Luiza Helena da Silva Vieira

Desenvolvimento de um roteiro para migração de dados armazenados em bancos de dados relacionais para bancos de dados não-relacionais

São Luís - MA

Julho - 2017

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Núcleo Integrado de Bibliotecas/UFMA

Vieira, Luiza Helena da Silva.
Desenvolvimento de um roteiro para migração de dados armazenados em bancos de dados relacionais para bancos de dados não-relacionais / Luiza Helena da Silva Vieira. - 2017.
61 f.

Orientador(a): Simara Vieira da Rocha.
Monografia (Graduação) - Curso de Ciência da Computação, Universidade Federal do Maranhão, São Luis, 2017.

1. Bancos de Dados Não-Relacionais. 2. Bancos de Dados Relacionais. 3. Migração de Dados. 4. MongoDB. I. Rocha, Simara Vieira da. II. Título.

Luiza Helena da Silva Vieira

Desenvolvimento de um roteiro para migração de dados armazenados em bancos de dados relacionais para bancos de dados não-relacionais

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof[a]. Dr[a]. Simara Vieira da Rocha

São Luís - MA

Julho - 2017

Luiza Helena da Silva Vieira

Desenvolvimento de um roteiro para migração de dados armazenados em bancos de dados relacionais para bancos de dados não-relacionais

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Trabalho aprovado em: São Luís - MA, 18 de Julho de 2017

Profª. Drª. Simara Vieira da Rocha - UFMA
Orientador

Prof. Msc. Carlos Eduardo Portela Serra de Castro - UFMA
Examinador 1

Prof. Msc. Vandecia Rejane Monteiro Fernandes - UFMA
Examinador 2

São Luís - MA

Julho - 2017

Agradecimentos

A Deus, por ter me permitido chegar onde cheguei.

A mim mesma, por não ter vacilado e nem desistido diante de todos os desafios impostos.

Aos meus pais, Arnaldo e Odete, por todos os sacrifícios que fizeram por mim, todo o apoio incondicional, autonomia em decidir o meu futuro e terem acreditado em mim.

Ao meu irmão, Pedro Henrique, por estar sempre ao meu lado em todos os momentos bons e ruins.

À minha avó Rosa que sempre me coloca em suas orações.

Aos meus familiares que sempre me ajudaram e apoiaram, em especial minhas tias Helena e Mirinalva.

A todos os meus amigos e amigas que me permitiram fazer parte da vida deles, em especial George Douglas que sempre me ouviu e aconselhou, mesmo quando não fui uma boa amiga.

A todos os livros que li porque, se não fossem por eles, meu processo de escrita teria sido um desastre.

Ao Victor, por me mostrar que não há nada de errado em colocar seus objetivos profissionais acima de qualquer coisa.

À minha orientadora, professora Simara, pela paciência e ensinamentos que me foram repassados nos últimos meses.

Ao corpo docente do curso por todo conteúdo que foi passado em sala de aula.

E por último, mas não menos importante, agradeço ao Google e ao Wolfram pela ajuda dada desde o início do curso.

Resumo

Nos últimos anos, houve um aumento importante nas pesquisas sobre bancos de dados não-relacionais. Dentre elas, pesquisas sobre modelos de migração de dados entre banco de dados relacionais para banco de dados não-relacionais estão crescendo. O principal motivo desse crescimento é que os bancos de dados relacionais não conseguem atender uma grande demanda de dados gerados atualmente de modo eficiente, fazendo com que surja a necessidade da migração para os bancos de dados não-relacionais. Esse trabalho se propõe a servir de roteiro para auxiliar a migração de dados entre banco relacionais para bancos não-relacionais de forma confiável e mantendo a integridade dos dados. Para tanto, foi feita uma exemplificação através de um estudo de caso baseado em um domínio de um hotel, visando demonstrar como seria o passo-a-passo da migração de um sistema de banco de dados relacional para o sistema de banco de dados não-relacional MongoDB.

Palavras-chaves: Banco de Dados Não-Relacionais. Banco de Dados Relacionais. Migração de Dados. MongoDB.

Abstract

Recently, there has been a significant increase in researches on non-relational database. Among them, research on data migration between relational databases to non-relational databases are growing. The main reason for this growth is that relational databases can not meet a large demand for currently generated data efficiently, leading to the need to migrate to non-relational databases. This paper proposes to serve as a guide to help the migration of data between relational databases to non-relational databases, while maintaining data integrity. To do so, an exemplification was done through a case study based on a domain of a hotel, aiming to demonstrate how it would be the step-by-step migration from a relational database system to the MongoDB database non-relational system.

Keywords: Data Migration. MongoDB. Relational Databases. Non-relational Databases.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ACID	Atomicidade, Consistência, Isolamento, Durabilidade
BASE	*Basically Available, Soft-state, Eventually consistency*
BLOB	*Binary Large Object*
CQL	*Cassandra Query Language*
DBRef	Referência de banco de dados
Modelo ER	Modelo Entidade-Relacionamento
NoSQL	*Not only SQL*
OLAP	Processamento Analítico Online
SGBD	Sistemas Gerenciadores de Banco de Dados
SGBDR	Sistemas Gerenciadores de Banco de Dados Relacionais
SQL	*Structured Query Language*

Sumário

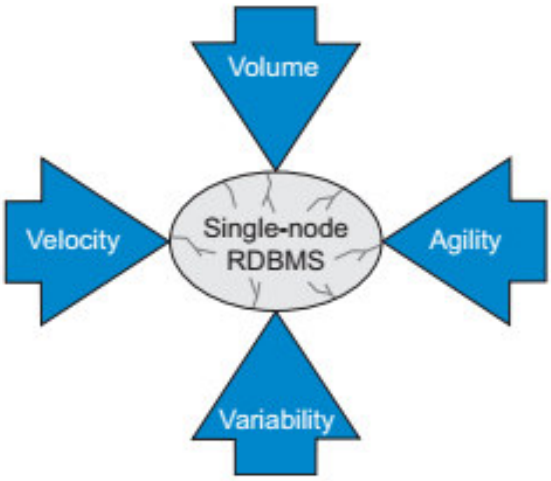
1 Introdução

Um grande volume variado de dados é produzido diariamente por diversos sistemas, dispositivos e aplicações. A problemática do tratamento desta grande quantidade de dados é caracterizado como *Big Data* e motiva o desenvolvimento de sistemas gerenciadores de banco de dados (SGBDs) diferentes dos tradicionais SGBDs relacionais (COSTA; VILAIN; MELLO, 2016).

Durante muito tempo, os sistemas gerenciadores de banco de dados relacionais (SGBDR), baseados no Modelo Relacional, atenderam os requisitos de diversas aplicações fornecendo simplicidade, robustez, desempenho e alta compatibilidade. Entretanto esses SGBDRs têm se mostrado ineficientes para manipular essa nova demanda de aplicações, focadas, em sua maioria, em grandes volumes de dados não estruturados (VALE; ROCHA, 2014). Nesse cenário, surgiram os sistemas de banco de dados não-relacionais ou sistemas de bancos de dados *NoSQL*. *NoSQL* significa *"Not only SQL"*, ou seja, que esses bancos não são baseados somente na linguagem *SQL*.

A Figura 1 mostra como a demanda de volume, velocidade, variação e agilidade tem o papel-chave nessa necessidade de uma possível migração. Segundo McCreary e Kelly (2014), volume e velocidade referem-se à capacidade do banco de dados lidar com grandes conjuntos de dados que chegam rapidamente; variabilidade refere-se a como diversos tipos de dados não se encaixam em tabelas estruturadas e a agilidade refere-se à rapidez com que uma organização responde à mudança de negócio.

Figura 1 – Manipulação de grande volume de dados em um SGBD relacional



Fonte: Adaptado de (MCCREARY; KELLY, 2014)

Cada uma dessas características aplica uma pressão ao banco relacional, fazendo

com que sua base se torna menos estável e, com o tempo, não atende mais às necessidades da organização. Portanto, aumenta a quantidade de aplicativos que necessitam migrar o banco de dados relacionais para banco de dados não-relacionais para atender a demanda de manipulação de grandes volumes de dados (MCCREARY; KELLY, 2014).

Se comparados com bancos de dados relacionais, os bancos de dados *NoSQL* fornecem uma abordagem mais eficiente ao armazenamento de dados, que inclui fácil escalabilidade e baixo custo de manutenção (LIANG; LIN; DING, 2015). Ainda dentre as principais características dos bancos *NoSQL*, destacam-se a facilidade de particionamento e replicação de dados (VALE; ROCHA, 2014).

Diversos SGBDs não-relacionais estão à disposição no mercado, sendo categorizados pelos seus modelos de dados, que são mais flexíveis em termos de representação de dados e com interfaces simples de acesso (COSTA; VILAIN; MELLO, 2016). Segundo Bugiotti et al. (2014), o desenvolvimento de metodologias de alto nível e ferramentas que suportam o projeto de banco de dados não-relacionais são necessários e existem diferentes alternativas na organização de dados nos bancos de dados *NoSQL*, com consequências significativas nos principais requisitos de qualidade, incluindo escalabilidade, desempenho e consistência

Sistemas com bancos relacionais podem ter interesse em migrar para um SGBD não-relacional por uma série de motivos, como a redução de custo com a administração de dados, maior escalabilidade, elasticidade e disponibilidade do banco de dados (COSTA; VILAIN; MELLO, 2016). Porém, ainda não há modelos e/ou ferramentas padrões designadas a executar essa migração entre bancos, apesar de existirem na literatura algumas soluções que visam adaptar os bancos relacionais para os não-relacionais, bem como proposta de migração automática (VALE; ROCHA, 2014).

Vale e Rocha (2014) pontuam alguns fatores para a dificuldade da migração de dados. Um deles é o volume de dados a ser migrado, visto que a decisão da migração parte da constatação que o SGBDR não vem atendendo às expectativas de desempenho. Outro fator é a necessidade de manter o banco novo semanticamente igual ao banco original, representando corretamente os relacionamentos existentes sem que nenhuma informação seja perdida ou distorcida.

Entrentato, a principal dificuldade da migração de dados se resume em descobrir as diferenças de modelos entre bancos de dados relacionais tradicionais e bancos de dados não-relacionais (LIANG; LIN; DING, 2015). O projeto em bancos não-relacionais é baseado em práticas e diretrizes que estão especificamente relacionadas ao sistema selecionado para ser utilizado, sem uma metodologia sistemática (BUGIOTTI et al., 2014). Por exemplo, no MongoDB os dados são armazenados em documentos, enquanto o Riak utiliza do armazenamento chave-valor. Logo, o projeto do banco e a modelagem dos dados para a migração serão diferentes, influenciados pela escolha do SGBD não-relacional a ser usado.

1.1 Objetivo geral

Desenvolver um roteiro de migração entre banco de dados relacionais para não-relacionais utilizando o MongoDB que, como resultado, mantenha a integridade dos dados e que seja confiável.

1.1.1 Objetivos específicos

- Analisar as principais características dos bancos de dados não-relacionais
- Investigar o MongoDB
- Propor um roteiro para migração de dados armazenados em um banco de dados relacional para um banco de dados não-relacional.
- Fazer um estudo de caso como forma de validação da proposta apresentada

1.2 Organização do trabalho

Essa monografia está dividida em quatro capítulos. O primeiro contém a introdução, objetivo geral e objetivos específicos. O segundo capítulo é formado pela fundamentação teórica necessária para a compreensão da teoria abordada. O terceiro capítulo é composto por um estudo de caso como forma de exemplificar o roteiro proposto, tendo uma comparação entre os dois SGBDs utilizados, ressaltando as principais vantagens e desvantagens de cada um. O quarto e último capítulo é feita uma conclusão do trabalho, comentando também sobre algumas dificuldades encontradas e sugestões de trabalhos futuros.

2 Fundamentação Teórica

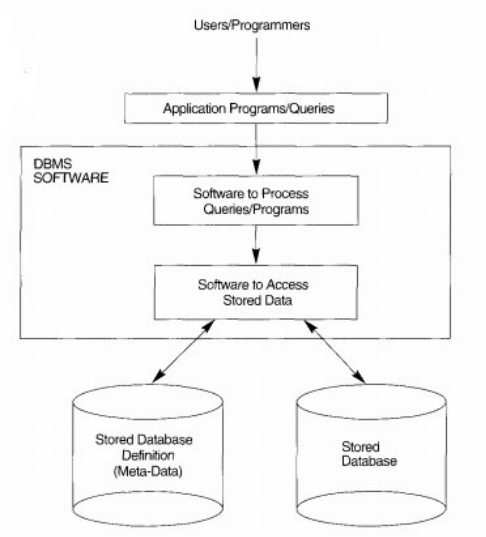
Esse capítulo trata-se da fundamentação teórica necessária para entendimento de conceitos que serão abordados no desenvolvimento dessa monografia. Serão descritos conceitos básicos de banco de dados relacionais e banco de dados não-relacionais para melhor entendimento do desenvolvimento da proposta.

Após essas seções, é apresentada uma comparação os modelos transacionais dos dois tipos de bancos, seguido por uma breve introdução a alguns dos sistemas de banco de dados não-relacionais *open source* mais utilizados.

2.1 Banco de Dados Relacionais

Segundo Elmasri e Navathe (2011), um banco de dados é uma coleção de dados que representam fatos relacionados a um domínio específico; e um sistema de gerenciamento de banco de dados (SGBD) é uma coleção de programas que permite usuários criar e manter um banco de dados. O principal objetivo de um SGBD é facilitar o processo de definição, construção, manipulação e distribuição de banco de dados entre usuários e aplicações. Um sistema de banco de dados é a junção do banco de dados em si e um SGBD, ilustrado na Figura 2.

Figura 2 – Configuração de um sistema de banco de dados simplificado



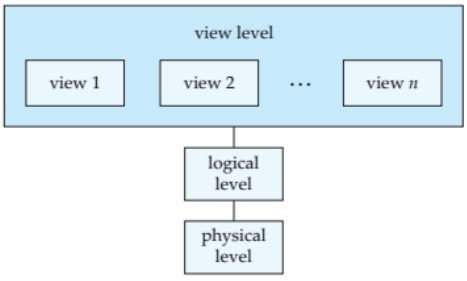
Fonte: Adaptado de (ELMASRI; NAVATHE, 2004)

Uma importante finalidade de um banco de dados é fornecer uma visão abstrata dos dados aos usuários, ocultando detalhes de como os dados são armazenados e mantidos

(SILBERSCHATZ et al., 2011). Essa abstração é simplificada em níveis para facilitar o uso do banco de dados pelos usuários, principalmente na recuperação dos dados, que precisa ser de forma eficiente. Esses níveis são: nível lógico, nível físico e nível de visão, como ilustra a Figura 3.

- Nível físico: é o nível de abstração mais baixo e descreve como os dados são realmente armazenados. Também descreve em detalhes as estruturas de dados;
- Nível lógico: descreve os dados armazenados e as relações entre eles. Ou seja, o banco inteiro é descrito em termos simples;
- Nível de visão: descreve apenas uma parte do banco. Esse nível existe para simplificar a interação do usuário com o sistema, que pode fornecer muitas visões para um mesmo banco.

