LAÍS PEREIRA OLIVEIRA

MIGRAÇÃO DE MODELO DE DADOS RELACIONAL PARA NOSQL

Monografia apresentada ao PECE – Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo como parte dos requisitos para conclusão do curso de MBA em Tecnologia de Software.

São Paulo
2013

LAÍS PEREIRA OLIVEIRA

MIGRAÇÃO DE MODELO DE DADOS RELACIONAL PARA NOSQL

Monografia apresentada ao PECE – Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo como parte dos requisitos para a conclusão do curso de MBA em Tecnologia de Software.

Área de Concentração: Tecnologia de Software

Orientador: Prof. Dr. Kechi Hirama

São Paulo
2013

FICHA CATALOGRÁFICA

Oliveira, Laís Pereira
Migração de modelo de dados relacional para NoSQL / L.P. Oliveira. -- São Paulo, 2013.
65 p.

Monografia (MBA em Tecnologia de Software) – Escola Politécnica da Universidade de São Paulo. Programa de Educação Continuada em Engenharia.

1.Banco de dados 2.NoSQL I.Universidade de São Paulo. Escola Politécnica. Programa de Educação Continuada em Engenharia II.t.

DEDICATÓRIA

Dedico este trabalho aos meus pais e minha irmã que sempre me apoiam e são a felicidade da minha vida.

AGRADECIMENTOS

Aos meus pais, minha irmã e Luiz que são minha grande inspiração, incentivam sempre meu crescimento profissional e me dão forças para não desistir dos meus objetivos.

Ao professor Kechi Hirama, pela paciência, confiança, orientação e apoio no desenvolvimento deste trabalho.

Aos meus amigos e colegas de trabalho da Summa, Cardif, Avenue Code e Wal-Mart, pelo apoio.

RESUMO

Este trabalho apresenta um estudo sobre a migração de banco de dados relacional para NoSQL. A partir dos trabalhos de Schram, Anderson (2012) e Cruz et al. (2011) foi possível identificar um processo para realizar a migração do modelo de dados bem como as principais dificuldades e decisões envolvidas.

Os bancos de dados NoSQL estão ganhando cada vez mais popularidade e consequentemente o interesse em migrar sistemas existentes que utilizam bancos de dados relacionais também é cada vez maior. Diversas soluções de banco de dados NoSQL foram criadas para suprir as necessidades não atendidas pelos bancos de dados relacionais. Contudo, devido a especificidade dos bancos de dados NoSQL, realizar a migração envolve a difícil decisão de escolher o banco de dados mais apropriado e saber como construir o modelo de dados de forma a atender os requisitos de negocio e de sistema.

ABSTRACT

This work presents a study concerning the migration of relational databases to NoSQL databases. Based on the work of Schram, Anderson (2012) and Cruz et al. (2011) a process to accomplish the data model migration as well as the main difficulties and decisions involved could be identified.

The NoSQL databases are gaining more popularity and consequently the interest in migrating existing systems that currently rely on relational databases is also increasing. Several NoSQL databases solutions were created to meet the flaws of relational databases. However, due to the specificity of NoSQL databases, perform the migration involves the difficult decision of choosing the most appropriate database and also build the data model to meet the business and system requirements.

LISTA DE ILUSTRAÇÕES

Pág.

LISTA DE ABREVIATURAS E SIGLAS

ORM	Object Relational Mapping
CAP	Consistency Availability Partition Tolerance
ACID	Atomicity, Consistency, Isolation, Durability
SQL	Structured Query Language

SUMÁRIO

Pág.

1. INTRODUÇÃO

Este capitulo apresenta as motivações, objetivo e justificativa de realizar este trabalho.

1.1 Motivações

Os bancos de dados relacionais vêm sido utilizados com sucesso por mais de 20 anos fornecendo persistência, controle de concorrência e um mecanismo de integração de aplicações.

O aumento cada vez maior dos dados na Web é um problema enfrentado por grandes empresas como Google e Amazon. Estas empresas têm que lidar com tera/peta bytes de dados e as requisições devem ser respondidas sem latência notável. Estas empresas foram pioneiras na utilização de banco de dados NoSQL, tendo criado o próprio sistema de banco de dados.

O fator que impulsionou as mudanças na forma de armazenamento dos dados foi a necessidade de manipular grande quantidade de dados de forma distribuída.

A abordagem NoSQL, defende a utilização dos bancos de dados relacionais e não relacionais em conjunto, no mesmo sistema ou até na mesma aplicação, visando fazer uso das tecnologias que sejam mais apropriadas para o negócio (Sadalage & Fowler, 2012).

A utilização de bancos de dados não relacionais é cada vez mais uma realidade para aplicações de menor porte, já que os todos buscam de alguma forma: escalabilidade, desempenho, menor custo e agilidade no desenvolvimento do software.

As decisões que devem ser tomadas para a utilização destes bancos de dados são muitas e por ser uma mudança recente, não há padronização.

Assim, uma das formas de enfrentar este cenário é através do estudo da migração de modelos de relacionais para não relacionais, pois através deste estudo é possível compreender os conceitos dos diversos modelos de dados, as preocupações e decisões envolvidas nesta nova abordagem.

1.2 Objetivo

O objetivo do trabalho apresentar um processo de migração de um modelo de dados relacional para um modelo de dados com um banco de dados NoSQL. Para que este objetivo seja cumprido são apresentados os passos para realizar a migração para NoSQL bem como uma aplicação do processo.

1.3 Justificativas

Existem diversos trabalhos que tratam do assunto NoSQL. Dentre eles os trabalhos de Schram & Anderson (2012) e Cruz et al. (2011) relatam a migração de banco de dados relacional para NoSQL.

O trabalho de Schram & Anderson (2012) descreve as dificuldades e preocupações desta migração de forma detalhada, porém não há discussão profunda das alternativas para a modelagem dos dados e para a escolha do tipo de banco de dados utilizado.

Já o trabalho de Cruz et al. (2011) relata a migração de uma forma menos detalhada que também não discute profundamente as decisões tomadas, mas pode-se observar que existe um processo informal sendo utilizado nesta migração.

1.4 Estrutura do Trabalho

O Capítulo 1 INTRODUÇÃO apresenta as motivações, o objetivo, as justificativas e a estrutura do trabalho.

O Capítulo 2 BANCOS DE DADOS NOSQL apresenta conceitos e visão geral dos bancos de dados NoSQL, bem como características das suas categorias e as

respectivas técnicas de modelagem de dados. No fim do capitulo, são apresentadas também algumas limitações dos bancos de dados relacionais e pontos de atenção ao se trabalhar com bancos de dados NoSQL.

O Capítulo 3 PROCESSO DE MIGRAÇÃO PARA NOSQL apresenta as fases do processo para realizar a migração extraídas principalmente dos trabalhos de Schram & Anderson (2012) e Cruz et al. (2011).

O Capítulo 4 APLICAÇÃO DO PROCESSO DE MIGRAÇÃO descreve um exercício de migração de um sistema fictício que aplica o processo e evidencia a importância das alternativas de modelagem de dados apresentadas nos capítulos anteriores.

O Capítulo 5 CONSIDERAÇÕES FINAIS descreve a conclusão do trabalho, as dificuldades encontradas, as contribuições e os trabalhos futuros.

REFERÊNCIAS relaciona os trabalhos citados.

APÊNDICE A apresenta os requisitos de negócio do sistema utilizado na aplicação do processo apresentado no capitulo 4.

ANEXO A apresenta uma comparação de banco de dados NoSQL

2. BANCO DE DADOS NOSQL

O termo NoSQL é um neologismo acidental. Não existe uma definição coerente sobre o termo, que para alguns significa *No SQL* (Não ao SQL) e para alguns *Not only SQL* (Não somente SQL). NoSQL é também o nome de um banco de dados relacional criado no final dos anos 90, que não tem influência alguma na abordagem NoSQL (Sadalage & Fowler, 2012).

Quando aplicado aos bancos de dados, o termo NoSQL refere-se a um conjunto destes que em sua maioria compartilham as seguintes características : (Sadalage & Fowler, 2012).

- Não usam o modelo relacional
- Operam em cluster com naturalidade
- *Open-source*
- Construídos para as aplicações Web do século 21
- Não possuem estrutura rígida de modelo de dados (*schemaless*)

Ainda é comum encontrar aplicações que utilizam apenas o banco de dados relacional, mas com o tempo está se tornando popular usar várias bancos de dados de tipos diferentes em conjunto, potencializando suas forças para criar um ecossistema que é mais poderoso, capaz e robusto do que a soma de suas partes. Esta prática é conhecida como persistência poliglota (Redmond & Wilson, 2012).

A maioria dos bancos de dados NoSQL são *open source*, acompanhando os desenvolvimentos do mercado global de software, o que permite aos usuários realizar avaliações técnicas de baixo custo (Sadalage & Fowler, 2012).

Diferentes bancos de dados NoSQL possuem diferentes abordagens. O que eles têm em comum é que eles não são relacionais. Sua principal vantagem é que, ao contrário de bancos de dados relacionais, eles lidam com dados não estruturados, como arquivos de texto, e-mail e multimídia, de forma eficiente. Bancos de dados relacionais trabalham melhor com dados estruturados, que facilmente podem ser organizados e acomodados em tabelas (Leavitt, 2010).

A escalabilidade em um banco de dados relacional pode ser alcançada através da escalabilidade vertical, adicionando mais poder de processamento ao servidor de banco de dados e consequentemente gerando mais custos. Para atingir um nível maior de escalabilidade deve-se adotar a escalabilidade horizontal, que consiste em distribuir os dados por múltiplos servidores. (Leavitt, 2010)

Durante anos, o desenvolvimento de sistemas de informação vem provendo escalabilidade vertical (também chamado de *scale-up*), investindo em novos e caros grandes servidores (Pokorny, 2011).

A distribuição de dados em tempo real pode causar diminuição do desempenho do sistema. O particionamento no banco de dados em vários servidores baratos adicionados dinamicamente, provendo escalabilidade horizontal (também chamado *scale-out*), aparentemente pode garantir a escalabilidade de uma forma mais eficaz e mais barata (Pokorny, 2011).

Para alcançar a escalabilidade horizontal, os bancos de dados NoSQL dispensaram algumas características de banco de dados comuns, como por exemplo, restrições do modelo de dados relacional e exigências de processamento de transações (Pokorny, 2011).

As próximas seções apresentam conceitos importantes, controle transacional e o teorema CAP, sobre as características que os bancos de dados NoSQL mudaram com relação aos bancos de dados relacionais.

2.1 Controle transacional

Uma das características básicas dos bancos de dados é o controle de transação caracterizado por Atomicidade, Consistência, Isolamento e Durabilidade, as chamadas propriedades ACID (*Atomicity*, *Consistency*, *Isolation*, *Durability*) (Pokorny, 2011).

Segundo Leavitt (2010), a Atomicidade (*Atomicity*) dos bancos de dados garantem que as operações executadas na mesma transação são persistidas somente se todas forem executadas com sucesso, caso contrário todas são descartadas. A Consistência (*Consistency*) significa que uma transação é realizada com sucesso quando as restrições do banco de dados são respeitadas, como por exemplo, as chaves primárias dos bancos de dados relacionais. O Isolamento (*Isolation*) garante que as transações em execução de forma concorrente são isoladas de forma que uma não interfere na execução das demais. A Durabilidade (*Durability*) garante que após finalizada, as operações de uma transação são persistidas no banco de dados e assim devem permanecer.

Bancos de dados que não implementam totalmente ACID são eventualmente consistentes. Em princípio, ao ceder um pouco de consistência, ganha-se mais disponibilidade e uma melhora significativamente a escalabilidade do banco de dados (Pokorny, 2011).

2.2 Teorema CAP

O teorema CAP (*Consistency*, *Availability*, *Partition Tolerance*) afirma que qualquer sistema de dados compartilhados operando numa rede pode ter somente duas das três propriedades desejadas. (Brewer, 2012).

Consistência (*Consistency*) significa que sempre que os dados são gravados, todos que leiam a partir do banco de dados obterão sempre a atualização mais recente dos dados. (Pokorny, 2011).

