Instituto Federal de Educação Ciência e Tecnologia do Rio Grande do Norte
Campus Pau dos Ferros
Curso de Análise Desenvolvimento de Sistemas

Migração de Banco de Dados Relacional para Orientado a documentos
Estudo de caso do Projeto SIADE

Pau dos Ferros – RN
Outubro de 2015

Roldão Gameleira do Rego Junior

Migração de Banco de Dados Relacional para Orientado a documentos
Estudo de caso do Projeto SIADE

Trabalho de Conclusão de Curso submetido ao Curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte, em cumprimento às exigências legais como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador: Fernando Wagner Brito Hortencio Filho

Pau dos Ferros – RN
Outubro de 2015

Roldão Gameleira do Rego Junior

Migração de Banco de Dados Relacional para Orientado a documentos
Estudo de caso do Projeto SIADE

Trabalho apresentado e aprovado em 21 de outubro de 2015

Fernando Wagner Brito Hortencio Filho
Orientador
Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte

---

Clayton Maciel Costa
Membro da banca
Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte

---

Jeferson Queiroga Pereira
Membro da banca
Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte

Pau dos Ferros
Outubro de 2015

RESUMO

A evolução dos dispositivos portáteis e a criação dos smartphones, permitiu o acesso fácil e rápido a vários tipos de aplicações. O sistema SIADE foi criado com objetivo de informatizar o processo de registro dos dados obtidos pelo agente de endemias. O sistema consiste em um aplicativo para tablets usado para registrar as atividades realizadas pelos agentes de endemias e um aplicativo web usado pelos supervisores dos agentes para visualizar os dados coletados. Em seu funcionamento, os dados precisam ficar armazenados no dispositivo e depois serem transferidos para o servidor. Porém, foi identificado que a sincronização de dados entre o dispositivo móvel e o servidor possuía problemas relacionados a consistência de dados sincronizados. Para resolver esse problema de sincronização, buscou-se verificar a possiblidade de se alterar o sistema gerenciador de banco de dados utilizado para CouchDB, aproveitando seu suporte à sincronização de dados e à dispositivos móveis. O esquema de dados relacional foi adaptado para o modelo orientado a documentos e foram feitas adaptações no sistema web e dispositivo móvel. Em seguida criou-se testes de desempenho para a verificar a viabilidade da adaptação o sistema.

Palavras chave: banco de dados, NoSQL, aplicação web, aplicativo móvel.

ABSTRACT

The evolution of handheld devices and the creation of smartphones, allowed easy and quick access to various applications. The SIADE system was created in order to computerize the registration process of data obtained by endemics agent. The system consists of an application for tablets used to record the activities of the endemics agents and a web application used by supervisors of agents to view the collected data. In its operation, the data needs to be stored on the device and then be transferred to the server. However, it was identified that the synchronization of data between the mobile device and the server had problems related to consistency of synchronized data. To resolve this synchronization problem, we sought to verify the possibility of changing the database management system used to CouchDB, taking advantage of its support for data synchronization and mobile devices. The relational data schema has been adapted to the driven model documents and adaptations were made in the web system and mobile. Then it was created performance tests to verify the feasibility of adapting the system.

Keywords: database, NoSQL, web aplication, mobile application.

Sumário

Lista de Ilustrações

1 Introdução

A forma como nos comunicamos, trocamos informações e criamos conteúdo mudou muito ao longo dos anos. Estamos vivenciando uma época onde as aplicações têm revolucionado o mundo em diversos sentidos e a tendência é que este crescimento habilite a criação de uma série de novas aplicações.

A evolução dos dispositivos portáteis e a criação dos smartphones, permitiu que as aplicações estejam sempre na palma da mão. Hoje em dia temos aplicativos em nossos celulares que nos fornecem acesso a serviços variados. A comunicação nesses dispositivos envolve várias questões e desafios de pesquisa. Neste cenário as desconexões são frequentes, fazendo-se necessário um serviço de comunicação que leve em consideração o estado da rede e o tipo de conexão.

Além disso, a quantidade de dados gerada diariamente por aplicações Web, rede sociais, redes de sensores, dados de sensoriamento, entre diversos outros, é muito grande. Essa imensa quantidade de dados gerados também traz novos grandes desafios na forma como os dados são armazenados e manipulados e processados.

Nesse contexto, os sistemas de banco de dados tradicionais se mostraram não ser os mais adequados a esses novos desafios: tratamento de grandes volumes de dados e suporte simples a replicação e distribuição dos dados (SADALAGE e FOWLER, 2013). Os sistemas de banco de dados NoSQL (*Not Only SQL*) surgiram como um esforço no intuito de superar esses desafios. Tais bancos trabalham de forma diferente dos sistemas de banco de dados tradicionais, oferecendo recursos que para lidar com grande quantidade de dados.

Neste trabalho iremos abordar alguns aspectos desses sistemas de banco de dados e avaliar a migração de um sistema que utiliza banco de dados tradicional para um sistema de banco de dados NoSQL.

O restante deste trabalho está dividido da seguinte forma: no restante desse capítulo são apresentados a justificativa e os objetivos do trabalho; no capítulo 2 trata do referencial teórico acerca dos conceitos e tecnologias utilizadas neste trabalho; no capítulo 3 é apresentado uma visão geral do sistema SIADE, sua arquitetura e tecnologias utilizadas; no capítulo 4, fala-se como foi realizada a adaptação do sistema para o banco de dados CouchDB; no capítulo 5, são descritos os

experimentos realizados e seus resultados. No capítulo 6, é feita uma recapitulação geral do trabalho e as conclusões sobre os experimentos.

1.1 Justificativa

A dengue constitui um dos principais problemas de saúde pública no Brasil e no mundo. Trata-se de doença viral de transmissão vetorial, e dentre estas, é a que mais causa impacto em termos de mortalidade na população mundial. O governo brasileiro, na tentativa de sanar esse problema, dispõem de agentes de combate de endemias. Estes realizam a vistoria em residências, depósitos, terrenos baldios e estabelecimentos comerciais, na busca de focos endêmicos. Para facilitar esse trabalho, foi criado o sistema SIADE cujo objetivo é informatizar o processo de registro dos dados obtidos pelo agente de endemias. O sistema consiste em um aplicativo para tablets usado para registrar as atividades realizadas pelos agentes de endemias e um aplicativo web usado pelos supervisores dos agentes para visualizar os dados coletados.

Por se tratar de um dispositivo móvel, poderá haver momentos em que o tablet não tenha acesso à Internet. Então para melhor funcionamento do sistema, os dados precisam ficar armazenados no dispositivo e depois serem transferidos para o servidor. O sistema implementa um protocolo de comunicação para transferir esses dados. Porém foi identificado que a parte do sistema que é responsável pela sincronização de dados entre o dispositivo móvel e o servidor possuía problemas relacionados a consistência de dados sincronizados. O sistema foi utilizado em fase de testes pelos agentes de endemias da Secretaria Municipal de Saúde do município de Pau dos Ferros. Eles relataram que o sistema por vezes os dados inseridos no tablet não sincronizavam ou que os dados do servidor não apareciam no tablet.

Para resolver esse problema, necessitou-se buscar melhorias na sincronização de dados entre o dispositivo móvel e o servidor. Após diversas pesquisas sobre sincronização de dados, verificou-se que o sistema gerenciador de banco de dados CouchDB tem como uma de suas principais características suporte nativo à sincronização de banco de dados. Esse gerenciador também possui uma versão para dispositivos móveis.

Portanto este trabalho busca verificar a possiblidade de resolver o problema de sincronização no SIADE alterando o sistema gerenciador de banco de dados utilizado

para CouchDB, aproveitando o suporte à sincronização de dados e à dispositivos móveis.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo deste trabalho é realizar a migração do banco de dados do sistema SIADE de uma tecnologia relacional para uma solução não-relacional e orientada a documentos. Será utilizado CouchDB[1], com o objetivo de permitir ao sistema SIADE utilizar os recursos oferecidos por ele para consistência e sincronização de dados.

1.2.2 Objetivos específicos

Como objetivos específicos deste trabalho, citam-se:

- Abordar as soluções de persistência NoSQL relativas a orientação a documentos
- Estudar o esquema relacional de banco de dados do SIADE e adaptá-lo para ser utilizando em banco de dados orientado a documentos
- Migrar o banco de dados relacional do servidor do SIADE para uma solução orientada a documentos.
- Migrar o banco de dados relacional do aplicativo móvel SIADE para uma solução orientada a documentos.
- Substituir a API de sincronização existente pela disponibilizada pelo CouchDB.
- Realizar testes de desempenho.

[1] CouchDB é um servidor de banco de dados baseado em documentos que utiliza JSON para armazenar e transmitir os dados (REDMOND e WILSON, 2012).

2 Fundamentação Teórica

2.1 Banco de dados relacionais

O modelo de dados mais popular das duas últimas décadas é o modelo relacional. O modelo relacional consiste em armazenar os dados em um conjunto de tabelas, onde cada tabela possui várias linhas e cada linha é uma entidade de interesse. A entidade é descrita por meio de colunas, cada uma possuindo apenas um valor. Uma coluna pode se referir a uma outra coluna na mesma tabela ou em uma tabela diferente (SADALAGE e FOWLER, 2013, p. 41).

Uma das características mais importantes do sistema de gerenciamento de bancos de dados (SGDB) relacionais é o conceito de transação. Uma transação é uma sequência de operações que são tratadas como uma única operação. As transações em banco de dados relacionais possuem quatro propriedades, popularmente conhecidas como propriedades ACID. ACID é um acrônimo para *Atomic* (Atomico) *Consistent* (Consistente) *Isolated* (Isolado) *Durable* (Durável).

Com o surgimento de aplicações que necessitam manipular uma quantidade cada vez maior de dados, percebeu-se que os bancos de dados relacionais não eram mais eficientes. Ao utilizar bancos de dados relacionais com grandes quantidades de dados surgem problemas como falta de eficiência no processamento, paralelização não efetiva, alto custo e escalabilidade limitada (WANZELLER, 2013, p. 11).

2.2 Big-data

A quantidade de informação disponível atualmente é enorme e cresce à medida que o conhecimento humano se expande. Além da quantidade há um aumento também na variedade e na velocidade com que precisamos acessar essas informações.