Figura 3 – Os três níveis de abstração de dados



Fonte: Adaptado de (SILBERSCHATZ et al., 2011)

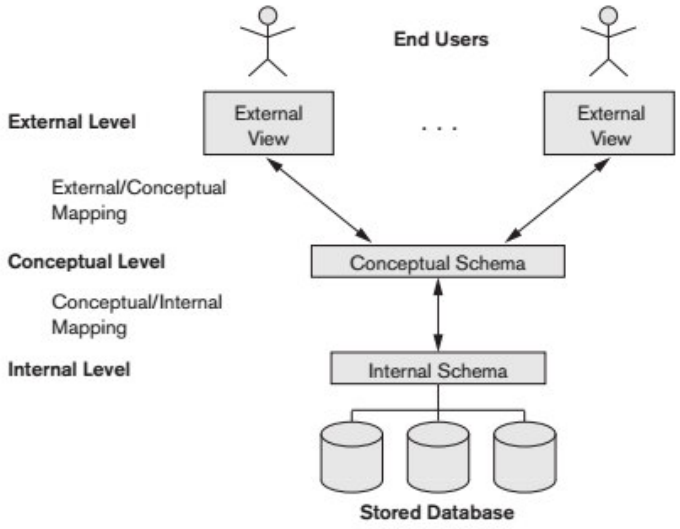
Para auxiliar na abstração dos dados, foi desenvolvida a arquitetura de três camadas. A arquitetura de três camadas é uma ferramenta que permite o usuário visualizar os níveis de esquema em um sistema de banco de dados e tem como objetivo separar as aplicações do usuário do banco de dados físico (ELMASRI; NAVATHE, 2011).

Segundo Elmasri e Navathe (2011), esses esquemas podem ser definidos em três níveis, ilustrados na Figura 4:

- Nível interno: possui um esquema interno, que descreve a estrutura do armazenamento físico do banco de dados. O esquema interno utiliza de um modelo de dados físico e descreve detalhes completos do armazenamento de dados e caminhos de acesso ao banco de dados;
- Nível conceitual: possui um esquema conceitual, que descreve a estrutura do banco de dados inteiro para uma comunidade de usuários. Um modelo de dados representativo é usado para descrever o esquema conceitual quando um sistema de dados é implementado;

- Nível externo ou de visão: possui uma série de esquemas externos ou visões de usuário, cada uma descrevendo uma parte do banco de dados que o usuário está interessado e oculta o restante do banco de dados do grupo de usuários. Assim como o nível anterior, cada esquema externo é implementado usando um modelo representativo.

Figura 4 – Arquitetura de três esquemas



Fonte: Adaptado de (ELMASRI; NAVATHE, 2011)

É importante lembrar que esses três esquemas são apenas descrições dos dados; os dados armazenados que existem estão apenas no nível físico (ELMASRI; NAVATHE, 2011).

2.1.1 Projeto de um banco de dados relacional

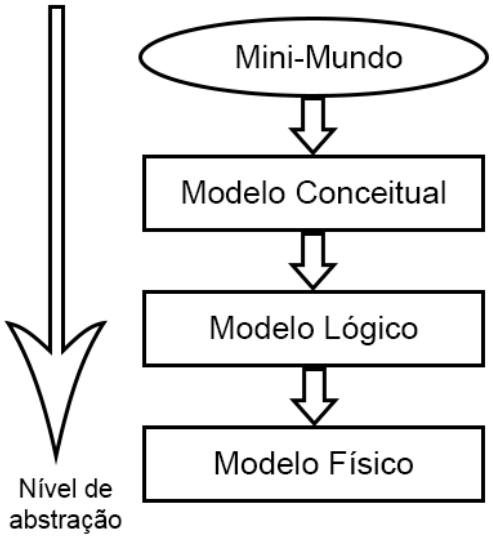
O desenvolvimento de um projeto geralmente ocorre em três etapas: modelo conceitual, modelo lógico e modelo físico.

A etapa de modelo conceitual fornece uma descrição detalhada do projeto, utilizando um modelo de dados, gerando um esquema conceitual (SILBERSCHATZ et al., 2011). Na etapa de modelo lógico acontece o mapeamento do esquema conceitual de alto nível para o modelo de dados de implementação que será utilizado (SILBERSCHATZ et al., 2011). Por fim, na etapa do modelo físico, os recursos físicos do banco de dados são especificados com base no modelo de dados lógico (SILBERSCHATZ et al., 2011). A Figura 5 ilustra a ordem que essas etapas ocorrem.

As etapas referentes ao modelo conceitual e ao modelo lógico serão melhor desenvolvidas nos próximos parágrafos.

O projeto de um banco de dados envolve em sua grande parte o projeto do esquema de banco de dados (SILBERSCHATZ et al., 2011). Tendo como papel central a necessidade

Figura 5 – Fases de um projeto de banco de dados



Fonte: Acervo do autor

dos usuários, o projetista de banco de dados tem de interagir com os usuários a fim de caracterizar quais são as necessidades a serem atendidas, resultando em especificações dessas necessidades.

Essa especificação de necessidades é o levantamento e análise de requisitos, primeiro passo para um projeto de banco de dados, seguido pela criação de um esquema conceitual (ELMASRI; NAVATHE, 2011). Essa fase é chamada de projeto conceitual e utiliza-se um modelo de dados de alto nível. O projeto conceitual fornece uma descrição detalhada e concisa dos requisitos dos dados dos usuários e o mais utilizado é o modelo Entidade-Relacionamento (ER).

Esse modelo foi criado para facilitar o projeto do banco (SILBERSCHATZ et al., 2011). Muito útil no mapeamento de interações em um esquema conceitual, visto que nem todas as informações contidas no levantamento de requisitos podem ser mapeadas diretamente para o modelo relacional. O modelo ER contêm três noções básicas: o conjuntos de entidades, o conjuntos de relacionamento e o conjunto de atributos.

Uma entidade é “algo” ou “objeto” no mundo real, diferente de outros objetos, representada por um conjunto de atributos (SILBERSCHATZ et al., 2011). Uma entidade contém um conjunto de propriedades e valores que a identificam de maneira única.

Os atributos são propriedades que descrevem o membro de um conjunto de entidades. Segundo Silberschatz et al. (2011), ao designar um atributo a um conjunto de entidades indica que o banco de dados guarda informações semelhantes relacionadas a cada entidade do conjunto. Porém, cada entidade poder ter seu próprio valor a um atributo.

Um relacionamento entre entidades existe quando um atributo de uma entidade se refere a outra entidade (ELMASRI; NAVATHE, 2011). Ou seja, relacionamento é associações entre várias entidades. Um conjunto de relacionamentos é definido por relacionamentos do mesmo tipo. Cada instância de um relacionamento é uma associação de entidade e representa o fato das entidades participantes estarem relacionadas à situação do minimundo correspondente (ELMASRI; NAVATHE, 2011). Na Figura 6 temos um exemplo de relacionamento.

Instructor e *student* são duas entidades. Nesse exemplo, as duas entidades tem como mesmos atributos *id* (referente a um número de identificação) e *name* (referente ao nome). O que difere entre elas é que a entidade *instructor* tem como terceiro atributo *salary* (referente ao salário do instrutor); e a entidade *student*, *tot_cred* (referente ao total de créditos de aula).

O relacionamento entre essas duas entidades é chamado *advisor*, denotando a associação entre instrutores e estudantes, porém tendo suas restrições de participação. Essas restrições são chamadas de cardinalidade, que indica a quantidade de vezes que cada entidade participa do relacionamento (SILBERSCHATZ et al., 2011).

Entre *instructor* e *advisor* há uma restrição de cardinalidade 0..*, que significa que o instrutor pode ser responsável por zero, um, dois ou mais estudantes. Isso indica que o relacionamento *advisor* de *instructor* para *student* é de um-para-muitos.

Já entre *advisor* e *student* há uma restrição de cardinalidade 1..1, que significa que cada estudante será designado somente a um instrutor e é a cardinalidade mínima dessa relação. Isso indica que o relacionamento *advisor* de *student* para *instructor* é de um-para-um.

Figura 6 – Exemplo de um relacionamento: relacionamento *advisor*

Fonte: Adaptado de (SILBERSCHATZ et al., 2011)

A partir do projeto conceitual é desenvolvido o esquema do banco de dados (ELMASRI; NAVATHE, 2011). O esquema do banco de dados corresponde ao projeto lógico do banco. Ferramentas de projetos gráficos têm incorporadas um algoritmo que geram automaticamente um esquema relacional a partir de um esquema conceitual (ELMASRI; NAVATHE, 2011). Esse algoritmo consiste em sete passos cujo foco está nos construtores básicos do modelo ER, como entidades, relacionamentos e tributos.

Segundo Elmasri e Navathe (2011), os sete passos são: mapeamento de entidades regulares (fortes), mapeamento de entidades fracas, mapeamentos de relacionamentos binários 1:1, mapeamentos de relacionamentos binários 1:N, mapeamento de relacionamentos binários N:M, mapeamento de atributos multivalorados, mapeamento de relacionamentos n-ários. A Tabela 1 resume as correspondências entre as construções e restrições dos modelos ER e relacional (ELMASRI; NAVATHE, 2011).

Tabela 1 – Tabela de correspondência entre os modelos ER e relacional

Modelo ER	Modelo Relacional
Tipo de entidade	Relação de entidade
Tipo de relacionamento 1:1 ou 1:N	Chave estrangeira (ou relação de relacionamento)
Tipo de relacionamento N:N	Relação de relacionamento e duas chaves estrangeiras
Tipo de relacionamento n-ário	Relação de relacionamento e n chaves estrangeiras
Atributo simples	Atributo
Atributo composto	Conjunto de atributos componentes simples
Atributo multivalorado	Relação e chave estrangeira
Conjunto de valores	Domínio
Atributo-chave	Chave primária (ou secundária)

Fonte: Adaptado de (ELMASRI; NAVATHE, 2011)

Com o projeto conceitual e o projeto lógico desenvolvidos, o próximo passo é a criação do banco de dados. O modelo relacional é o mais utilizado pelos SGBDs atuais. Ele conquistou esse patamar devido à sua simplicidade, a qual facilitou bastante o trabalho do programador (SILBERSCHATZ et al., 2011).

2.1.2 Características do banco de dados relacionais

Os bancos de dados relacionais utilizam o modelo transacional ACID. A sigla ACID vem de Atomicidade, Consistência, Isolamento e Durabilidade, que são propriedades impostas pelo controle de concorrência e restauração do SGBD. Elmasri e Navathe (2011) definem as propriedades ACID como:

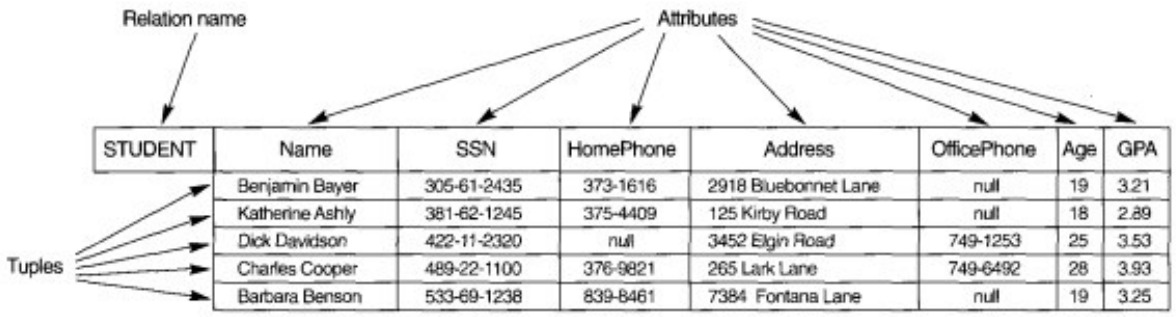
- Atomicidade: transação é uma unidade atômica de processamento; sendo executada totalmente ou de modo nenhum;
- Consistência: transação será preservadora de consistência se sua execução completa fizer o banco de dados passar de um estado consistente para outro;
- Isolamento: transação deve ser executada como se estivesse isolada das demais, não sofrendo interferência de quaisquer outras transações concorrentes;

- Durabilidade: mudanças feitas no banco de dados por uma transação efetivada devem persistir no banco de dados e não devem ser perdidas em razão de uma falha.

Um banco de dados relacional é representado como uma coleção de relações, com cada relação sendo representada como uma tabela (ELMASRI; NAVATHE, 2011). Cada linha dessa tabela é chamada de tupla e representa um conjunto de dados relacionados. O cabeçalho de uma coluna se chama atributo, pertencente a um domínio, que é um conjunto de valores atômicos e valores permitidos.

Uma relação é definida por um conjunto de n-tuplas. Em cada tupla existe uma lista ordenada de n vetores, onde cada valor é um elemento de um domínio (ELMASRI; NAVATHE, 2011). É importante lembrar de duas tuplas não podem ter mesma combinação de valores para todos os atributos. Na figura 7 temos um exemplo de relação com seus atributos e tuplas.

Figura 7 – Os atributos e as tuplas de uma relação ALUNO



STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25

Fonte: Adaptado de (ELMASRI; NAVATHE, 2004)

2.2 Banco de Dados Não-Relacionais

Um grande volume variado de dados é produzido diariamente por diversos sistemas, dispositivos, aplicações e pessoas. E essa grande quantidade de dados gerado são classificados como *Big Data* (LI; MANOHARAN, 2013).

Big Data são grandes quantidades de dados estruturados, semi-estruturados e/ou não estruturados, que requer um processamento rápido e flexível (LI; MANOHARAN, 2013), produzindo novos desafios diariamente.

Durante muito tempo, os SGBDs baseados no Modelo Relacional atenderam os requisitos de várias aplicações. Porém, esses sistemas de gerenciamento têm se mostrado ineficientes na manipulação desses dados (VALE; ROCHA, 2014). Os SGBDs tradicionais não conseguem atender completamente às necessidades de *Big Data* e uma das mais

variadas soluções para esse atender essas necessidades são banco de dados não-relacionais, também denominados bancos *NoSQL*.

NoSQL, ou banco de dados não-relacionais, é definido como um conjunto de conceitos que permite o processamento rápido e eficiente de dados com foco na performance, agilidade e segurança (MCCREARY; KELLY, 2014). Para a comunidade *NoSQL*, o termo significa *"Not only SQL"* (Não somente *SQL*). Esse termo surgiu a partir de uma solução de banco de dados que não oferecia uma interface *SQL*, mas tendo o sistema ainda baseado na arquitetura relacional (BRITO, 2010).

O propósito dos bancos *NoSQL* não é substituir o Modelo Relacional totalmente, mas apenas em situações que seja necessária uma maior flexibilidade de estruturação do banco de dados (BRITO, 2010). Apesar da definição do termo *NoSQL* parecer excluir a linguagem *SQL*, ela também é utilizada em banco de dados não-relacional, assim como outras linguagens menos conhecidas. Entretanto, essa falta de padronização de uma linguagem para esse tipo de banco de dados ainda é uma das maiores desvantagens que a comunidade enfrenta.

2.2.1 Projeto de um banco de dados não-relacional

A metodologia utilizada para a modelagem de dados para um banco de dados *NoSQL* é contrária à usada em um banco de dados relacional. Em um banco de dados relacional, começamos com um modelo de domínio e desenvolvemos um modelo de dados físicos em torno dele, criando um esquema de banco de dados. Os bancos de dados *NoSQL* normalmente não possuem um esquema, que é uma de suas características. Em vez disso, temos uma aplicação orientada a dados e, portanto, nos concentramos no modelo de consulta e construímos nosso modelo de dados em torno dele, a fim de satisfazer de forma eficiente suas necessidades (SILVA, 2011). Basicamente, o banco de dados será definido para suportar as demandas da aplicação.

Apesar dessa falta de esquema, os bancos de dados não-relacionais ainda utilizam de casos de uso para definir padrões na aplicação. Segundo Silva (2011), ao examinar as interdependências entre os diferentes elementos de dados, é permitido alcançar diferentes decisões de projeto.