Disponibilidade (*Availability*) significa que se pode sempre esperar que cada operação termine em uma resposta pretendida. A alta disponibilidade é normalmente realizada através de um grande número de servidores físicos que agem como um único banco de dados com os dados compartilhados entre vários nós de banco de dados e replicações de dados. (Pokorny, 2011).

Tolerância a Particionamento (*Partition Tolerance*) significa que o banco de dados ainda pode ler e armazenar dados mesmo quando partes dele sejam completamente

inacessíveis. Por exemplo, quando a conexão de rede entre um número significativo de servidores de banco de dados é interrompida a tolerância a particionamento pode ser alcançada através do envio das requisições de escrita destinadas aos servidores inacessíveis para os que estão ainda acessíveis. Então, quando os servidores com falha voltarem a se conectar na rede, eles recebem os dados escritos que foram perdidos (Pokorny, 2011).

Após mais de uma década da afirmação do teorema CAP, Brewer (2012) explica que ao manipular partições explicitamente, os desenvolvedores podem otimizar a consistência e disponibilidade, conseguindo assim um balanceamento de todas as três propriedades (Brewer, 2012).

Os conceitos de controle transacional e do teorema CAP são importantes para a compreensão das principais limitações dos bancos de dados relacionais apresentadas na próxima seção.

2.3 Limitações dos Bancos de Dados Relacionais

Os bancos de dados relacionais possuem algumas limitações assim como os bancos de dados NoSQL. Conhecer as limitações dos bancos de dados relacionais é importante para compreender as principais motivações para a criação dos bancos de dados NoSQL e também quais são as principais diferenças entre eles.

Desempenho

Bancos de dados NoSQL geralmente processam dados mais rapidamente do que os bancos de dados relacionais. As operações realizadas nos bancos de dados relacionais, sejam operações que requerem grande precisão ou não, estão sujeitos às restrições de ACID (Leavitt, 2010).

Ter que executar essas restrições em todos os dados faz com que os bancos de dados relacionais sejam mais lentos. A maioria dos bancos de dados NoSQL não suportam ACID a fim de aumentar o desempenho, e isso pode causar problemas quando usados para aplicações que necessitam de grande precisão (Leavitt, 2010).

Bancos de dados NoSQL também são muitas vezes mais rápidos, pois seus modelos de dados são mais simples. Existe um balanço entre a velocidade e a complexidade do modelo e na maioria dos casos é uma troca vantajosa (Leavitt, 2010).

Escalabilidade

Bancos de dados relacionais não foram criados para operar de forma distribuída com partição de dados. Realizar consultas que envolvem varias tabelas que estão distribuídas não é fácil (Leavitt, 2010).

SQL

Usar a linguagem SQL é conveniente com dados estruturados. Porém, com outros tipos de dados é difícil porque o SQL foi criado para funcionar com dados estruturados, fixos e relacionalmente organizados.

Trabalhar com a linguagem SQL e consequentemente com banco de dados relacionais pode envolver uma grande quantidade de código complexo e não funciona bem com o desenvolvimento ágil moderno (Leavitt, 2010). Isto porque no desenvolvimento de software o projeto do modelo de dados exige uma fase, que pode ser longa, antes do desenvolvimento e que se não for bem projetado acarreta em muitas manutenções.

Flexibilidade

A estrutura dos dados num banco de dados relacional é pré-definida pelo layout das tabelas e pelos nomes e tipos fixos das colunas (Leavitt, 2010).

A manutenção e evolução dos modelos de dados relacionais é normalmente trabalhosa, principalmente acomodar os dados já existentes no banco de dados em uma nova estrutura. Enquanto que com os bancos de dados NoSQL muitas vezes não há um modelo de dados fixo, portanto fica a cargo da aplicação lidar com as diferentes versões dos dados.

Complexidade

Com os bancos de dados relacionais, todos os dados devem ser convertidos para tabelas. Quando os dados não se encaixam facilmente em uma tabela, a estrutura acaba ficando complexa, lenta e difícil de se manipular (Leavitt, 2010).

Recursos do banco de dados

Bancos de dados relacionais oferecem um grande conjunto de recursos e integridade dos dados. Porém, os defensores dos bancos de dados NoSQL dizem que os usuários muitas vezes não precisam de todos os recursos, bem como o custo e a complexidade que eles podem trazer (Leavitt, 2010). A segurança é um exemplo, porque pode ser obtida através da aplicação responsável pelos dados (Sadalage & Fowler, 2012).

Incompatibilidade de Impedância / Mapeamento Objeto-Relacional

Conhecido como *impedance mismatch*, é a diferença entre o modelo relacional e a estrutura de dados que está em memoria na aplicação.

Nos bancos de dados relacionais, os relacionamentos fornecem simplicidade e padronização porém também adicionam limitações. Os valores de uma tupla não podem conter estruturas complexas como de registros aninhados e listas, enquanto que estruturas manipuladas em memória pela aplicação podem.

No desenvolvimento de aplicações orientadas a objetos, o trabalho de mapeamento das diferentes estruturas foi facilitado pelos *frameworks* ORM (*object-relational mapping*), como o Hibernate, que vêm sendo largamente utilizados, mas podem acarretar em problemas de desempenho e adicionar ainda mais complexidade quando mal utilizados (Sadalage & Fowler, 2012).

Conforme apresentado no inicio deste capítulo, os bancos de dados NoSQL não usam o modelo de dados relacional, justamente para evitar os problemas como a flexibilidade, complexidade e a incompatibilidade com o mapeamento objeto-relacional.

2.4 Modelos de Dados dos Bancos de Dados NoSQL

Um modelo de dados descreve a organização dos dados de uma aplicação no banco de dados. Um modelo de armazenamento descreve como o banco de dados armazena e manipula os dados internamente (Sadalage & Fowler, 2012).

Idealmente os usuários dos bancos de dados não precisariam conhecer profundamente o modelo de armazenamento, porém na prática são necessários conhecimentos um pouco mais do que básicos para alcançar um bom desempenho na sua utilização (Sadalage & Fowler, 2012).

O termo modelo de dados é também utilizado com o sentido de modelo de armazenamento quando se refere aos tipos de modelos de armazenamento de bancos de dados NoSQL. (Sadalage & Fowler, 2012).

Os bancos de dados NoSQL são categorizados de acordo com o modelo de dados que utilizam internamente. Essencialmente os tipos de modelos de armazenamento, ou modelos de dados NoSQL, são : chave-valor, documento, coluna e grafo.

Portanto, quatro categorias, das quais chave-valor, documento e coluna são também categorizadas como orientadas a agregados por compartilharem algumas características de seus modelos de dados. Eles permitem que os dados persistidos possuam estruturas complexas, como de objetos aninhados e listas, trabalhando de forma natural com agregados (Sadalage & Fowler, 2012).

Agregado é um termo vindo do DDD (*Domain Driven Design*). No DDD, um agregado é uma coleção de objetos relacionados com a qual se interage como uma unidade (Sadalage & Fowler, 2012).

Nos bancos de dados orientados a agregados, o agregado é uma unidade para manipulação e gerenciamento de consistência dos dados. Como os dados de um agregado são sempre manipulados, na escrita e leitura, como uma unidade, a comunicação com o banco de dados é atômica, não sendo necessário controle transacional adicional, além de facilitar a operação do banco de dados de forma

distribuída, já que um agregado é também uma unidade natural para replicação e particionamento (Sadalage & Fowler, 2012).

Não existe uma forma bem definida para a modelagem dos agregados, depende totalmente de como a aplicação pretende manipular os dados. Devido a este fato, muitas vezes a modelagem dos agregados envolve outras disciplinas (Sadalage & Fowler, 2012).

Um modelo de armazenamento que não suporta a estrutura de agregados, como por exemplo os modelos relacionais, permite que os dados sejam acessados de formas diferentes, portanto é a melhor escolha para os casos em que não haja uma estrutura dos dados e bem definida, bem como sua forma de utilização (Sadalage & Fowler, 2012).

A grande vantagem dos agregados é a facilidade de distribuição dos dados, pois estes estão no mesmo servidor devido a sempre serem manipulados juntos, evitando o problema de fazer uma consulta que precise juntar dados distribuídos em vários servidores (Sadalage & Fowler, 2012).

2.5 Modelagem de Dados com Bancos de Dados NoSQL

O formato JSON (*Javascript Object Notation*) é normalmente utilizado para representar as estruturas dos dados nos bancos de dados NoSQL (Pokorny, 2011).

Na figura 1 pode ser visto um exemplo de um arquivo em formato JSON.

```
{
    “ID”: 1,
    “ERR”: “Out of Memory”,
    “TIME”: “2004-09-16T23:59:58.75”,
    “DC”: “NYC”,
    “NUM”: “212-223-2332”
}
{
    “ID”: 2,
    “ERR”: “ECC Error”,
    “TIME”: “2004-09-16T23:59:59.00”,
    “DC”: “NYC”,
    “NUM”: “212-223-2332”
}
```

Figura 1 – Exemplo de um arquivo em formato JSON (Couchbase, 2013)

Enquanto a modelagem de dados de aplicações que utilizam bancos de dados relacionais tem a preocupação em formar relacionamentos normalizados, na modelagem de dados de aplicações que utilizam bancos de dados NoSQL a desnormalização é fortemente encorajada. (Schram & Anderson, 2012)

Desnormalização pode ser descrita como um processo para reduzir o grau de normalização de um modelo de dados com o objetivo de melhorar o desempenho e reduzir o tempo de resposta do processamento de consultas realizadas no banco de dados (Sanders & Shin, 2001).

Uma das áreas mais úteis para aplicação de técnicas de desnormalização é em implementações de Data Warehouse para mineração de dados (Sanders & Shin, 2001).

Na Figura 2, pode ser visto um exemplo da desnormalização do modelo de dados relacional, onde as tabelas *User Info* e *Address Info* são representadas em um único arquivo JSON, fazendo com que estes dados sejam recuperados mais rapidamente.

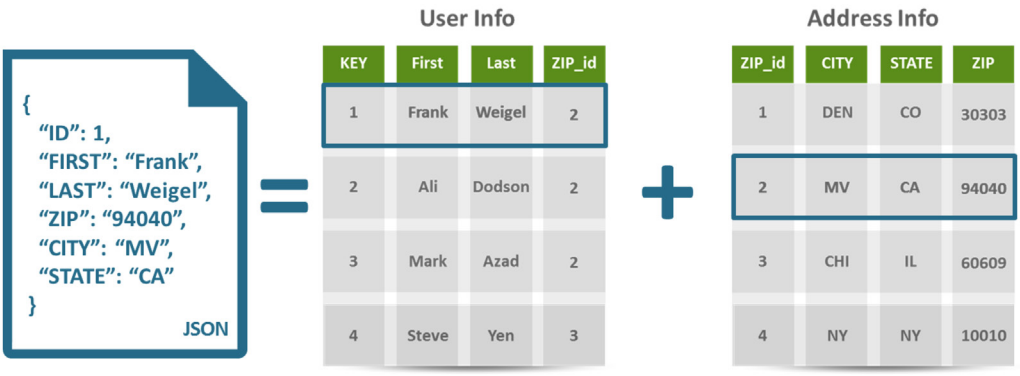

Figura 2 – Dados desnormalizados em formato JSON (Couchbase, 2013)

Os conceitos de desnormalização e agregados são importantes para a construção de modelos de dados para bancos de dados NoSQL. A partir destes conceitos e dos estudos e literatura apresentados por Schram & Anderson (2012); Cruz et al. (2011); Saladage & Fowler (2012); Redmond & Wilson (2012) é possível identificar diversas alternativas e decisões de modelagem, algumas se aplicam a várias categorias de bancos de dados NoSQL e outras somente a categorias específicas.

Algumas destas alternativas e decisões são apresentadas nas seções que seguem juntamente com mais detalhes sobre cada categoria de banco de dados NoSQL.

2.6 Tipos de Banco de Dados NoSQL

Nas seções seguintes são apresentadas, para cada tipo de banco de dados, as características do modelo de dados, casos de utilização recomendada e não recomendada e alternativas e decisões de modelagem de dados.