> Geramos enormes quantidades de dados todos os dias, e essa quantidade só tende a aumentar. Redes sociais, dispositivos móveis que guardam nossas informações, sites que armazenam nossas preferências, dispositivos de busca que indexam as páginas da web e a popularização da computação em nuvem nos colocam em uma época de grande volume de dados, uma época em que tudo é informação, tudo é valioso, tudo pode ser extraído. (WANZELLER, 2013).

Anos atrás, grandes bancos de dados eram medidos em megabytes. Hoje, é em terabytes, petabytes ou mesmo exabytes, portanto as formas tradicionais de organização e acesso de dados, os sistemas de gerenciamento de banco de dados, devem ser repensadas, reinventadas ou substituídas (WASCHKE, 2012, p. 1).

Ainda não há uma definição precisa para Big Data, mas pode-se usar o termo para designar “um conjunto de tendências tecnológicas que permite uma nova abordagem para o tratamento e entendimento de grandes conjuntos de dados” (SILVA e BRETERNITZ, 2013, p. 107). Big Data descreve um conjunto de problemas e suas soluções tecnológicas em computação aplicada com características que tornam seus dados difíceis de tratar. Há consenso de que três características são as principais: volume, velocidade e variedade. (XÉXEO, 2013, p. 19).

O aumento do volume de dados exige um aumento dos recursos computacionais. Para comportar esse crescimento, a alternativa mais barata é a utilização de clusters. Porém a utilização de clusters leva a um novo problema: banco de dados relacionais não foram projetados para executar em clusters (SADALAGE e FOWLER, 2013, p. 32). Para sanar essa necessidade surgiram os sistemas de banco de dados NoSQL.

2.3 Banco de dados NoSQL

Os sistemas de banco de dados NoSQL que, diferentemente dos relacionais, são pensados e otimizados para serem executado em clusters de modo que são mais apropriados para grandes quantidades de dados (SADALAGE e FOWLER, 2013, p. 15). Ou seja, diferentemente dos bancos de dados SQL, os bancos de dados NoSQL foram pensados e projetados para utilizar computação distribuída.

Esses sistemas de bancos de dados também permitem que sejam armazenadas estruturas mais complexas do que simples tabelas. Essas estruturas são comumente chamadas de agregados e são tratadas e manipuladas como uma unidade. (SADALAGE e FOWLER, 2013).

Em oposição as propriedades ACID, os SGDBs não relacionais possuem BASE (*Basically Available, Soft-state, Eventual Consistency*).

Basically Available (Disponível Basicamente) – foco na disponibilidade de dados mesmo em caso de falhas.

Soft-state (Estado leve) – A reponsabilidade de validação da estrutura dos dados é do aplicativo e não do banco de dados.

Eventual Consistency (Consistência eventual) – Poderá haver intervalos de tempo de inconsistência. Por isso, não há garantias que as operações de leitura retornarão sempre os dados atualizados na última operação de escrita.

Os bancos de dados NoSQL podem ser classificados, segundo seus modelos de dados, em: chave-valor, família de colunas, orientado a grafos e orientado a documentos.

2.3.1 Chave-valor

Os bancos de dados chave-valor são considerados os mais simples, pois o armazenamento dos dados consiste em uma simples tabela *hash*, onde o acesso aos dados se dá exclusivamente pela sua chave. Funciona similar uma tabela de um banco de dados relacional com apenas duas colunas (chave e valor), onde a coluna chave é a chave primária e a valor pode ser qualquer informação. (SADALAGE e FOWLER, 2013, p. 123).

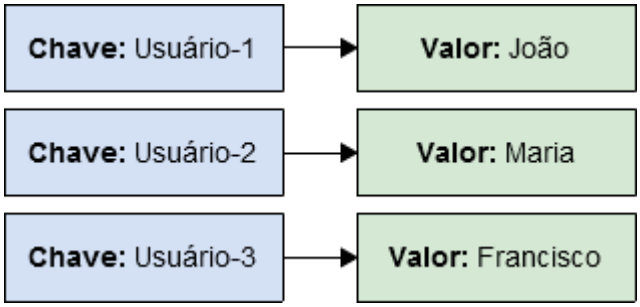


Figura 1 – Banco de dados chave-valor

A vantagem essa abordagem é que ela possui excelente desempenho e é capaz de lidar facilmente com grande quantidade de dados. A desvantagem é que uma vez armazenados, os dados só podem ser recuperados pela sua chave. Alguns exemplos de banco de dados desse tipo são: RIAK, LevelDB, Voldemort, Redis.

2.3.2 Família de colunas

Os bancos de dados de família de colunas são similares aos chave-valor. Porém eles permitem que sejam armazenados vários valores utilizando um conjunto de colunas associadas a chave. (SADALAGE e FOWLER, 2013, p. 147).

Nesse tipo de banco de dados o número de colunas das linhas não são fixos e podem variar conforme a necessidade. Alguns exemplos são de banco dados de família de colunas são o Cassandra e Hypertable.

Chave da linha	Colunas			
Linha1	Coluna1	Coluna2	Coluna3	Coluna4
	Valor	Valor	Valor	Valor
Linha2	Coluna1	Coluna2	Coluna3	
	Valor	Valor	Valor	
Linha3	Coluna1	Coluna4		
	Valor	Valor		

Figura 2 – Banco de dados de família de colunas

2.3.3 Orientado a grafos

Os bancos de dados orientado a grafos armazenam os dados em nós de um grafo e cujas arestas representam a associação entre os nós. Cada nó possui um conjunto de propriedades. “A organização em grafos permite que os dados sejam armazenados os dados uma vez e depois sejam interpretados de formas diferentes, baseada em relacionamentos” (SADALAGE e FOWLER, 2013, p. 161).

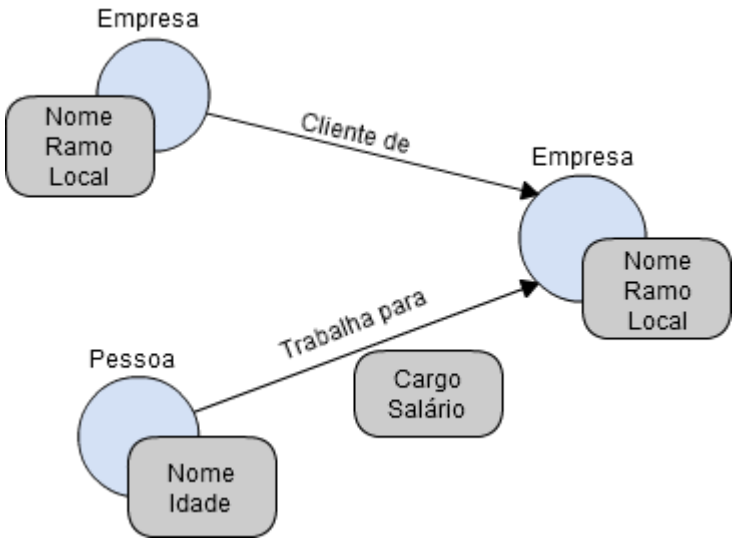


Figura 3 – Banco de dados de grafo

Em banco de dados de grafo acessar os registros relacionados é muito rápido, pois os relacionamentos não precisam ser calculados em tempo de consulta. Alguns exemplos são: Neo4J, infoGrid e FlockDB.

2.3.4 Orientado a documentos

Figura 4 – Banco de dados orientado a documento

Nos bancos de dados orientado a documentos os dados são encapsulados em pares de chave-valor. Cada documento possui um identificador único dentro de uma coleção de documentos. Os documentos são unidades básicas e não tem nenhuma estrutura definida, ou seja, não há um esquema de dados definido (WANZELLER, 2013). Ao armazenar os dados em JSON há uma vantagem adicional que é o suporte a tipos de dados, o que torna a forma de armazenamento mais amigável para os desenvolvedores. Alguns bancos de dados orientados a documentos são CouchDB, MongoDB e RavenDB.

2.4 Map-Reduce (Mapear-Reduzir)

A ascensão dos bancos de dados NoSQL deve-se, em grande parte na utilização de clusters. Executar uma aplicação em um cluster implica equilibrar o armazenamento dos dados de maneira diferente daquela executada em somente uma máquina. Isso significa que, se os dados estão armazenados em um cluster, para processá-los de forma eficiente é necessário um novo modo de organizar o processamento (SADALAGE e FOWLER, 2013, p. 107).

Map-reduce é um modelo de programação desenhado para processar grandes volumes de dados. Os usuários especificam uma função mapeadora (*map*) que processa um par chave-valor e gera um conjunto pares chave-valor intermediários, e uma função redutora (*reduce*) que juntas todos os valores intermediários com a mesma chave (DEAN e GHEMAWAT, 2004, p. 1).

Cada aplicação da função *map* é independente de todas as outras. Isso permite que elas executem em paralelo com segurança, de modo que uma operação map-reduce pode ser distribuída de forma eficiente em um cluster (SADALAGE e FOWLER, 2013, p. 108).

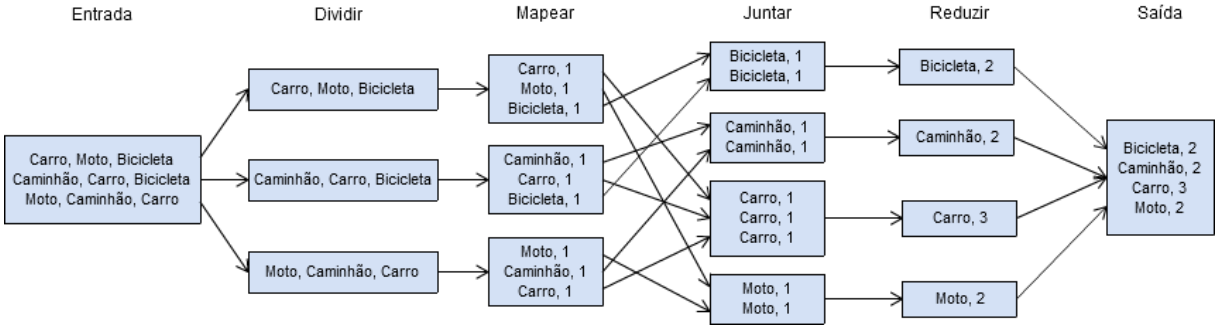


Figura 5 – Processo geral de uma operação Map-Reduce de contar palavras

A Figura 5 mostra o como poderia ser feita a contagem de palavras em um texto usando Map-Reduce. Primeiramente o conjunto de entradas é dividido em vários grupos para que possam ser processados em paralelo. Em seguida, é executada a função de mapeamento para cada entrada. Esses dados são ordenados e agrupados pela chave emitida pela função mapeadora. Para cada um desses grupos é executada a função redutora. Ao final, os resultados das funções de redutoras são unidos em enviada como resultado final da operação.

