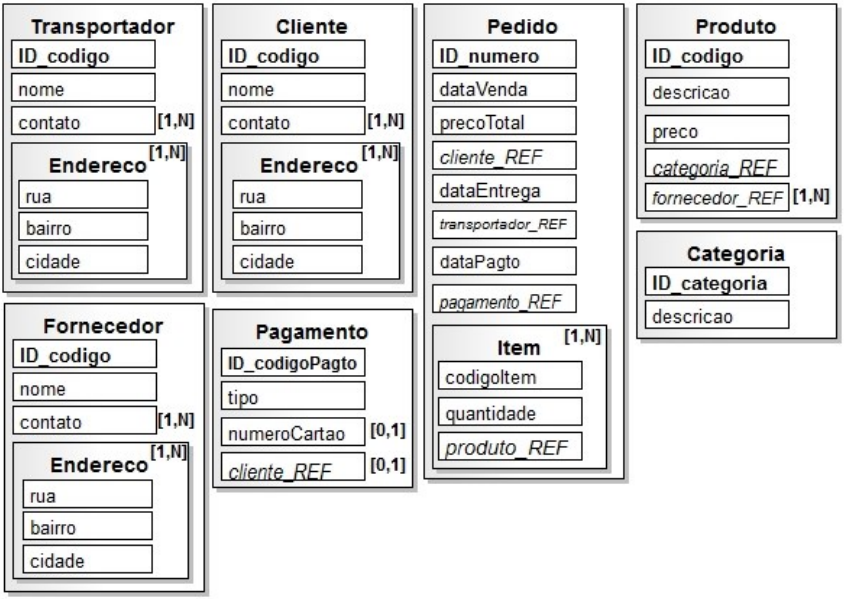
Os esquemas lógicos produzidos são avaliados, nesta seção, quanto ao desempenho das operações sobre documentos adequados aos esquemas apresentados. O desempenho das operações é medido pelo volume de acesso gerado pela operação sobre cada esquema lógico NoSQL documento, utilizando-se a mesma metodologia aplicada para mensurar o volume de acesso das operações sobre o esquema conceitual. Consultas foram executadas sobre documentos JSON (JSON, 2014) armazenados no BD NoSQL orientado a documentos MongoDB (MONGODB, 2014) a fim de evidenciar o volume de acesso gerado pelas operações sobre cada esquema.

5.3.1 Abordagem do Experimento

Com objetivo de avaliar os esquemas produzidos pela abordagem de conversão (*convencional* e *otimizada*), é avaliado um esquema lógico produzido artificialmente através de recomendações de fabricantes (MONGODB, 2015; JBOSS TEIID, 2015) para o mapeamento de esquemas físicos (BDRs) para esquemas NoSQL documento. O esquema físico utilizado

compreende os dados de uma aplicação real, compatível com o esquema conceitual EER da Figura 28. Para fins de identificação, o esquema lógico produzido é chamado de esquema *aplicação real* e é apresentado na Figura 31.

Figura 31 - Esquema Lógico NoSQL documento obtido por mapeamento de aplicação real.



Fonte: Desenvolvido pelo autor.

Observa-se que o esquema EER da Figura 28 foi obtido através de um processo de *engenharia reversa*, realizado manualmente, do BDR de uma aplicação real no domínio de *e-commerce*. Neste caso, o mapeamento realizado para a produção do esquema lógico *aplicação real* não utilizou o esquema conceitual EER da Figura 28. Ao invés disso, o esquema lógico foi obtido a partir do esquema físico do BDR através da aplicação de recomendações de fabricantes (MONGODB, 2015; JBOSS TEIID, 2015). Este esquema NoSQL gerado foi então devidamente traduzido para o modelo lógico NoSQL documento definido nesta dissertação.

Assim como os esquemas *convencional* e *otimizado*, o esquema *aplicação real* foi gerado de forma a não permitir redundância de dados e optando-se pela geração de *fragmento* mais enxuto, sempre que possível. Em outras palavras, durante o mapeamento realizado, ao aplicar as regras de

mapeamento recomendadas pelos fabricantes (mapeamento relacional-MongoDB), evitou-se gerar redundância de dados e optou-se pela geração de menor número de *coleções*. Para mapeamentos (1:1), por exemplo, (JBOSS TEIID, 2015) fornece duas opções: (i) *Fusão* (*merge*); e (ii) *Aninhamento* (*embeddable*). Nestes casos de mapeamentos optou-se pela *Fusão* (sempre que possível), pois esta opção gera o *fragmento* (coleção) mais enxuto. Em mapeamentos (1:N) e (N:N), quando se utiliza a opção *Aninhamento* pode ocorrer redundância de dados (JBOSS TEIID, 2015), que é então eliminada na geração do esquema *aplicação real*.

(MONGODB, 2015) sugere duas opções para o mapeamento de tabelas: (i) *Aninhamento* (*embedding*); e (ii) *Referência* (*referencing*). Os mapeamentos (1:1) e (1:N) são candidatos a utilizar a opção de *Aninhamento*. Já os mapeamentos (N:N) são candidatos a utilizar a opção *Referência*. No geral, as recomendações dos fabricantes considerados são semelhantes, prevalecendo, nas exceções, o conjunto de recomendações e regras de mapeamento de (JBOSS TEIID, 2015), que *utiliza* as diretrizes de (MONGODB, 2015).

A principal diferença do esquema *aplicação real*, apresentado na Figura 31, em relação aos esquemas *convencional* e *otimizado*, é a criação de coleções distintas para *Cliente* e *Pedido*, ocasionada pela aplicação das regras de mapeamento (1:N) definidas em (JBOSS TEIID, 2015). O esquema aplicação real apresenta maior número de coleções e menor número de blocos se comparado aos esquemas *convencional* e *otimizado*.

Cabe salientar aqui que o foco desta dissertação é a produção de esquemas NoSQL documento (adequada aos objetivos do projeto lógico de um BD) a partir de regras e processo de conversão detalhados e que consideram os construtores do modelo conceitual EER. Por este motivo, o mapeamento de *esquema físico de BDR* para esquema NoSQL documento não é detalhado nesta seção e também não foi adicionado ao Capítulo 3, destinado a trabalhos relacionados. Os detalhes das regras e recomendações de mapeamentos utilizados (para a geração do esquema *aplicação real*) estão disponíveis nos *sites* dos fabricantes (MONGODB, 2015; JBOSS TEIID, 2015). Uma abordagem recente (SCHREINER, 2016) detalha a questão do mapeamento de esquemas relacionais para esquemas adequados a BDs NoSQL.

Ressalta-se que o objetivo da produção do esquema *aplicação real* é possibilitar a comparação com os esquemas produzidos pelo processo de conversão proposto por este trabalho, visto que os trabalhos relacionados a projeto lógico de BDs NoSQL documento apresentados no Capítulo 3 (BUGIOTTI et al., 2014; JOVANOVIC; BENSON, 2013) não detalham processos de conversão entre esquemas conceituais e lógicos, dificultado o

desenvolvimento e a aplicação dos seus respectivos processos para a execução de comparações com este trabalho.

O volume de acesso gerado pelas operações sobre os três esquemas é mensurado considerando as informações de volume de dados estimado para os conceitos do esquema EER (Figura 28) e os conceitos e a frequência estimada para cada operação levantada pela modelagem de carga do BD. A mesma metodologia aplicada para produzir os dados da Tabela 8 é utilizada aqui, porém agora considerando a execução destas operações sobre a estrutura apresentada por cada um dos três esquemas lógicos NoSQL documento.