Foram considerados somente os bancos de dados orientados a agregados (ver seção 2.4) para detalhamento e apresentação das decisões de modelagem, portanto a categoria de banco de dados NoSQL de Grafos não é apresentada.
Os bancos de dados de grafos são menos utilizados, e são excelentes para lidar com dados interligados. Um banco de dados de grafo consiste em nós e as relações entre os nós. Ambos os nós e os relacionamentos podem ter propriedades, pares de chave-valor, que armazenam os dados. Os dados são acessados através da navegar pelos nós seguindo os relacionamentos (Redmond & Wilson, 2012).

2.6.1 Chave-Valor

Como o nome indica, os bancos de dados desta categoria armazenam valores indexados para que sejam recuperados por sua chave correspondente. Estes sistemas podem conter dados estruturados ou não estruturados.

Este é o modelo mais simples entre os modelos das categorias NoSQL. O modelo de dados é caracterizado por pares formados por uma chave e um valor, como na estrutura de dados de tabela *hash* (Redmond & Wilson, 2012).

Um sistema de arquivos pode ser considerado um armazenamento de chave-valor, onde o caminho do arquivo é a chave e o conteúdo do arquivo como o valor (Redmond & Wilson, 2012)

Utilização recomendada :
Com pouca ou nenhuma necessidade de manter índices, os bancos de dados do tipo chave-valor são muitas vezes projetados para serem escaláveis horizontalmente e extremamente rápidos. Eles são particularmente adequados para problemas onde os dados não são altamente relacionados. Por exemplo, em uma aplicação Web, dados de sessão dos usuários atendem a esse critério, a atividade da sessão de cada usuário será diferente e não relacionada com a atividade de outros usuários (Redmond & Wilson, 2012)

Utilização não recomendada :
Em casos que seja necessário manipular os dados com operações além básicas CRUD (*Create*, *Read*, *Update*, *Delete*). (Redmond & Wilson, 2012)

Decisões de modelagem

1) Modelo simples

Neste modelo a chave é formada por um valor único e o valor recebe um agregado.

Fornece atomicidade dada pelo agregado (v. seção 2.4.1), porém o trata como um valor opaco, o que significa que pode-se obter o valor apenas pela chave. O banco de dados não tem conhecimento do conteúdo do valor, por isso não se pode executar consultas para retornar apenas uma parte do agregado (Sadalage & Fowler, 2012).

Armazenar diversos agregados sem segmentá-los por tipo aumenta as chances de conflitos com as chaves. Exemplo : a chave *java* pode referenciar a linguagem de programação e também a ilha da Indonésia (Sadalage & Fowler, 2012).

2) Modelo com chave composta

Neste modelo a chave é formada por um valor único concatenado com o nome do objeto que será atribuído ao valor. Desta forma, é possível acessar os objetos individualmente, resolvendo o problema de acesso de partes do agregado do modelo simples, e são reduzidas as chances de conflitos de chaves (Sadalage & Fowler, 2012).

Contudo a consistência é aplicada somente para operações em chaves únicas, no caso de atomicidade para agregados a consistência fica a cargo da aplicação, já que os objetos estão sendo persistidos separadamente (Sadalage & Fowler, 2012).

3) Modelo segmentado por domínio

As características deste modelo se assemelham as do modelo simples, a chave é formada por um valor único e o valor recebe um agregado. Porém para poder acessar os objetos individualmente dentro do agregado sem alterar a estrutura da chave, como no modelo com chave composta, alguns bancos de dados do tipo chave-valor possuem os *buckets,* que permitem que as chaves sejam segmentadas por domínio, também reduzindo as chances de conflitos de chaves.

Neste modelo, assim como no anterior, a consistência é aplicada somente para operações em chaves únicas, no caso de atomicidade para agregados a

consistência fica a cargo da aplicação, já que os objetos estão sendo persistidos separadamente, segmentados por domínio.

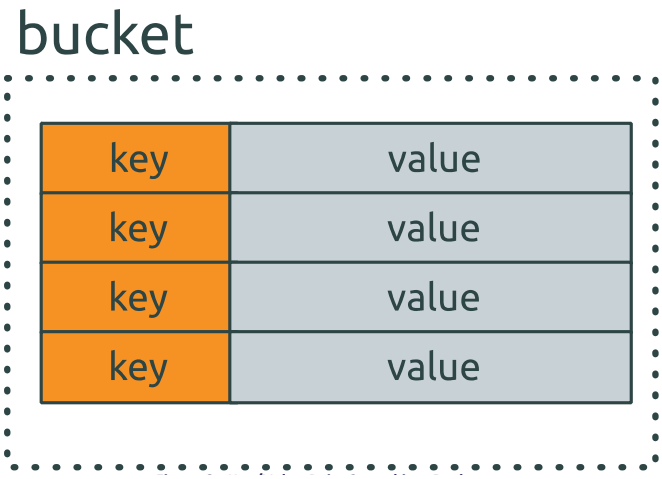

Figura 3 - Pares de chave-valor armazenados em um *bucket* (Basho, 2012)

Na figura 3 é possível visualizar a organização do modelo de dados do tipo chave-valor com um *bucket*.

2.6.2 Documentos

Para os bancos de orientados a documentos, a palavra *documento* não está relacionada com o significado utilizado em um contexto geral, ou seja, não significa livros, cartas ou artigos. Um documento, neste contexto, refere-se a um registro de dados que é auto-descritivo com relação aos dados que nele são contidos. Documentos nos formatos XML, HTML e JSON são exemplos que se encaixam neste contexto (Couchbase, 2013).

Um documento é como um *hash*, com um campo de identificador único e valores que podem ser de variados tipos, incluindo um próprio *hash*.

Os documentos podem conter estruturas aninhadas, e assim eles apresentam um alto grau de flexibilidade, permitindo sua utilização para diversos domínios. O sistema impõe poucas restrições sobre os dados de entrada, contanto que eles atendam ao requisito básico de ser expresso como um documento.

Diferentes bancos de dados deste tipo possuem diferentes abordagens com relação à indexação, consultas ad-hoc, replicação, consistência e outras decisões de projeto. Fazer a melhor escolha entre eles requer a compreensão dessas diferenças e como elas impactam os casos de uso específicos.

Os bancos de dados orientados a documentos permitem qualquer número de campos por objeto e até mesmo permitem que os objetos a sejam aninhados em qualquer profundidade como valores de outros campos.

Uma vez que os documentos não se relacionam entre si, como bancos de dados relacionais, eles são relativamente fáceis de particionar e replicar em vários servidores, tornando as implementações distribuídas bastante comuns.

A Figura 4 compara a representação de quatro registros em um banco de dados relacional com quatro registros em um banco de dados orientado a documentos.

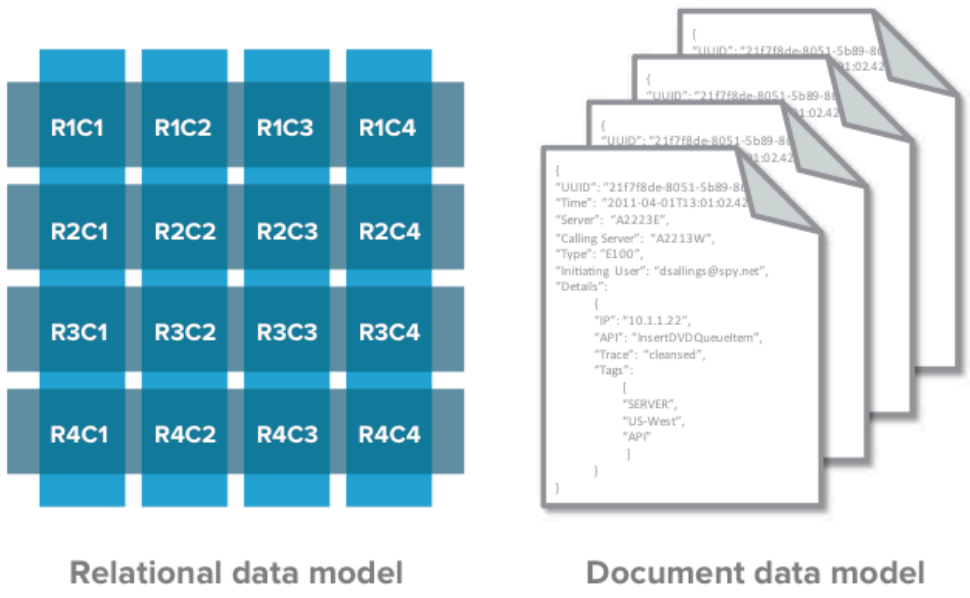

Figura 4 – Representação de registros comparadas relacional x documentos (Couchbase, 2013)

O modelo de dados relacional é caracterizado pela organização de tabelas estruturadas com formato dos dados e estrutura dos registros rigidamente definidos.

O modelo de dados de documentos possui coleções de documentos complexos com formato arbitrário e aninhado de dados e formato dos registros variado. (Couchbase, 2013)

Utilização recomendada :
Bancos de dados orientados a documentos são adequados para os problemas que envolvem domínios altamente variáveis e também para quando não se sabe de antemão exatamente como serão os dados da aplicação. Além disso, devido à natureza dos documentos, muitas vezes eles são facilmente mapeados para modelos de programação orientada a objetos. Isso significa menos incompatibilidade (*impedance mismatch*) ao mover dados entre o modelo de dados e o modelo de aplicação (Redmond & Wilson, 2012).

Utilização não recomendada :
Para realizar consultas elaboradas com modelos altamente normalizados.
Um documento geralmente deve conter a maioria ou todas as informações relevantes necessárias para o uso. Assim, enquanto em um banco de dados relacional, naturalmente os dados são normalizados para reduzir ou eliminar as cópias que podem ficar dessincronizadas, com bancos de dados orientados a documentos, dados desnormalizados é a regra (Redmond & Wilson, 2012).

Decisões de modelagem

1) Embeded

Este modelo visa a desnormalização dos dados e trabalha de forma transparente com agregados, permitindo a execução de consultas para obter partes do agregado. Porém como o documento não possui uma definição da sua estrutura, o banco de dados não pode agir muito na estrutura do documento para otimizar o armazenamento e leitura de partes dos agregados (MongoDB, 2013).

Benefícios deste modelo (MongoDB, 2013):

- Normalmente melhor desempenho para operações de leitura.
- Habilidade de requisitar e obter dados relacionados em uma única operação no banco de dados.

Embutir dados relacionados em documentos pode ocasionar situações onde os documentos cresçam depois da criação. A criação de um documento pode impactar o desempenho de escrita dos dados e levar a fragmentação dos dados (MongoDB, 2013).

Além disso, o tamanho dos documentos pode ser limitado, por exemplo para o MongoDB que é um banco de dados popular orientado a documentos, eles devem ser menores do que o tamanho máximo permitido para um documento BSON (*Binary Script Object Notation*) (MongoDB, 2013).

2) Reference

Este modelo visa a normalização dos dados, armazenando referências entre dois documentos para indicar um relacionamento entre eles (MongoDB, 2013).

Esta modelo pode ser usado quando (MongoDB, 2013) :

- O modelo Embeded resultar em duplicação de dados que não forneça vantagens suficientes no desempenho de leitura para compensar as implicações da duplicação.
- Para representar relacionamentos mais complexos do tipo Muitos-para-Muitos.
- Para modelar grandes hierarquias, como estruturas de dados de árvore.

Este modelo oferece mais flexibilidade do que o Embeded, porém requer acessos ao servidor, pois para resolver as referências, as aplicações clientes devem requisitar consultas adicionais (MongoDB, 2013).

2.6.3 Colunas

Os bancos de dados orientados a colunas são assim chamados porque o aspecto importante de sua estrutura é que os dados de uma determinada coluna (no sentido de tabela bidimensional) são armazenado em conjunto. Por outro lado, um banco de dados orientado a linhas (como o relacional) mantém informações sobre uma linha juntos (Redmond & Wilson, 2012).