2.5 Modelagem de banco de dados NoSQL

Os bancos de dados NoSQL não requerem um esquema associado aos dados, porém esses dados devem possuir alguma estrutura. Essa estrutura é definida pelo aplicativo. Os dados devem ser mapeados para elementos disponíveis no SGDB de destino, como documentos ou pares chave-valor, por exemplo.

O projeto tradicional de banco de dados é um processo com três fases de modelagem de dados, os projetos conceitual, lógico e físico. Enquanto o objetivo do projeto conceitual é produzir um esquema expressivo e capaz de representar os dados de um domínio de informação, o objetivo do projeto lógico é transformar um esquema conceitual em uma representação equivalente em um modelo lógico que se aproxima do modelo de implementação do banco de dados. Porém para NoSQL existem poucos trabalhos que propõem metodologias de projeto lógico, com ênfase em processos que convertam modelagens conceituais para representações lógicas em modelos de dados (LIMA e MELLO, 2015, p. 3).

Modelagem de banco de dados NoSQL, diferentemente do modelo relacional, inicia-se a partir de consultas específicas da aplicação, pois é necessário saber como os dados serão acessados para saber como eles devem ser estruturados. Além disso é comum o uso de dados duplicados ou desnormalização (KATSOV, 2012).

2.5.1 Agregados atômicos

Uma razão pela qual as transações são parte de um banco relacional é porque quando os dados são normalizados, geralmente eles requerem a atualização em várias tabelas do banco de dados. Os bancos de dados NoSQL permitem que se armazene esses dados agregados como uma única entidade (documento, linha ou par chave-valor) e, portanto, atualizá-la de forma atômica (KATSOV, 2012).

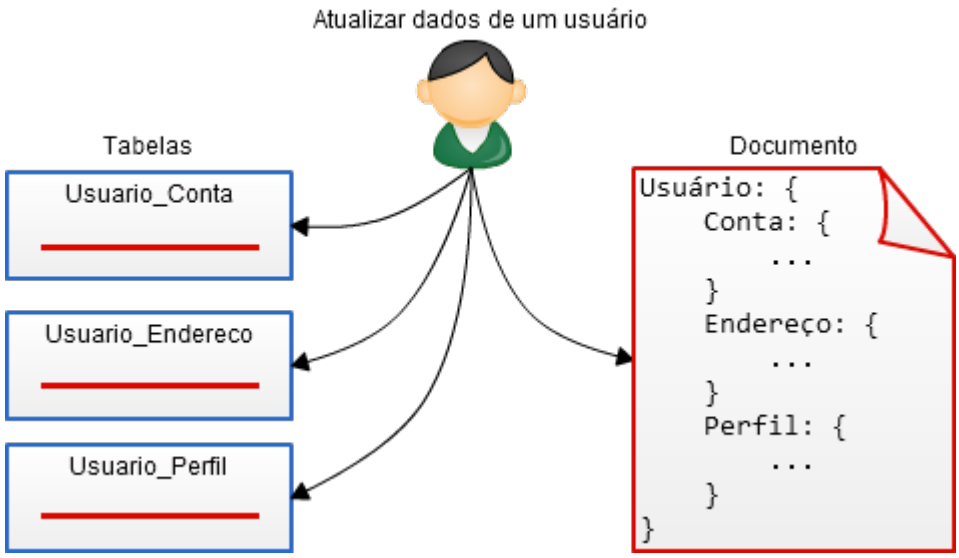
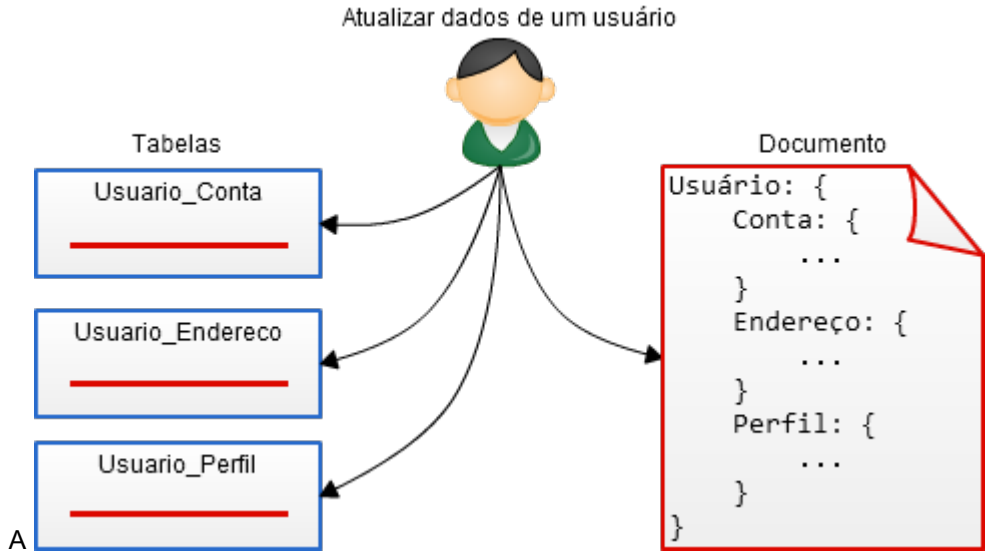


Figura 6 – Atualização de uma entidade de negócio na forma normalizada e agregada



A

Figura 6 mostra como seria atualização de um usuário em um sistema qualquer. Quando os dados estão normalizados é necessário executar a atualização em 3 tabelas (*Usuario_Conta, Usuario_Perfil, Usuario_Endereco*). Quando os dados estão agregados é necessário atualizar apenas um documento.

2.5.2 Junções em nível de aplicativo

Junções (*joins*) raramente são suportadas por tecnologias NoSQL. Porém usando banco de dados não-relacionais pode-se evitar junções na maioria dos casos. Porém existem situações em que elas são inevitáveis. Para esses casos, elas deverão ser feitas pelo aplicativo. Alguns casos são:

Relacionamentos muitos-para-muitos que são modelados como ligações entre registros e precisam ser usar junções.

Estruturas agregadas podem não ser aplicáveis quando a entidade mais interna está sujeita a modificações frequentes. Normalmente é melhor manter registros separados e juntá-los durante a consulta ao invés de alterar o valor. Por exemplo, um sistema de mensagens pode ser modelado como um usuário com vários registros de mensagem agregados. Porém se novas mensagens são adicionadas frequentemente, pode ser melhor criar as mensagens como registros separados e juntar durante a consulta (Figura 7).

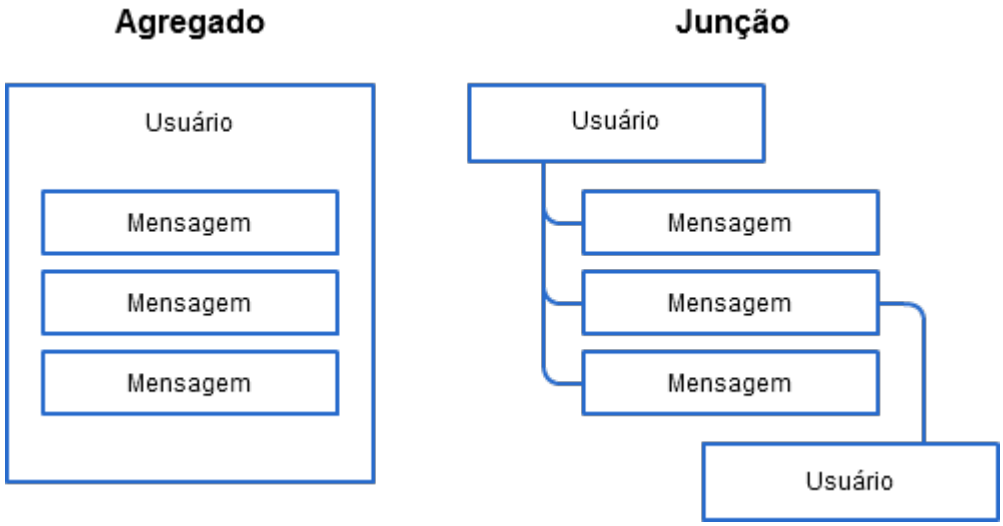


Figura 7 – Modelo de um sistema de mensagens usando agregados e usando junções

2.5.3 Índice inverso

Esta técnica consiste em modelar os relacionamentos 1-para-n utilizando uma lista de ids. Pode ser utilizado para substituir ou complementar a forma comum referência

Figura 8 – Usuários, categorias e cidades usando índices direto e inverso

A Figura 8 mostra como fica o relacionamento Usuário e Categorias usando índice direto e inverso. O índice inverso aparece como uma lista de usuários no documento da categoria e o índice direto aparece como uma categoria no documento do usuário. O mesmo acontece para cidade, o índice reverso aparece como uma lista de usuários do documento cidade.

2.6 CouchDB

CouchDB é um servidor de banco de dados escrito em ErLang[2], baseado em documentos e que utiliza JSON para armazenar e transmitir os dados (REDMOND e WILSON, 2012). Ele foi pensado para web, por isso, todas as chamadas são feitas através de uma interface REST[3] usando o protocolo *HyperText Transfer Protocol* (HTTP).

Operações de leitura no banco de dados nunca serão bloqueadas nem precisam esperar as operações de escrita, mesmo que seja no mesmo documento. As operações de leitura usam o conceito de controle de concorrência multi-versão (*Multi-Version Concurrency Control* – MVCC) onde há várias versões de um

[2] Erlang é uma linguagem de programação de uso geral e um sistema para execução. Foi desenvolvida pela Ericsson para suportar aplicações distribuídas e tolerantes a falhas a serem executadas em um ambiente de tempo real e ininterrupto (WIKIPEDIA, 2015).