Silva (2011) dá um exemplo: ao definir as cardinalidades entre relações (nesse caso, de um para muitos e muitos para muitos), é importante estimar também quão grande será esse muitos e quanto irá crescer. Essas decisões podem ter efeitos diferentes no desempenho e podem depender de como os dados crescem. Ou seja, é importante conhecer os dados a serem modelados visto que, diferente dos bancos relacionais, geralmente existe mais de um modo de modelar nos bancos não-relacionais (SILVA, 2011), por não existir uma arquitetura padrão.

Enquanto em bancos relacionais é comum a prática de normalizar os dados a fim de eliminar redundância e evitar inconsistência no banco, em bancos não-relacionais é necessário desnormalizar os dados a fim de corresponder aos requisitos da aplicação (SILVA, 2011). Isso se deve ao fato de que bancos *NoSQL* ainda não possuem uma maneira flexível para operar os dados como os bancos relacionais, que possuem a linguagem *SQL* (*Structured Query Language*).

Resumindo, ao dar início a um projeto de banco não-relacional, o foco está nas consultas que serão esperadas na aplicação e, a partir disso, começar a trabalhar no modelo de dados (SILVA, 2011).

2.2.2 Características dos bancos de dados não-relacionais

Uma das principais características dos sistemas *NoSQL* é que eles são *schema-free* (MCCREARY; KELLY, 2014). Essa característica dá a liberdade de guardar informações sem a necessidade de desenvolver um esquema, podendo começar a codificar, guardar e recuperar informações sem saber como o banco de dados funciona internamente. O grande benefício é que o tempo de desenvolvimento é encurtado, o que aumenta conforme a aplicação vai passando por várias versões e necessidade de alteração interna dos dados no banco.

Os bancos não-relacionais permitem que extraia o dado usando interfaces sem junção (MCCREARY; KELLY, 2014). Apesar de não armazenarem informações sobre como registros individuais se relacionam com outros registros no banco de dados, o que pode soar como uma limitação, esse tipo de banco de dados armazenam e recuperam dados de muitos formatos, tornando-os mais flexíveis em termos de estruturas de dados.

Os sistemas *NoSQL* tamtém permitem armazenar o banco de dados em múltiplos processadores e mantêm a performance de alto nível (MCCREARY; KELLY, 2014). A principal vantagem dessa abordagem é no caso de conjuntos de dados muito grandes, pois mesmo o maior servidor único disponível não poderia armazenar ou processar todos os dados necessários. Alguns dos sistemas são de baixo custo, tendo que separar a memória RAM do disco, mas se adicionado mais processadores, consegue-se um aumento na performance.

Muitos SGDBs restringem a localização das transações a um único processador. Sistemas ACID focam na consistência e integridade dos dados acima de outras considerações. Os sistemas não-relacionais utilizam o método transacional BASE para controlar as transações no banco. A sigla BASE vem do inglês *Basic availability, Soft-state, Eventual consistency* e, segundo McCreary e Kelly (2014), se refere aos seguintes conceitos:

- *Basic availability*: permite que os sistemas sejam temporariamente inconsistentes para que as transações sejam gerenciáveis. Nos sistemas BASE, as capacidades de informação e serviço estão "basicamente disponíveis";

- *Soft-state*: reconhece que alguma imprecisão é temporariamente permitida e dados podem mudar enquanto estiver sendo usado para reduzir a quantidade de recursos consumidos;
- *Eventual consistency*: significa eventualmente, quando toda a lógica de serviço é executada, o sistema é deixado em um estado consistente.

Os sistemas BASE focam na disponibilidade e são notáveis porque seu objetivo número um é permitir novos dados serem armazenados, mesmo que ocorra o risco de ficar dessincronizado por um curto período de tempo (MCCREARY; KELLY, 2014). Esses sistemas tendem a ser mais simples e rápidos porque não é necessário escrever códigos para lidar com bloqueio e desbloqueio de recursos.

2.2.3 Arquitetura

Para os bancos não-relacionais, há muitos tipos de arquitetura de dados que podem ser utilizados. Porém, as arquiteturas mais comuns são: armazenamento orientado a documentos, armazenamento de chave-valor, de famílias de colunas e de grafos.

2.2.3.1 Armazenamento orientado a documentos

Os bancos de dados orientados a documentos possuem documentos indexados e um mecanismo de consulta simples é fornecido (CATTELL, 2011). Em geral, esse tipo de banco não possui esquema e essa característica faz dele uma boa opção para armazenamento de dados semiestruturados (DIANA; GEROSA, 2010). Os bancos de dados populares desse tipo de arquitetura são o MongoDB e o CouchDB. A Figura 8 ilustra esse tipo de armazenamento.

Figura 8 – Exemplo de armazenamento em documentos

Fonte: Adaptado de (MONIRUZZAMAN; HOSSAIN, 2013)

Ao adicionar um novo documento, ele é automaticamente indexado no documento já existente (MCCREARY; KELLY, 2014). Apesar dos grandes índices, tudo é pesquisável. Sabendo a propriedade do documento procurado, encontra-se todos os documentos que contêm essa mesma propriedade, mostrando a localização exata do armazenamento. Essa indexação pode gerar duplicação de dados. Em um SGBD relacional poderia ser um problema de consistência, mas no SGBD não-relacional, essa duplicação facilita a distribuição de dados no sistema.

2.2.3.2 Armazenamento chave-valor

Também conhecidos como tabela de *hash* distribuídas, os objetos são indexados por chaves (*string*), que possibilitam as buscas por esses objetos (CATTELL, 2011). Esse tipo de armazenamento não tem uma linguagem de consulta própria; ele fornece uma maneira de adicionar e remover os pares de chave-valores (uma combinação de chave e valor onde a chave está vinculada ao valor até que um novo valor seja atribuído) (MCCREARY; KELLY, 2014). Os bancos mais famosos com essa arquitetura são o Riak, o Redis, o MemcacheDB e o Dynamo. Na Figura 9 temos um exemplo desse armazenamento.

Figura 9 – Exemplo de armazenamento em chave-valor

Car	
Key	Attributes
1	Make: Nissan Model: Pathfinder Color: Green Year: 2003
2	Make: Nissan Model: Pathfinder Color: Blue Color: Green Year:2005 Transmission: Auto

Fonte: Adaptado de (MONIRUZZAMAN; HOSSAIN, 2013)

Esses sistemas geralmente fornecem um mecanismo de persistência e funcionalidade adicional também: replicação, controle de versão, bloqueio, transações, classificação e/ou outros recursos (CATTELL, 2011). Um dos benefícios desse tipo de arquitetura é a liberdade em armazenar qualquer tipo de dados. O sistema armazenará a informação como BLOB (*Binary Large Object*) e retorna esse objeto quando há uma requisição (MCCREARY; KELLY, 2014). É papel da aplicação determinar que tipo de dado está sendo usado. Tanto a chave quanto o valor são flexíveis e podem ser representados em muitos formatos.

2.2.3.3 Armazenamento em famílias de colunas

A arquitetura família de colunas utiliza identificadores de linhas e colunas como chaves na procura de dados (MCCREARY; KELLY, 2014). Esse tipo de arquitetura é importante por poder manipular grandes volumes de dados. Essa arquitetura torna a escrita de dados muito mais rápido, pois os dados de um registro são colocados de uma só escrita no banco, além de ser bastante eficiente na leitura de registros inteiros (CATTELL, 2011). A Figura 10 ilustra esse tipo de armazenamento.

Figura 10 – Exemplo de armazenamento em coluna

Fonte: Adaptado de (MONIRUZZAMAN; HOSSAIN, 2013)

Segundo Cattell (2011), o modelo básico de escalabilidade dessa arquitetura é dividir linhas e colunas em vários nós. As linhas são divididas entre nós e são divididas em intervalos, o que facilita a consulta, visto que as consultas não precisam passar por todos os nós. As colunas são distribuídas em vários nós utilizando "grupos de colunas", que são uma maneira do cliente indicar quais colunas funcionam melhor quando armazenadas em conjunto. Esse grupo de colunas tem de ser pré-definido.

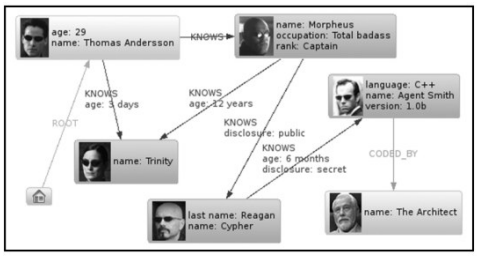
Esse tipo de arquitetura é utilizado bastante no processamento analítico online (*OLAP*) por causa da sua rapidez em calcular agregação de colunas (MCCREARY; KELLY, 2014). Os bancos mais conhecidos dessa estrutura são o Cassandra, o HBase e o Hypertable, que foi baseado no artigo sobre o BigTable (CHANG et al., 2008).

2.2.3.4 Armazenamento em grafos

Diferente dos outros tipos de armazenamentos já falados, esse está relacionado a um modelo de dados estabelecido, o modelo de grafos (CATTELL, 2011). Essa arquitetura é importante em aplicações que possuem a necessidade de analisar relações entre objetos ou visitar todos os nós de um grafo particular, sendo altamente otimizado na eficiência

de armazenamento (MCCREARY; KELLY, 2014). A Figura 11 ilustra esse tipo de armazenamento.

Figura 11 – Exemplo de um armazenamento em grafos



Fonte: Adaptado de (MONIRUZZAMAN; HOSSAIN, 2013)

Esse modelo contém uma sequência de nós e relações que, quando combinados, criam um grafo (MCCREARY; KELLY, 2014). Esse grafo representa os dados e/ou esquema de dados como grafos dirigidos e dá suporte ao uso de restrições sobre os dados (CATTELL, 2011). Os bancos mais conhecidos dessa arquitetura são o Neo4j e InfoGrid.

2.2.4 Comparando os modelos transacionais dos bancos de dados relacionais e não-relacionais

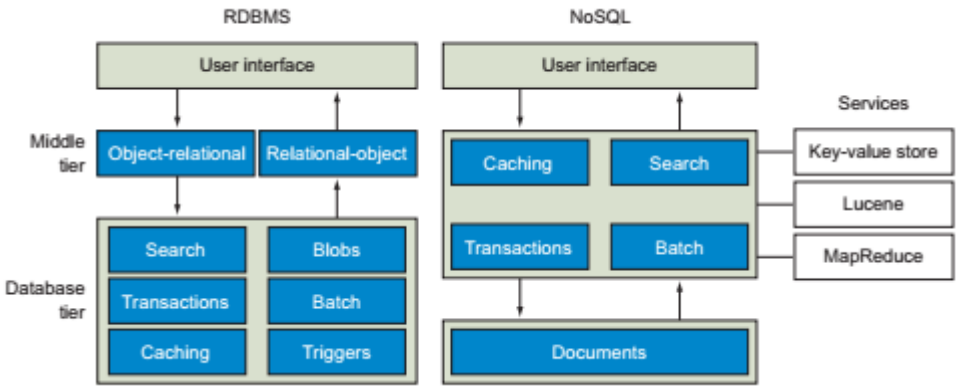
McCreary e Kelly (2014), ao comparar os dois modelos de banco de dados em uma aplicação, relatam que em um banco de dados relacional a maioria das funcionalidades da aplicação se encontra na camada de banco de dados. Já em um banco de dados não-relacional, a maioria das funcionalidades se encontram na camada que faz ligação entre a camada de dados e a interface do usuário.

A Figura 12 ilustra essa comparação. O lado esquerdo representa um SGBD relacional, que foca em colocar as funções na camada de dados, garantindo segurança e integridade nas transações. A camada intermediária é utilizada na conversão dos objetos em tabelas. No SGBD não-relacional, representado na direita, não utiliza do conceito de mapeamento de dados, movendo as funções para a camada intermediária e serviços externos.

As vantagens dos bancos de dados relacionais são: as transações ACID a nível de banco de dados faz com que o desenvolvimento seja mais fácil; utilização de visões previne mudanças feitas por usuários não autorizados; maioria do código *SQL* é compatível para outros bancos de dados *SQL*, incluindo opções *open source*; colunas e restrições digitadas valida os dados antes de serem adicionados ao banco de dados e aumenta a qualidade dos dados (MCCREARY; KELLY, 2014).

Porém, suas desvantagens são: a camada de mapeamento relação-objeto pode ser complexa; modelo de entidade-relacionamento deve ser completado antes dos testes

Figura 12 – Comparação entre um SGBD relacional e um SGBD não-relacional



Fonte: Adaptado de (MCCREARY; KELLY, 2014)

começarem, o que atrasa o desenvolvimento; SGBDs não são escalonados quando junções são necessários; fragmentação sobre muitos servidores pode ser feito, mas requer código de aplicativo e será operacionalmente ineficiente; busca completa no documento exige ferramentas externas; dificuldade em armazenador dados de grande variabilidade (MCCREARY; KELLY, 2014).

As vantagens dos bancos de dados não-relacionais são: os dados de teste podem ser carregados usando ferramentas *drag and drop* antes do modelo entidade-relacionamento estar completo; arquitetura modular permite troca de componentes; o escalonamento linear ocorre quando novos nós de processamento são adicionados ao *cluster*; custos operacionais mais baixos são obtidos por auto fragmentação; funções de pesquisa integradas fornecem resultados de pesquisa classificados de alta qualidade; não há necessidade de uma camada de mapeamento objeto-relacional; é fácil armazenar dados de alta variabilidade (MCCREARY; KELLY, 2014).

Já suas desvantagens são: os armazenamentos de documentos não fornecem segurança de alto nível no nível do elemento; os sistemas *NoSQL* são novos para muitos funcionários e treinamento adicional pode ser requeridos; o arquivo de documentos possui sua própria linguagem de consulta não padronizada, que proíbe a portabilidade; o arquivo de documentos não funcionará com as ferramentas de relatório e *OLAP* existentes (MCCREARY; KELLY, 2014).

2.2.5 Sistemas de banco de dados não-relacionais *open source* mais utilizados

Apesar de ser uma tecnologia mais recente, já existe uma variedade de sistemas de banco de dados não-relacionais. Essa seção aborda alguns dos sistemas *open source* mais utilizados por desenvolvedores, que são o MongoDB, o CouchDB, o Riak, o Cassandra e o HBase.

2.2.5.1 MongoDB

O MongoDB é um sistema de banco de dados *open-source* orientado a documentos, escrito em C++, focado em alta performance e desenvolvimento ágil (SILVA, 2011). Esse sistema possui características parecidas com as de um banco de dados relacional; tem suporte a complexos tipos de dados; possui uma poderosa linguagem de consulta; e rapidez ao acessar um grande volume de dados (HAN et al., 2011). Por ser um banco de dados orientado a documentos, o MongoDB permite que os dados persistam em um estado aninhado, podendo fazer consulta a esses dados de forma *ad hoc* (REDMOND; WILSON, 2012).

O MongoDB armazena os dados como objetos BSON (variante binária do JSON) (BSON, 2011), suportando todos os tipos de dados que são partes do JSON (JSON, 2011), mas também definindo novos tipos de dados (HARRISON, 2015). Segundo Silva (2011), esses documentos são organizados em coleções e podem ser vistos como equivalente a tabela dos bancos relacionais. Essas coleções podem conter qualquer tipo de documento, nenhuma relação é aplicada e documentos dentro de uma coleção geralmente têm a mesma estrutura que fornece uma maneira lógica de organizar dados.