A diferença pode parecer irrelevante, mas o impacto desta decisão de projeto é mais profunda. Nos bancos de dados orientados a colunas, adicionar colunas é uma operação de pouca oneração e é feita de linha em linha. Cada linha pode ter um conjunto diferente de colunas, ou nenhum, permitindo que as tabelas permaneçam dispersas sem incorrer em um custo de armazenamento para valores nulos (Redmond & Wilson, 2012).

A Figura 5 apresenta a organização do modelo dados de colunas do banco de dados Cassandra, que é um dos mais populares desta categoria.

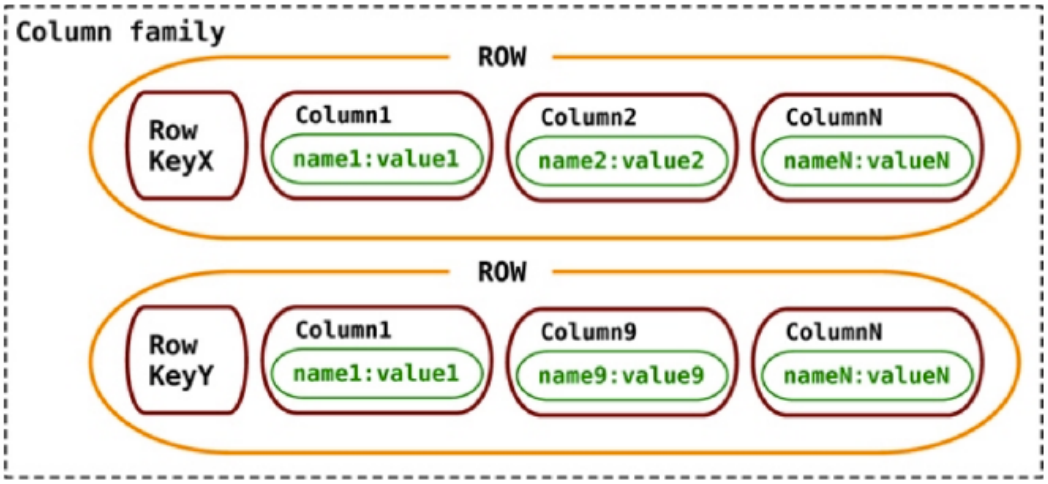

Figura 5 – Modelo de dados orientado a colunas do Cassandra (Sadalage & Fowler, 2012)

A família de colunas (*column family*) é formada por linhas (*rows*) que possuem um identificador (*row key*). As linhas são compostas de colunas, onde cada coluna possui um nome e um valor (Pokorny, 2011).

Utilização recomendada:
Os bancos de dados orientados a colunas têm sido tradicionalmente desenvolvidos com escalabilidade horizontal como objetivo principal. Por isso, eles são

particularmente adequados para lidar com grandes volumes de dados, presentes em grupos de dezenas, centenas ou milhares de nós. Eles também tendem a ter suporte integrado para recursos como compressão e versionamento. O exemplo canônico de um bom problema de armazenamento de dados em um modelo de colunas é indexar páginas da Internet. Isto porque elas são altamente textuais (benefícios da compressão), pouco interrelacionadas, e mudam ao longo do tempo (benefícios do versionamento) (Redmond & Wilson, 2012).

Utilização não recomendada:
Diferentes bancos de dados orientados a colunas têm características diferentes entre eles e, portanto, diferentes desvantagens. Porém, o que eles têm em comum é que é melhor projetar o modelo de dados da aplicação com base em como se pretende consultar os dados. Isso significa que se deve ter alguma ideia com antecedência de como os dados serão utilizados pela aplicação, não apenas do que eles serão consistidos. Se os padrões de uso de dados não podem ser definidos com antecedência, por exemplo, relatórios ad-hoc, então este tipo de banco de dados pode não ser o mais adequado (Redmond & Wilson, 2012).

Decisões de modelagem

1) Skinny rows

Este modelo é aplicável aos bancos de dados orientados a colunas, e é mais similar ao modelo de dados relacional.

As características deste modelo são famílias de colunas que definem o tipo de dado, poucas colunas e linhas com as mesmas colunas usadas em várias linhas diferentes (Sadalage & Fowler, 2012).

Nesta técnica o identificador da linha (row key) recebe o valor do identificador de uma entidade. As linhas são formadas por colunas que representam os atributos da entidade. O nome de cada coluna é o nome de cada atributo da entidade. O valor da coluna recebe o valor do atributo.

Tweet Id 1	createdAt	text	screen_name	•••
	Date	String	String	•••
•••				
Tweet Id N	createdAt	text	screen_name	•••
	Date	String	String	•••

Figura 6 - Exemplo da técnica de modelagem Skinny rows (Schram & Anderson, 2012).

A Figura 6 apresenta um exemplo deste modelo, que armazena os dados da entidade *Tweet* (obtidos do *Twitter)*.

2) Wide rows

As características deste modelo são muitas colunas (da ordem de milhares) diferentes para cada linha, modela uma lista, com cada coluna sendo um elemento desta lista e cada linha representa um agregado (Sadalage & Fowler, 2012).

Neste modelo, o identificador da linha (row key) recebe o valor do identificador de uma entidade. As linhas são formadas por colunas que representam os atributos da entidade. O nome de cada coluna é o nome de cada atributo da entidade. O valor da coluna recebe o valor do atributo. (Sadalage & Fowler, 2012).

A família de colunas é baseada no tipo de consulta/resultado que a aplicação necessita/espera, ou seja, dados que normalmente são acessados juntos. O identificador da linha é único e representa uma entidade do tipo da família de coluna. (Sadalage & Fowler, 2012).

A linha é formada por várias colunas e cada coluna representa uma entidade, portanto, o número de colunas de cada linha vai variar de acordo com a quantidade de entidades associadas ao identificador da linha (pode chegar a dezenas de milhões ou mais...) (Sadalage & Fowler, 2012).

Event Name 1	Tweet Id 1	•••	Tweet Id N
	JSON	•••	JSON
•••			
Event Name N	Tweet Id 1	•••	Tweet Id N
	JSON	•••	JSON

Figura 7 – Exemplo da técnica de modelagem Wide rows (Schram & Anderson, 2012)

No exemplo da Figura 7, na família de colunas chamada eventos, o identificador da linha “Event Name 1”, é o identificador único do evento. As colunas são modeladas de forma que o nome seja o identificador da entidade, e o valor sejam os atributos da entidade em um formato opaco para a aplicação, por exemplo em formatos JSON/ XML / String, ou seja para a coluna de nome “Tweet Id 1”, seus atributos como *createdAt*, *text* e outros (v. Figura 6) estão contidos em um formato JSON.

3) Modelo com chave composta

Este modelo pode ser utilizado quando a manipulação da estrutura de dados se tornar custosa devido a grandes volumes de dados (Schram & Anderson, 2012).

O modelo com chave composta traz muitos benefícios quando um repositório com chaves ordenadas é utilizado. Chaves compostas em conjunto com ordenação secundária permite que sejam construídos um tipo de índice multidimensional. (Katsov, 2012)

Event Name 1	Tweet Id 1	•••	Tweet Id N
	JSON	•••	JSON
•••			
Event Name N	Tweet Id 1	•••	Tweet Id N
	JSON	•••	JSON

Figura 8 - Exemplo de modelagem com chave composta (Schram & Anderson, 2012)

No exemplo da Figura 8, o modelo armazena os objetos que representam um *Tweet* em formato JSON relacionados a eventos, o identificador da linha é formado pelo

identificador do evento. Com este modelo, podem ser armazenados centenas de milhões de *Tweets* por evento causando o problemas devido a quantidade de colunas.

Neste modelo o identificador da linha é composto pelo identificador de uma entidade e outros valores que se queira utilizar como filtro.

Event Name 1: Day W	Tweet Id 1	•••	Tweet Id N
	JSON	•••	JSON
•••			
Event Name 1: Day X	Tweet Id 1	•••	Tweet Id N
	JSON	•••	JSON
•••			
Event Name N: Day Y	Tweet Id 1	•••	Tweet Id N
	JSON	•••	JSON
•••			
Event Name N: Day Z	Tweet Id 1	•••	Tweet Id N
	JSON	•••	JSON

Figura 9 – Exemplo de modelagem com chave composta (Schram & Anderson, 2012)

Na Figura 9, foi adicionado no identificador da linha o dia do evento (Evento:Periodo, que poderia ser hora, minuto, conforme a necessidade), limitando a quantidade de colunas por linha.

Desta forma, quando a aplicação buscar os dados por esta chave composta a quantidade de colunas será menor, de um tamanho que a aplicação consiga alocar na memória para manipulação, além de melhorar no tempo de replicação/distribuição dos dados no cluster.

O ponto negativo desta técnica é que o modelo de dados fica dependente dos requisitos de consulta da aplicação. Se mudar, na maioria das vezes é inviável reformular a estrutura de dados já coletados. Então, normalmente a solução seria

que as novas consultas terão resultados apenas para os novos dados coletados. (Schram & Anderson, 2012)

2.7 Pontos de Atenção com os Bancos de Dados NoSQL

Após conhecer os conceitos, vantagens e principais características dos bancos NoSQL, vistos nas seções anteriores, finalmente é importante conhecer também alguns dos cuidados que devem ser tomados ao se trabalhar com estes bancos de dados para que possa ser feita uma boa avaliação de quando utilizá-los

Complexidade

Devido aos bancos de dados NoSQL não trabalharem com a linguagem SQL, eles exigem programação manual das consultas, que pode ser rápida para tarefas simples, mas demorada para os outras mais complexas (Leavitt, 2010).

Confiabilidade

Bancos de dados relacionais suportam nativamente ACID, enquanto a maioria dos bancos de dados NoSQL não. Bancos de dados NoSQL, portanto, não oferecem nativamente o grau de confiabilidade que ACID oferece. Fica a cargo da aplicação responsável pelos dados garantir que estes sejam confiáveis. (Leavitt, 2010).

Consistência

Devido aos bancos de dados NoSQL não suportarem nativamente transações ACID, a consistência dos dados pode ser comprometida, a não ser que suporte manual seja fornecido. Não fornecer consistência permite um melhor desempenho e escalabilidade, mas é um problema para certos tipos de aplicações e transações, tais como as envolvidas no setor bancário (Leavitt, 2010).

Falta de familiaridade com a tecnologia

A maioria das organizações não está familiarizada com bancos de dados NoSQL, portanto, podem não sentir-se confiante o suficiente para escolher um produto ou até mesmo para determinar qual abordagem poderia ser melhor para os seus fins (Leavitt, 2010).

Ambiente limitado

Ao contrário dos bancos de dados relacionais comerciais, muitos bancos de dados NoSQL *open-source* ainda não possuem ferramentas de apoio aos usuários ou de gestão (Leavitt, 2010).

2.8 Considerações do Capítulo

Neste capitulo, os bancos de dados NoSQL foram introduzidos através da descrição das principais características, conceitos importantes como o ACID e o teorema CAP e das categorias chave-valor, documentos e colunas com suas respectivas características e decisões de modelagem de dados.

Foram também apresentadas as principais limitações dos bancos de dados relacionais e cuidados a serem tomados com os bancos de dados NoSQL, com o propósito de compreender as motivações para utilização dos bancos de dados NoSQL e ajudar na avaliação de quando utilizá-los.

3. PROCESSO DE MIGRAÇÃO PARA NOSQL

Os bancos de dados não relacionais existem desde o final da década de 60, porém somente nos últimos anos que os novos tipos de banco de dados não relacionais, chamados de NoSQL, começaram a atrair o mercado (Leavitt, 2010).

Inicialmente, a adoção dos bancos de dados NoSQL vem ocorrendo em projetos específicos que tratam de sistemas distribuídos, que envolvem grande quantidade de dados e que precisam de larga escalabilidade. Com o passar do tempo a adoção deve ocorrer em maiores escalas, inclusive porque trabalhar com os bancos de dados NoSQL é mais fácil para os desenvolvedores que não estão familiarizados com SQL. Haverá uma crescente percepção de que os bancos de dados relacionais em uso nos dias de hoje são na maioria dos casos boas ferramentas, mas que também existe espaço para outras (Leavitt, 2010),

É importante ter a consciência de que os bancos de dados NoSQL não substituirão os bancos de dados relacionais, mas serão uma melhor opção para certos tipos de projetos.(Edlich, 2012) (Leavitt, 2010).