A Tabela 10 apresenta o volume de acesso gerado por cada esquema em cada conceito (entidade ou relacionamento) envolvido nas operações consideradas. Para exemplificar, considere o esquema lógico NoSQL documento da Figura 29 (convencional) e a operação *O4* da Tabela 10. Visto que a estimativa para a execução de *O4* é 300 vezes por dia, a quantidade de acessos sobre o bloco *Cliente* é considerada também 300 para o esquema convencional.

Em virtude de o conceito *efetivacao* ser representado como bloco aninhado do bloco *Pedido* que, por sua vez, é bloco aninhado de *Cliente*, é necessário executar *junções por estrutura*, devendo multiplicar-se a quantidade de acessos diários ao bloco *Cliente* (300) pelas respectivas cardinalidades médias associadas dos conceitos *Pedido* (no relacionamento *consumo* é 34,4) e *efetivação* (no relacionamento *efetivacao* é 1), resultando em 10.320 (300 * 34,4 * 1) acessos sobre o bloco *efetivacao*.

Porém, para obter-se os pagamentos relacionados a um *Cliente* é necessário comparar o identificador do bloco *Pagamento* com o atributo de referência *pagamento_REF* presente em todos os blocos *efetivacao*. Isto significa que para obter os pagamentos associados aos 300 clientes consultados por dia, é necessário comparar o valor dos 10.320 atributos de referência *pagamento_REF* com as 1.113.951 instâncias de *Pagamento*. Desta forma, esta *junção por valor* ocasiona 11.495.974.320 (10.320 * 1.113.951) acessos por dia sobre o bloco *Pagamento* do esquema convencional.

A mesma operação (*O4*) gera 10.320 acessos diários sobre *Pagamento* no esquema otimizado, devido ao fato de *Pagamento* estar representado como um bloco aninhado de *Pedido*. Neste caso, como a cardinalidade média para *Pedido* no relacionamento *consumo* é 34,4, para obter os pagamentos associados a um cliente é necessário executar uma *junção por estrutura* que retorna 34,4 blocos aninhados de *Pedido* em média. Considerando as 300 vezes, são contabilizados 10.320 acessos.

Tabela 10 - Volume de acesso produzido pelas operações sobre os esquemas lógicos NoSQL documento.

O	***f(O)***	t_i	$fO(t_i)$		
			Convencional	***Otimizado***	***Aplicação Real***
O1	1500	Cliente	1.500	1.500	1.500
		consumo	-	-	-
		Pedido	51.600	51.600	1.670.926.500
		composicao	-	-	-
		Item	82.560	82.560	2.673.482.400
		referencia	-	-	-
		Produto	74.304.000	74.304.000	2.406.134.160.000
		Subtotal:	**74.439.660**	**74.439.660**	**2.410.478.570.400**
O2	900	Pedido	900	900	900
		consumo	-	-	-
		Cliente	900	900	29.176.200
		efetivacao	900	-	-
		Pagamento	1.002.555.900	900	1.002.555.900
		Subtotal:	**1.002.558.600**	**2.700**	**1.031.733.000**
O3	450	Cliente	450	450	450
		consumo	-	-	-
		Pedido	15.480	13.185	501.277.950
		Entrega	15.480	13.185	-
		Transportador	247.680	210.960	8.020.447.200
		Subtotal:	**279.090**	**237.780**	**8.521.725.600**
O4	300	Cliente	300	300	300
		consumo	-	-	-
		Pedido	10.320	10.320	334.185.300
		efetivacao	10.320	-	-
		Pagamento	11.495.974.320	10.320	372.266.049.120.300
		Subtotal:	**11.495.995.260**	**20.940**	**372.266.383.305.900**
O5	100	Produto	100	100	100
		referencia	-	-	-
		Item	178.908.200	178.908.200	178.908.200
		composicao	-	-	-
		Pedido	178.908.200	178.908.200	178.908.200
		Subtotal:	**357.816.500**	**357.816.500**	**357.816.500**
O6	100	Fornecedor	100	100	100
		fornecimento	90.000	90.000	-
		Produto	90.000	90.000	90.000
		catalogo	-	90.000	-
		Categoria	5.490.000	5.490.000	5.490.000
		Subtotal:	**5.670.100**	**5.760.100**	**5.580.100**
Frequência de Acesso			**12.936.759.210**	**438.277.680**	**374.686.778.731.500**

Fonte: Desenvolvido pelo autor.

Consultas foram executadas sobre documentos JSON (JSON, 2014) armazenados no BD NoSQL orientado a documentos MongoDB (MONGODB, 2014), com a finalidade de verificar o desempenho das operações apresentadas na Tabela 10 sobre um BD NoSQL documento.As consultas foram produzidas de acordo com as operações consideradas neste estudo de caso. A versão *trial* da ferramenta *NoSQL Manager for MongoDB Professional 2.9.3.1* (NOSQL MANAGER, 2015) foi utilizada para a execução das consultas no *MongoDB-shell*. Além disso, a versão *trial* da ferramenta *MongoVUE 1.6.9* (MONGOVUE, 2015) foi utilizada para a criação de coleções e para a importação de documentos JSON nas respectivas coleções.

Os experimentos foram realizados em uma máquina com processador Intel Core i7 de 2.40GHz, equipada com 8GB de memória RAM e capacidade de 1TB de armazenamento em disco rígido (com 8GB de SSD). O sistema operacional utilizado foi o *Microsoft Windows 8.1 Pro 64 bits*. Todas as consultas executadas sobre o MongoDB foram processadas sobre as mesmas condições do ambiente, utilizando a configuração padrão do *MongoDB Server 3.0.3*.

A produção dos documentos JSON, adequados aos esquemas lógicos NoSQL documento (*convencional*, *otimizado* e *aplicação real*) foi concebida através do desenvolvimento de uma aplicação Java, utilizando a biblioteca *json-simple* (JSON-SIMPLE, 2015). A aplicação Java inicialmente produz objetos JSON, os quais representam diretamente as construções lógicas dos três esquemas analisados. Na sequência, uma visão (*view*) do BD *e-commerce* utilizado é percorrida e, utilizando os objetos JSON criados para cada esquema, são produzidos documentos JSON adequados aos esquemas lógicos analisados neste experimento. Para cada esquema foi gerado um conjunto de documentos JSON, produzindo um volume de dados de acordo com o que foi estimado na modelagem de carga do BD. Por exemplo, foram geradas 32.418 instâncias da coleção *Cliente*, 900 instâncias para a coleção *Produto*, e assim por diante. A Figura 2 (Seção 2.1.2) apresenta um exemplo de documento JSON gerado e que representa uma instância da coleção *Produto* do esquema lógico *otimizado* (Figura 30).

As consultas executadas em *MongoDB-shell* foram produzidas de acordo com o esquema de navegação entre os conceitos estabelecido para cada operação da Tabela 8. Os *Apêndices A*, *B* e *C* apresentam as consultas aplicadas sobre os documentos JSON conformados aos esquemas *convencional*, *otimizado* e *aplicação real*, respectivamente. Cada uma destas consultas foi executada sobre coleções equivalentes ao esquema atendido pela consulta em questão.