Cada documento é identificado por um único ID, que é dado pelo usuário no momento da criação do documento ou gerado automaticamente pelo banco de dados (SILVA, 2011). As relações entre os documentos podem ser modeladas de duas formas diferentes: referência de documentos ou incorporação de documentos

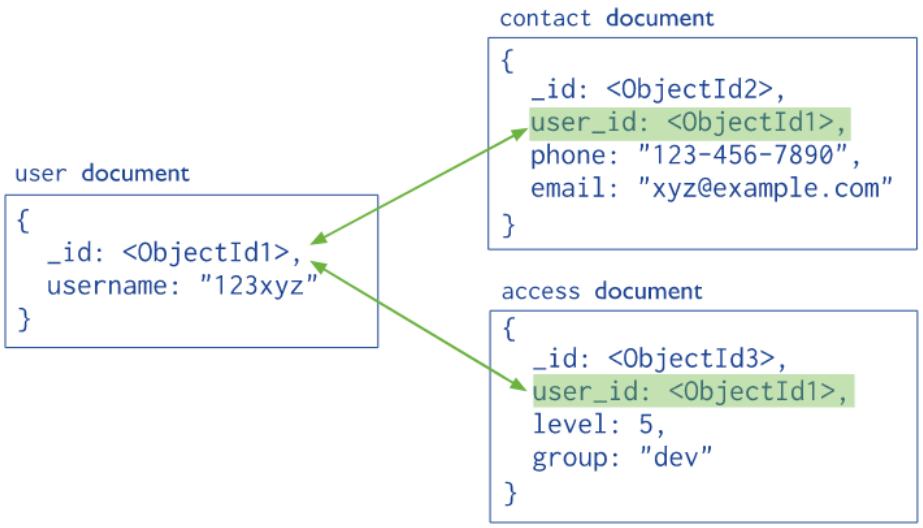
A modelagem por referência pode ser definidas manualmente atribuindo algum campo de referência ao valor do campo de identificação do documento referenciado (STRAUCH; SITES; KRIHA, 2011). Além disso, o MongoDB fornece uma maneira mais formal de especificar referências denominadas *DBRef* ("Referência de banco de dados"). Segundo Strauch, Sites e Kriha (2011), as vantagens de usar o DBRefs são que os documentos em outras coleções podem ser referenciados por eles e que alguns dispositivos de linguagem de programação os descarguem automaticamente. Essa referência pode ser vista como equivalente à chave estrangeira no banco relacional (SILVA, 2011).

Segundo a documentação do MongoDB (MONGODB, 2017), o modelo por referência é usado:

- quando a incorporação de documentos resultaria na duplicação de dados, sem resultar em vantagens de desempenho de leitura suficientes para superar as implicações da duplicação;
- para representar relações complexas de muito para muitos;
- para modelar grandes conjuntos de dados hierárquicos.

Na Figura 13 temos um exemplo de modelagem por referência. Em *user document*, temos um _id de identificação do documento. Esse _id é utilizado nos documentos *contact document* e *acess document* para referenciar o seu documento de origem. Ou seja, esse _id tem a mesma função da chave estrangeira do banco relacional, como já foi dito antes.

Figura 13 – Exemplo de modelagem de dados por referência



Fonte: Adaptado de (MONGODB, 2017)

A modelagem incorporação de documentos significa que esse documento pode conter campos de dados referentes a outros documentos (SILVA, 2011). De acordo com a documentação do MongoDB (2017), esse tipo de modelo é mais eficiente porque oferece um melhor desempenho para as operações de leitura, bem como a capacidade de solicitar e recuperar dados relacionados em uma única operação de banco de dados, visto que os modelos de dados incorporados permitem atualizar dados relacionados em uma única operação de gravação atômica. Ao usar o modelo de referência, para cada cruzamento de referência resulta em uma consulta no banco, também resulta em pelo menos uma adição de latência entre o servidor e o banco de dados (STRAUCH; SITES; KRIHA, 2011).

Segundo a documentação do MongoDB (MONGODB, 2017), o modelo por incorporação é usado:

- quando se tem relações entre entidades;
- quanto tem relações um-para-muito entre entidades, onde os documentos-filhos ("muitos") sempre aparecem com ou são visualizados no contexto dos documentos-pais ("um");
- quando um objeto não está sendo referenciado em outro objeto.

Na Figura 14 temos um exemplo de modelagem por incorporação de documentos. No documento, percebe-se que os atributos *contact* e *access* são os dois documentos do exemplo anterior que foram incorporados ao documento *user document*.

Figura 14 – Exemplo de modelagem de dados por incorporação de documentos

Fonte: Adaptado de (MONGODB, 2017)

O MongoDB suporta consultas dinâmicas sobre seus documentos armazenados em uma coleção, incluindo os documentos incorporados em outros (SILVA, 2011). Essas consultas são expressas em uma sintaxe JSON e enviadas para o MongoDB como objetos BSON pelo drive do banco de dados (OREND, 2010). Consultas mais complexas podem ser expressas utilizando a operação MapReduce (DEAN; GHEMAWAT, 2008), podendo ser útil para o processamento em lote de dados e operações de agregação, com o resultado das operações sendo armazenados em uma coleção temporária, que é automaticamente removida após o cliente obter os resultados (SILVA, 2011).

A maior vantagem do MongoDB é sua habilidade de manipular grandes volumes de dados por causa da replicação e escalonamento horizontal (REDMOND; WILSON, 2012). Adiciona-se o fato que ele tem um modelo de dados bastante flexível, resultante da não-necessidade de criar um esquema. MongoDB foi desenvolvido para ser de fácil usabilidade (REDMOND; WILSON, 2012). Esse fato deve-se à semelhança entre os comandos do Mongo com os comandos *SQL*, exceto aos que dizem respeito ao servidor.

Segundo Redmond e Wilson (2012), apesar de ser a maior vantagem do MongoDB, alguns desenvolvedores creem que a falta de um esquema é algo negativo. Para alguns, as restrições do banco relacional são uma garantia contra a inserção de algum dado errado, o que pode causar diversos problemas. Essa flexibilidade do banco não é de muita importância se o modelo de dados já está definido.

2.2.5.2 CouchDB

O CouchDB é um banco de dados *open source* orientado a documentos escrito em Erlang, uma linguagem de programação voltada para aplicações concorrentes e distribuídas (STRAUCH; SITES; KRIHA, 2011). O CouchDB foi projetado com a Web em mente e todas as inúmeras aplicações que acompanham. Consequentemente, o CouchDB oferece uma robustez incomparável pela maioria dos outros bancos de dados, prosperando mesmo quando a conectividade é raramente disponível (REDMOND; WILSON, 2012).

O armazenamento de dados nesse banco é feito através de documentos contendo campos nomeados que possuem uma chave/nome e um valor, sendo o nome do campo de valor único dentro do documento (STRAUCH; SITES; KRIHA, 2011). Um documento é um arquivo JSON, *schema-free* e sem seguir qualquer tipo de estrutura, exceto a herdada do JSON (SILVA, 2011). Cada documento pode ter qualquer número de atributos e cada atributo pode conter listas e até objetos (OREND, 2010), assim como cada documento é identificado por um único ID (SILVA, 2011).

Na Figura 15 temos um exemplo de um documento armazenado no CouchDB. Nesse exemplo, o documento contém informações necessárias para descrever um livro (SILVA, 2011). A identificação do documento é dada pelo campo _id, declarado logo no começo, seguido dos atributos referentes aos dados a serem armazenados.

Figura 15 – Exemplo de um documento armazenado no CouchDB

```
{
 "_id":"282e1890a8095bcc0cb1318bed85bcb377e2700f",
 "_rev":"946B7D1C",
 "Type":"Book"
 "Title":"Flatland",
 "Author":"Edwin A. Abbot",
 "Date":"2009-10-09",
 "Language":"English",
 "ISBN":"1449548660",
 "Tags":["flatland", "mathematics", "geometry"],
}
```

Fonte: Adaptado de (SILVA, 2011)

O modelo de consultas nesse banco consistem em visões que são criadas utilizando funções do MapReduce (DEAN; GHEMAWAT, 2008) ou uma API HTTP, que permite os clientes acessarem e consultarem as visões (OREND, 2010).

As consultas feitas através da API RESTFul (Transferência de Estado Representacional) HTTP permite a leitura e atualização de documentos (STRAUCH; SITES; KRIHA, 2011). Essa mesma API também executa as operações básicas de CRUD (*Create, Read, Delete, Update*) em todos os itens armazenados, utilizando os métodos *POST, GET, PUT* e *DELETE* (SILVA, 2011). As consultas mais complexas podem ser implementadas na forma de visão.

Segundo Orend (2010), uma visão no CouchDB é basicamente uma coleção de pares de valores-chave, que são ordenados por sua chave, criadas por funções MapReduce especificadas pelo usuário, que são chamadas incrementalmente sempre que um documento no banco de dados é atualizado ou criado. Esta é uma diferença para outros bancos de dados distribuídos em que o modelo MapReduce é usado para instrumentar índices já existentes para agregações.

Para Silva (2011), essas visões são métodos para agregar e reportar documentos existentes no banco de dados, não afetando os dados existentes subjacentes na base; eles simplesmente mudam a forma como os dados são representados e definem o modelo da aplicação, além de ser um modo de organização interna.

O CouchDB não é apenas um banco de dados *NoSQL*, mas também um servidor web para aplicações escritas em JavaScript. A vantagem de usar o CouchDB como servidor web é que os aplicativos podem ser implantados apenas os armazenando no banco de dados e que as aplicações podem acessar diretamente o banco de dados sem a sobrecarga de um protocolo de consulta (OREND, 2010).

Uma das maiores desvantagens do CouchDB é a replicação do banco. Segundo Redmond e Wilson (2012), a replicação do banco é tudo ou nada, o que significa que todos os servidores replicados terão o mesmo conteúdo. Não há cortes para distribuir conteúdo em torno do *datacenter*. Outro problema são as visualizações baseadas em MapReduce (DEAN; GHEMAWAT, 2008). Elas não executam do modo esperado como em um banco relacional e não é aconselhável fazer consultas *ad hoc*.

2.2.5.3 Riak

O Riak é um sistema distribuído *open source* com armazenamento chave-valor, onde seus valores podem ser qualquer coisa e todos são acessíveis através de uma interface HTTP (REDMOND; WILSON, 2012). É baseado no banco Dynamo (DECANDIA et al., 2007), utilizado pela Amazon, desenvolvido na linguagem Erlang e C e, apesar de ser *open source*, seu desenvolvimento é também supervisionado pela companhia Basho (SILVA, 2011).

O Riak também é tolerante a falhas, suportando alta disponibilidade e proporcionando níveis ajustáveis de garantias de durabilidade e consistência (SILVA, 2011). Como o CouchDB, esse banco também provê uma interface web e cria réplicas dos seus arquivos distributivamente (MUHAMMAD, 2011). Os dados são organizados de maneira simples, consistindo em *buckets*, chaves e valores, que também são referenciados como objetos.

Bucket é um objeto que contém todas as informações dos dados não tratados, ou seja, todas as chaves e valores serão armazenados em um *bucket* (MUHAMMAD, 2011). Embora não haja exigência sobre a estrutura de dados armazenada dentro de um *bucket*,

geralmente é desejável armazenar dados semelhantes dentro do mesmo *bucket* (SILVA, 2011).

Os valores mantêm os dados a serem armazenados e podem armazenar qualquer tipo de dados, conforme exigido pelo usuário, identificados e acessados por uma chave exclusiva e cada par chave-valor é armazenado em um *bucket* (SILVA, 2011).

Complementando este modelo de dados, o conceito de links é introduzido. Segundo Silva (2011), os links definem relações unidirecionais entre objetos diferentes, fornecendo ao usuário uma maneira simples de criar relacionamentos entre dados. Um único objeto pode conter vários links apontando para objetos diferentes. Todas os acessos de dados feitos no Riak são através de uma chave, provendo a capacidade de adicionar, obter, atualizar e excluir objetos, utilizando a API RESTful HTTP e seus métodos (SILVA, 2011).

A companhia Basho desenvolveu a plataforma Riak Search. Essa plataforma fornece um mecanismo de pesquisa completo no armazenamento do banco, provendo ao usuário métodos de acesso adicionais (SILVA, 2011). Esta tecnologia ainda não é muito utilizada por ainda estar em fase de desenvolvimento.

Um dos pontos fortes do Riak é o foco em remover pontos de falha na intenção de suportar o máximo de tempo de atividade para atender às demandas de mudança (REDMOND; WILSON, 2012). E, segundo Redmond e Wilson (2012), você precisar de mais velocidade do que o HTTP pode manipular, você também pode tentar a sua comunicação através do Protobuf, que é um protocolo de codificação e transporte binário mais eficiente.

Redmond e Wilson (2012) afirma que o Riak ainda está em atraso em termos de uma estrutura de consulta *ad hoc* fácil e robusta, embora seja certamente no caminho certo. E, para os que não programam em Erlang, usar o JavaScript nesse banco pode ter algumas limitações, como indisponibilidade pós-*commit* e execução lenta do MapReduce (DEAN; GHEMAWAT, 2008).

2.2.5.4 Cassandra

O Cassandra é um banco de dados distribuído *open source* orientado a família de colunas, desenvolvido inicialmente pelo Facebook e projetado para lidar com grandes quantidades de dados, oferecendo alta disponibilidade e escalabilidade (SILVA, 2011).

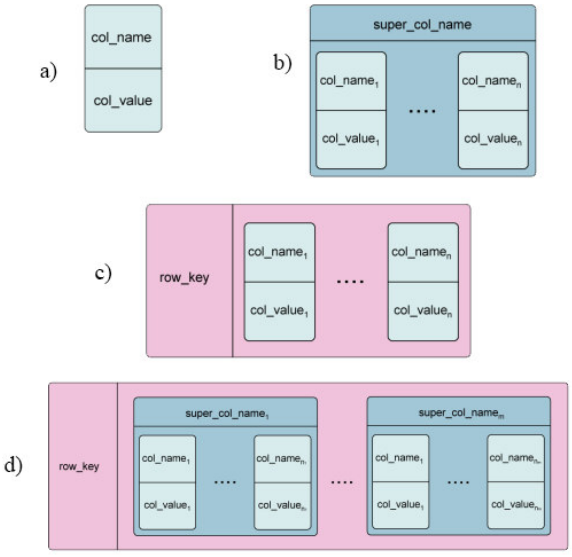
O Cassandra é um banco escalonadamente escalável. Os dados de uma tabela são divididos e distribuídos entre os nós por uma função *hash* consistente que também preserva a ordem das chaves de linha (STRAUCH; SITES; KRIHA, 2011). A tabela nesse banco é um mapa multidimensional distribuído indexado por uma chave e o valor é um objeto altamente estruturado (SILVA, 2011). A chave da linha na tabela é uma *string* sem restrições de tamanho, mas geralmente tem entre 16 a 32 bytes; e cada operação feita sob uma única linha de linha é atômica por réplica, não importa quantas colunas estão sendo

lidas ou gravadas (LAKSHMAN; MALIK, 2010).

Segundo Silva (2011), as múltiplas dimensões do banco de dados podem ser representadas por:

- Coluna: é a menor entidade de dados e é uma tupla que contém nome e valor, ilustrada na Figura 16 a);
- Super Coluna: uma super-coluna pode ser vista como uma coluna com subcolunas. Similar a uma coluna, é uma tupla que contém um nome e um valor, apesar de que o valor nesse caso é um mapeamento de colunas, ilustrada na Figura 16 b);
- Família de Coluna: uma família de coluna contém uma infinidade de linhas. Cada linha tem uma chave e um mapeamento de colunas, ordenados por seus nomes, ilustrada na Figura 16 c).
- Super Família de Coluna: similar a famílias de colunas, uma super família de colunas é um recipiente de super colunas, ilustrada na Figura 16 d);
- Espaço de Chave: um espaço de chave é uma coleção de famílias de colunas. Geralmente, agrupa todos os dados relacionados a um aplicativo.