A adoção de grandes mudanças de tecnologia causa resistência das empresas, principalmente pelo alto custo, como já ocorreu no passado com as mudanças de plataformas Mainframe para Cliente-Servidor, depois de Cliente-Servidor para a aplicações que aplicam os conceitos de SOA, SOA para WEB, e agora de bancos de dados relacionais para não relacionais (NoSQL). As empresas que estão tentando rapidamente integrar com banco de dados NoSQL estarão melhor posicionadas para o futuro (Edlich, 2012).

A migração para NoSQL possibilita que no mesmo sistema ou até na mesma aplicação, diversos tipos bancos de dados, relacionais ou não relacionais, possam ser utilizados para atender apropriadamente aos requisitos do sistema, ou seja, a migração pode ocasionar a utilização de bancos de dados não relacionais para alguns requisitos e o relacional para outros. Isto não significa que os bancos de dados relacionais serão totalmente substituídos (Schram & Anderson, 2012)

A convivência de diversos tipos de bancos de dados traz diversos desafios. A partir de estudos de migração para NoSQL apresentados por Schram & Anderson (2012) e Cruz et al. (2011) foi possível identificar um processo de migração. Este processo de migração é composto de três fases.

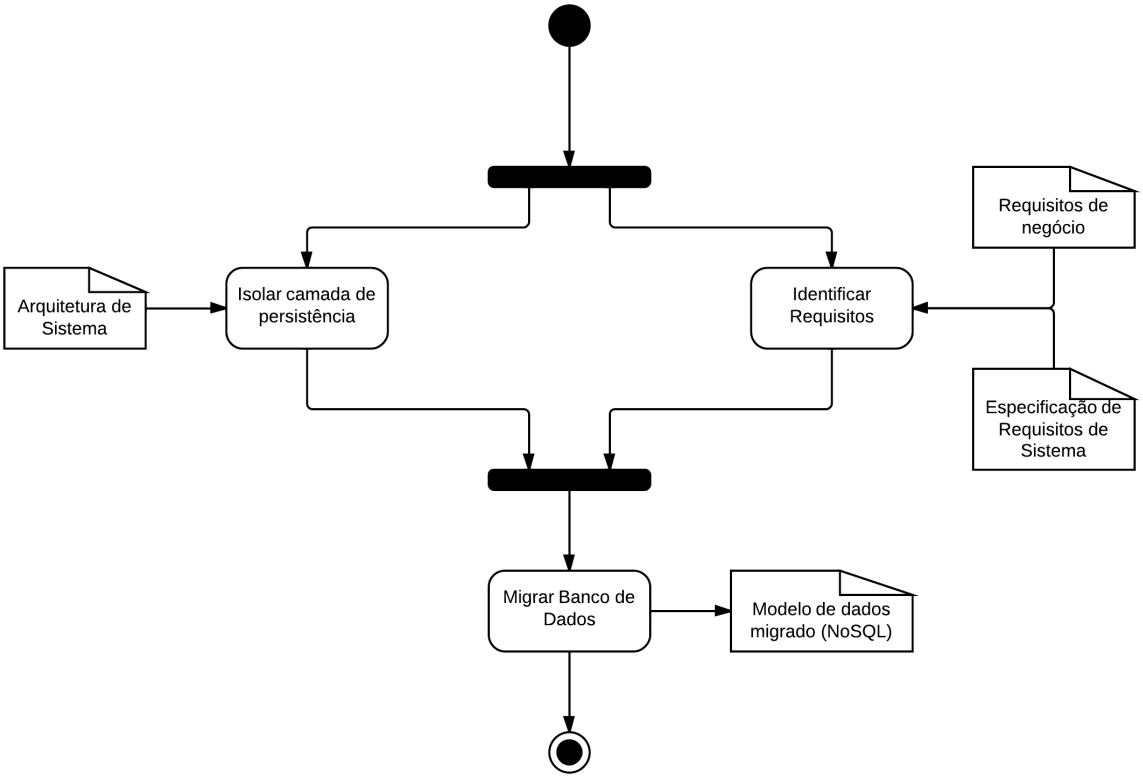

Figura 10 - Processo de migração para NoSQL

Conforme pode ser visto na Figura 10, a primeira fase Isolar Camada de Persistência tem como objetivo isolar a camada de persistência da aplicação afim de facilitar que esta possa ser facilmente substituída ou alterada sem grandes impactos. Esta fase, a depender do resultado da análise feita a partir da especificação da arquitetura do sistema e do código implementado supondo que os requisitos do sistema já são conhecidos, pode gerar muito trabalho ou nenhum.

Paralelamente, a fase Identificar Requisitos tem como objetivo identificar os requisitos funcionais que serão implementados com uma nova tecnologia de banco de dados.

A terceira fase Migrar Banco de Dados tem como objetivo realizar a migração do modelo de dados existente a partir dos requisitos funcionais e não funcionais.

As próximas seções apresentam mais detalhes sobre as fases do processo de migração para NoSQL.

3.1 Isolar Camada de Persistência

A aplicação deve permitir que o acesso aos dados seja exposto através de uma camada de persistência abstrata (sem dependência da fonte de dados).
(Schram & Anderson, 2012) (Sadalage & Fowler, 2012)

Nesta fase, deve ser feita uma análise para determinar se com a arquitetura atual é possível fazer a migração sem grandes impactos. Caso o resultado da análise seja positivo a migração segue para a próxima fase do processo, caso o resultado seja negativo é necessário que mudanças sejam feitas na aplicação.

O objetivo destas mudanças é que a aplicação tenha suporte à persistência hibrida e para isto, práticas de como utilizar SQL como um mecanismo de integração entre aplicações deve ser reavaliado. Neste cenário, o banco de dados relacional age como um banco de dados de integração, com várias aplicações, normalmente desenvolvidas por diferentes equipes, armazenando seus dados em um banco de dados comum.

Esta prática melhora a comunicação entre as aplicações pois todas estarão em operação num consistente conjunto de dados persistidos. Contudo, existem pontos negativos nesta integração feita com banco de dados compartilhado. Uma estrutura que é projetada para integrar muitas aplicações acaba sendo mais complexa de manter do que manter uma aplicação individual.

O cenário ideal é que o banco de dados seja da aplicação. Com um banco de dados de aplicação, somente a equipe responsável pela aplicação precisa conhecer a estrutura do banco de dados, o que permite que o modelo do banco de dados seja

mantido e evoluído mais facilmente. Como uma única aplicação controla o banco de dados, a responsabilidade da integridade dos dados pode ser atribuída à ela.

Desta forma, com os bancos de dados não sendo mais utilizados como mecanismo de integração entre aplicações, muitos adotaram a tecnologia Web Services como uma nova forma de comunicação para substituir os banco de dados relacionais como mecanismo de integração.

A mudança para Web Services como mecanismo de integração resulta em mais flexibilidade para a estrutura dos dados que estão sendo trafegados. Com a integração via banco de dados, os dados tem que ser estruturados de forma relacional. Porém, com serviços é possível construir estruturas de dados mais ricas, como por exemplo, registros aninhados e listas. Estas estruturas são normalmente representadas como documentos XML ou JSON. (Sadalage & Fowler, 2012)

Assim que a decisão de adotar um banco de dados de aplicação é tomada, tem-se mais liberdade para a escolha de um banco de dados. Como existe um desacoplamento entre o banco de dados e os serviços que se comunicam para fazer as integrações, os clientes destes serviços não precisam conhecer a forma de armazenamento dos seus dados, permitindo que bancos de dados não relacionais possam ser considerados como opção (Sadalage & Fowler, 2012) e que a implementação da persistência dos dados possa ser alterada e evoluída sem grandes impactos para os clientes (Schram & Anderson, 2012).

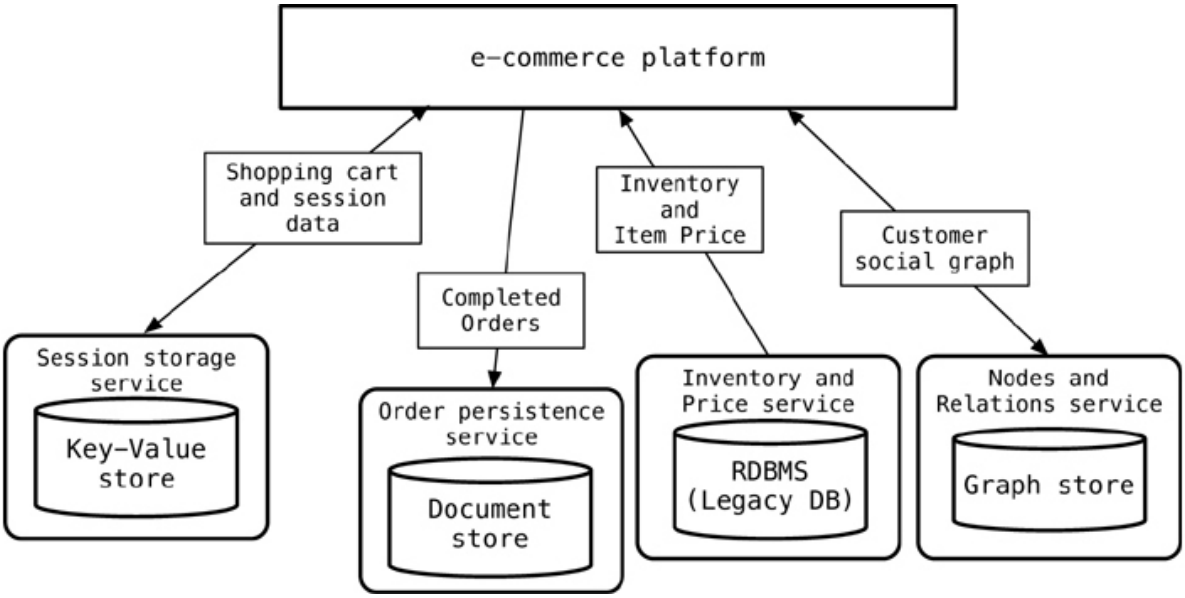

Figura 11 – Utilização de serviços para abstrair o acesso aos dados ao invés de acessar diretamente os bancos de dados (Sadalage & Fowler, 2012).

Na Figura 11, pode-se ver um exemplo de um sistema de comercio eletrônico em que ao invés de cada aplicação acessar diretamente o banco de dados, elas acessam os serviços que são responsáveis pelos dados. A ideia de possuir serviços responsáveis pelos dados é que podem existir outras aplicações que se beneficiem do uso destes dados. Por exemplo, outras aplicações que precisam obter os dados de pedidos não precisam acessar a base de dados do tipo documentos, *Document Store*. Para isto, elas podem simplesmente utilizar o serviço, *Completed Orders* que é responsável por estes dados (Sadalage & Fowler, 2012).

3.2 Identificar Requisitos

O objetivo desta fase do processo é identificar e avaliar os requisitos para que estes possam ser utilizados como insumo para fase seguinte do processo de migração.

A motivação para realizar a migração para NoSQL é a resolução de algum problema com a forma de manipular os dados do sistema. Normalmente, este problema está relacionado a limitações dos bancos de dados relacionais, listados na seção 2.3.

Estas limitações são critérios para eleger os requisitos funcionais que serão implementados com uma nova tecnologia de banco de dados, cuja motivação pode

ser originada de um requisito de negócio, de um requisito funcional ou de um requisito não funcional, que se transforma em um problema de manipulação dos dados.

Contudo, somente estes critérios não são suficientes. Identificar estes requisitos muitas vezes não é uma tarefa fácil, visto que é necessário isolar corretamente as funções do sistema para fazer o melhor uso destas novas tecnologias, levando-se em conta a modularização não só a nível de código, mas também a nível de sistema. Para isto, intenso esforço de engenharia de software é necessário.

A escolha dos requisitos cuja implementação pode ser migrada para NoSQL requer o entendimento dos processos de negócio e dos respectivos próprios requisitos implementados pela aplicação.

Os serviços implementados que possuem operações detalhadamente definidas, com entradas e saídas pré-definidas e especificas são candidatos a serem implementados usando NoSQL.