A Tabela 11 apresenta o tempo de resposta em segundos para a execução de cada uma das 6 operações sobre os documentos JSON

conformados aos três esquemas lógicos gerados. Para cada esquema é apresentado o tempo gasto em uma execução única (*Única*[25]) de uma consulta e o tempo de ocupação diário do sistema (*Acumulada*) para executar uma consulta considerando sua frequência diária. Por exemplo, para o esquema convencional, uma única consulta *O1* apresenta o tempo de resposta de 0,573 segundos. Considerando a frequência de 1500 vezes ao dia, o tempo diário de ocupação do sistema para executar esta operação (acumulada) é de 859,500 segundos (0,573 * 1500).

Tabela 11 - Tempo de resposta em segundos para a execução única e acumulada das operações sobre documentos dos esquemas apresentados.

O	***f(O)***	***Convencional***		***Otimizado***		***Aplicação Real***	
		Única	Acumulada	Única	Acumulada	Única	Acumulada
O1	1500	0,573	859,500	0,569	853,000	1,170	1.755,500
O2	900	1,018	915,900	0,750	675,300	0,510	459,300
O3	450	0,884	397,800	0,601	270,600	1,283	577,200
O4	300	0,776	232,800	0,143	42,900	1,189	356,600
O5	100	2,097	209,667	2,080	208,033	1,825	182,500
O6	100	0,471	47,100	0,524	52,400	0,518	51,800
Total:		**5,818**	**2.662,767**	**4,667**	**2.102,233**	**6,495**	**3.382,900**

Fonte: Desenvolvido pelo autor.

A Figura 32 apresenta o tempo de resposta em segundos para a execução de cada uma das 6 operações em forma gráfica. Na seção seguinte, os resultados apresentados pelas Tabelas 10 e 11 (e pela Figura 32) são devidamente discutidos.

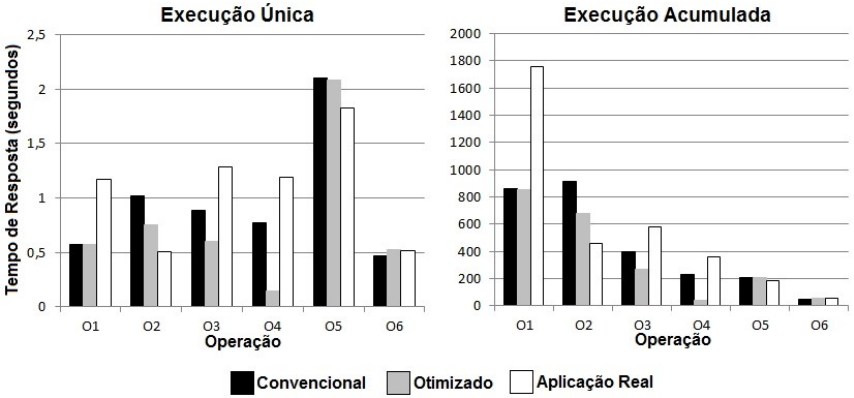
5.3.2 Considerações sobre os Resultados

Inicialmente, comparando a FAT de 1.340.990 estimada para o esquema conceitual EER com os valores da última linha da Tabela 10, verifica-se que a FAT para os esquemas lógicos NoSQL documento aumenta substancialmente. Este aumento se deve ao grande número de comparações ocasionadas por *junções por valor*. Junções por valo*r* são necessárias para recuperar relacionamentos conceituais representados por relacionamentos de referência nos esquemas lógicos NoSQL documento. Estes esquemas lógicos, conforme apresentado nas Figuras 29, 30 e 31, estabelecem um total de 6, 5 e 7 relacionamentos por referência, respectivamente. A FAT apresentada por cada

[25] A *execução única* refere-se à média de cinco execuções de cada operação.

um destes esquemas é diferenciada, e a maior diferença se dá entre a FAT do esquema otimizado (438.277.680 acessos diários) para a FAT do esquema aplicação real (374.686.778.731.500 acessos diários).

Figura 32 - Tempo de processamento das operações em segundos.



Fonte: Desenvolvido pelo autor.

Embora apresentando uma pequena diferença na quantidade de relacionamentos por referência, as estruturas de aninhamento apresentadas por estes esquemas produzem um volume de acessos diferenciado frente às operações consideradas. O esquema *otimizado* apresenta um aninhamento para o bloco *Pagamento* diferente do esquema *convencional*. Esta estrutura de aninhamento diferenciada foi produzida pela prioridade dada à entidade *Pedido* na conversão do relacionamento *efetivacao*, sendo que *Pedido* detinha maior FAG e tornou-se "$\boldsymbol{e_1}$" no relacionamento em questão, pela conversão baseada em informações de carga de dados (otimizada). Esta estrutura de aninhamento possibilita que apenas 900 acessos sejam necessários para alcançar as instâncias de pagamentos dos 900 pedidos através da operação *O2*. Para a mesma operação, os esquemas *convencional* e *aplicação real* geram 1.002.555.900 acessos para obter os pagamentos. Este número elevado de acessos é devido ao relacionamento de referência estabelecido entre *Pedido* e *Pagamento* para representar o relacionamento *efetivacao*. Ainda, considerando o aninhamento do bloco *Pagamento*, a operação *O4* gera um número de acessos bastante menor para o esquema *otimizado* (20.940), em relação aos demais esquemas (11.495.995.260 acessos para o esquema *convencional* e 372.266.383.305.900 acessos para o esquema *aplicação real*).

É importante esclarecer que índices não estão sendo considerados, uma vez que constituem decisões do *projeto físico* de um BD. No entanto, observa-

se que os dados produzidos pela modelagem de carga poderiam auxiliar na identificação de eventuais índices para melhorar o desempenho de consultas mais custosas.

A quantidade de acessos diários gerada pelo esquema *aplicação real* é substancialmente pior, em relação aos demais esquemas, nas operações O1, O3 e O4. Nestas operações, um maior número de *junções por valor* é executado para o esquema aplicação real. É importante ressaltar que este esquema foi obtido artificialmente através de boas práticas de migração de esquemas físicos (relacionais) para esquemas NoSQL documento, especificamente para o BD NoSQL MongoDB. Nesta migração, também foi evitado qualquer redundância de dados com o objetivo de manter esta característica semelhante aos demais esquemas produzidos (*convencional* e *otimizado*). Conforme comentado anteriormente, o esquema *aplicação real* possui um maior número de coleções e também de relacionamentos de referência, em relação aos demais esquemas. Metade das operações consideradas executa uma quantidade de acessos diários elevado, em relação aos demais esquemas, devido às *junções por valor* geradas por relacionamentos de referência.

Observa-se que a abordagem proposta por este trabalho pode dar suporte à migração e ao mapeamento de dados de BDRs para BDs NoSQL da categoria documento. Os documentos JSON conformados aos esquemas lógico NoSQL documento gerados pelo processo de conversão proposto (*convencional* e *otimizado*) produzem melhores resultados, em relação ao número de acessos diários e aos tempos de resposta, quando comparados ao esquema *aplicação real* considerado.

O tempo de resposta das consultas apresentado na Tabela 11 e na Figura 32 confirma a diferença no tempo de ocupação gerado pelas consultas executadas sobre o esquema *otimizado* frente aos demais esquemas. Observa-se que o somatório das execuções únicas de todas as consultas do esquema *convencional* (5,818 segundos) apresenta um valor próximo ao somatório de execução única sobre o esquema *otimizado* (4,667 segundos). Isso se deve ao equilíbrio gerado entre estes dois esquemas, pois o esquema otimizado apresenta um menor volume de acesso em algumas operações (*O2*, *O3*, *O4*) e o esquema convencional gera um menor volume de acessos em outras operações.