Figura 16 – Exemplo das dimensões de dados do Cassandra



Fonte: Adaptado de (SILVA, 2011)

Cassandra Query Language (*CQL*) é a linguagem utilizada para consultas e tem uma sintaxe parecida com *SQL*. Diferente do *SQL*, *CQL* não suporta operações binárias, como junções (CHEBOTKO; KASHLEV; LU, 2015). Todas as consultas feitas no Cassandra são através de uma chave e que tenha uma coluna associada a essa chave como retorno,

apesar de métodos para obter múltiplos valores ou várias colunas de uma vez também são possíveis (SILVA, 2011).

Chebotko, Kashlev e Lu (2015) listam um conjunto de regras a serem seguidas no momento da consulta, que garatem eficiência e escabilidade:

- somente chaves primárias das colunas podem ser utilizadas nas consultas;
- todas as colunas das chaves de partição devem ser restritas por valores;
- todas, algumas ou nenhuma das colunas referentes à chave utilizada podem ser utilizadas na consulta;
- se uma coluna de chave de agrupamento for usada em um predicado de consulta, todas as colunas de chave de agrupamento que precedem essa coluna de agrupamento na definição da chave primária também devem ser usadas no predicado;
- se uma coluna de chave de agrupamento for restrita por intervalo (ou seja, busca de desigualdade) em um predicado de consulta, todas as colunas de chave de agrupamento que precedem esta coluna de agrupamento na definição da chave primária devem ser restritas por valores e nenhuma outra coluna de agrupamento pode ser usada no predicado.

Também é possível executar tarefas do MapReduce (DEAN; GHEMAWAT, 2008) no banco de dados, para processar consultas complexas, usando o Apache Hadoop. Para simplificar esta tarefa, o Cassandra também oferece suporte Pig (FOUNDATION, 2016), o que permite ao usuário escrever trabalhos complexos do MapReduce em uma linguagem de alto nível semelhante ao *SQL* (SILVA, 2011).

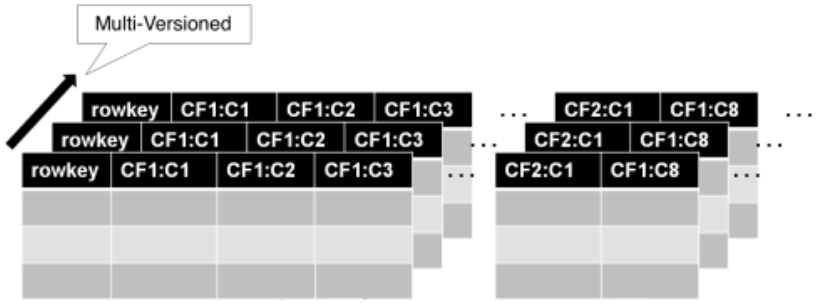
2.2.5.5 HBase

O HBase é um sistema de banco de dados distribuído *open source* orientado a colunas, com base no BigTable (CHANG et al., 2008), escrito em Java e utiliza como sistema de arquivos o HDFS (*Hadoop Distribuited Filesystem*), garantindo a tolerância a falhas quando armazenando grande quantidade de dados escassos (SILVA, 2011).

O modelo de dados desse sistema é bem parecido com o do Cassandra, visto que ambos foram baseados no BigTable (SILVA, 2011). Segundo a documentação do HBase (HBASE, 2017), os dados são armazenados em tabelas, que possuem linhas e colunas.

Uma tabela no HBase consiste em múltiplas linhas. Uma linha é composta por uma chave de linha com uma ou mais colunas associadas a ela; cada linha é organizada alfabeticamente pela chave de linha à medida que elas são armazenadas (SILVA, 2011), como ilustrado na Figura 17.

Figura 17 – Exemplo de armazenamento no HBase



Fonte: Adaptado de (SILVA, 2011)

O planejamento da chave da linha é muito importante, visto que o objetivo é armazenar dados de modo que as linhas relacionadas fiquem próximas uma das outras. Cada linha em uma tabela tem as mesmas famílias de colunas, embora uma determinada linha possa não armazenar nada em uma determinada família de colunas (HBASE, 2017), como pode ser visto na Figura 18.

As colunas contêm um histórico de versões de seus conteúdos, ordenados por um *timestamp*. As colunas são agrupadas em famílias de colunas e todas têm um prefixo comum: qualificador (SILVA, 2011). Um qualificador de coluna é adicionado a uma família de colunas para fornecer o índice de um determinado dado, sendo eles mutáveis e podem diferir entre as linhas (HBASE, 2017).

Na Figura 18 temos um exemplo de uma tabela armazenada no HBase. A primeira coluna é referente à chave da linha; a segunda refere-se ao timestamp do conteúdo. As três últimas colunas são famílias de colunas. Na família de coluna *anchor*, temos duas colunas diferentes, enquanto na família de coluna *contents* possui somente uma coluna.

Figura 18 – Exemplo de uma tabela armazenada no HBase

Row Key	Time Stamp	ColumnFamily contents	ColumnFamily anchor	ColumnFamily people
"com.cnn.www"	t9		anchor:cnnsi.com = "CNN"	
"com.cnn.www"	t8		anchor:my.look.ca = "CNN.com"	
"com.cnn.www"	t6	contents:html = "<html>..."		
"com.cnn.www"	t5	contents:html = "<html>..."		
"com.cnn.www"	t3	contents:html = "<html>..."		

Fonte: Adaptado de (HBASE, 2017)

As famílias de colunas colocam fisicamente um conjunto de colunas e seus valores,

muitas vezes por motivos de desempenho. Segundo a documentação do HBase (HBASE, 2017), cada família de colunas possui um conjunto de propriedades de armazenamento, tais como se seus valores devem ser armazenados em cache na memória, como seus dados são compactados ou suas chaves de linha são codificadas e outras.

Em um primeiro momento, o HBase é muito semelhante a um banco relacional (REDMOND; WILSON, 2012). Apesar de armazenar dados em tabelas, essas tabelas não funcionam como relações. Todas as linhas de uma tabela são identificadas por uma única chave (HARRISON, 2015). Uma das vantagens do HBase são características que outros bancos não têm, como compressão de dados, coleta de lixo (para dados antigos) e tabelas em memória (REDMOND; WILSON, 2012).

Apesar o HBase ser projetado para escalar, ele não retrocede. Segundo Redmond e Wilson (2012), esse detalhe pode ser um problema na hora de administrar os dados. Além disso, o HBase nunca é instalado sozinho; é sempre acompanhado pelo Hadoop, pelo sistema de arquivos distribuídos HDFS e por um serviço que auxilia na coordenação de nós chamado Zookeeper. Todo esse ambiente pode ser tanto uma vantagem quanto desvantagem: apesar de proporcionar uma grande robustez arquitetônica, esse ambiente sobrecarrega o administrador da aplicação por conta da manutenção constante (REDMOND; WILSON, 2012).

Concluindo, nesse capítulo abordamos tópicos essenciais para a fundamentação teórica necessária, a fim de gerar compreensão dos assuntos relacionados, principalmente no tocante de bancos não-relacionais, visto que é uma tecnologia mais recente.

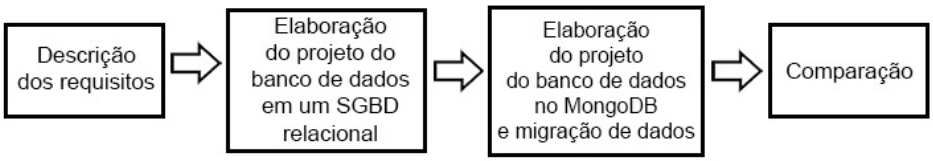
O próximo capítulo apresentará um estudo de caso como forma de exemplificar a teoria abordada nesse trabalho, um roteiro de migração de dados armazenados em um banco relacional para um banco não-relacional.

3 Estudo de caso

Esse capítulo irá abordar um estudo de caso, como forma de exemplificar como se daria a migração de um projeto de banco de dados relacional para um banco de dados não-relacional. O SGBD escolhido é o MongoDB versão 3.4, por ser um dos mais utilizados atualmente e possuir algumas semelhanças com o banco relacional.

Primeiramente será apresentado o domínio escolhido para executar esse estudo de caso, juntamente com seus requisitos principais. Após mapeado os requisitos, será mostrado como é feito um projeto de banco de dados e a criação do banco em um banco de dados relacional utilizando o MySQL Workbench versão 6.3. Em seguida, para efeitos de comparações, será feito um projeto de banco de dados e a criação do banco no MongoDB. Por último, será feita uma comparação entre os dois tipos de bancos, ressaltando suas vantagens e desvantagens. Essas etapas estão ilustradas na Figura 19.

Figura 19 – Etapas da migração de dados



Fonte: Acervo do autor

3.1 Descrição dos requisitos

O domínio escolhido para condução desse estudo de caso foi o de um hotel, ressaltando os principais requisitos descritos em linguagem textual em nível de usuário. O sistema de hotel serve para automatizar o processo de reserva e locação de quartos para hóspedes. Esse sistema serve para ser utilizado tanto no local do hotel quanto pela *internet*. O sistema deve manter os dados dos hóspedes que se cadastraram (seja pela *internet* ou no local), os dados dos quartos presentes e os dados das reservas feitas. Além de realizar cadastros, o sistema deverá oferecer consultas dos quartos e das reservas e cancelamentos de reserva.

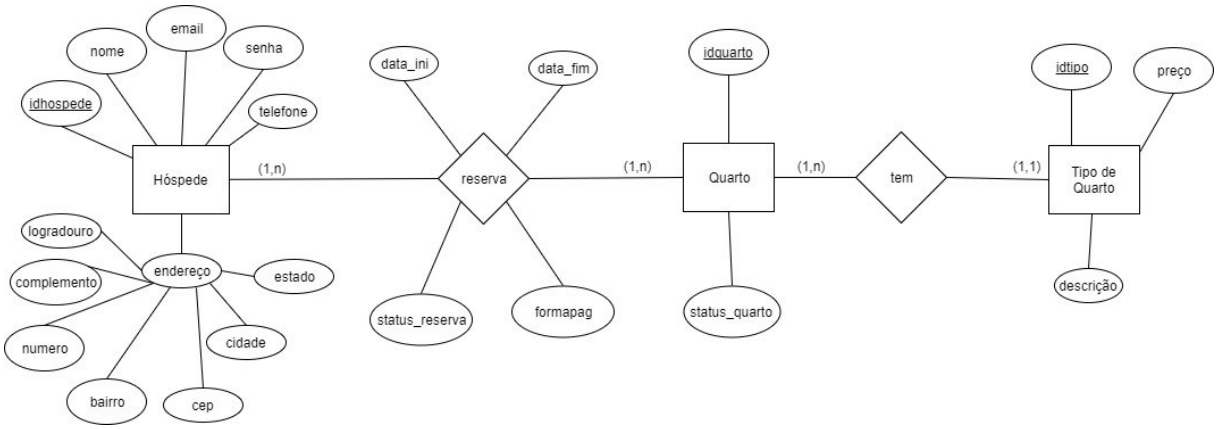
O cadastro de cliente terá de conter: o número de identificação do hóspede (que será gerado automaticamente), nome, endereço, e-mail, senha (para acesso pela *internet*). O cadastro dos quartos deverá ter: o número do quarto (utilizado para identificação), tipo do quarto, preço e status do quarto (livre, ocupado, reservado). O cadastro de reserva de

quarto deverá ser composto por: número do quarto, número de identificação do hóspede, situação da reserva (confirmada, a confirmar), data da entrada do hóspede, data da saída e o tipo de pagamento.

3.2 Elaboração do projeto do banco de dados em SGBD relacional

Como já foi abordado anteriormente, o desenvolvimento de um projeto de banco de dados se dá em três etapas. A primeira é gerar o projeto conceitual, geralmente utilizando o modelo ER. A Figura 20 ilustra o modelo ER do sistema a ser utilizado nesse estudo de caso.

Figura 20 – Modelo ER do sistema de gerenciamento de hotel utilizando a abordagem do Peter Chen



Fonte: Acervo do autor

Após criado o modelo ER, é feito o mapeamento desse modelo para o modelo relacional. Anteriormente, já foi citado que Elmasri e Navathe (2011) elencam sete passos para a criação do projeto lógico do banco. O resultado do modelo relacional do sistema utilizado está ilustrado no bloco de texto abaixo.

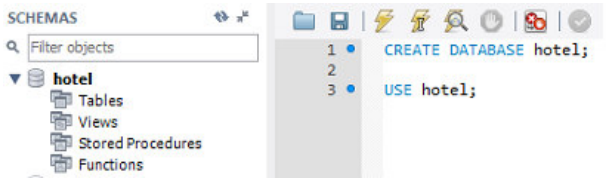
```
Hospede (idhospede, nome, email, senha, telefone, logradouro,
        complemento, numero, cep, cidade, estado)
PK(idhospede)

Quarto (idquarto, idtipo, status_quarto)
PK (idquarto)
FK (idtipo) referencia Tipo_Quarto

Tipo_Quarto (idtipo, descricao, preco)
PK (idtipo)
```

```
Reserva (idhospede, idquarto,
        data_ini, data_fim, status_reserva, formapag)
PK (idhospede, idquarto, data_ini)
FK (idhospede) referencia Hospede
FK (idquarto) referencia Quarto
```

Feitos os modelos ER e relacional, o próximo passo é a criação do banco de dados. Nos bancos de dados relacionais, a linguagem usada para a criação do banco de dados é a linguagem *SQL*. O SGBD utilizado para a criação do banco relacional foi o MySQL Workbench versão 6.3.

Primeiramente, antes de criar as tabelas com os dados, cria-se o banco de dados em si. No MySQL utiliza-se o comando *CREATE DATABASE* passando como parâmetro o nome do banco. Após criado, utiliza-se o termo *USE* e o nome do banco para especificar que o banco **hotel** esteja sendo utilizado. Na Figura 21, a primeira linha de comando está criando o banco **hotel** e a segunda linha está especificando que esse banco está sendo utilizado (que está em negrito na coluna ao lado).

Figura 21 – Criação do banco de dados **hotel** utilizando o MySQL Workbench 6.3



Fonte: Acervo do autor

Para criar uma tabela, utiliza-se o comando *CREATE TABLE*, passando como parâmetro o nome da tabela a ser criada e suas propriedades. Na Figura 22 temos um exemplo de criação das tabelas **hospede, quarto, reserva, tipo_quarto** com suas respectivas propriedades.

Para inserir dados no banco relacional, é utilizado o comando *INSERT* e utilizando como parâmetro a tabela a qual vai ser inserida os dados e os dados a serem inseridos. A Figura 23 exemplifica a inserção de dados nas tabelas **hospede**, **quarto** e **tipo**.

A tabela **reserva** foi populada posteriormente como visto na Figura 24. Isso se deve ao fato que a tabela possui duas chaves estrangeiras, uma vinda da tabela **hospede** e outra da tabela **quarto**.