Outros serviços que necessitam de mais flexibilidade no acesso aos dados, por exemplo quando são realizadas consultas ad-hoc, ou serviços analíticos, são candidatos a continuarem trabalhando com o banco de dados relacional, pois com o SQL tem-se esta flexibilidade no acesso aos dados, que podem ser consultados de diversas formas atendendo a vários requisitos. (Schram & Anderson, 2012)

3.3 Migrar Banco de Dados

O objetivo desta fase é realizar a migração do modelo de dados da aplicação usando os requisitos eleitos na fase Identificar Requisitos. Antes disso, é necessário escolher o tipo de bancos de dados NoSQL.

3.3.1 Escolha do tipo de banco de dados NoSQL

Realizar a transição de um modelo de dados relacional, que possui uma estrutura fixa e bem definida, para um modelo da dados não relacional, que possui uma estrutura flexível, não é trivial pois existe pouco suporte de ferramentas e os *frameworks* e APIs disponíveis não possuem documentação detalhada e completa. Além de serem necessários conhecimentos profundos em sistemas distribuídos e habilidades em administração de sistemas (Schram & Anderson, 2012).

Estes conhecimentos são necessários pois na prática é de responsabilidade do desenvolvedor da aplicação decidir usar o banco de dados relacional ou não relacional (Leavitt, 2010), e com esta decisão outros aspectos, como consistência e integridade dos dados, disponibilidade e escalabilidade, que não são considerados normalmente devem ser estudados para a escolha do banco de dados.

Para a escolha do banco de dados, primeiramente, os dados que serão manipulados devem ser avaliados para identificar um modelo de dados compatível. Isto para evitar complexidade desnecessária devido a transformações e mapeamento destes dados (Hecht & Jablonski, 2011).

As consultas que a aplicação necessita que o banco de dados suporte também devem ser consideradas, pois esses requisitos de consulta tem grande influencia no design do modelo de dados, como também foi apresentado nas Técnicas de Modelagem, capitulo 2.3.1. (Hecht & Jablonski, 2011).

Além da escolha do tipo de banco de dados, é preciso escolher também o produto (ferramenta) que será utilizado. Isto porque os produtos de um mesmo tipo, apresentam características e formas de lidar com certos atributos diferentes.
O trabalho de Hetch e Jablonski (2011), cita os principais pontos a serem avaliados na escolha do produto e também do tipo de banco de dados :

- Possibilidades de query/ consulta
- Modelo de dados
- Controle de concorrência
- Particionamento

- Replicação e consistência
- Disponibilidade
- Mecanismo de Durabilidade
- Suporte do produto pela comunidade
- Versionamento

Mais detalhes para auxiliar nesta escolha podem ser vistos no ANEXO A – Comparação de bancos de dados NoSQL.

3.3.2 Migração do Modelo de Dados

Um dos principais desafios para uma migração de sucesso para banco de dados NoSQL é a conversão do modelo relacional existente para o modelo não relacional escolhido.

A forma de como o desenvolvimento de software utilizando a programação orientada a objetos vem sendo feito em conjunto com os bancos de dados relacionais evoluiu a ponto de permitir que as tabelas do banco de dados sejam geradas automaticamente a partir das classes, possibilitando que o desenvolvedor mantenha o foco em outros aspectos da construção do software.

O desenvolvimento de software com banco de dados relacionais já possui processos e técnicas amplamente conhecidas e por isso é facilmente mapeado com os conceitos de orientação a objetos.

Normalmente o modelo de dados é construído a partir do modelo de classes, as classes e atributos são traduzidos em tabelas e suas respectivas colunas. O relacionamento entre as classes pode ser facilmente mapeado com *frameworks* ORM nos diversos tipos (um para um, muitos para muitos, etc.). (Schram & Anderson, 2012)

Já no desenvolvimento com bancos de dados NoSQL diferentemente do desenvolvimento com bancos de dados relacionais, a construção do modelo de

dados começa pela definição de como os dados serão utilizados na implementação dos requisitos. (Schram & Anderson, 2012) (Cruz et al., 2011)

Para fazer isto, é necessário identificar os dados de entrada e saída esperados de cada operação. Com as entradas e saídas, o modelo de dados é criado de forma que a partir da entrada, a saída esperada seja facilmente obtida. Para isto, as consultas SQL existentes em cada operação que são baseadas em JOINS podem ser traduzidas para estruturas de dados onde eles são naturalmente indexados. (Cruz et al., 2011)

Para realizar a migração desta estrutura existem alternativas de modelagem que devem ser consideradas na construção do modelo ideal baseado nos requisitos funcionais e não funcionais. Essas alternativas de modelagem possuem também grande influência na escolha do tipo de banco de dados NoSQL mais apropriado. Algumas decisões de modelagem, vantagens e desvantagens de cada tipo de banco de dados estão descritas na seção 2.5.

3.4 Considerações do Capítulo

Este capitulo discutiu aspectos sobre a migração para NoSQL e apresentou um processo para migrar o modelo de dados relacional para um dos modelos utilizados pelos bancos de dados NoSQL. De acordo com este processo, a partir dos requisitos de negocio, requisitos de sistema e da arquitetura atual do sistema é possível realizar a migração do modelo de dados .

A migração do modelo de dados é feita logo após as atividades de Isolar a Camada de persistência e Identificar Requisitos que consistem respectivamente em uma avaliação da arquitetura do sistema, que deve permitir a utilização de um novo repositório de dados sem grandes impactos, e a identificação dos requisitos, que visa avaliar os requisitos, as necessidades e motivações da migração. Estas atividades são importantes para a tomada de decisões na migração do modelo de dados.

4. APLICAÇÃO DO PROCESSO DE MIGRAÇÃO

Este capítulo apresenta um exercício baseado em um sistema fictício, com intuito de mostrar a aplicação do processo de migração descrito no capitulo anterior.

Primeiramente, é feita uma breve apresentação do sistema e suas principais características e, em seguida, é apresentada a aplicação do processo de migração para este sistema. A necessidade de uso de um banco de dados NoSQL será explicada durante a fase Identificar requisitos.

O foco do exercício é mostrar a aplicação de algumas das técnicas de modelagem e as decisões envolvidas na migração do modelo de dados.

4.1 Descrição do Sistema

O sistema Adventurous é utilizado para criar e gerenciar planos de viagens através da Internet. Os principais requisitos de negocio estão listados no Apêndice A.

4.2 Processo de Migração para NoSQL

As seções a seguir descrevem a aplicação do processo de migração apresentado no capítulo 3 para o sistema Adventurous.

4.2.1 Isolar Camada de Persistência

O sistema possui uma arquitetura de 3 camadas: Apresentação, Serviço e Dados, conforme pode ser visto na Figura 12.

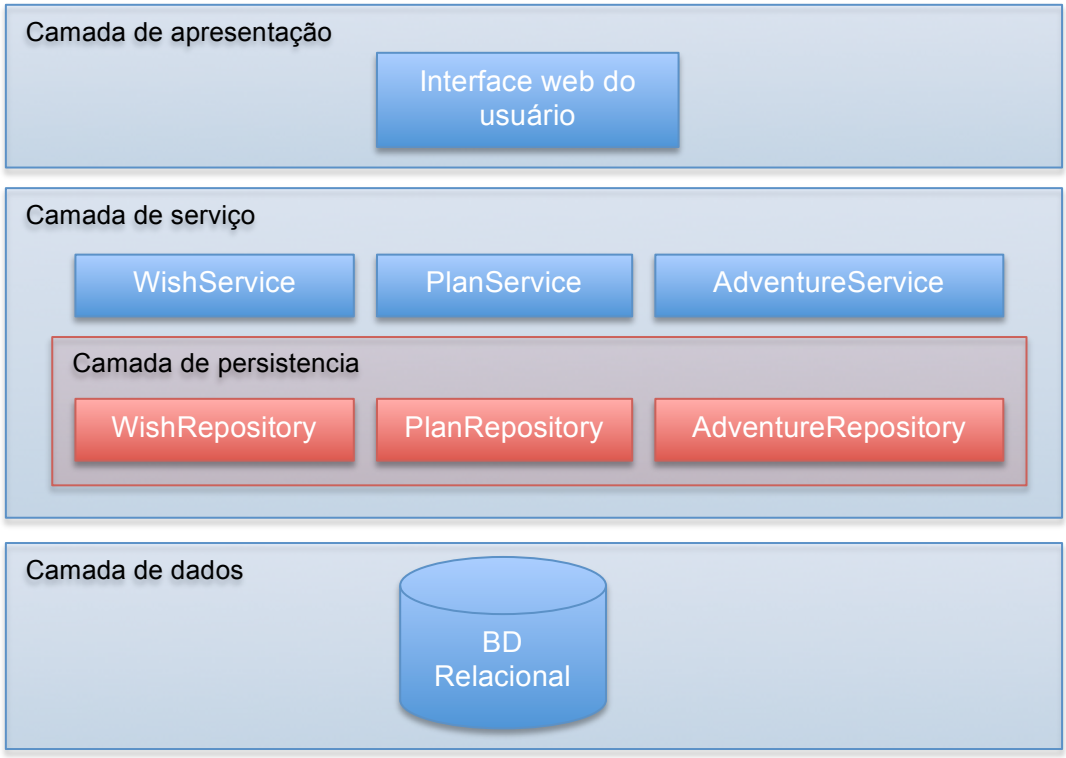

Figura 12 - Visão das três camadas do sistema antes da migração

A implementação dos requisitos principais do sistema está distribuída entre os três serviços: WishService, PlanService e AdventureService.

Na camada de Serviço, existe uma camada lógica de persistência que é utilizada por cada serviço. Cada serviço é responsável pelo acesso de um tipo de dado, por exemplo, o serviço WishService é responsável por todas as operações relacionadas aos desejos do usuário, e este acesso é feito através do repositório que também é responsável por um tipo de dado.

O acesso ao banco de dados relacional é feito através da camada lógica de persistência. Os repositórios fazem uso dos componentes presentes na camada de persistência que abstraem o acesso ao banco de dados relacional.

A arquitetura atual do sistema já possui uma camada de persistência desacoplada suficientemente para que as mudanças necessárias no acesso ao banco de dados sejam feitas com o menor impacto para os serviços.

4.2.2 Identificar Requisitos

Nesta fase, conforme descrito no seção 3.2 é necessário ter uma motivação para realizar a migração.

Para o sistema Adventurous, uma das motivações identificadas foi originada de um novo requisito de negócio que teve impacto em dois requisitos do sistema.

Requisito 2 (ver APÊNDICE A) : o sistema deve realizar a Coleta de informações para cada desejo, estas informações podem ser *links*, comentários, fotos, documentos e anotações.
Foi solicitado que o sistema também possa coletar informações obtidas de redes sociais, por exemplo: likes do Facebook, tweets do Twitter e pins do Pinterest.

Requisito 6 (ver APÊNDICE A) : o sistema deve realizar a Consulta de desejos do usuário. Deve ser possível consultar todas as informações disponíveis para o desejo. Devido a solicitação do Requisito 2, a consulta de um desejo do usuário deve exibir as informações disponibilizadas pelo Requisito 2.

O requisito original está implementado no serviço WishService (Figura 12), que além deste implementa também outros requisitos relacionados a um desejo :

- Requisito 1 : Criação de um desejo de realizar uma atividade ou experiência (sem associação á aventura).
- Requisito 5: Listagem de desejos do usuário que pode opcionalmente ser filtrada por categoria.

O serviço WishService utiliza o repositório WishRepository para manipulação dos dados. O WishRepository por sua vez é responsável pelo acesso aos dados no banco de dados relacional, e acessa estrutura de tabelas apresentada na Figura 13.