Considerando a frequência destas consultas, a execução acumulada total do esquema *otimizado* possui um percentual de 22% de ganho, em relação ao tempo total de execução, se comparado ao esquema *convencional*. Uma das operações responsáveis por esta diferença é *O2*. A execução das 900 vezes de *O1* gera um total de 915,900 segundos para o esquema *convencional* e 675,300 segundos para o esquema *otimizado*. Assim sendo, pode-se concluir que em termos de execuções isoladas das consultas, os esquemas *convencional* e *otimizado* apresentam um tempo de execução aproximado. Porém, considerado

o total de execuções diárias dessas operações, o tempo de ocupação do sistema considerando o esquema otimizado é reduzido.

Quanto ao esquema *aplicação real*, o valor total das execuções únicas já se mostra superior aos demais esquemas, totalizando 6,495 segundos. Este valor superior é decorrente, principalmente, da execução única das operações *O1*, *O3* e *O4*. Em decorrência disto e também devido à execução acumulada de *O1*, o total da execução acumulada por este esquema é também superior: 3.382,900 segundos. Observa-se que a proporção entre o total de acessos de cada operação apresentado pela Tabela 10 e a execução acumulada de uma operação na Tabela 11 não é sempre equivalente. Por exemplo, embora o total de acessos gerado por *O3* nos esquemas *convencional*, *otimizado e aplicação real* sejam 279.090, 237.780 e 8.521.725.600, respectivamente, a execução acumulada não sofre uma variação proporcional: 397,8 segundos para o esquema *convencional*, 270,6 segundos para o esquema *otimizado* e 577,2 segundos para o esquema *aplicação real*. Assume-se que estas diferenças podem ser ocasionadas por variações no tamanho dos documentos considerados e características específicas das instâncias utilizadas para cada documento em cada execução de consulta. Estas e outras variações que podem ocorrer não são avaliadas por este trabalho, visto que o objetivo do experimento realizado sobre o MongoDB (MONGODB, 2014) é apenas confirmar a diferença entre o volume de acesso gerado para os três esquemas, de acordo com a Tabela 10.

Embora as operações consideradas neste estudo de caso não representem um grande volume de acesso diário ao BD, é possível constatar, pela comparação realizada, que as informações da carga estimada para o BD guiam o processo de conversão para a geração de um esquema NoSQL documento mais otimizado. O tempo de ocupação deste esquema é consideravelmente inferior ao tempo produzido pelos demais esquemas. Observa-se que o ganho gerado pela aplicação da conversão baseada em análise de carga é *dependente* das otimizações possíveis de serem aplicadas sobre o esquema conceitual e as informações de carga fornecidas. A existência de diversas opções para o aninhamento dos conceitos do esquema e o indicativo de que determinados aninhamentos têm maior impacto nas operações, constituem dados que permitem que otimizações sejam realizadas em um esquema NoSQL documento. Algumas situações podem minimizar o efeito da aplicação do processo de conversão baseado em análise de carga, como, por exemplo, a inexistência de opções de aninhamento entre os conceitos do esquema, ou a indicação de que as opções de aninhamento apresentam um impacto equilibrado para a carga do BD. Tal processo tende a apresentar ganhos mais significativos para casos em que existem operações que visivelmente serão executadas com maior frequência, gerando grande carga para o BD.

6 CONCLUSÃO

Os BDs NoSQL são soluções adequadas à gestão de dados na Web, bem como na nuvem, e às necessidades do contexto do *Big Data*, como a escalabilidade elástica horizontal e o suporte a modelos flexíveis de armazenamento de dados. Um modelo de dados associado aos dados permite a definição de melhores estratégias para a persistência e manipulação desses dados no BD destino. Neste contexto, uma representação lógica baseada em *agregados* fornece suporte para escalabilidade e consistência, pois agregados são considerados unidades naturais para o particionamento horizontal e a manipulação de dados em ambientes distribuídos.

Este trabalho apresenta uma abordagem de conversão para o projeto lógico de BDs NoSQL da categoria documento, que estabelece um processo de conversão de esquemas conceituais definidos pelo modelo EER em esquemas lógicos documento definidos por um modelo lógico NoSQL documento. A proposta é constituída por regras de conversão capazes de transformar *cada um dos construtores do modelo conceitual EER* em uma representação lógica baseada em agregados e abstrata quanto ao modelo de implementação comum de BDs NoSQL da categoria documento.

Um processo de conversão EER-NoSQL automático é proposto neste trabalho, com a finalidade de ordenar a aplicação das regras na produção de um esquema NoSQL documento que *não habilita a redundância de dados* e, ao mesmo tempo, constitui uma representação bem estruturada das informações modeladas pelo projeto conceitual.

Esquemas NoSQL documento otimizados são produzidos levando em consideração informações relativas à principal carga estimada para o BD que está sendo modelado. As melhorias realizadas no esquema durante o processo de conversão tendem a produzir ganhos significativos no desempenho de consultas previstas durante o projeto lógico como o conjunto de operações que produzirão a maior carga do BD.

O estudo de caso apresentado demonstra o ganho obtido por documentos conformados a estes esquemas no tempo de ocupação diário do sistema para a execução das operações mais frequentes do BD, conforme apresentado no Capítulo 5. Este capítulo apresenta ainda as contribuições deste trabalho, juntamente com as limitações e sugestões para trabalhos futuros.

6.1 CONTRIBUIÇÕES

A principal contribuição deste trabalho é a proposta para produção de esquemas NoSQL documento adequada aos objetivos do projeto lógico de um BD, a partir de regras e processo de conversão detalhados e que consideram

integralmente os construtores de um modelo conceitual. Desta forma, acredita-se estar contribuindo para a área de gerência de dados na nuvem rumo à consolidação de uma metodologia para o projeto de BDs NoSQL da categoria documento. Dentre as demais contribuições deste trabalho destacam-se:

- A formalização de um modelo de dados lógico para BDs NoSQL documento. A escolha por uma representação lógica baseada em agregados, tendência dos trabalhos correlatos, justifica-se pelo fato de que eles apoiam os requisitos típicos dos BDs NoSQL, sendo considerados unidades de distribuição (*escalabilidade*) e consistência, e podem ser divididos em pequenos elementos de dados por questões de desempenho. Diferente de trabalhos relacionados, que utilizam agregados no nível lógico, o modelo lógico proposto destaca a categoria de BDs NoSQL documento e apresenta um modelo simples, com notação esquemática (*robustez*), e que representa integralmente um esquema conceitual;

- Um conjunto de regras de conversão que considera todos os construtores de um modelo conceitual para a produção de composições NoSQL documento equivalentes. Diferente de trabalhos relacionados, que propõem modelo lógico para NoSQL, o conjunto de regras proposto provê alternativas de conversão para *todos os construtores do modelo conceitual EER*, modelo esse consagrado para o projeto conceitual de BDs;

- Um processo *automático* para a conversão de um esquema EER em um esquema lógico NoSQL documento. O processo ordena a aplicação das regras na produção de um esquema NoSQL documento que não habilita a redundância de dados. Diferente de trabalhos relacionados, que propõem modelo lógico para NoSQL, a produção de esquemas NoSQL documento ocorre através de processo *automático*;