Para consultar dados em um banco relacional, utiliza-se o comando *SELECT*, utilizando como parâmetros o nome e algum campo existente na tabela, como critério de seleção. Se não for colocada nenhuma condição no critério de seleção do comando *SELECT*, serão resgatados todos os dados da tabela consultada. A Figura 25 mostra um exemplo de

Figura 22 – Criação de tabelas no banco **hotel**

```
CREATE TABLE Hospede (
    idhospede INT PRIMARY KEY,
    nome VARCHAR(30),
    email VARCHAR(50),
    senha VARCHAR(20),
    telefone VARCHAR(12),
    logradouro VARCHAR(50),
    complemento VARCHAR(50),
    numero INT(3),
    cep VARCHAR(9),
    bairro VARCHAR(25),
    cidade VARCHAR(25),
    estado VARCHAR(2));

CREATE TABLE Tipo_Quarto (
    idtipo INT PRIMARY KEY,
    descricao VARCHAR(10),
    preco DECIMAL);

CREATE TABLE Quarto (
    idquarto INT PRIMARY KEY,
    idtipo INT,
    status_quarto VARCHAR(20),
FOREIGN KEY (idtipo) REFERENCES Tipo_Quarto (idtipo));

CREATE TABLE Reserva (
    idhospede INT,
    idquarto INT,
    data_ini DATE PRIMARY KEY,
    data_fim DATE,
    status_reserva VARCHAR(15),
    formapag VARCHAR(20),
FOREIGN KEY (idhospede) REFERENCES Hospede (idhospede),
FOREIGN KEY (idquarto) REFERENCES Quarto (idquarto));
```

Fonte: Acervo do autor

Figura 23 – Criação de dados nas tabelas **hospede, quarto, tipo**

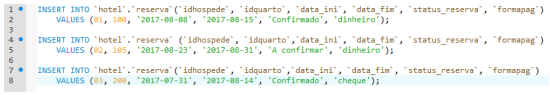
```
INSERT INTO `hotel`.`hospede`(`idhospede`, `nome`, `email`, `senha`,
    `telefone`, `logradouro`, `complemento`, `numero`, `cep`, `bairro`, `cidade`, `estado`)
    VALUES (01, 'Victor Reuben', 'victorreuben@gmail.com', '#somostodosvictor',
    '11-986314578', 'Avenida das Flores', 'Quadra B', 555, '90450-000', 'Arquipelago','Porto Alegre', 'RS');

INSERT INTO `hotel`.`hospede`(`idhospede`, `nome`, `email`, `senha`,
    `telefone`, `logradouro`, `complemento`, `numero`, `cep`,`bairro`, `cidade`, `estado`)
    VALUES (02, 'Dante Montini', 'dantemontini@fatosefuros.com.br', 'superhomem',
    '86-984567123', 'Rua Antares', 'Conjunto 2', 70, '60000-000', 'Aldeota','Fortaleza', 'CE');

INSERT INTO `hotel`.`hospede`(`idhospede`, `nome`, `email`, `senha`,
    `telefone`, `logradouro`, `complemento`, `numero`, `cep`, `bairro`, `cidade`, `estado`)
    VALUES (03, 'Will Sumner', 'willsumner@gmail.com', 'ameixa',
    '21-989631475', 'Rua Anita Garibaldi', 'Quadra J', 21, '90450-000', 'Lapa','Rio de Janeiro', 'RJ');

INSERT INTO `hotel`.`quarto`(`idquarto`, `tipo`, `status_quarto`) VALUES (100, 1, 'Livre');

INSERT INTO `hotel`.`quarto`(`idquarto`, `tipo`, `status_quarto`) VALUES (105, 2, 'Livre');

INSERT INTO `hotel`.`quarto`(`idquarto`, `tipo`, `status_quarto`) VALUES (110, 3, 'Livre');

INSERT INTO `hotel`.`quarto`(`idquarto`, `tipo`, `status_quarto`) VALUES (170, 4, 'Livre');

INSERT INTO `hotel`.`quarto`(`idquarto`, `tipo`, `status_quarto`) VALUES (200, 5, 'Livre');

INSERT INTO `hotel`.`tipo_quarto`(`idtipo`, `descricao`, `preco`) VALUES (1, 'Solteiro', '50.00');

INSERT INTO `hotel`.`tipo_quarto`(`idtipo`, `descricao`, `preco`) VALUES (2, 'Duplo', '70.00');

INSERT INTO `hotel`.`tipo_quarto`(`idtipo`, `descricao`, `preco`) VALUES (3, 'Casal', '100.00');

INSERT INTO `hotel`.`tipo_quarto`(`idtipo`, `descricao`, `preco`) VALUES (4, 'Triplo', '180.00');

INSERT INTO `hotel`.`tipo_quarto`(`idtipo`, `descricao`, `preco`) VALUES (5, 'Suite', '250.00');
```

Fonte: Acervo do autor

consulta de dados, utilizando como parâmetro **idquarto**. Como retorno para o usuário, tem-se todos os dados correspondente ao quarto de número de identificação 100.

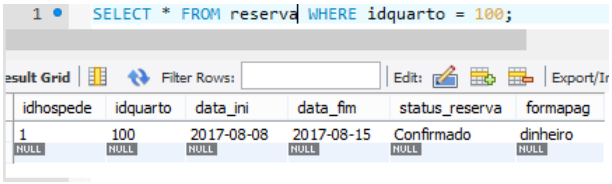
Para alterar dados no banco de dados relacional, utiliza-se o comando *UPDATE*, passando como parâmetro a tabela, os valores a serem alterados e a critério de identificação da linha a ser alterada. Na Figura 26, temos um exemplo de alteração de dados na tabela Quarto, alterando o campo **status_quarto** de "Livre"para "Reservado", visto que na tabela Reserva há registro de reservas para cada quarto.

Figura 24 – Criação de dados na tabela **reserva**



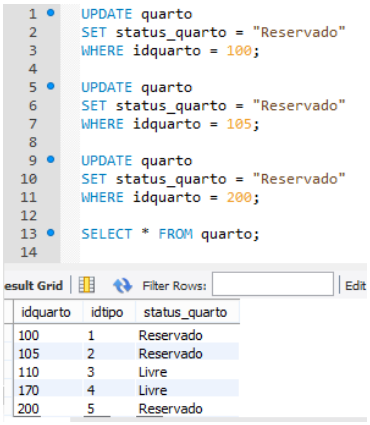
Fonte: Acervo do autor

Figura 25 – Consulta de dados utilizando como parâmetro **idquarto**



Fonte: Acervo do autor

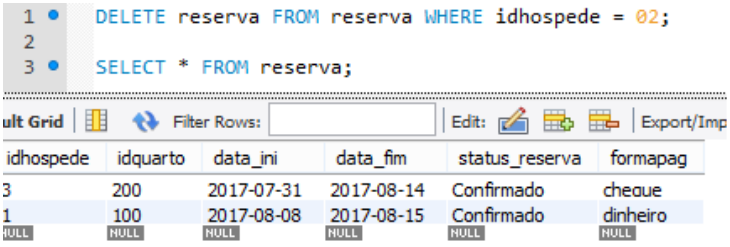
Figura 26 – Exemplo de uso do comando *UPDATE*



Fonte: Acervo do autor

Para excluir dados de uma tabela, há dois modos. O comando *DELETE* exclui todos os dados que estão associados ao parâmetro que foi passado no comando. Na Figura 27, temos um exemplo da exclusão da reserva que contenha o parâmetro idhospede = 02. Com isso, todos os dados da reserva do usuário Dante Montini será excluído.

Figura 27 – Exemplo de uso do comando *DELETE*

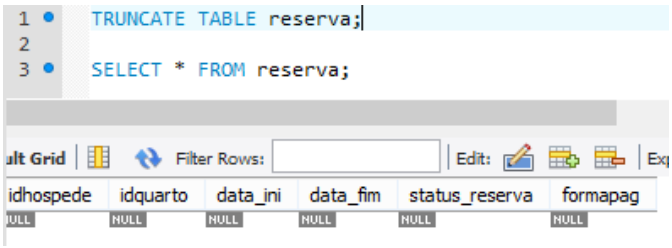


Fonte: Acervo do autor

Caso queira excluir todos os dados de uma tabela, no caso do MySQL, utiliza-se o

comando *TRUNCATE*. Esse comando exclui todos os dados da tabela, mas a mantém, como ilustrado na Figura 28, ao utilizar o comando para excluir todos os dados da tabela Reserva.

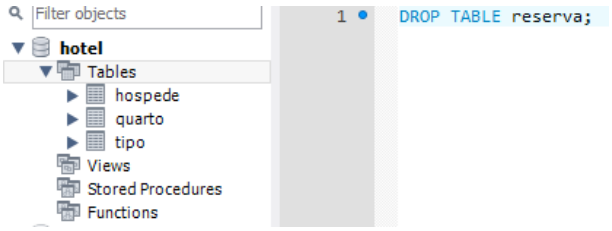
Figura 28 – Exemplo de uso do comando *TRUNCATE*



Fonte: Acervo do autor

Para excluir uma tabela inteira, utiliza-se o comando *DROP TABLE*. Ao ser executado, esse comando exclui toda a tabela, estando vazia ou não, sem modo de recuperação dos dados, como visto na Figura 29.

Figura 29 – Exemplo de uso do comando *DROP TABLE*



Fonte: Acervo do autor

3.3 Elaboração do projeto do banco de dados no MongoDB e migração de dados

O MongoDB, assim como outros bancos de dados não-relacionais, tem como característica ser *schema-free*. Sua programação é voltada para as consultas que a aplicação irá exigir (SILVA, 2011). Apesar dessa falta de normalização, os bancos não-relacionais ainda utilizam de casos de usos e, ao definir a cardinalidade dos entre as relações, estima-se o quanto o banco irá crescer (SILVA, 2011).

O MongoDB é um banco orientado a armazenamento de documentos. Por conseguinte, todos os dados migrados do banco relacional serão armazendados em documentos, seja esses dados originados de tabelas de entidades ou de relacionamentos.

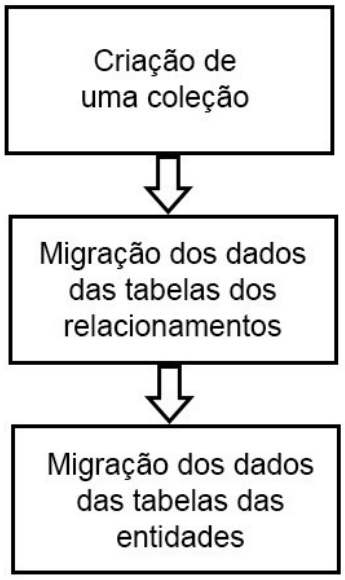
Como explicado no capítulo anterior, o MongoDB possui dois tipos de estrutura de armazenamento de dados: armazenamento por referência de documentos e armazenamento

por incorporação de documentos. O tipo de estrutura de dados utilizado para essa migração é a incorporação de documentos pois, essa estrutura é a mais indicada por oferecer um melhor desempenho nas operações de leitura, capacidade de solicitar e recuperar dados relacionados em uma única operação. (MONGODB, 2017).

Primeiramente, como disse Silva (2011), temos de definir a cardinalidade entre as relações. Ao se tratar de um sistema de hotel e dos requisitos apresentados no começo do capítulo, tem-se em mente que, no geral, um hóspede pode reservar um ou mais quartos, assim como um quarto pode ser reservado por um ou mais hóspedes. Logo, a cardinalidade entre os relacionamentos é N:M (muitos-para-muitos).

Esse processo de migração ocorrerá em três etapas. A primeira etapa é criar a coleção que irá armazenar os documentos no banco. A segunda etapa é migrar os dados das tabelas que representam relacionamentos entre entidades. E a terceira e última etapa é migrar os dados das as tabelas referentes a entidades e suas propriedades. A Figura 30 ilustra a ordem que essas etapas ocorrem.

Figura 30 – Etapas da migração de dados



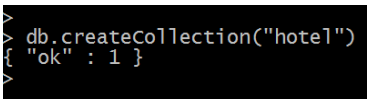
Fonte: Acervo do autor

É válido deixar claro que esse roteiro de migração foi exclusivamente desenvolvido para o MongoDB e utilizando a estrutura de incorporação de documentos. As três etapas de migração serão melhor desenvolvidas e explicadas nos próximos parágrafos.

No MongoDB, o banco Hotel e as quatro tabelas (Hospede, Quarto, Tipo_Quarto, Reserva) descritos anteriormente serão unidas em uma só coleção chamada **hotel**. Essa coleção irá armazenar documentos referentes aos usuários/hóspedes, aos quartos e às reservas do hotel. Ou seja, essa coleção irá armazenar todos os dados referentes ao sistema de hotel, fazendo o papel representado nas Figuras 21 e 22. Para criar essa coleção, somente

usa-se o método *db.createCollection()*. A Figura 31 ilustra a criação da coleção **hotel** utilizando o método *db.createCollection()*.

Figura 31 – Criação da coleção **hotel** utilizando o método *db.createCollection()*



Fonte: Acervo do autor

Cada tupla de cada tabela de relacionamento presente no banco será transformada em um documento. Para cada tupla presente nas tabelas de entidades também será criado um documento.

Após criada a coleção que irá armazenar os documentos, inicia-se a migração dos dados das tabelas referentes aos relacionamentos. Cada tupla de cada tabela de relacionamento se transformará em um documento. A estrutura de dado do documento é composta por campos e valores (MONGODB, 2017). Esse estrutura é denotada como **campo: valor**, sendo campo o equivalente à propriedades da tabela e valor o dado referente à propriedade na tupla.

Apesar do MongoDB criar um identificador próprio para o documento, a chave primária também irá funcionar como identificador. Essa função da chave primária permanece porque o valor do ObjectID do documento é uma *string* de 24 caracteres, composta de letras e números, e não é especificada no documento. Já as chaves primárias nos bancos relacionais são de menor tamanho, logo de mais fácil manipulação. Fora que elas já foram especificadas, visto que estamos somente migrando os dados.

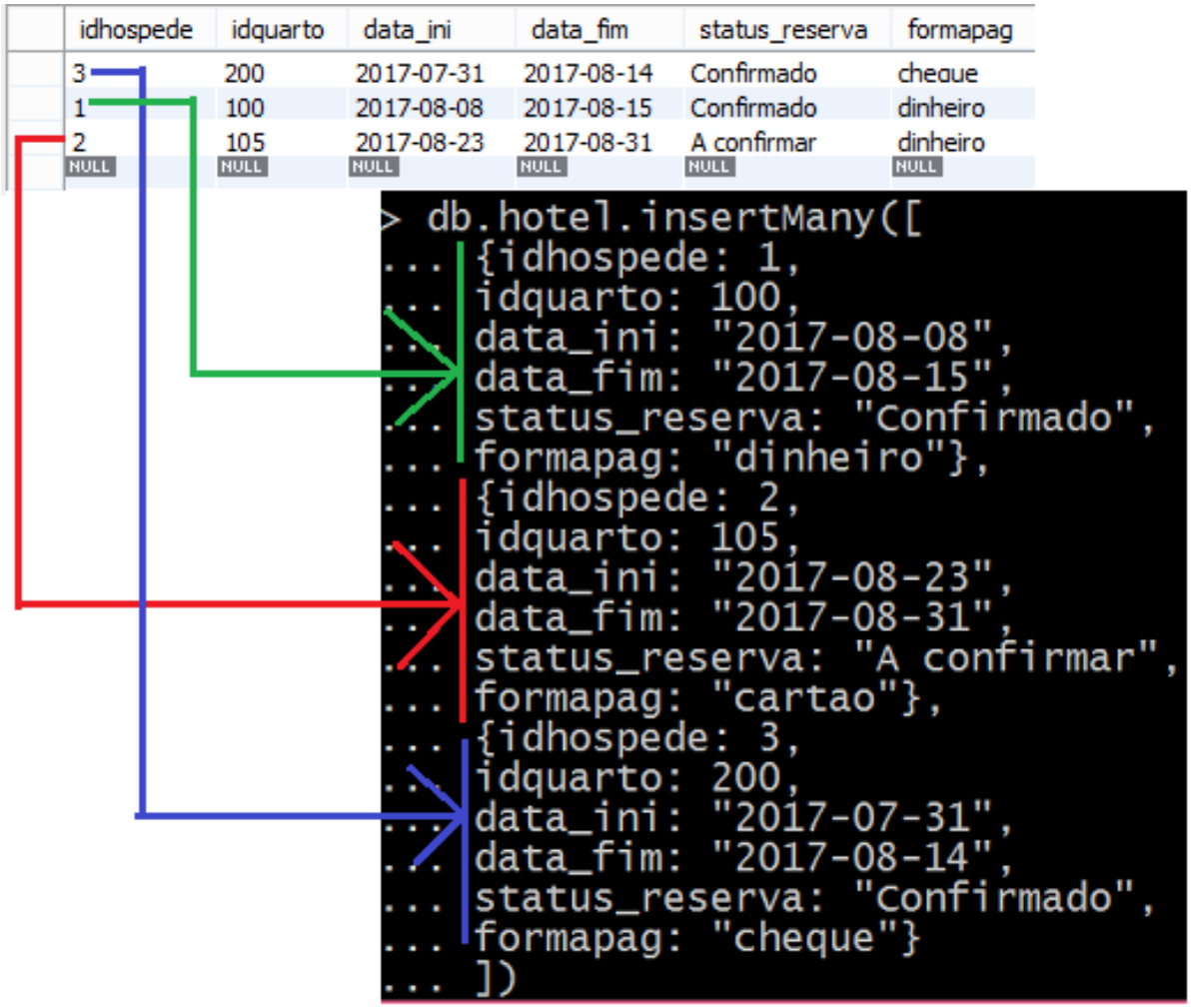
Na Tabela 2 temos a tabela referente ao relacionamento Reserva, com alguns dados armazenados. No documento no MongoDB, as propriedades **idhospede, idquarto, data_ini, data_fim, status_reserva, formapag** serão os campos que irão identificar os dados no documento: **idhospede** representa o número de identificação do hóspede que fez a reserva, **idquarto** representa o número do quarto reservado; **data_ini** e **data_fim** representam as datas que irá iniciar e terminar a reserva, respectivamente; **status_reserva**, o status da reserva; e **formapag**, a forma de pagamento. A Figura 32 ilustra essa etapa, sinalizando o documento que cada tupla originou.