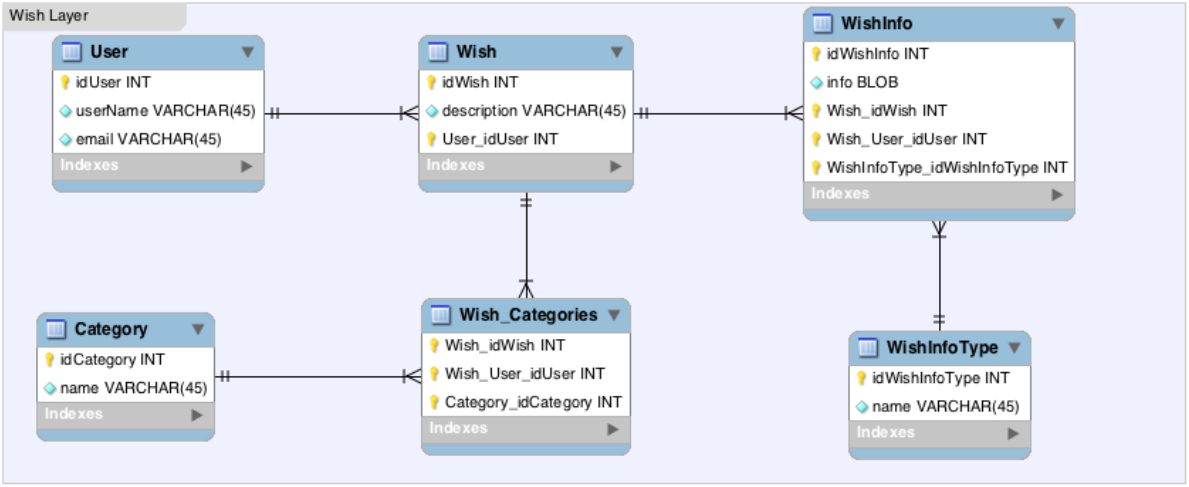

Figura 13 - Modelo de dados relacional utilizado pelo serviço WishService

Na tabela WishInfo, são armazenadas as informações de cada desejo. A fim de evitar a criação de uma estrutura rígida e complexa para suportar o conteúdo destas informações que podem variar entre um simples texto até dados mais complexos como a representação binária de arquivos ou listas de valores, estes dados foram armazenados em um campo do tipo BLOB.

Com esta solução, os dados ficam sem visibilidade para o banco de dados, ou seja não existe no banco de dados a estrutura para representar um link, uma foto, um uma anotação, um documento, pois estes dados estão sendo armazenados em sua representação binária num campo do tipo BLOB. Impedindo por exemplo que a aplicação utilize estes dados como filtro de pesquisa.

Estas informações coletadas para cada desejo são variáveis, cada uma possui atributos diferentes e também são de tipos diferentes. Idealmente, estes dados estariam armazenados em uma estrutura que os comporte de forma natural, o que certamente tornaria complexo o modelo de dados relacional que deveria possuir uma tabela para cada tipo de informação a ser coletada.

Como não há garantias de que novos tipos de informações não irão surgir, isto resultaria em uma quantidade ilimitada e imprevisível de tabelas fazendo com que as consultas se tornem complexas com muitos *joins*.

Com esta análise, chega-se a conclusão que para o sistema Adventurous, as principais motivações identificadas para a migração para NoSQL são a flexibilidade e complexidade dos dados (ver seção 2.3).

Portanto estes requisitos são elegíveis para migração para banco de dados NoSQL pois, como pode ser visto na Figura 13, o modelo de dados relacional relativo a este serviço não possui a flexibilidade requerida e se fosse alterado para atender aos requisitos se tornaria complexo.

4.2.3 Migrar Banco de Dados

Com base no modelo de dados relacional existente, são apresentadas técnicas de modelagem para atender aos requisitos escolhidos para serem migrados.

Para fazer a consulta de desejos do usuário, o sistema executa a consulta SQL apresentada na Figura 14.

```
SELECT wish.*,
       category.*,
       wishinfo.*
FROM   wish wish
       INNER JOIN wish_categories wishCategories
               ON wishCategories.wish_idwish = wish.idwish
       INNER JOIN category category
               ON wishCategories.category_idcategory=category.idcategory
       INNER JOIN wishinfo wishInfo
               ON wishinfo.wish_idwish = wish.idwish
       INNER JOIN wishinfotype wishInfoType
               ON wishinfotype.idwishinfotype =
                  wishinfo.wishinfotype_idwishinfotype
WHERE  wish.user_ = 1
       AND wish.idwish = 1
```

Figura 14 – Consulta SQL do Requisito 6

Analisando as entradas, nota-se que a consulta de um desejo é feita sempre pelo seu identificador único e pelo identificador do usuário.

Analisando a saída nota-se que são retornados todos os campos das tabelas, pois todos as informações serão utilizadas para a visualização do usuário.

Com isto é necessário um modelo que suporte a consulta com estas entradas e saídas. O modelo de dados orientado a documentos possui suporte e será utilizado com base nas principais motivações para realizar a migração, flexibilidade e complexidade dos dados (v. seção 4.2.1) e nas características dos tipos de bancos de dados NoSQL apresentadas na seção 2.4.

a. Documentos

As decisões de modelagem de documentos envolvem determinar como estruturar os documentos da melhor forma. Para isto, as duas decisões principais de modelagem: Documentos Embeded e Reference são aplicadas.

- Embeded

```
{
  "id": "Mergulho na Ilha de pascoa",
  "user": {
    "userName": "laisoliveira",
    "email": "laisoliveira@gmail.com"
  },
  "wishInfos": [
    {
      "info": "Info sobre quando",
      "wishInfoType": "WHEN"
    },
    {
      "info": "Info sobre onde",
      "wishInfoType": "WHERE"
    }
  ],
  "categories": [
    {
      "name": "TRAVEL"
    },
    {
      "name": "SPORTS"
    }
  ]
}
```

Figura 15 – Modelo de dados representado em Documento - Embeded

Conforme pode ser visto na Figura 15, o modelo conta com uma coleção de **wishes,** que resulta em um documento com as informações de um Wish. Neste modelo, o atributo **categories** representa um relacionamento N para N e o atributo **wishInfos** representa um relacionamento 1 para N.

Dentro do documento de **wish**, eles são estruturados dentro de um *array*. Neste caso é necessário avaliar a possibilidade de crescimento destes *arrays* dentro do documento de wish, pois os documentos em algumas ferramentas possuem um tamanho máximo (16MB no caso do MongoDB). Por exemplo, a coleção de **wishInfos**, que possivelmente pode crescer.

A vantagem deste modelo é que a quantidade de acessos ao banco de dados pela aplicação diminui, pois com uma única requisição todos os dados de um wish são recuperados.

- Reference

A aplicação deste modelo resulta em 4 documentos :
- Coleção de **wishes,** de documentos com as informações de um Wish. (Figura 16)

```
{
  "id": "1",
  "title": "Mergulho na Ilha de pascoa",
}
```

Figura 16 – Modelo de dados representado em Documento - Reference

- Coleção de **categories,** de documentos com as informações de um Category. (Figura 17)

```
"categories": [
   {
    "name": "TRAVEL",
    "id": 1
   },
   {
    "name": "SPORTS",
    "id": 2
   }
```

Figura 17 – Modelo de dados representado em Documento - Reference

- Coleção de **user,** de documentos com as informações de um User. (Figura 18)

```
{
   "id": 1,
   "userName": "laisoliveira",
   "email": "laisoliveira@gmail.com"
 }
```

Figura 18 – Modelo de dados representado em Documento - Reference

- Coleção de **wishInfos,** de documentos com as informações de um WishInfo (Figura 19)

```
{
   "id": 1,
   " wish": 1,
   "info": "Info sobre quando",
   "wishInfoType": "WHEN"
}
```

Figura 19 – Modelo de dados representado em Documento - Reference

Com este modelo, é necessário que que os JOINS que eram feitos anteriormente com a consulta SQL (Figura 14) sejam agora feitos programaticamente pela aplicação para obter todos os dados de um WishInfo, pois os dados estão espalhados em diversos documentos e não existe *join* de documentos. Este modelo de documentos representa um modelo relacional, a diferença é que está armazenado num banco de dados de documentos.

A consistência e integridade dos dados pode ser mais facilmente afetada com este modelo também, pois para a criação de um Wish, por exemplo, a escrita é feita em diversos documentos .

4.3 Arquitetura Após a Migração

Após a migração a arquitetura da aplicação pode ser representada como na Figura 20.

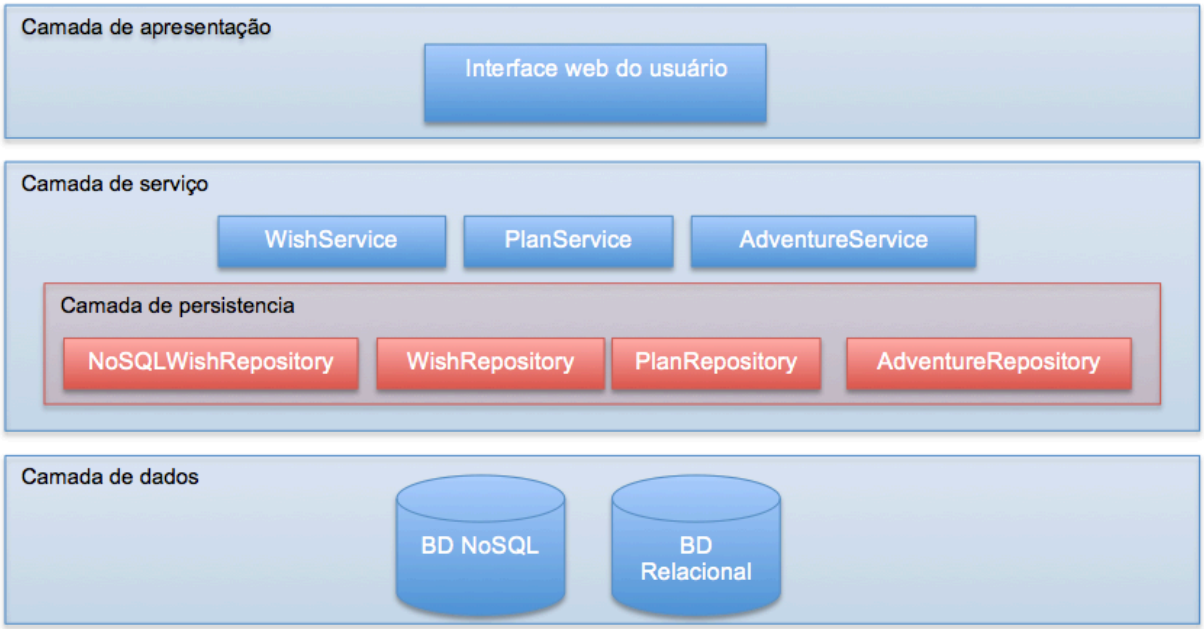

Figura 20 - Visão das três camadas do sistema após a migração

Um novo repositório, NoSQLWishRepository foi adicionado para manipular os dados no banco de dados NoSQL. Este novo repositório será utilizado pelo WishService, para a operação de consulta de desejos do usuário.

4.4 Considerações do Capítulo

Por ser um exercício de um sistema fictício, o proposito do exercício foi ilustrar a aplicação do processo com foco nas técnicas de modelagem. Por esta razão a lista de requisitos disponível no Apêndice A que foi utilizada na aplicação do processo foi descrita de forma superficial. Em um caso real, espera-se que o insumo para a fase

de Identificar Requisitos seja conforme descrito na seção 3.2, com os requisitos funcionais e requisitos não funcionais de sistema.

A motivação para migração foi impulsionada por um novo requisito do sistema. Nos trabalhos de Schram & Anderson (2012) e Cruz et al. (2011) utilizados como principal referência sobre a migração para NoSQL, a motivação de ambos foi o grande volume de dados que estava causando problemas de desempenho. Porém, a migração desta aplicação se justifica pelos requisitos de flexibilidade e complexidade.

5. CONSIDERAÇÕES FINAIS

Neste trabalho é possível ver que mais de um resultado pode ser obtido na aplicação do processo de migração para NoSQL a partir de um mesmo modelo de dados relacional. A variação destes resultados está ligada à escolha do tipo de banco de dados NoSQL e as técnicas de modelagem aplicadas.