[3] A Transferência de Estado Representativo (Representational State Transfer) é uma técnica de engenharia de software para sistemas hipermídia distribuídos. Trata-se de uma forma de enviar mensagens de servidor a outro através do protocolo HTTP.

documento e o cliente sempre vê uma versão consistente do banco de dados do início até o fim da operação de leitura (APACHE SOFTWARE FOUNDATION, 2015).

Em disco, o CouchDB nunca sobrescreve os dados gravados e estruturas associadas, eles são sempre adicionados ao fim do arquivo. Isso permite que a gravação dos dados em disco seja rápida e que o arquivo esteja sempre em um estado consistente (APACHE SOFTWARE FOUNDATION, 2015). Caso ocorra qualquer falha na gravação de um documento e este fique corrompido, pode-se simplesmente utilizar a versão anterior gravada no arquivo.

Essa técnica faz com que o arquivo de dados do CouchDB cresça rapidamente. Como forma de recuperar o espaço consumido pelo banco, há um processo de compactação do banco de dados que remove dados não utilizados decorrentes das atualizações no banco de dados. Ele fica disponível o tempo inteiro permitindo que todas as operações de leitura e escrita funcionem normalmente (APACHE SOFTWARE FOUNDATION, 2015).

Exceto pelas consultas feitas utilizando o identificador do documento, no CouchDB não é possível fazer consultas diretamente. Ao invés disso, ele utiliza o conceito de visões que permitem estruturar os documentos de forma que eles possam ser utilizados pelas aplicações.

2.6.1 Recursos de consulta

Para procurar e agregar informações armazenadas no banco de dados CouchDB, é necessário escrever uma visão (*View*). Visões convertem os documentos individuais em seu banco de dados em uma lista de informações. A partir dessa lista, pode-se consultar e selecionar a informação desejada ou pesquisar e encontrar grupos e documentos individuais (BROWN, 2012, p. 46).

Visões permitem criar novas formas de acessar os dados armazenados. São janelas para os documentos em um banco de dados (REDMOND e WILSON, 2012, p. 186). Elas permitem filtrar os documentos e encontrar qualquer valor existentes neles.

Uma visão consiste em uma função mapeadora (*map*) e uma função redutora (*reduce*) que são usadas para gerar uma lista ordenada de pares chaves-valor. Ambas chave e valor são valores válidos JSON. (REDMOND e WILSON, 2012, p. 186).

As funções map e reduce são escritas em *Javascript* e são armazenadas no banco de dados em documentos especiais denominados *design documents*. Como pode ser visto Figura 9, o documento possui um identificador especial e um atributo **views** que contém o código das views map-reduce do CouchDB.

```
{
   "_id": "_design/carros",
   "_rev": "1-6d8019c21cf68168e8f283725c6815c3",
   "language": "javascript",
   "filters": {
       "carros_por_tipo": "function(doc, req) { ... }"
   },
   "views": {
       "carros_por_marca_modelo_ano": {
           "map": "function(doc) { ... }"
           "reduce": "function (keys, values) { ... }"
       },
       "carros_por_preço": {
           "map": "function(doc) { ... }",
           "reduce": "function (keys, values) { ... }"
       },
   }
}
```

Figura 9 - Exemplo de um Design Document

Para permitir a consulta de dados rápida, o CouchDB armazena dentro do banco de dados os resultados da função map executada nos documentos e vai atualizando à medida que os documentos são inseridos ou alterados, funcionando de maneira similar a uma visão materializada em banco de dados relacional (APACHE SOFTWARE FOUNDATION, 2015).

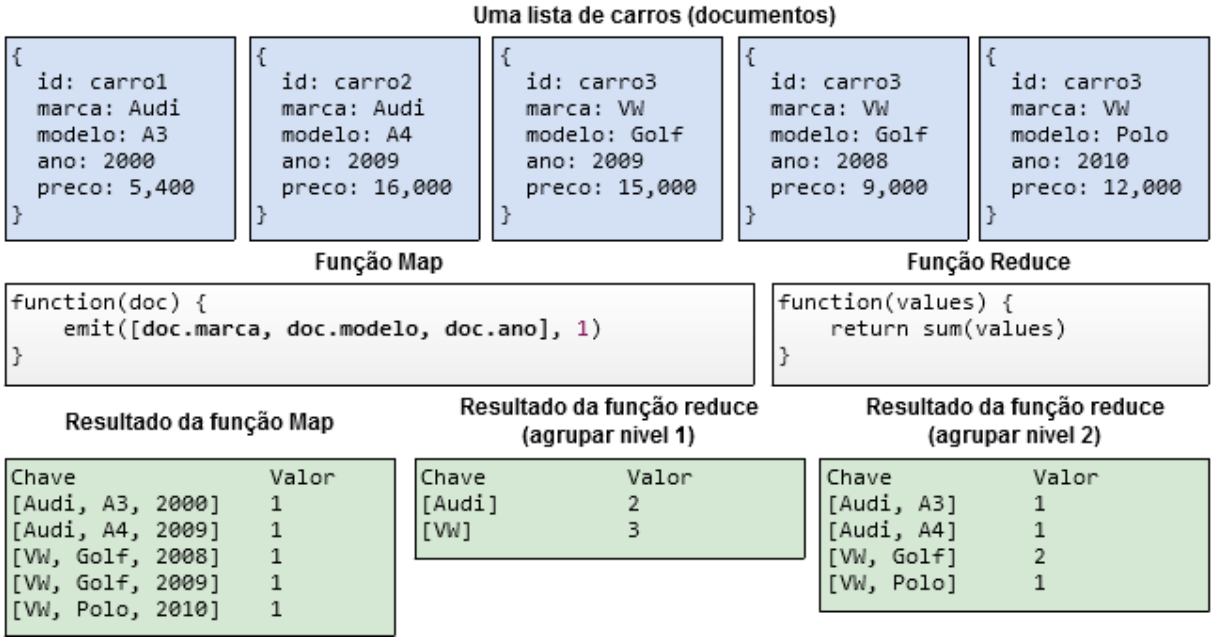


Figura 10 - Exemplo de uma view em CouchDB

A Figura 10 exemplifica o funcionamento da uma view *map-reduce* no CouchDB. Cada documento da lista de passa pela função mapeadora (através do parâmetro *doc*) e resulta em par chave-valor (retornado pelo comando *emit*). Nesse exemplo trata-se de uma chave composta. Essa chave pode ser utilizada para agrupar os dados na hora de executar a função redutora. Isso permite que, através da função *reduce*, o banco de dados conte os carros agrupando por marca, por marca e modelo ou por marca, modelo e ano.

2.6.2 Replicação de dados

Um dos recursos principais do CouchDB é a facilidade com que pode ser feita replicação de dados. Ela usa a mesma API REST utilizada para acesso aos documentos e suporta filtros e durante o processo. (BROWN, 2012, p. 4).

Para manter a consistência durante a replicação, o CouchDB realiza uma replicação incremental, um processo que consiste em copiar apenas as diferenças entre os bancos de dados (ANDERSON, LEHNARDT e SLATER, 2010, p. 16). Deve-se notar que a replicação de dados ocorre entre banco de dados e não entre servidores. Isso significa que é possível replicar bancos de dados na mesmo máquina ou diferentes banco de dados por vários servidores e dispositivos (BROWN, 2012, p. 4).

A replicação pode ser feita de suas formas: sob demanda ou contínua. O aplicativo requisita ao CouchDB sincronização de dois bancos de dados e o servidor

irá enviar as alterações do banco de dados de origem para o de destino. Também é possível requisitar ao servidor que mantenha uma replicação contínua das alterações do banco.

A replicação é feita em uma única direção, ou seja, o banco de dados A é replicado para o banco de dados B. Para que seja feita a replicação nas duas direções, devem ser feitas duas chamadas uma para replicar de A para B e outra de B para A.

Além disso, é possível realizar replicação de apenas uma parte dos documentos armazenados. No processo de replicação, o SGBD permite que seja especificada uma função de filtro que é executada para cada documento no banco de dados. Caso a função retorne verdadeiro o documento é replicado. Assim como as outras funções, elas ficam armazenadas em *Design Documents* (Figura 9).

O sistema de replicação do CouchDB vem com detecção e resolução de conflitos automática. Quando um documento é alterado em ambos bancos de dados, ele detecta que um documento foi alterado, sinaliza que o documento está em conflito. Quando duas versões conflitam durante uma replicação, o CouchDB automaticamente escolhe uma revisão como a mais recente. Porém ao invés de descartar a outra versão, ele a salva no histórico de revisões (ANDERSON, LEHNARDT e SLATER, 2010, p. 20).

2.6.3 Limitações do CouchDB

Apesar se ser muita poderosa e aplicável a diversos problemas, a técnica de Map-Reduce não é resposta para todos os problemas. O índice gerado pela função *map* é unidimensional e a função *reduce* não pode gerar muito dados. Além disso essas funções não permitem se trabalhar com mais que um documento (HOLT, 2011). As funções map-reduce não podem fazer tudo que se pode fazer usando banco de dados relacionais. (REDMOND e WILSON, 2012, p. 217).

Diferente dos bancos de dados SQL onde é possível executar uma atualização de registro informando apenas as colunas a serem atualizadas, no CouchDB só possível executar atualizações utilizando o documento inteiro.

Apesar do CouchDB armazenar os resultados da função *map* para consulta rápida, isso não é feito quando um documento é salvo, apenas quando as views são acessadas. Como resultado, ao fazer a consulta na view do CouchDB, a mesma

poderá demorar algum tempo para ser executada, dependendo da quantidade de documentos inseridos, alterados ou excluídos.

Os bancos de dados relacionais utilizam tabelas para separar e armazenar os dados, porém no CouchDB não existe essa separação, portanto é necessário criar um atributo nos documentos para identificar a que tipo de entidade ele se refere.

Em bancos de dados relacionais é possível configurar um controle de acesso para cada tabela existente. Como o CouchDB não faz essa separação de dados em tabelas, significa que qualquer usuário pode ler qualquer documento armazenado.