- A produção de esquemas NoSQL documento otimizados quanto à carga estimada para um BD. Os experimentos realizados demonstram que esquemas lógicos NoSQL documento gerados pelo processo de conversão baseado em análise da carga são capazes de responder com mais eficiência às principais operações do BD, gerando um *tempo de ocupação diário do sistema reduzido* se comparado à esquemas não otimizados quanto às operações mais frequentes previstas para um BD. Diferente de trabalhos relacionados, que propõem modelo lógico para

NoSQL, o processo de conversão considera informações de carga de dados estimadas para o BD, utilizadas para geração de estruturas NoSQL documento otimizadas;

- O suporte à migração de dados, a partir de SGBDs relacionais para BDs NoSQL da categoria documento. Os experimentos realizados demonstram que documentos conformados aos esquemas lógicos NoSQL documento gerados pela abordagem proposta produzem melhores resultados, em relação ao número de acessos diários e aos tempos de resposta, quando comparados ao esquema gerado seguindo boas práticas de fabricantes, a partir de mapeamento de SGBD relacional para um BD NoSQL documento. Desta forma, a aplicação de engenharia reversa, a partir de SGBDs relacionais, seguida da aplicação da abordagem proposta, a partir de esquema conceitual obtido através da engenharia reversa, tende a gerar esquemas lógicos mais eficientes;

- O processo de conversão proposto neste trabalho gerou um artigo publicado como *full paper* no *17th International Conference on Information Integration and Web-based Applications & Services* (iiWAS2015) – evento qualificado pela CAPES (LIMA; MELLO, 2015a). Uma versão melhorada será submetida em março deste ano, a convite, como um dos melhores artigos daquela edição do iiWAS2015, ao *International Journal of Web Information Systems* (IJWIS). Os estudos realizados para a realização deste trabalho também gerou um artigo publicado na *XI Escola Regional de Banco de Dados* (ERBD 2015) (LIMA; MELLO, 2015b).

6.2 LIMITAÇÕES E TRABALHOS FUTUROS

Considera-se como fator limitante deste trabalho a comparação com trabalhos correlatos no estudo de caso. As abordagens relacionadas (BUGIOTTI et al., 2014; JOVANOVIC; BENSON, 2013) não detalham processos de conversão entre esquemas conceituais e lógicos (documento), impossibilitando a implementação e aplicação dos respectivos processos, a fim de compará-los com o processo de conversão da abordagem proposta (*baselines*). Por esse motivo, um esquema lógico foi produzido através das recomendações de fabricantes (MONGODB, 2015; JBOSS TEIID, 2015) para mapeamento de dados de SGBDs relacionais para BDs NoSQL documento.

Ainda com relação ao estudo de caso, outra dificuldade encontrada para a sua realização diz respeito à obtenção de BDs NoSQL gratuitos, com grandes volumes de dados e que pudessem gerar esquemas conceituais (EER) adequados, incluindo a questão da quantidade de elementos do esquema conceitual. Esquemas conceituais com poucos elementos necessitam de uma quantidade mínima de conversões, além de gerar poucas opções de aninhamento entre os conceitos do esquema, e consequentemente, poderiam *minimizar os ganhos* obtidos com a aplicação da abordagem proposta.

Uma ferramenta que implementa a abordagem proposta nesta dissertação está em desenvolvimento no Grupo de Banco de Dados da UFSC e será integrada à ferramenta *brModeloNext* (MENNA; RAMOS; MELLO, 2011).

Considera-se como principal interesse na continuação deste trabalho o aprimoramento do processo de conversão baseado em análise de carga para atuar em procedimentos de *tuning* de BD. Isto inclui a avaliação da aplicação do processo em um maior volume de dados em um ambiente distribuído, e também a consideração de detalhes do projeto físico do BD NoSQL, incluindo a definição de índices. A abordagem proposta por este trabalho poderia ser utilizada para a sugestão de índices a serem criados para melhorar o acesso às porções da estrutura NoSQL documento mais frequentemente acessadas.

Também se deve considerar a realização de adaptações no modelo lógico proposto, a fim de estendê-lo, *mantendo a simplicidade*, para considerar os modelos de dados NoSQL *chave-valor* e *colunar*. Um modelo lógico baseado em agregados também favorece a utilização destes modelos. Este aprimoramento poderia habilitar uma maior gama de soluções NoSQL adequadas à abordagem proposta, tornando-a mais abrangente.

Em virtude do custo gerado por relacionamentos de referência presentes em esquemas NoSQL documento, considera-se a possibilidade de estender a abordagem para reduzir o número deste tipo de relacionamento em um esquema NoSQL documento, identificando o tipo de acesso gerado (consulta ou atualização) pelas operações estimadas durante a modelagem de carga do BD. Através deste levantamento seria possível eliminar o uso de relacionamentos de referência para os casos em que se identifica que o relacionamento é relevante para a carga do BD, porém os conceitos envolvidos são pouco acessados para fins de atualização. Referências são utilizadas pelo processo de conversão, proposto neste trabalho, para evitar a redundância de dados. Esta nova abordagem a ser proposta permitiria a redundância de dados em alguns pontos do esquema, porém as anomalias de atualização seriam controladas e minimizadas se os conceitos envolvidos nestes pontos de redundância são pouco acessados para fins de atualização.

Por fim, outro trabalho futuro relevante é a realização de uma avaliação estatística dos experimentos, além da realização de comparação estatística com os *baselines* (quando maiores detalhes destas abordagens estiverem disponíveis). Além disso, uma avaliação em termos de simplicidade do modelo lógico proposto é interessante e poderia ser conduzida com o auxílio de usuários especialistas.

REFERÊNCIAS

AGRAWAL, D. et al. Database management as a service: Challenges and opportunities. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, ICDE'09, p.1709–1716, 2009.

ATZENI, P.; BUGIOTTI, F.; ROSSI, L. Uniform access to non-relational database systems: The SOS platform. In: INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING, CAiSE'12, p. 160–174, 2012.

ATZENI, P. et al. The relational model is dead, SQL is dead, and I don't feel so good myself. SIGMOD Record, v.42, n.2, p.64–68, 2013.

BADIA, A.; LEMIRE, D. A call to arms: revisiting database design. SIGMOD Record, v. 40, n. 3, p.61–69, 2011.

BATINI, C.; CERI, S.; NAVATHE, S. B. **Conceptual Database Design**: An Entity Relationship Approach. Benjamin/Cummings, 1992.

BENSON, S. R. **Polymorphic Data Modeling**. Electronic Theses & Dissertations. 2014. Disponível em: <http://digitalcommons.georgiasouthern.edu/etd/1126>. Acesso em: 27/08/2015.

BIRD, L.; GOODCHILD, A.; HALPIN, T. Object Role Modeling and XML-Schema. In: INTERNATIONAL CONFERENCE ON CONCEPTUAL MODELING, ER 2000. p.661–705, 2000.

BISKUP, J.; MENZEL, R.; POLLE, T. Transforming an Entity-Relationship Schema into Object-Oriented Database Schemas. In: ADVANCES IN DATABASES AND INFORMATION SYSTEMS, ADBIS'95. p.109–136, 1995.

BREWER, E. A. Towards robust distributed systems. In: PRINCIPLES OF DISTRIBUTED COMPUTING, PODC. v.7, 2000.

BUGIOTTI, F. et al. Database Design for NoSQL Systems. In: INTERNATIONAL CONFERENCE ON CONCEPTUAL MODELING, ER 2014. p.223-231, 2014.

BUGIOTTI, F. et al. A logical approach to NoSQL databases. 2013. Disponível em: <http://cabibbo.dia.uniroma3.it/pub/noam.pdf>. Acesso em: 03/05/2014.