Tabela 2 – Tabela referente ao relacionamento Reserva

idhospede	idquarto	data_ini	data_fim	status_reserva	formapag
1	100	2017-08-08	2017-08-15	Confirmado	dinheiro
2	110	2017-08-23	2017-08-31	A confirmar	cartao
3	200	2017-07-31	2017-08-14	Confirmado	cheque

Fonte: Acervo do autor

Figura 32 – Exemplo de migração de dados da tabela relacionamento Reserva



Fonte: Acervo do autor

A migração dos dados das tabelas pode ser feitas de duas maneiras: utilizando o método *db.collection.insertOne()* ou *db.collection.insertMany()*. Os métodos tem em comum o fato de que, se o documento não possuir um campo de identificação específico, o MongoDB adiciona um campo de identificação com um valor no documento (MONGODB, 2017).

Ao utilizar o método *db.collection.insertOne()*, só poderá criar/inserir um documento de cada vez. O método *db.collection.insertMany()*, responsável por criar/inserir vários documentos ao mesmo tempo na coleção, passando vetores como parâmetros. Esse método é o que vai ser utilizado e é mais indicado quando se está começando migrando o banco de dados visto que, ao inserir vários documentos ao mesmo tempo, há uma otimização de tempo (MONGODB, 2017). É aconselhado usar o método *db.collection.insertOne()* quando for inserir um novo documento que contenha relações entre documentos como, por exemplo, no momento de registrada uma nova reserva.

Na Figura 32 já apresentada, temos um exemplo da migração de dados das tabelas que representam o relacionamento Reserva utilizando o método *db.collection.insertMany()*.

Após a migração de todos os dados das tabelas de relacionamentos, migra-se os dados das tabelas de entidades. Para a migração das tabelas que representam entidades é utilizado o mesmo conceito da migração das tabelas de relacionamentos: cada tupla gerará um documento, com a estrutura dos dados compostas de campos e valores. Entretanto, essa migração se dará através dos seguintes passos:

- identifica chave estrangeira na tabela de relacionamento;
- identifica a tabela referenciada pela chave estrangeira;
- compara valor da chave estrangeira na tabela de relacionamento com o valor do campo no documento-relacionamento criado;
- identifica a tupla procurada na tabela de entidade;
- identifica o documento-relacionamento a ser inserido o novo documento;
- adiciona no documento-relacionamento um novo documento com os dados da tupla encontrada na tabela de entidade.

Para realizar essa etapa da migração, utilizaremos o método *db.collection.updateOne()*. O método *db.collection.updateOne(filter, update, options)* atualiza um documento somente. Já o método *db.collection.updateMany(filter, update, options)* atualiza vários documentos ao mesmo tempo. Ambos os métodos possuem os mesmos parâmetros. Segundo a documentação do MongoDB (2017), os parâmetros são:

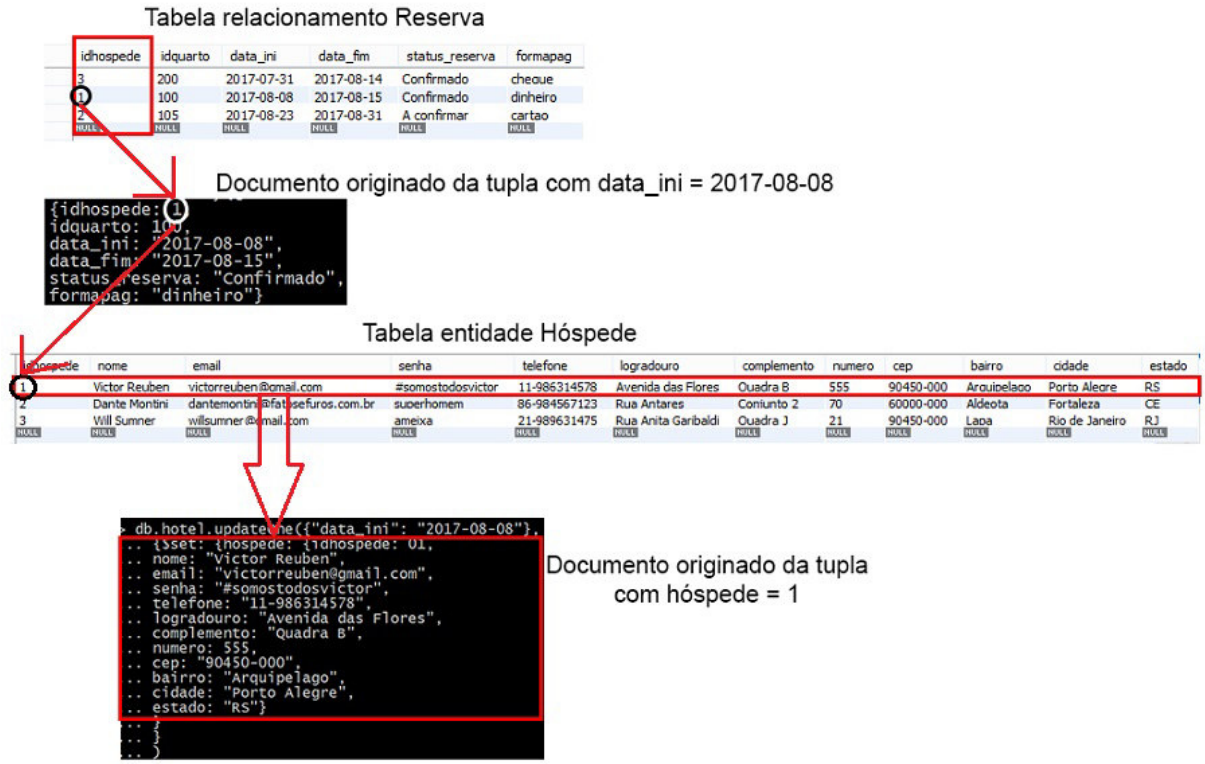
- *filter*: campo do critério de seleção para a alteração do documento;
- *update*: campo para as modificações a serem aplicadas;
- *options*: como o próprio nome diz, é opcional; geralmente usado como sinalizador se as alterações foram sucedidas ou não.

Na Figura 33 temos o passo a passo dessa etapa da migração:

- A tabela Reserva possui duas chaves estrangeiras: uma referenciando a tabela Hospede e a outra, a tabela Quarto;
- Seguindo a ordem das propriedades da tabela Reserva, trabalha-se primeiro com a tabela Hospede;

- Identificada a tabela Hospede, compara-se o valor da chave estrangeira da tupla da tabela Reserva com o valor de mesmo campo no documento-relacionamento referente; nesse caso, é o hóspede de identificação 1, no documento-relacionamento de data_ini: "2017-08-08";
- Encontra a tupla que contém os dados do hóspede de identificação 1, no caso é a primeira tupla;
- Adiciona no documento-relacionamento um novo documento com os dados da tupla encontrada na tabela de entidade.

Figura 33 – Exemplo de migração de uma tupla de dados da tabela entidade Hospede



Fonte: Acervo do autor

Para fazer consultas em documentos no MongoDB, utiliza-se o método *db.collection.find()*. Esse método sempre inclui o campo de identificação quando retorna os documentos, a não ser que seja especificado para isso não acontecer (MONGODB, 2017). Se no momento da consulta não for especificado nenhum parâmetro, terá como resultado todos os documentos pertencentes àquela coleção. A Figura 34 mostra um exemplo de consulta de todos documentos possuem o parâmetro **idquarto: 100**.

Além dos métodos de criação de coleção, criação/inserção de documentos e consulta, o MongoDB contém métodos específicos para manipulação dos dados contidos nos documentos. A manipulação dos dados nos documentos também pode ser feita através dos

Figura 34 – Consulta usando método *db.collection.find()* com parâmetro

Fonte: Acervo do autor

métodos *db.collection.updateOne()* e *db.collection.updateMany()*, explicados anteriormente. Acrescentando a essa lista, temos *db.collection.replaceOne()*.

Na Figura 35 é ilustrado a alteração dos dados de um documento utilizando o método *db.collection.updateOne()*. O documento a ser alterado será um que representa um relacionamento de reserva. Com uma reserva sinalizada para o quarto 100, iniciando na data "2017-08-08", o campo **status_quarto** tem de ser alterado de "Livre"para "Reservado". Então, passa-se como parâmetro o campo de identificação do quarto e as mudanças a serem feitas, no caso o estado do quarto. Esse mesmo método será repetido para todos os outros documentos, alterando a situação do quarto, tendo como resultado os documentos mostrados na Figura 36.

Figura 35 – Alteração de **status_quarto** utilizando o método *db.collection.updateOne()* no documento de **data_ini = "2017-08-08"**

Fonte: Acervo do autor

O método *db.collection.replaceOne()* é utilizado seja necessário substituir completamente um documento, que tem como parâmetros *filter, replacement, options*. O campo *filter* serve como critério de seleção do documento a ser substituído; o campo *replacement* refere-se ao novo documento; e o campo *options*, assim como nos métodos de atualização, é opcional (MONGODB, 2017).

Assim como para criação/inserção de documentos e manipulação de dados, para excluir documentos existem dois métodos: *db.collection.deleteOne()* e *db.collection.deleteMany()*. Diferente dos métodos de inserir e atualizar, esses métodos de exclusão possuem somente um parâmetro fixo, o critério de seleção do(s) documento(s) a ser(em) excluído(s).

No caso do método *db.collection.deleteMany()*, se não for passado nenhum parâme-

Figura 36 – Alteração do **status_quarto** em todos os documentos

Fonte: Acervo do autor

tro, irá excluir todos os documentos da coleção (MONGODB, 2017). No caso do método *db.collection.deleteOne()*, se não for especificado o critério de seleção, será excluído o primeiro documento que for encontrado na coleção (MONGODB, 2017).

Abaixo, na Figura 37 temos um exemplo de exclusão de somente um documento. Ao desejar excluir o documento que contém os dados da reserva do hóspede Dante Montini, foi passado como parâmetro a data do início de sua reserva. O servidor retorna para o usuário se o documento foi realmente excluído e a quantidade de documentos excluídos. Ao executar o método *db.collection.find()*, note que ao excluir a reserva, foram excluídos também os dados do hóspede que havia feito a reserva e o quarto reservado. Esse detalhe será discutido mais a frente.

Figura 37 – Exclusão de documento utilizando o método *db.collection.deleteOne()*

Fonte: Acervo do autor

Já a Figura 38 ilustra a exclusão de todos os documentos presentes na coleção, utilizando o método *db.collection.deleteMany()*. O método tem como retorno a confirmação

da exclusão dos documentos e quantos foram deletados. O método *db.collection.find()* foi chamado e não retornou nenhum documento, visto que todos foram excluídos.

Figura 38 – Exclusão de documento utilizando o método *db.collection.deleteMany()*

```
> db.hotel.deleteMany({})
{ "acknowledged" : true, "deletedCount" : 2 }
> db.hotel.find()
>
```

Fonte: Acervo do autor

Caso o usuário não queria mais usar uma coleção, usa-se o método *db.collection.drop()*. Esse método exclui toda a coleção, esta estando vazia ou não. Na Figura 39 temos um exemplo da utilização desse método, ao deletar toda a coleção **hotel**.

Figura 39 – Exclusão da coleção **hotel** usando o método *db.collection.drop()*

```
> db.hotel.drop()
true
>
```

Fonte: Acervo do autor

Como dito no começo da seção, nesse roteiro foi utilizado o armazenamento por incorporação de documentos. Apesar dessa abordagem ser mais eficiente, ela possui algumas desvantagens.

A principal delas é a perda dos dados dos documentos incorporados no documento-pai caso esse seja excluído, como foi ilustrado na Figura 37. Em toda a documentação do MongoDB não existe um método para salvar os documentos incorporados aos documentos-pai. Trazendo pro estudo de caso, ao deletar um documento que represente um relacionamento, os documentos com dados das entidades que fazem parte do relacionamento serão excluídos e não tem como salvar esses dados antes, a não ser que se crie logo um documento com esses dados, o que gera a desvantagem a seguir.

Esse tipo de estrutura causa repetição de dados, fazendo com que o banco fique muito grande: exemplificando no roteiro do estudo de caso, toda vez que for criado um documento originado de um relacionamento, os dados das entidades que fazem parte desse relacionamento também serão criados. No caso do estudo de caso, o banco é pequeno, mas se for pensar em um banco maior, isso pode gerar outros problemas, como perda de performance e necessidade maior de armazenamento.

A documentação do MongoDB (2017) diz que, apesar da estrutura de incorporação ser mais eficiente na hora da leitura, a estrutura por referência evita repetição de dados: como foi explicado na seção 2.2.5.1, a estrutura por referência é mais indicada para representar relações de muito para muitos e quando a incorporação de documentos resultar

em duplicação de dados (MONGODB, 2017) e esses dois quesitos estão presentes no estudo de caso.

No início dessa seção, estabelecemos que a cardinalidade entre as relações é N:M (muitos-para-muitos), o que já corresponde a um dos problemas descritos. Outro fato correspondente aos problemas descritos é que, ao se criar um documento novo de reserva para um hóspede que já havia se hospedado antes, esses dados do hóspede serão duplicados ao serem incorporados nesse novo documento de reserva. O mesmo raciocínio se aplica aos quartos: toda vez que for criado um documento de reserva para quarto X, os dados desse quarto serão duplicados ao serem incorporados no novo documento.

3.4 Comparação entre o MySQL e o MongoDB

Após o desenvolvimento dos projetos e criação dos SGBDs em *SQL* e MongoDB, essa seção irá abordar uma comparação entre os dois tipos de SGBDs.