3 Sistema SIADE

3.1 Visão Geral

O sistema é utilizado pela secretara de saúde do município de Pau dos Ferros e o objetivo o sistema é facilitar a coleta de dados pelos agentes de endemias e a facilitar a visualização pelos supervisores.

Ele se divide em dois aplicativos: um aplicativo móvel voltado para o sistema operacional Android e outro aplicativo voltado para ambiente de servidor web.

O aplicativo Web consiste basicamente de três módulos: cadastro de agentes, gerenciamento de imóveis e gerenciamento de ciclo (período de trabalho dos agentes de endemias). O módulo gerenciamento de imóveis é responsável pelo cadastro dos imóveis do município bem como das ruas e dos bairros onde eles de localizam. O módulo gerenciamento de ciclo possui a função de gerenciar o trabalho dos agentes de endemias. Através desse módulo é possível criar ciclos de trabalho, distribuir imóveis para cada agente e cadastrar as visitas realizadas.

No sistema há dois tipos de usuários, o agente de endemias e o supervisor. O agente é responsável por visitar os imóveis e inserir os dados no sistema através do aplicativo móvel. O supervisor tem a responsabilidade de designar os imóveis que o agente irá trabalhar, estipular no prazo para conclusão das visitas e acompanhar o trabalho dos agentes. O acesso do supervisor é feito através de um navegador web (Figura 11).

3.2 Tecnologias utilizadas

O sistema baseia-se em uma arquitetura cliente-servidor. O servidor é executado em um servidor web e o cliente é executado em um navegador web. Há também um cliente para dispositivos móveis desenvolvido para executar em sistema operacional Android.

O aplicativo móvel foi desenvolvido em Java e utiliza banco de dados SQLite[4] e executa em sistema operacional Android 4.0 ou superior. O servidor foi desenvolvido

[4] SQLite é uma biblioteca em linguagem C que implementa um banco de dados SQL embutido. Programas que usam a biblioteca SQLite podem ter acesso a banco de dados SQL sem executar um SGBD separado.

em Python utilizando-se o framework Django[5] e do sistema de gerenciamento de banco de dados MySQL[6] (Figura 11).

Para a comunicação entre o aplicativo e o servidor é utilizada troca de mensagens JSON através de uma API REST. Somente usuários cadastrados e autenticados podem usar essa API. Essa autenticação é implementada usando o protocolo OAuth[7]

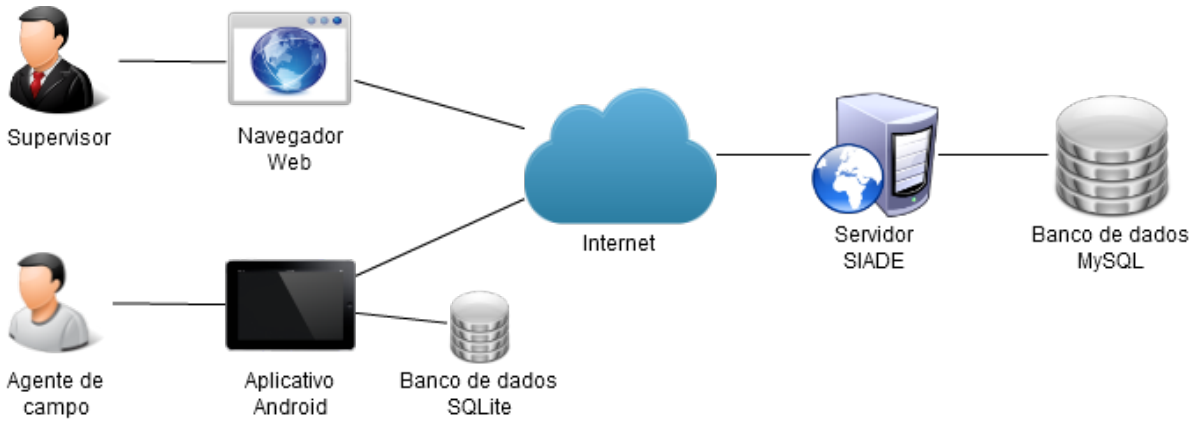


Figura 11 – Arquitetura do sistema SIADE

3.3 Esquema do banco de dados do SIADE

A Figura 12 mostra a estrutura do banco de dados do SIADE

[5] Framework para desenvolvimento rápido de aplicações web.
[6] MySQL é um sistema de gerenciamento de banco de dados relacional (SGBDR).
[7] OAuth é um protocolo de autorização de padrão aberto que permite que terceiros acessem os dados do usuário sem precisar saberem a senha dele.

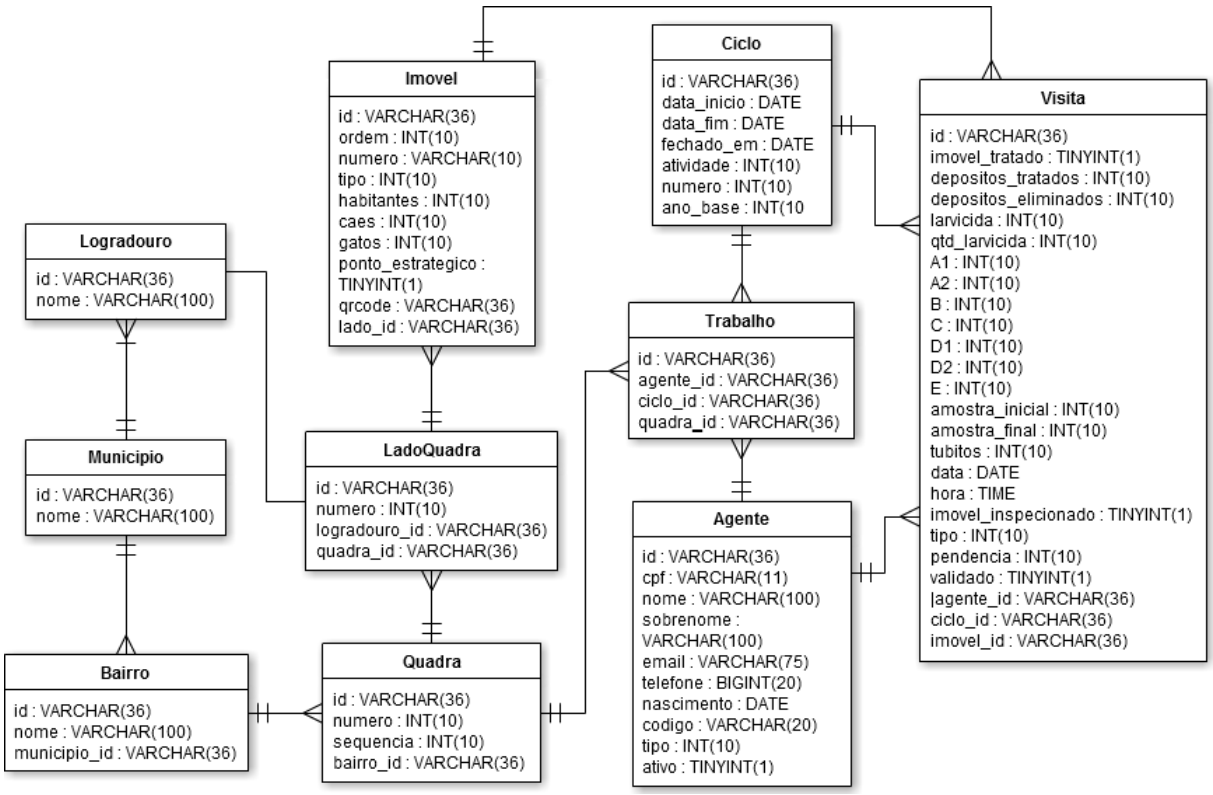


Figura 12 – Entidades e relacionamentos do banco de dados do SIADE

- **Agente**: Representa um agente de endemias seja ele um supervisor ou agente de campo.
- **Município**: Cadastro do município onde sistema está em funcionamento.
- **Bairro**: Bairro do município. Cada bairro possui diversas quadras.
- **Quadra**: Toda quadra possui um número, que deve ser sequencial, e vários lados.
- **LadoQuadra**: Representa o lado de uma quadra. Esta possui vários imóveis, um número sequencial e faz parte de um logradouro.
- **Logradouro**: Representa um logradouro do município (rua, avenida, etc).
- **Imóvel**: Representa um imóvel do dentro da quadra (residência, prédio comercial, terreno). O imóvel possui um número de ordem que serve para manter os imóveis na sequência em que eles estão na quadra. O imóvel pode ou não ser ponto estratégico que significa que este é um ponto de alto risco.
- **Ciclo**: Um ciclo é um período que um agente tem para visitar todas as casas que o supervisor lhe determinou. Ao iniciar um ciclo o supervisor

determina a data de início e de encerramento do ciclo e a atividade a ser realizada pelos agentes nas casas durante esse ciclo.

- **Trabalho**: Representa o trabalho designado aos agentes pelo supervisor. Está associado a um determinado ciclo, a um agente e a uma quadra.
- **Visita**: A visita realizada por um agente. Um agente pode desenvolver dois tipos de atividades de campo: pesquisa e tratamento**.** Há casos também em que é feita a pesquisa em conjunto com o tratamento. Quando o agente entrar na casa, ele coletará os dados do formulário de acordo com a atividade escolhida pelo supervisor na hora de iniciar o ciclo.

3.4 Sincronização

Para sincronizar os dados com o aplicativo móvel, o servidor implementa uma API REST específica para isso. O aplicativo envia todos os registros alterados desde a última sincronização para o servidor e recebe os dados atualizados. Para que o sistema saiba quais alterações foram feitas, ele guarda a data da última modificação (Figura 13).