BUYYA, R. et al. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Gener. Comput. Syst., v.25, n.6, p. 599–616, 2009.

CATTELL, R. Scalable SQL and NoSQL Data Stores. SIGMOD Record, v. 39, n.4, p.12–27, 2010.

CHEBOTKO, A.; KASHLEV, A. ; LU, S. A Big Data Modeling Methodology for Apache Cassandra. In: INTERNATIONAL CONGRESS ON BIG DATA. p.238 - 245, IEEE, 2015.

CHEN, P. P. The entity-relationship model - toward a unified vies of data. In: ACM TRANSACTIONS ON DATABASE SYSTEMS, TODS 1976. v.1, n.1, p.9-36, 1976.

CHOI, M.; LIM, J.;JOO, K. Developing a Unified Design Methodology based on Extended Entity-Relationship Model for XML. In: INTERNATIONAL CONFERENCE ON COMPUTATIONAL SCIENCE, ICCS 2003. Springer Heidelberg, p. 920–929, 2003.

COUCHBASE. **Why NoSQL?**. 2014. Disponível em: <http://www.couchbase.com/nosql-resources/what-is-no-sql>. Acesso em: 11/12/2014.

ELMASRI, R.; JAMES, S.;KOURAMAJIAN, V. Automatic Class and Method Generation for Object-Oriented Databases. In: DEDUCTIVE AND OBJECT-ORIENTED DATABASES, DOOD'93, Springer LNCS 760. p. 395-414, 1993.

ELMASRI, R. et al. Conceptual Modeling for Customized XML Schemas. In: INTERNATIONAL CONFERENCE ON CONCEPTUAL MODELING, ER'02. p.429–443, 2002.

ELMASRI, R.; NAVATHE, S. B. **Fundamentals of Database Systems**. 6th ed., Pearson Addison Wesley, 2011.

EMBLEY, D. **Object Database Development**: Concepts and Principles. Addison Wesley, 1998.

EVANS, E. **Domain-Driven Design**: Tackling Complexity in the Heart of Software. Addison-Wesley, 2003.

FONG, J. Mapping Extended Entity-Relationship Model to Object Modeling Technique. SIGMOD Record, v. 24, n.3, p.18-22, 1995.

FONG, J. et al. Translating Relational Schema with Constraints into XML Schema. In: INTERNATIONAL JOURNAL OF SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING, v.16, n.2, p.201–243, 2006. Disponível em: <http://www.worldscientific.com/doi/abs/10.1142/S0218194006002744 >. Acesso em: 15/11/2014.

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

HELLAND, P. Life beyond distributed transactions: an apostate's opinion. In: CONFERENCE ON INNOVATIVE DATASYSTEMS RESEARCH, CIDR 2007. p.132–141, 2007.

HSIEH, D. **NoSQL data modeling**. Ebay tech blog. 2014. Disponível em: <http://www.ebaytechblog.com/2014/10/10/nosql-data-modeling>. Acesso em: 10/03/2015.

HEUSER, C. A. **Projeto de Banco de Dados**. 6. ed., Bookman, 2008.

JBOSS TEIID. **MongoDB Translator**. 2015. Disponível em: <https://docs.jboss.org/author/display/teiid812final/MongoDB+Translator>. Acesso em: 03/09/2015.

JOVANOVIC, V.; BENSON, S. Aggregate Data Modeling Style. In: SOUTHERN ASSOCIATION FOR INFORMATION SYSTEMS, SAIS 2013. p.70-75, 2013.

JSON. **Introducing JSON**. 2014. Disponível em <http://json.org/>. Acesso em 10/02/2014.

JSON-SIMPLE. **JSON.simple**: A simple Java toolkit for JSON. 2015. Disponível em: <https://code.google.com/p/json-simple/>. Acesso em 05/04/2015.

KAUR, K.; RANI, R. Modeling and Querying Data in NoSQL Databases. In: INTERNATIONAL CONFERENCE ON BIG DATA. p.1-7, IEEE, 2013.

KATSOV, I. **NoSQL data modeling techniques**. Highly Scalable Blog. 2012. Disponível em: <http://highlyscalable.wordpress.com/2012/03/01/nosql-datamodeling- techniques/>. Acesso em: 15/03/2014.

LIMA, C.; MELLO, R. S. A Workload-Driven Logical Design Approach for NoSQL Document Databases. In: INTERNATIONAL CONFERENCE ON INFORMATION INTEGRATION AND WEB-BASED APPLICATIONS AND SERVICES, iiWAS 2015. 2015**a**.

LIMA, C.; MELLO, R. S. Um Estudo sobre Modelagem Lógica para Bancos de Dados NoSQL. In: XI ESCOLA REGIONAL DE BANCO DE DADOS. 2015**b**. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/erbd/2015/002.pdf >. Acesso em: 15/07/2015.

MAGUIRE, J.; O'KELLY, P. **Does data modeling still matter, amid the market shift to XML, NoSQL, big data, and cloud?**. 2013. Disponível em:

<https://www.embarcadero.com/phocadownload/new-papers/okellywhitepaper-071513.pdf>. Acesso em 01/08/2015.

MCMURTRY, D. et al. **Data Access for Highly-Scalable Solutions**: Using SQL, NoSQL, and Polyglot Persistence. MICROSOFT, 2013. Disponível em: <http://www.microsoft.com/en-us/download/details.aspx?id=40327>. Acesso em: 05/06/2014.

MENNA, O. S.; RAMOS, L. A.; MELLO, R S. BrModeloNext: Nova Versão de uma Ferramenta para Modelagem de Bancos de Dados Relacionais. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, SBBD 2011, Florianópolis, 2011.

MOHAN, C. History repeats itself: sensible and NonsenSQL aspects of the NoSQL hoopla. In: INTERNATIONAL CONFERENCE ON EXTENDING DATABASE TECHNOLOGY EDBT'13. p.11–16, 2013.

MOK, W. Y.; EMBLEY, D. Generating compact redundancy-free xml documents from conceptual-model hypergraphs. In: IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING. v.18, n.8, p.1082–1096, 2006.

MONGODB. **MongoDB**. 2014. Disponível em: <http://www.mongodb.org>. Acesso em: 01/09/2014.

MONGODB. **RDBMS to MongoDB Migration Guide**. 2015. Disponível em: <https://www.mongodb.com/collateral/rdbms-mongodb-migration-guide>. Acesso em: 03/09/2015.

MONGOVUE. **MongoVUE**. 2015. Disponível em: <http://www.mongovue.com>. Acesso em: 10/05/2015.

NACHOUKI, J.; CHASTANG, M.P.; BRIAND, H. From Entity-Relationship Diagram to an Object-Oriented Database. In: INTERNATIONAL CONFERENCE ON CONCEPTUAL MODELING, ER 1991, p.459-482, 1991.

NARASIMHAN, B.; NAVATHE, S.; JAYARAMAN, S. On Mapping ER and Relational Models onto OO Schemas. In: INTERNATIONAL CONFERENCE ON CONCEPTUAL MODELING, ER'93, Springer LNCS 823. p.402–413, 1993.

NOSQL MANAGER. **NoSQL Manager for MongoDB GUI tool** . 2015. Disponível em: <http://www.mongodbmanager.com>. Acesso em: 10/05/2015.

SADALAGE, P. J.; FOWLER, M. J. **NoSQL Distilled**. Addison-Wesley, 2013.