A diferença entre os dois tipos de SGBDs já se inicia na criação do projeto do banco. O banco relacional já tem um padrão em três etapas: modelo conceitual, modelo lógico e modelo físico. O banco não-relacional ainda carece dessa padronização e essa é uma das maiores dificuldades enfrentada pela comunidade. Como explicado anteriormente, a grande variedade de armazenamentos dificulta essa padronização e temos esse obstáculo no próprio estudo de caso.

O MongoDB oferece dois tipos de armazenamentos de documento: por referência de documentos e por incorporação de documentos. O roteiro descrito no estudo de caso foi feito utilizando a incorporação de documentos, porém esse mesmo roteiro não se aplica ao armazenamento por referência de documentos.

Apesar de ser mais eficiente se comparada com o armazenamento por referência, a incorporação de documentos possui algumas desvantagens que foram encontradas durante o desenvolvimento do roteiro. Como explicadas antes, as principais desvantagens dessa abordagem são a perda dos documentos incorporados caso o documento-pai seja excluído e a repetição de dados.

Essas desvantagens vêm do fato que todos os documentos são guardados em uma coleção gigante, onde os documentos são armazenados sem organização alguma. No banco relacional todos os dados são armazenados em tabelas diferentes, que facilita o controle da manipulação dos dados.

No tocante dos comandos da linguagem *SQL* e os métodos utilizados no MongoDB, percebe-se que semanticamente são similares, motivo esse que o MongoDB foi utilizado nesse estudo de caso. A Tabela 3 ilustra a comparação entre os comandos do *SQL* e os métodos do MongoDB.

Tabela 3 – Comparação dos comandos *SQL* e métodos do MongoDB

Função	Linguagem SQL	MongoDB
Criar banco	CREATE DATABASE	db.createCollection()
Criar tabelas	CREATE TABLE	db.collection.insertOne()
		db.collection.insertMany()
Inserir dados	INSERT	db.collection.insertOne()
		db.collection.insertMany()
Atualizar dados	UPDATE	db.collection.updateOne()
		db.collection.updateMany()
		db.collection.replaceOne()
Consultar dados	SELECT	db.collection.find()
Excluir dados	DELETE	db.collection.deleteOne()
	TRUNCATE	db.collection.deleteMany()
Excluir tudo	DROP TABLE	db.collection.drop()

Fonte: Acervo do autor

Logo no começo da tabela é ilustrado uma divergência entre os dois bancos: o processo de criação dos bancos de dados. No banco relacional, primeiro cria-se o banco em si e depois as tabelas, onde os dados serão armazenados e ordenados. As tabelas que representam relacionamentos entre entidades têm como ligação entre elas chaves estrangeiras, que geralmente são as chaves primárias das tabelas de entidade.

No MongoDB, todo o banco e todas essas tabelas são convertidas em uma única coleção, onde serão armazenados todos os dados em forma de documentos. Nesse caso, a coleção **hotel** abrange o papel do banco relacional Hotel e os papéis das tabelas Hospede, Quarto, Tipo_Quarto e Reserva em si. Apesar dessa diferença, o conceito de chaves primárias e estrangeiras permanece no MongoDB, quando se tratando do armazenamento por referência.

A vantagem das tabelas nos bancos relacionais é que os dados serão organizados de acordo com seus atributos e propriedades, sejam dados de uma entidade ou de um relacionamento. Porém quanto mais dados forem adicionados, maior a tabela vai crescer verticalmente, podendo chegar a um ponto que prejudique a aplicação.

Já o MongoDB foi projetado para tratar um grande volume de dados, que são armazenados em documentos. Entretanto, esses documentos não possuem uma organização precisa como as tabelas no banco relacional, como dito anteriormente. Em uma mesma coleção, pode-se armazenar documentos com dados de entidades e dados de um relacionamento e nenhum deles possui um rótulo de identificação se aquele documento contém dados referentes a uma entidade ou a um relacionamento.

Outra grande diferença são os métodos de inserção de dados. No *SQL* só existe um comando para uma dessas ações, que é o comando *INSERT*. Para inserir dados na tabela,

é necessário passar como parâmetros desse comando: primeiramente os campos a serem preenchidos e depois os seus valores. Essa inserção de dados é feita uma tupla de cada vez, que demanda uma grande quantidade de tempo, principalmente no momento que está se inserindo vários dados no banco.

No MongoDB existem dois tipos para inserção de dados, que também é utilizado para criação de documentos: um voltado para inserir/criar um documento somente (*db.collection.insertOne()*) e outro para inserir/criar vários documentos ao mesmo tempo (*db.collection.insertMany()*). Essa possibilidade de inserir/criar vários documentos de uma vez garante uma otimização no momento da população dos dados em documentos na coleção ou, como sugere esse trabalho, na migração de dados.

No banco relacional utiliza-se o comando *UPDATE* para atualizar/alterar os dados em uma tabela, que tem como parâmetro a tabela, o campo a ser atualizado e um critério de seleção. Assim como o parâmetro *INSERT*, esse comando só altera uma tupla de cada vez, o que gera a mesma desvantagem do comando de inserção.

Já no MongoDB, essa atualização/alteração de dados pode ser feita de duas maneiras, assim como o método de criação/inserção de documentos: um atualizar/alterar um documento somente (*db.collection.updateOne()*) e outro para atualizar/alterar vários documentos ao mesmo tempo (*db.collection.updateMany()*). Dependendo da alteração a ser feita, alterar vários documentos ao mesmo tempo também gera uma otimização de tempo. No *SQL*, o comando mais próximo do método *db.collection.updateMany()* é o *ALTER TABLE*, que altera toda a coluna de uma tabela de uma vez. Esse método também se assemelha ao método *db.collection.deleteMany()*, visto que também é possível ser usado para excluir uma coluna da tabela.

As funções de exclusão de dados e excluir tudo do banco são funções que não divergem muito entre os dois tipos de bancos. No *SQL*, o comando *DELETE* serve para deletar dados específicos a partir de um dado parâmetro e o comando *TRUNCATE* para excluir todos os dados de uma tabela. No MongoDB, pode-se excluir somente um documento, utilizando um parâmetro de critério (*db.collection.deleteOne()*), ou muitos documentos (*db.collection.deleteMany()*). Em ambos os bancos, o resultado esperado influencia na utilização das funções.

No caso do banco relacional, ao se excluir uma tupla de dados, todas as outras tuplas permanecem. Já se tratando ao armazenamento por incorporação de documentos do MongoDB, ao se excluir um documento, todos os documentos incorporados a ele também serão excluídos, sem escolha de salvar ou não aqueles dados.

A função de excluir tudo de um banco de dados tem a mesma semântica tanto no *SQL* quando no MongoDB. Enquanto no *SQL* utiliza-se o comando *DROP TABLE*, passando como parâmetro a tabela a ser excluída, no Mongo se utiliza *db.collection.drop()*.

Apesar da mesma semântica, o resultado entre eles é diferente.

No *SQL*, o comando *DROP TABLE* irá excluir somente a tabela indicada. Enquanto no MongoDB ao chamar o método *db.collection.drop()*, toda a coleção será excluída, ou seja, todos os documentos e seus dados serão excluídos. Nesse cenário, a linguagem *SQL* exerce vantagem sobre o banco não-relacional.

Nesse capítulo abordamos um estudo de caso como forma de exemplificar o roteiro de migração de dados entre um banco relacional para um banco não-relacional, utilizando o MongoDB como alvo da migração. O próximo capítulo apresentará a conclusão desse trabalho, bem como as dificuldades encontradas e as propostas para trabalhos futuros.

4 Conclusão

Esse trabalho apresentou o desenvolvimento de um roteiro de migração de dados armazenados em um banco de dados relacional para um banco de dados não-relacional. Esse roteiro tem como objetivo facilitar a migração dos dados de forma confiável e mantendo sua integridade.

No primeiro capítulo foi feita uma introdução sobre o foco do trabalho proposto, que é a migração de dados. Primeiramente, foi feito um breve contexto do cenário que influenciou no surgimento dos bancos não-relacionais. Em seguida, comentou-se sobre alguns motivos que poderiam levar alguns aplicativos a necessitar fazer a migração de dados. Após esses motivos, também foi abordado algumas dificuldades encontradas para executar essa migração de dados. Por último, comentou-se sobre o objetivo geral e os objetivo específicos do trabalho, assim com sua organização.

O segundo capítulo foi abordada a fundamentação teórica necessária para entendimento de conceitos que seriam abordados ao longo do trabalho. Esse capítulo foi dividido em duas seções macros. A primeira macro seção foca nos bancos de dados relacionais. Nessa seção aborda como se dá a criação de um projeto de banco de dados em bancos relacionais e suas principais características.

A segunda macro seção foca nos bancos de dados não-relacionais. Assim com a seção anterior, foi discorrido a criação de um projeto de bancos de dados em bancos não-relacionais, assim como suas principais características. Além desses dois tópicos, foi apresentado as principais arquiteturas de armazenamento desse tipo de banco e também uma pesquisa sobre os bancos não-relacionais *open-source* mais utilizados atualmente, visando escolher o melhor SGBD a ser utilizado no estudo de caso. O MongoDB foi escolhido para o desenvolvimento do estudo de caso por ter similaridades ao banco relacional e o roteiro aqui apresentado é totalmente voltado para esse SGBD, que é de fácil usabilidade.

Para exemplificar o roteiro proposto, foi elaborado um estudo de caso baseado no domínio de um hotel, explanado no terceiro capítulo. Após descrever os principais requisitos desse domínio, foi explicado e ilustrado o projeto de banco de dados *SQL*, utilizando o MySQL Workbench. Em seguida, foi apresentado o projeto de banco de dados no MongoDB, seguido da explicação da proposta e demonstrando como seria essa migração. Por último, foi feita uma comparação entre os dois tipos de SGBDs, ressaltando as vantagens e desvantagens de cada um.

As etapas referentes a descrição dos requisitos e a elaboração do projeto em banco relacional utilizando o MySQL foram de fácil desenvolvimento, visto que são áreas conhecidas. A etapa sobre a elaboração do projeto no MongoDB e a migração já foram um

pouco trabalhosas, por conta da necessidade configurar o SGBD e aprender a utilizá-lo, visto que é uma tecnologia mais recente no mercado. Entretanto, no geral, o trabalho teve um bom desenvolvimento e creio que cumpriu sua pretensão.

Foram encontradas dificuldades na pesquisa sobre a documentação de alguns bancos não-relacionais. Por alguns ainda estarem em fase de desenvolvimento, a documentação é bem escassa e esse detalhe influenciou bastante na escolha do MongoDB como alvo da migração. Também encontrou-se dificuldades na abordagem de armazenamento escolhida para esse roteiro, como a exclusão total de documentos incorporados a outros e duplicação de dados.

Como trabalhos futuros, sugere-se adaptar esse roteiro de migração para o outro tipo de armazenamento que o MongoDB fornece, o armazenamento por referência. Também é pensado desenvolver um teste que compare a eficiência do roteiro proposto com o adaptado para o outro armazenamento. E não é descartada a ideia adaptá-lo também para outros tipos de bancos não-relacionais, como CouchDB e Cassandra.

Referências

BRITO, R. W. Bancos de dados nosql x sgbds relacionais: análise comparativa. *Faculdade Farias Brito e Universidade de Fortaleza*, p. 6, 2010.

BSON. *BSON - Binary JSON*. 2011. <http://bsonspec.org/>. Acesso em 11 de junho de 2017.

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

CATTELL, R. Scalable sql and nosql data stores. *Acm Sigmod Record*, ACM, v. 39, n. 4, p. 12–27, 2011.

CHANG, F. et al. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, ACM, v. 26, n. 2, p. 4, 2008.

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

COSTA, A. V. da; VILAIN, P.; MELLO, R. dos S. Uma camada para o mapeamento de instruções sql dml para o banco de dados nosql chave-valor voldemort. 2016.

DEAN, J.; GHEMAWAT, S. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, ACM, v. 51, n. 1, p. 107–113, 2008.

DECANDIA, G. et al. Dynamo: amazon's highly available key-value store. *ACM SIGOPS operating systems review*, ACM, v. 41, n. 6, p. 205–220, 2007.

DIANA, M. D.; GEROSA, M. A. Nosql na web 2.0: Um estudo comparativo de bancos não-relacionais para armazenamento de dados na web 2.0. In: *IX Workshop de Teses e Dissertações em Banco de Dados*. [S.l.: s.n.], 2010. v. 9.

ELMASRI, R.; NAVATHE, S. *Database systems, 4th edition*. [S.l.]: Pearson Education Boston, MA, 2004. 1330 p.

ELMASRI, R.; NAVATHE, S. *Database systems, 6th edition*. [S.l.]: Pearson Education Boston, MA, 2011. 770 p.

FOUNDATION, A. S. *Welcome to Apache Pig!* 2016. <http://pig.apache.org/>. Acesso em 11 de junho de 2017.

HAN, J. et al. Survey on nosql database. In: IEEE. *Pervasive computing and applications (ICPCA), 2011 6th international conference on*. [S.l.], 2011. p. 363–366.

HARRISON, G. *Next Generation Databases: NoSQL and Big Data*. [S.l.]: Apress, 2015. 235 p.

HBASE, T. A. *Apache HBase^{TM} Reference Guide*. 2017. <http://hbase.apache.org/apache_hbase_reference_guide.pdf>. Acesso em 11 de junho de 2017.

JSON. *Introducing JSON*. 2011. <http://json.org/>. Acesso em 11 de junho de 2017.

LAKSHMAN, A.; MALIK, P. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, ACM, v. 44, n. 2, p. 35–40, 2010.

LI, Y.; MANOHARAN, S. A performance comparison of sql and nosql databases. In: IEEE. *Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference on*. [S.l.], 2013. p. 15–19.

LIANG, D.; LIN, Y.; DING, G. Mid-model design used in model transition and data migration between relational databases and nosql databases. In: IEEE. *Smart City/SocialCom/SustainCom (SmartCity), 2015 IEEE International Conference on*. [S.l.], 2015. p. 866–869.

MCCREARY, D.; KELLY, A. *Making sense of NoSQL*: A guide for managers and the rest of us. [S.l.]: Manning, 2014. 314 p.

MONGODB, I. *Data Model Design*. 2017. <https://docs.mongodb.com/manual/core/data-model-design/>. Acesso em 11 de junho de 2017.

MONIRUZZAMAN, A.; HOSSAIN, S. A. Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. *International Journal of Database Theory and Application*, v. 6, n. 4, 2013.

MUHAMMAD, Y. Evaluation and implementation of distributed nosql database for mmo gaming environment. *Uppsala University*, 2011.

OREND, K. Analysis and classification of nosql databases and evaluation of their ability to replace an object-relational persistence layer. *Architecture*, p. 100, 2010.

REDMOND, E.; WILSON, J. R. *Seven databases in seven weeks: a guide to modern databases and the NoSQL movement*. [S.l.]: Pragmatic Bookshelf, 2012. 338 p.

SILBERSCHATZ, A. et al. *Database system concepts, 6th edition*. [S.l.]: McGraw-Hill, 2011. 1349 p.

SILVA, C. A. R. F. O. *Data modeling with NoSQL: How, when and why*. 95 p. Dissertação (Mestrado em Informática e Engenharia da Computação) — Faculdade de Engenharia da Universidade do Porto, Porto, 2011.

STRAUCH, C.; SITES, U.-L. S.; KRIHA, W. Nosql databases. *Lecture Notes, Stuttgart Media University*, 2011.

VALE, F.; ROCHA, L. Nosqlayer: a framework for migrating relational datasets to nosql models. *Revista de Iniciação Científica*, v. 14, n. 3, 2014.