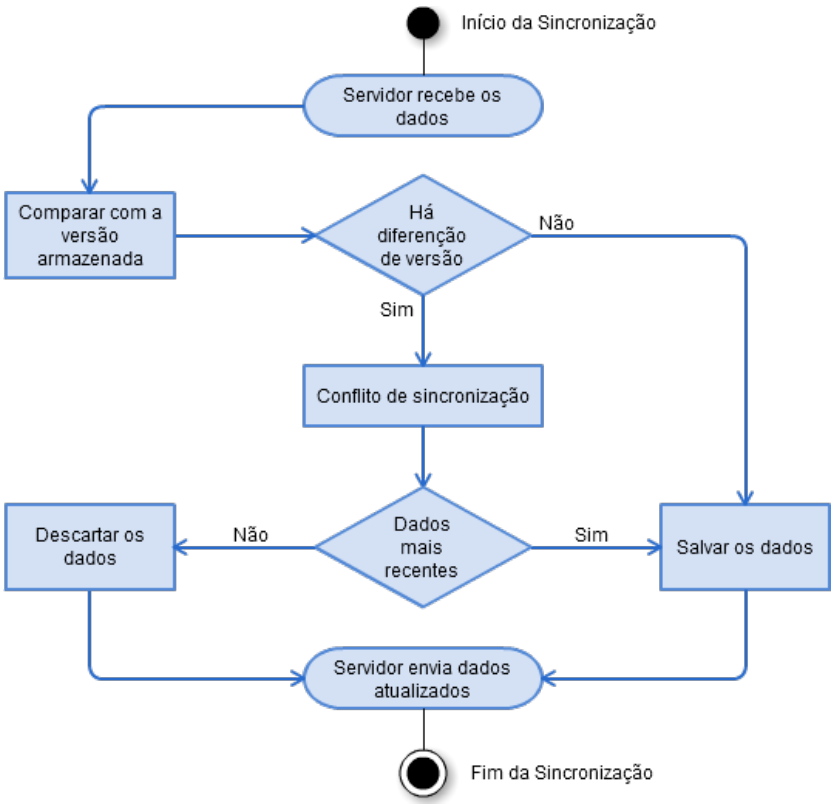


Figura 13 – Tratamento de conflitos de sincronização do servidor SIADE

A API de sincronização possui algumas limitações. Para cada entidade do sistema é necessário fazer uma requisição. Essas requisições implicam em uma maior transferência de dados, resultando em um tempo maior para completar a sincronização. Quando há um conflito de sincronização, o servidor descarta os dados mais antigos, ou seja, há perda de dados. Além disso, o registro das alterações é baseado em tempo. Caso haja uma diferença de hora entre o servidor e o dispositivo móvel, poderá haver perda de dados durante a sincronização. Portanto é primordial que o cliente e o servidor estejam com os relógios sincronizados para evitar possíveis inconsistências.

4 Adaptação do SIADE para CouchDB

4.1 Modelo de dados

O aplicativo organiza os dados seguindo um formado específico. Como os bancos de dados orientados a documentos não possuem esquema, eles podem se adaptar a qualquer esquema definido no aplicativo. Por isso, foram utilizados técnicas de modelagem NoSQL (KATSOV, 2012) para a criação de um modelo de dados baseado em documentos a partir das classes de domínio do sistema SIADE.

Como o banco de dados NoSQL não suporta junções (JOIN), foi necessário utilizar a técnica de desnormalização para que fosse possível realizar as consultas necessárias ao sistema, porém o processo não pode ser aplicado em todos os casos. Portanto, para esses casos foram feitas utilizados várias consultas para obter os dados necessários e depois feita a junção no aplicativo ao invés de ser feita no SGDB.

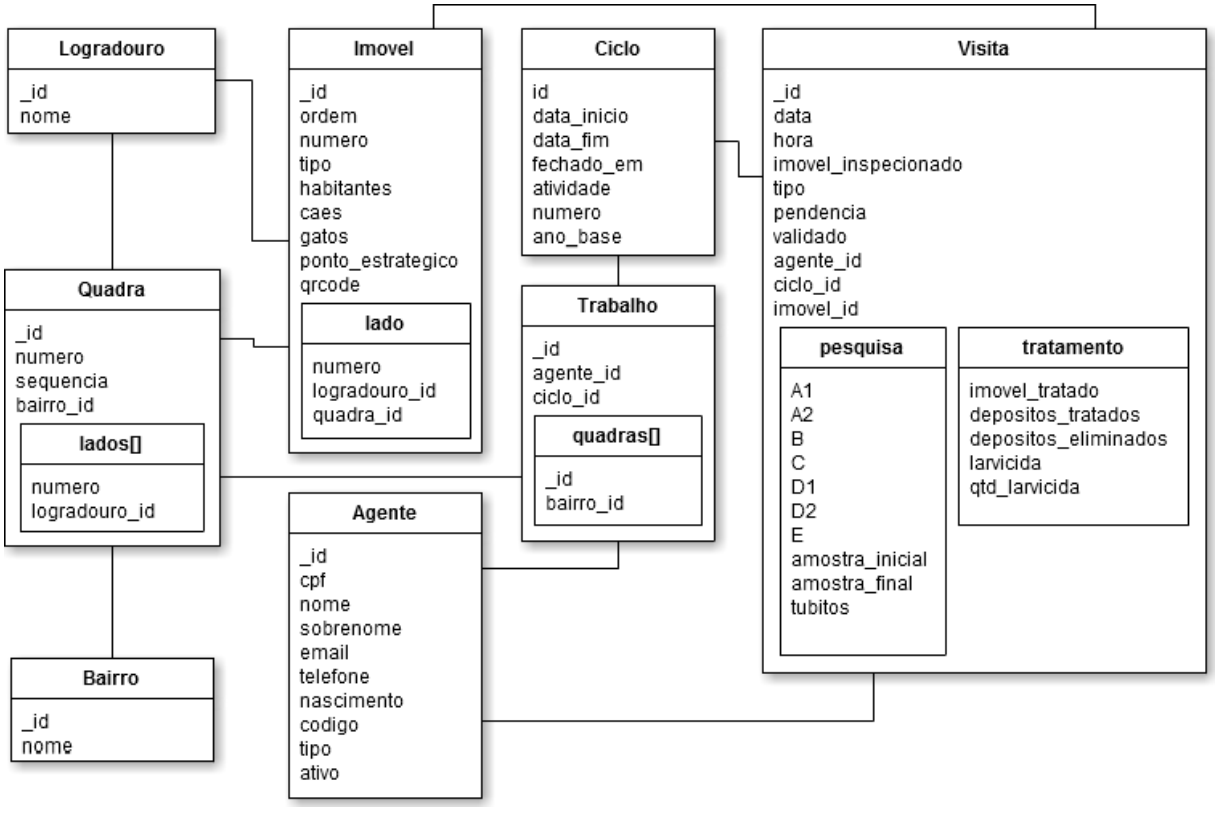


Figura 14 – Modelo de dados após a migração para o CouchDB

4.2 Migração dos dados

Para migração dos dados foi criado um módulo no sistema para ler os dados do banco MySQL convertê-los em JSON (com o auxílio do Django REST Framework) e depois grava-os no banco de dados CouchDB. Para cada classe de domínio do

sistema foi criado uma classe *Serializer* cujo objetivo de converter o objeto na representação JSON a ser armazenada. A Figura 15 mostra o funcionamento da migração de dados para um objeto da classe *Imovel*.

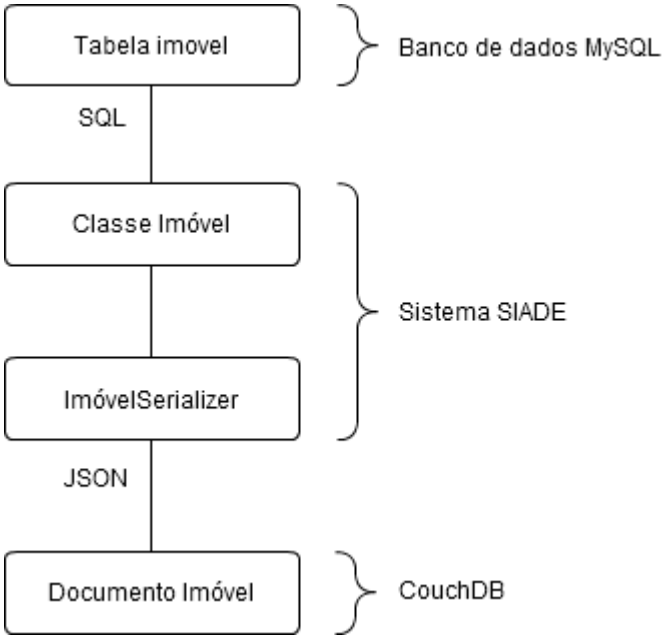


Figura 15 – Exemplo de processo de migração dos dados para JSON

4.3 Adaptação do servidor

Inicialmente foi identificado quais as consultas que o sistema realiza no banco de dados para que fosse possível criar *views* do CouchDB que fossem equivalentes as consultas SQL. Porém, na maioria dos casos, isso não foi possível, pois a forma de consulta no CouchDB é completamente diferente da utilizada nos bancos de dados relacionais. Portanto foi necessário adaptar o código do sistema para que fosse alcançado o mesmo resultado. Muitas consultas ao banco de dados tiveram que ser transformadas em duas ou mais consultas do CouchDB para que fosse possível extrair a mesma informação.

Foi utilizado a biblioteca *django_couchdb*, cujo objetivo é permitir ao framework Django acessar bancos de dados CouchDB.

Para consultar os dados do sistema foram criadas views map-reduce. Todas elas utilizam a mesma função reduce para fazer contagem e algumas delas utilizam o conceito de chave composta para permitir o agrupamento, ordenação e filtro dos dados.

```
function(keys, values, rereduce) {
  if (rereduce) {
    return sum(values);
  } else {
    return values.length;
  }
}
```

Figura 16 - Função reduce de contagem

Foram criadas duas **views bairros_por_nome** e **logradouros_por_nome** para listar, respectivamente, bairros e logradouros cadastrados usando o nome como chave no resultado. A Figura 17 mostra a função map da view **bairros_por_nome**.

```
function(doc) {
  if(doc.doc_type == 'Bairro') {
    emit(doc.nome);
  }
}
```

Figura 17 - Função map para bairros_por_nome

Para consultar dados dos imóveis foi criada uma view **imoveis_por_quadra** (Figura 18) para listar imóveis usando bairro, quadra, e lado como chave. Isso permite usar a função reduce agrupando os dados por bairro, por quadra ou por lado de quadra. Nesta view foi incluída, também, o número de ordem do imóvel para este seja usado para ordenação. A view **quadras_por_bairro** trabalha de forma similar, porém utilizando somente o identificador do bairro e da quadra.

```
function(doc) {
  if(doc.doc_type == 'Imovel') {
    emit([doc.lado.quadra.bairro._id,
          doc.lado.quadra._id,
          doc.lado.numero,
          doc.ordem]);
  }
}
```

Figura 18 - Função map para imoveis_por_quadra

Foram criadas ainda views para consultar ciclos, quadras a serem visitadas e as visitas feitas por cada agente em determinado ciclo (Figura 19). Essas views possuem código similar aos já apresentados.

```
function(doc) {
  if(doc.doc_type == 'Visita') {
    emit([doc.ciclo._id,
          doc.agente._id,
          doc.pendencia]);
  }
}
```

Figura 19 – Função map para visitas_por_agente

Para consultar determinadas informações no banco de dados, foi necessário consultar dados de duas views diferentes. Por exemplo, para saber quantos imóveis um agente precisa visitar é necessário consultar quais quadras o agente precisa visitar e quantos imóveis que existem em cada uma dessas quadras.