SILVA, C. A. R. F. O. **Data Modeling with NoSQL**: How, When and Why. Dissertação (Mestrado) – Engenharia da Computação e Informática – Faculdade de Engenharia da Universidade do Porto. 2011.

SOUSA, F. R. C. et al. Gerenciamento de Dados em Nuvem: Conceitos, Sistemas e Desafios. 2010. Disponível em: <http://www.lia.ufc.br/~flavio/papers/sbbd2010.pdf>. Acesso em 01/09/2014.

SCHREINER, G. A. **SQLtoKeyNoSQL**: A Layer for Relational to Key-based NoSQL Database Mapping. 2016. Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, Centro Tecnológico, Universidade Federal de Santa Catarina. Florianópolis, 2016.

SCHROEDER, R.; MELLO, R. S. Improving Query Performance on XML Documents: A Workload-Driven Design Approach. In: ACM SYMPOSIUM ON DOCUMENT ENGINEERING, DocEng 2008. p.177-186, 2008.

SCHROEDER, R. **Uma Abordagem para Projeto Lógico de Banco de Dados XML baseada em Informações de Carga de Dados**. 2008. Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, Centro Tecnológico, Universidade Federal de Santa Catarina. Florianópolis, 2008.

SCHROEDER, R.; DUARTE, D.; MELLO, R. S. A workload-aware approach for optimizing the XML schema design trade-off. In: INTERNATIONAL CONFERENCE ON INFORMATION INTEGRATION AND WEB-BASED APPLICATIONS AND SERVICES, iiWAS 2011. p.12-19, 2011.

TIWARI, S. **Professional NoSQL**. John Wiley & Sons, 2011.

VERNON, V. **Implementing Domain-Driven Design**. Addison-Wesley, 2013.

VIEIRA, M. R. et al. Bancos de Dados NoSQL: Conceitos, Ferramentas, Linguagens e Estudos de Casos no Contexto de Big Data. In: SIMPÓSIO BRASILEIRO DE BANCOS DE DADOS, SBBD 2012, São Paulo, 2012. Disponível em: <http://data.ime.usp.br/sbbd2012/artigos/pdfs/sbbd_min_01.pdf>. Acesso em: 01/09/2014.

APÊNDICE A – Consultas Esquema Convencional

Quadro 1 – Consultas *MongoDB-shell* utilizadas sobre documentos JSON do esquema lógico *convencional*.

Operação	Script *MongoDB-shell*

O1	var aux;
	var clienteCursor = db.Cliente.find({ nome : 'CLAUDIO DE LIMA' },{
	nome:1, "Pedido.Item.produto_id":1, "Pedido.numero":1 });
	var cliente = clienteCursor.hasNext() ? clienteCursor.next() : null;
	var PedidoArray = cliente.Pedido;
	var ItemArray;
	var ProdutoArray = [];
	for (i = 0; i < PedidoArray.length; i++) {
	aux = PedidoArray[i];
	ItemArray = aux.Item;
	for (j = 0; j < ItemArray.length; j++) {
	aux = ItemArray[j];
	ProdutoArray.push(aux.produto_id);
	}
	}
	PedidoArray;
	db.Produto.find({ '_id':{$in:ProdutoArray} } , { descricao:1 });

O2	var aux;
	var clienteCursor = db.Cliente.find({ 'Pedido.numero' : 1066884},{ nome:1,
	Pedido:{$elemMatch:{numero: 1066884}} });
	var cliente = clienteCursor.hasNext() ? clienteCursor.next() : null;
	var PedidoArray = cliente.Pedido;
	var PagamentoArray = [];
	for (i = 0; i < PedidoArray.length; i++) {
	aux = PedidoArray[i];
	pagamento = aux.Efetivacao;
	PagamentoArray.push(pagamento.pagamento_id);
	}
	cliente.nome;
	var pagamentoCursor = db.Pagamento.find({ '_id':{$in:PagamentoArray} } , {
	_id:0, "CartaoCredito.numeroCartao":1 });
	var pagamento = pagamentoCursor.hasNext() ? pagamentoCursor.next() : null;
	pagamento.CartaoCredito.numeroCartao;

O3	var aux;
	var clienteCursor = db.Cliente.find({ nome : 'CLAUDIO DE LIMA' },{
	nome:1, "Pedido.Entrega":1, "Pedido.numero":1 });
	var cliente = clienteCursor.hasNext() ? clienteCursor.next() : null;
	var PedidoArray = cliente.Pedido;
	var TransportadorArray = [];
	var EntregaArray = [];
	var entrega;
	for (i = 0; i < PedidoArray.length; i++) {
	aux = PedidoArray[i];
	entrega = aux.Entrega;

	TransportadorArray.push(entrega.transportador_id);
	EntregaArray.push(entrega.dataEntrega);
	}
	EntregaArray;
	db.Transportador.find({ '_id':{$in:TransportadorArray} } , { nome:1 });

O4	var aux;
	var clienteCursor = db.Cliente.find({ nome:{$regex:"CLAUDIO DE LIMA*"} },{ 'Pedido.Efetivacao.pagamento_id':1, 'Pedido.Efetivacao.dataPagto':1 });
	var cliente = clienteCursor.hasNext() ? clienteCursor.next() : null;
	var PedidoArray = cliente.Pedido;
	var PagamentoArray = [];
	var DataPagamentoArray = [];
	var pagamento;
	for (i = 0; i < PedidoArray.length; i++) {
	aux = PedidoArray[i];
	pagamento = aux.Efetivacao;
	PagamentoArray.push(pagamento.pagamento_id);
	DataPagamentoArray.push(pagamento.dataPagto);
	}
	DataPagamentoArray;
	db.Pagamento.find({ '_id':{$in:PagamentoArray} });

O5	var aux;
	var produtoCursor = db.Produto.find({ descricao:{$regex:"CAMISETA.*"} }, {descricao:1});
	var produtoArray = [];
	while (produtoCursor.hasNext()) {
	aux = produtoCursor.next();
	produtoArray.push(aux._id);
	}
	db.Cliente.find({ 'Pedido.Item.produto_id':{$in:produtoArray} }, { _id:0, "Pedido.numero":1 });

O6	var aux;
	var fornecedorCursor = db.Fornecedor.find({ nome:{$regex:"mario*"} });
	var fornecedorArray = [];
	while (fornecedorCursor.hasNext()) {
	aux = fornecedorCursor.next();
	fornecedorArray.push(aux._id);
	}
	var produtoCursor = db.Produto.find({
	'Fornecimento.fornecedor_id':{$in:fornecedorArray} } , { categoria_id:1 });
	var categoriaArray = [];
	while (produtoCursor.hasNext()) {
	aux = produtoCursor.next();
	categoriaArray.push(aux.categoria_id);
	}
	db.Categoria.find({ _id:{$in:categoriaArray} }, {descricao:1});

Fonte: Desenvolvido pelo autor

APÊNDICE B – Consultas Esquema Otimizado

Quadro 1 – Consultas *MongoDB-shell* utilizadas sobre documentos JSON do esquema lógico *otimizado*.