As principais dificuldades foram :

• Mapeamento das diferentes decisões de modelagem.
As decisões de modelagem apresentadas para cada tipo de banco de dados foram identificadas a partir de uma grande quantidade de fontes, como exemplos de utilização dos bancos de dados nos livros e artigos e documentação de fornecedores dos bancos de dados, e acredito que foram um dos pontos chave para a compreensão da migração para NoSQL.

• Mapeamento das questões envolvidas nas decisões da migração e alternativas.
As decisões de modelagem são complexas e envolvem a análise de muitos requisitos funcionais e não funcionais do sistema, por isto foi difícil explicar estas decisões fora do contexto de um sistema real.

• Utilização das técnicas de modelagem no processo de migração.
No exercício do processo de migração apresentado foi difícil utilizar e avaliar as decisões de modelagem por se tratar de um sistema fictício que não possui a complexidade de um sistema real.

5.1 Contribuições do Trabalho

O processo de migração que foi proposto a partir da leitura dos dois trabalhos que foram as principais referencias, Schram & Anderson (2012) e Cruz et al. (2011) são complementares e coincidem. A primeira que foi a principal referência conta os desafios e o trabalho de migração feito com alto nível de detalhe, enquanto a segunda conta também o trabalho de migração com menos detalhes. Foi possível

identificar os passos para realizar a migração e observar as dificuldades e pontos de atenção.

O trabalho mostrou a importância que as decisões de modelagem de dados possuem no processo de migração. Além do modelo de dados NoSQL que melhor se encaixa para cada caso de uso, é preciso conhecer as diferentes técnicas de modelagem e suas implicações. Ou seja, cada tipo de banco de dados NoSQL possui um modelo de dados, porém isso não significa que a partir de um modelo de dados relacional, o resultado do modelo para o tipo escolhido será sempre igual.

Isto porque a variação destes resultados está ligada não somente a escolha do tipo de banco de dados NoSQL, mas também às decisões de modelagem aplicadas.

Para um caso real, as decisões são bem mais difíceis, pois existem muitos outros requisitos importantes como, desempenho, escalabilidade, entre outros.
De modo geral, o trabalho foi importante para visualizar os passos e as principais considerações a serem tomadas na migração para bancos de dados NoSQL.

5.2 Trabalhos Futuros

Como trabalho futuro é possível explorar mais as alternativas de modelagem de dados. Neste trabalho foram considerados modelos básicos e de mais fácil compreensão, sem explorar mais detalhadamente os recursos de cada banco de dados.

Poderia ser discutida a aplicação da persistência poliglota, que faria uso de diversos bancos de dados de tipos diferentes no mesmo sistema.

Também pode ser considerado aplicar o processo de migração em uma aplicação real, para que possam ser considerados requisitos funcionais e não funcionais variados e mais complexos.

Outro trabalho interessante seria pesquisar e aplicar alternativas de modelagem e o processo de migração para os bancos de dados do tipo Grafo, que não foi discutido neste trabalho.

REFERÊNCIAS

BASHO. **From Relational to Riak**. Disponível em <http://basho.com/assets/RelationaltoRiakDEC.pdf>. Acesso em: 12 mai. 2013

BREWER, E. **CAP Twelve Years Later**: How the “Rules” Have Changed. Computer, vol. 45, no.2, p. 23 – 29, Fev. 2012.

COUCHBASE. **Making the Shift from Relational to NoSQL**. Disponível em <http://info.couchbase.com/Relational-to-NoSQL.html>. Acesso em: 29 mai. 2013.a

COUCHBASE. **Why NoSQL**. Disponível em <http://www.couchbase.com/why-nosql/nosql-database>. Acesso em: 22 mai. 2013.b

CRUZ, F.; GOMES, P; OLIVEIRA, R; PEREIRA, J. Assessing NoSQL Databases for Telecom Applications. In**: Commerce and Enterprise Computing**, 2011, Luxembourg. p. 267 – 270.

EDLICH, S. **The State of NoSQL.** InfoQ NoSQL eMag, vol. 1, p. 4 – 8, Maio. 2013.

HECHT, R.; JABLONSKI, S. NoSQL Evaluation: A Use Case Oriented Survey. In: **Cloud and Service Computing**, 2011, Hong Kong. p. 336 – 341.

KATSOV, I. **NoSQL Data Modeling Techniques**. Disponível em: <http://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques>. Acesso em: 17 mar. 2013

LEAVITT, N. **Will NoSQL Databases Live Up to Their Promise.** Computer, vol. 43, no. 2, p. 12-14, Fev. 2010.

MONGODB. **Data Modeling**. Disponível em <http://docs.mongodb.org/manual/data-modeling>. Acesso em: 17 mar. 2013

POKORNY, J. NoSQL Databases**:** a step to database scalability in Web environment. In: **Information Integration and Web-based Applications and Services**, 2011, Ho Chi Minh City. p. 278 – 283.

REDMOND, E.; WILSON, J. R. **Seven Databases in Seven Weeks**: A Guide to Modern Databases and the NoSQL Movement. 1. ed. Dallas: Pragmatic Bookshelf, 2012. 333 p.

SALADAGE, P. J.; FOWLER, M. **NoSQL Distilled**: A Brief Guide to the Emerging World of Polyglot Persistence. 1. ed. Crawfordsville: Addison-Wesley, 2012. 192 p.

SANDERS, G.L.; SHIN, S. Denormalization Effects on Performance of RDBMS. In: **System Sciences**, 2001, Hawaii. p. 336 – 341.

SCHRAM, A.; ANDERSON, K. M. MySQL to NoSQL: Data Modelling Challenges in Supporting Scalability. In: **SPLASH**, 2012, Tucson. p. 191 – 202.

ZHANG, H.; WANG, Y.; HAN, J. Middleware Design for Integrating Relational Database and NOSQL Based on Data Dictionary. In: **Transportation, Mechanical, and Electrical Engineering (TMEE)**, 2011, Changchun. p. 1469 – 1472.

APÊNDICE A – REQUISITOS DE NEGÓCIO DO SISTEMA ADVENTUROUS

O sistema Adventurous é utilizado para criar e gerenciar planos de viagens através da Internet. Seus principais requisitos de negócio são:

1- Criação de um desejo de realizar uma atividade ou experiência. Os desejos podem ser opcionalmente associados à uma ou mais aventuras.

Exemplo:

Desejo 1: “Mergulho na Ilha de Pascoa”.

Desejo 2 : “Fazer um mochilão”.

Desejo 3 : “Escalar um vulcão”.

Aventura 1 : “Férias 2013”, que contém os desejos 1, 2 e 3.

Aventura 2 : “Férias 2014”, que contém o desejo 2.

2- Coleta de informações para cada desejo, que podem ser *links*, comentários, fotos, documentos, anotações e devem responder as seguintes perguntas:

a. O que é esta atividade ou experiência?

Exemplo de um link de um site que contém informações gerais sobre Mergulhos na Ilha de Pascoa:

http://www.brasilmergulho.com/port/points/inter/ilha_pascoa/index.shtml

b. Quando este desejo poderá ser realizado?

Exemplo de uma anotação de possíveis datas: “Estarei em férias no final do ano, mas talvez fevereiro seja um mes mais interessante de ir devido ao clima”

c. Onde este desejo pode ser realizado?

Exemplo de arquivo PDF: um guia sobre a ilha e os pontos de mergulho.

d. Quem está envolvido neste desejo?

Exemplo de uma anotação: “Pessoas interessadas: Maria – maria@gmail.com, José – jose@gmail.com”.

e. Por que realizar este desejo?

Exemplo de fotos que inspiraram este desejo.

f. Como realizar este desejo?

Exemplo de um link de um site de uma operadora de mergulho: http://www.mikerapu.cl.

Cada desejo pode conter mais de uma informação de cada tipo (o quê, quando, onde, quem, por quê e como)

3- Criação de um ou mais planos para os desejos. Os planos devem conter informações concretas sobre a execução da atividade ou experiência desejada, como o preço, duração e local. Um plano pode conter vários desejos.

4- Gerenciamento da execução de um plano. A Aventura contém um plano escolhido para ser executado e informações necessárias para a execução das atividades e experiências planejadas. Exemplo : Documentos de reservas, comprovantes e *checklists*.

5- Listagem de desejos do usuário que opcionalmente pode ser filtrada por categoria.

6- Consulta de desejos do usuário. Deve ser possível consultar todas as informações disponíveis para o desejo.

ANEXO A – COMPARAÇÃO DE BANCOS DE DADOS NOSQL

Redmond, Wilson (2012) disponibilizaram várias tabelas com comparações entre diversas ferramentas de banco de dados NoSQL. Estas tabelas auxiliam na escolha do banco de dados mais adequado.

	Genre	Version	Datatypes	Data Relations
MongoDB	Document	2.0	Typed	None
CouchDB	Document	1.1	Typed	None
Riak	Key-value	1.0	Blob	Ad hoc (Links)
Redis	Key-value	2.4	Semi-typed	None
PostgreSQL	Relational	9.1	Predefined and typed	Predefined
Neo4j	Graph	1.7	Untyped	Ad hoc (Edges)
HBase	Columnar	0.90.3	Predefined and typed	None

	Standard Object	Written in Language	Interface Protocol	HTTP/REST
MongoDB	JSON	C++	Custom over TCP	Simple
CouchDB	JSON	Erlang	HTTP	Yes
Riak	Text	Erlang	HTTP, protobuf	Yes
Redis	String	C/C++	Simple text over TCP	No
PostgreSQL	Table	C	Custom over TCP	No
Neo4j	Hash	Java	HTTP	Yes
HBase	Columns	Java	Thrift, HTTP	Yes

	Ad Hoc Query	Mapreduce	Scalable	Durability
MongoDB	Commands, mapreduce	JavaScript	Datacenter	Write-ahead journaling, Safe mode
CouchDB	Temporary views	JavaScript	Datacenter (via BigCouch)	Crash-only
Riak	Weak support, Lucene	JavaScript, Erlang	Datacenter	Durable write quorum
Redis	Commands	No	Cluster (via master-slave)	Append-only log
PostgreSQL	SQL	No	Cluster (via add-ons)	ACID
Neo4j	Graph walking, Cypher, search	No (in the distributed sense)	Cluster (via HA)	ACID
HBase	Weak	Hadoop	Datacenter	Write-ahead logging

	Secondary Indexes	Versioning	Bulk Load	Very Large Files
MongoDB	Yes	No	mongoimport	GridFS
CouchDB	Yes	Yes	Bulk Doc API	Attachments
Riak	Yes	Yes	No	Lewak (deprecated)
Redis	No	No	No	No
PostgreSQL	Yes	No	COPY command	BLOBs
Neo4j	Yes (via Lucene)	No	No	No
HBase	No	Yes	No	No

	Requires Compaction	Replication	Sharding	Concurrency
MongoDB	No	Master-slave (via replica sets)	Yes	Write lock
CouchDB	File rewrite	Master-master	Yes (with filters in BigCouch)	Lock-free MVCC
Riak	No	Peer-based, master-master	Yes	Vector-clocks
Redis	Snapshot	Master-slave	Add-ons (e.g., client)	None
PostgreSQL	No	Master-slave	Add-ons (e.g., PL/Proxy)	Table/row writer lock
Neo4j	No	Master-slave (in Enterprise Edition)	No	Write lock
HBase	No	Master-slave	Yes via HDFS	Consistent per row

	Transactions	Triggers	Security	Multitenancy
MongoDB	No	No	Users	Yes
CouchDB	No	Update validation or Changes API	Users	Yes
Riak	No	Pre/post-commits	None	No
Redis	Multi operation queues	No	Passwords	No
PostgreSQL	ACID	Yes	Users/groups	Yes
Neo4j	ACID	Transaction event handlers	None	No
HBase	Yes (when enabled)	No	Kerberos via Hadoop security	No

	Main Differentiator	Weaknesses
MongoDB	Easily query *Big Data*	Embed-ability
CouchDB	Durable and embeddable clusters	Query-ability
Riak	Highly available	Query-ability
Redis	Very, very fast	Complex data
PostgreSQL	Best of OSS RDBMS model	Distributed availability
Neo4j	Flexible graph	BLOBs or terabyte scale
HBase	Very large-scale, Hadoop infrastructure	Flexible growth, query-ability