4.4 Adaptação do aplicativo móvel

Para adaptação do aplicativo móvel foi utilizado a biblioteca *Couchbase Lite* que implementa o mesmo funcionamento do CouchDB dentro do Android usando banco de dados SQLite.

O aplicativo móvel do SIADE implementa o padrão de projeto DAO-VO (ORACLE, 2015) para a acesso ao banco de dados. Ou seja, para cada entidade de domínio há duas classes no sistema: uma para representar a entidade (VO) e outra para gravar e recuperar os dados do banco (DAO). Portanto, para realizar a adaptação foi necessário modificar apenas as classes DAO para utilizar a API do CouchBase Lite invés do SQLite.

Diferentemente do CouchDB, no Couchbase Lite a views map-reduce são implementadas em Java através de classe que implementa a interface *com.couchbase.lite.Mapper* para a função map e *com.couchbase.lite.Reducer* para a função reduce. A Figura 20 mostra o método ***getQueryByBairro*** que realiza a consulta das quadras do bairro indicado pelo parâmetro ***bairroId***.

```
private Query getQueryByBairro(String bairroId) {
    View view = mDatabase.getView("quadras_por_bairro");
    final ObjectMapper om = new ObjectMapper();
    if (view.getMap() == null) {
        Mapper map = new Mapper() {
            @Override
            public void map(Map<String, Object> document, Emitter emitter) {
                if ("Quadra".equals(document.get(docTypeProperty))) {
                    Map<String, Object> bairro = (Map<String, Object>) document.get("bairro");
                    List<Object> key = Arrays.asList(
                        bairro.get("_id"), document.get("numero"), document.get("sequencia")
                    );
                    emitter.emit(key, null);
                }
            }
        };
        view.setMap(map, "6");
    }
    Query query = view.createQuery();
    query.setStartKey(bairroId);
    query.setEndKey(Arrays.asList(bairroId, new HashMap<String, Object>()));
    query.setPrefetch(true);
    return query;
}
```

Figura 20 – Código de consultar quadras de um bairro no aplicativo móvel

De forma similar foram criadas consultas para listar imóveis em uma quadra e visitas feitas em um imóvel.

Assim como no servidor, em alguns casos, foi necessário a utilização e duas ou mais consultas a banco de dados para extrair os mesmos dados que conseguia utilizando uma consulta SQL. Por exemplo, há uma tela no aplicativo móvel que mostra as quadras de um bairro exibindo também quantidade de imóveis existentes e de imóveis visitados nessa quadra. Para realizar essa consulta foi necessário realizar duas consultas no banco de dados: uma para ler as informações da quadra e outra para ler os totais de imóveis e de visitas. Para exibir a lista de imóveis de uma quadra mostrando a situação do imóvel também foi necessário realizar duas consultas. Uma para ler a lista de imóveis da quadra e outra para ler a situação da última visita de cada imóvel. Para isso foi necessário criar uma view e que exibisse o status da visita de cada imóvel daquela quadra. Devido ao sistema permitir realizar mais de uma visita por imóvel, foi necessário criar uma função reduce especial, para que ele consulte o status apenas da última visita. Abaixo a função *map-reduce* da *view* **status_imovel**.

```
Mapper map = new Mapper() {
    @Override
    public void map(Map<String, Object> document, Emitter emitter) {
        if ("Visita".equals(document.get(docTypeProperty))) {
            Map<String, Object> imovel = (Map<String, Object>) document.get("imovel");
            if(imovel == null) return;
            Map<String, Object> lado = (Map<String, Object>) imovel.get("lado");
            if(lado == null) return;
            Map<String, Object> quadra = (Map<String, Object>) lado.get("quadra");
            List<Object> key = Arrays.asList(quadra.get("_id"), lado.get("numero"),
                    imovel.get("_id"), document.get("data"), document.get("hora"));
            emitter.emit(key, document.get("pendencia"));
        }
    }
};
Reducer reduce = new Reducer() {
    @Override
    public Object reduce(List<Object> keys, List<Object> values, boolean rereduce) {
        return values.get(values.size()-1);
    }
};
```

Figura 21 - Função map-reduce para status_imovel

4.4.1 Sincronização

O Couchbase Lite possui a funcionalidade de replicação de dados da mesma forma que o CouchDB. Portanto é relativamente simples realizar sincronização de dados entre os dois.

O aplicativo móvel deve manter apenas os dados relacionados ao ciclo de trabalho e ao agente que está utilizando. Para que isso seja possível, é necessário fazer replicação apenas de uma parte dos dados. Para realizar essas replicações parciais foram criadas funções de filtro no servidor.

O aplicativo móvel possui uma classe chamada **PerformSync** responsável por fazer todo o trabalho de sincronização de dados. Portanto, ela foi reimplementada para utilizar a sincronização do CouchBase Lite.

O Couchbase Lite implementa dois tipos de replicadores de dados: *pull* e *push*. Cada um dos tipos é responsável por replicar os dados em uma direção. O *pull* serve para receber os dados de um banco de dados remoto e o *push* serve para enviar os dados para um banco de dados remoto.

Para este caso foram utilizados dois replicadores um do tipo *push* para enviar todos os dados presentes no aplicativo para o servidor e outro do tipo *pull* para receber apenas os dados relacionados ao agente que está utilizando o aplicativo.

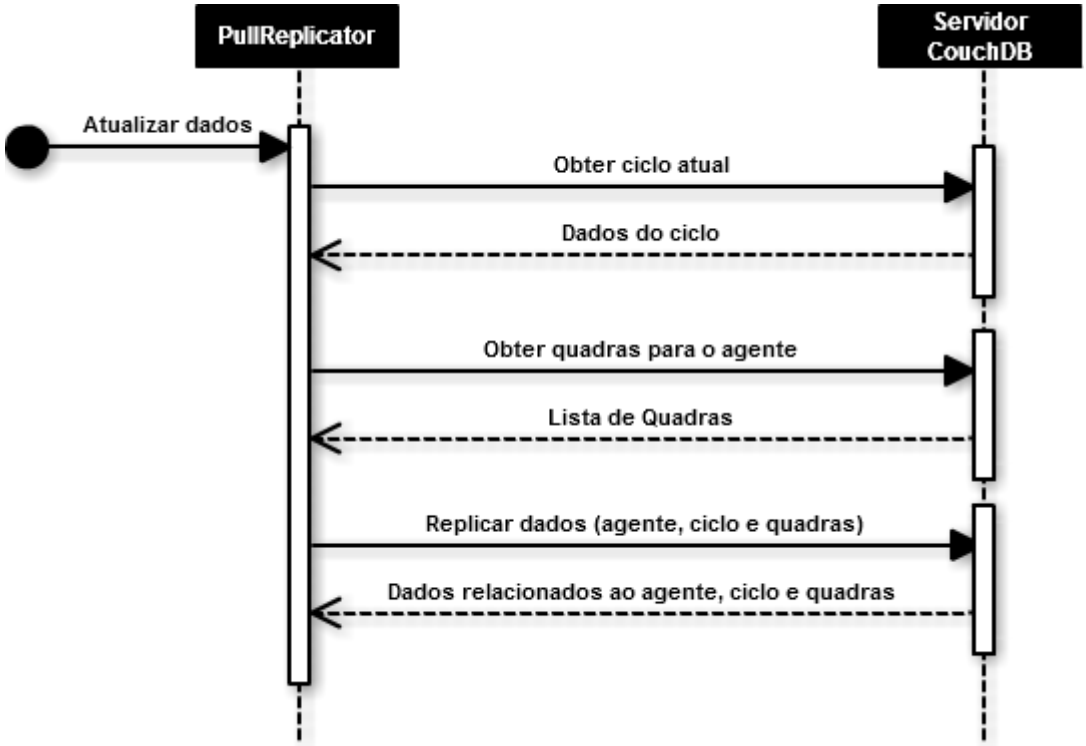


Figura 22 - Funcionamento do replicador pull no aplicativo móvel

O funcionamento do replicador pull é descrito na Figura 22. Para que seja possível a replicação dos dados relativos ao agente é preciso saber qual o ciclo atual e em quais quadras o agente vai trabalhar. Portanto, primeiro o aplicativo consulta esses dados do servidor, para depois utilizá-los como na função filtro, que por usa vez é executada na replicação dos dados. O replicador push simplesmente envia todos os dados modificados no aplicativo móvel sem realizar nenhum tipo de filtro.

5 Testes

5.1 Ambiente de teste

Os testes foram realizados em uma máquina física com a seguinte configuração

- Sistema Operacional: Windows 10 Pro
- Processador: Intel Core i3-2100 3.10GHz
- Quantidade de memória RAM: 4GB
- Versão do MySQL: 5.6.17
- Versão do CouchDB: 1.6.1

5.2 Conjunto de dados

Para realização dos testes foram criados dados fictícios através de um programa gerador de dados aleatórios.

O programa está dividido em 3 módulos: gerador de agentes, gerador de bairro e gerador de ciclos. Arquivos escritos em linguagem Python que ao serem executados geram dados fictícios de agentes, de um bairro, e de vários ciclos em um ano, respectivamente. Após os dados serem gerados são salvos em um banco de dados MySQL.