Operação	Script *MongoDB-shell*

O1	var aux;
	var clienteCursor = db.Cliente.find({ nome : 'CLAUDIO DE LIMA' },{
	nome:1, "Pedido.Item.produto_id":1, "Pedido.numero":1 });
	var cliente = clienteCursor.hasNext() ? clienteCursor.next() : null;
	var PedidoArray = cliente.Pedido;
	var ItemArray;
	var ProdutoArray = [];
	for (i = 0; i < PedidoArray.length; i++) {
	aux = PedidoArray[i];
	ItemArray = aux.Item;
	for (j = 0; j < ItemArray.length; j++) {
	aux = ItemArray[j];
	ProdutoArray.push(aux.produto_id);
	}
	}
	PedidoArray;
	db.Produto.find({ '_id':{$in:ProdutoArray} } , { descricao:1 });

O2	var clienteCursor = db.Cliente.find({ 'Pedido.numero' : 1066884},{ nome:1,
	Pedido:{$elemMatch:{numero: 1066884}} });
	var cliente = clienteCursor.hasNext() ? clienteCursor.next() : null;
	cliente.nome;
	var pedido = cliente.Pedido[0];
	pedido.numeroCartao;

O3	var aux;
	var clienteCursor = db.Cliente.find({ nome : 'CLAUDIO DE LIMA' },{
	nome:1, "Pedido.Entrega":1, "Pedido.numero":1 });
	var cliente = clienteCursor.hasNext() ? clienteCursor.next() : null;
	var PedidoArray = cliente.Pedido;
	var TransportadorArray = [];
	var EntregaArray = [];
	var entrega;
	for (i = 0; i < PedidoArray.length; i++) {
	aux = PedidoArray[i];
	entrega = aux.Entrega;
	TransportadorArray.push(entrega.transportador_id);
	EntregaArray.push(entrega.dataEntrega);
	}
	EntregaArray;
	db.Transportador.find({ '_id':{$in:TransportadorArray} } , { nome:1 });

O4	db.Cliente.find({ nome:{$regex:"CLAUDIO DE LIMA*"} }, { _id:0, 'Pedido.codigoPagto':1, 'Pedido.tipo':1, 'Pedido.numeroCartao':1, 'Pedido.dataPagto':1 });
O5	var aux; var produtoCursor = db.Produto.find({ descricao:{$regex:"CAMISETA.*"} },

	{descricao:1});
	var produtoArray = [];
	while (produtoCursor.hasNext()) {
	aux = produtoCursor.next();
	produtoArray.push(aux._id);
	}
	db.Cliente.find({ 'Pedido.Item.produto_id':{$in:produtoArray} }, { _id:0,
	"Pedido.numero":1 });

O6	var aux;
	var fornecedorCursor = db.Fornecedor.find({ nome:{$regex:"mario*"} });
	var fornecedorArray = [];
	while (fornecedorCursor.hasNext()) {
	aux = fornecedorCursor.next();
	fornecedorArray.push(aux._id);
	}
	var produtoCursor = db.Produto.find({
	'Fornecimento.fornecedor_id':{$in:fornecedorArray} } , {
	"Catalogo.categoria_id":1 });
	var categoriaArray = [];
	var catalogo;
	while (produtoCursor.hasNext()) {
	aux = produtoCursor.next();
	catalogo = aux.Catalogo;
	categoriaArray.push(catalogo.categoria_id);
	}
	db.Categoria.find({ _id:{$in:categoriaArray} }, {descricao:1});

Fonte: Desenvolvido pelo autor

APÊNDICE C – Consultas Esquema Aplicação Real

Quadro 1 – Consultas *MongoDB-shell* utilizadas sobre documentos JSON do esquema lógico *aplicação real*.

Operação	Script *MongoDB-shell*

O1	var aux;
	var clienteCursor = db.Cliente.find({ nome:{$regex:"CLAUDIO DE LIMA*"} },{ _id:1 });
	var cliente = clienteCursor.hasNext() ? clienteCursor.next() : null;
	var pedidoCursor = db.Pedido.find({ cliente_id : cliente._id },{ 'Item.produto_id':1 });
	var PedidoArray = [];
	var ItemArray;
	var ProdutoArray = [];
	while (pedidoCursor.hasNext()) {
	aux = pedidoCursor.next();
	PedidoArray.push(aux);
	ItemArray = aux.Item;
	for (j = 0; j < ItemArray.length; j++) {
	aux = ItemArray[j];
	ProdutoArray.push(aux.produto_id);
	}
	}
	PedidoArray;
	db.Produto.find({ '_id':{$in:ProdutoArray} } , { descricao:1 });

O2	var pedidoCursor = db.Pedido.find({ _id : 1066884},{ cliente_id:1, pagamento_id:1 });
	var pedido = pedidoCursor.hasNext() ? pedidoCursor.next() : null;
	var clienteCursor = db.Cliente.find({ _id: pedido.cliente_id },{ nome:1 });
	var cliente = clienteCursor.hasNext() ? clienteCursor.next() : null;
	cliente.nome;
	var pagamentoCursor = db.Pagamento.find({ '_id': pedido.pagamento_id } , { _id:0, numeroCartao:1 });
	var pagamento = pagamentoCursor.hasNext() ? pagamentoCursor.next() : null;
	pagamento.numeroCartao;

O3	var aux;
	var clienteCursor = db.Cliente.find({ nome : 'CLAUDIO DE LIMA' },{ nome:1 });
	var cliente = clienteCursor.hasNext() ? clienteCursor.next() : null;
	var pedidoCursor = db.Pedido.find({ cliente_id : cliente._id },{ transportador_id:1, dataEntrega:1 });
	var TransportadorArray = [];
	var EntregaArray = [];
	while (pedidoCursor.hasNext()) {
	aux = pedidoCursor.next();
	TransportadorArray.push(aux.transportador_id);
	EntregaArray.push(aux.dataEntrega);
	}
	EntregaArray;

	db.Transportador.find({ '_id':{$in:TransportadorArray} } , { nome:1 });

O4	var aux;
	var clienteCursor = db.Cliente.find({ nome:{$regex:"CLAUDIO DE LIMA*"} },{ _id:1 });
	var cliente = clienteCursor.hasNext() ? clienteCursor.next() : null;
	var pedidoCursor = db.Pedido.find({ cliente_id : cliente._id },{ dataPagto:1, pagamento_id:1 });
	var PagamentoArray = [];
	var DataPagamentoArray = [];
	while (pedidoCursor.hasNext()) {
	aux = pedidoCursor.next();
	PagamentoArray.push(aux.pagamento_id);
	DataPagamentoArray.push(aux.dataPagto);
	}
	DataPagamentoArray;
	db.Pagamento.find({ '_id':{$in:PagamentoArray} });

O5	var aux;
	var produtoCursor = db.Produto.find({ descricao:{$regex:"CAMISETA.*"} }, {descricao:1});
	var produtoArray = [];
	while (produtoCursor.hasNext()) {
	aux = produtoCursor.next();
	produtoArray.push(aux._id);
	}
	db.Pedido.find({ 'Item.produto_id':{$in:produtoArray} }, { _id:1 });

O6	var aux;
	var fornecedorCursor = db.Fornecedor.find({ nome:{$regex:"mario*"} });
	var fornecedorArray = [];
	while (fornecedorCursor.hasNext()) {
	aux = fornecedorCursor.next();
	fornecedorArray.push(aux._id);
	}
	var produtoCursor = db.Produto.find({ fornecedor_id:{$in:fornecedorArray} } , { categoria_id:1 });
	var categoriaArray = [];
	while (produtoCursor.hasNext()) {
	aux = produtoCursor.next();
	categoriaArray.push(aux.categoria_id);
	}
	db.Categoria.find({ _id:{$in:categoriaArray} }, {descricao:1});

Fonte: Desenvolvido pelo autor.