Iniciou pensou-se em usar a quantidade de imóveis de uma cidade grande, porém essa informação não foi encontrada. Portanto, foi utilizado um quantitativo suficientemente grande para se observar as diferenças de desempenho entre as duas versões do sistema. O quantitativo de dados é mostrado na Tabela 1.

Imóveis	93.900
Quadras	3.130
Logradouros	3.698
Bairros	22
Agentes	100
Ciclos	4
Visitas	394.385

Tabela 1 - Quantativo de dados gerados para testes

Depois de salvos no banco MySQL os dados são transferidos para um banco de dados CouchDB através do módulo de migração de dados desenvolvido e mencionado anteriormente na seção 4.2.

5.3 Desempenho

5.3.1 Aplicação web

Na aplicação web há uma página que mostra todos os bairros do município mostrando a quantidade de quadras e a quantidade de imóveis que existem naquele bairro. Ao clicar em um bairro, o sistema mostra uma página listando todas as quadras daquele bairro, a quantidade de lados em e de imóveis em cada quadra. Ao clicar em uma quadra é exibida uma página com todos imóveis daquela quadra, agrupados pelo lado. Além dessas páginas há também uma em é possível ver a quantidade e imóveis que precisam ser visitados e de visitas feitas por cada agente em um determinado ciclo.

Com base nisso, foram criadas operações de consulta em ambas versões do sistema para avaliar o tempo necessário para o processamento de cada uma delas. A partir desses tempos, foi criado um gráfico comparativo.

A consultas escolhidas para os testes são consultas realizadas pelo sistema SIADE em seu uso normal. Foram escolhidas operações de agregação (total) por serem mais demoradas. As consultas escolhidas foram:

Total de quadras e imóveis em um bairro: Consultar todos bairros mostrando o total de quadras e de imóveis em cada um deles. Abaixo um exemplo do resultado a ser retornado por ambas as consultas.

Bairro	Quadras	Imóveis
Bairro 1	10	300
Bairro 2	12	350
Bairro 3	8	120

Total de imóveis e lados em uma quadra: Consultar todas de quadras mostrando a contagem de lados e de imóveis para cada uma delas. Abaixo um exemplo do resultado a ser retornado por ambas as consultas.

Quadra	Lados	Imóveis
Quadra 1	4	30
Quadra 2	6	29
Quadra 3	3	15

Total de visitas dos agentes por ciclo: Consultar todos agentes e todos os ciclos e para cada par agente, ciclo mostrar o total de visitas realizadas. Abaixo um exemplo do resultado a ser retornado por ambas as consultas.

Ciclo	Agente	Total de Visitas
Ciclo 1	Agente 1	300
Ciclo 2	Agente 2	250
Ciclo 1	Agente 1	320
Ciclo 2	Agente 2	250

Total de imóveis dos agentes por ciclos: Consultar todos agentes e todos os ciclos e para cada par agente, ciclo mostrar o total de imóveis de deve ser visitado. Abaixo um exemplo do resultado a ser retornado por ambas as consultas.

Ciclo	Agente	Imóveis
Ciclo 1	Agente 1	300
Ciclo 2	Agente 2	250
Ciclo 1	Agente 1	320
Ciclo 2	Agente 2	250

Para todas as consultas acima, exceto a de total de visitas dos agentes por ciclos, são necessárias duas consultas ao banco de dados, tanto na versão MySQL quanto na versão CouchDB. A Figura 23 mostra o gráfico de tempo de execução das operações em ambas versões do sistema.

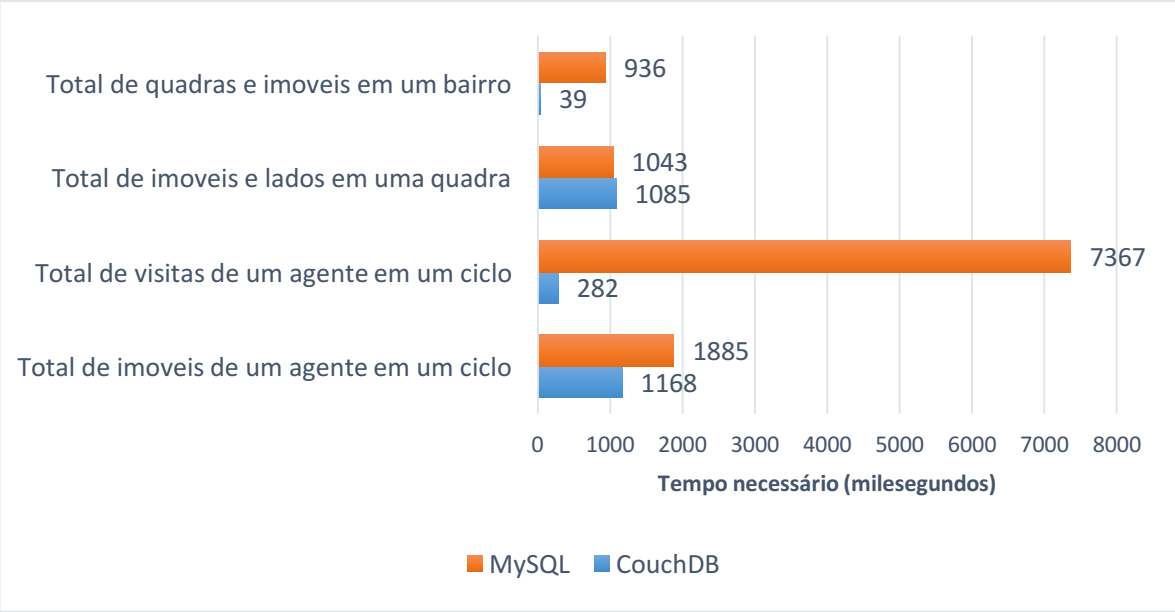


Figura 23 - Gráfico comparativo entre o desempenho do CouchDB e MySQL

Como pode ser visto no gráfico, o CouchDB foi mais rápido em três dos quatros testes. No outro ele mantém praticamente o mesmo tempo do MySQL. Esses tempos similares, deve-se ao fato de ambas implementações necessitam realizar duas consultas ao banco de dados. As operações onde há maior diferença de tempo são a de **total de quadras e imóveis em um bairro** e **total de visitas de um agente em um ciclo**.

Na primeira, essa diferença de tempo deve-se ao fato de que na versão SQL é necessário realizar uma junção de tabelas (*JOIN*) para que seja possível saber quais imóveis pertencem a cada bairro, enquanto o CouchDB já possui essa informação no documento.

Na operação de contar visitas por agente e ciclo foi onde houve a maior diferença. O MySQL levou mais de 7 segundos para executar a operação e o CouchDB levou apenas 282 milissegundos. Essa operação só envolve a consulta de uma view. Como o resultado da operação map é guardado em uma árvore balanceada, consequentemente é muito mais rápido para o CouchDB realizar essa operação.

5.3.2 Aplicativo móvel

Para o aplicativo móvel foram usados dados de apenas um agente e ciclo e apenas os imóveis existentes nas quadras associadas ao agente (cerca de 940 imóveis), pois esse é o subconjunto de dados utilizados no aplicativo móvel.

Para realizar os testes de desempenho foram utilizadas as duas principais telas do aplicativo: **lista de quadras de um bairro** e **lista de imóveis de um lado de quadra**. Para ambas foi medido o tempo necessário para realizar a consulta dos dados e criado um gráfico comparativo (Figura 24).

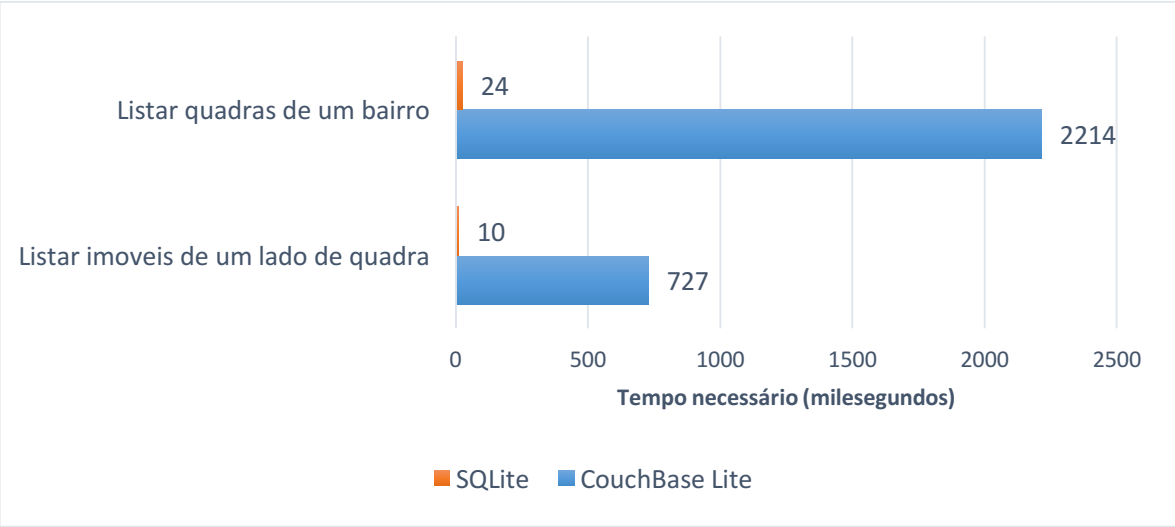


Figura 24 - Comparativos de tempo do aplicativo móvel

Como pode-se perceber o Couchbase Lite leva mais de 50 vezes o tempo que o SQLite leva para executar a mesma operação. Essa diferença de tempo deve-se ao fato de que o Couchbase Lite para Android ser implementado utilizando-se do SQLite para armazenamento de dados. O que significa quando for necessário ler ou gravar dados do banco será necessário executar uma operação SQL. Além disso, deve-se levar em conta também o tempo de conversão do objeto a partir do documento JSON.

6 Considerações finais

Diante das pesquisas e dos testes realizados, nota-se que o gerenciador de banco de dados CouchDB possui características que se adequam bem ao caso do sistema SIADE. Pois já possui o recurso de sincronização e resolução de e conflitos.

Porém um dos principais componentes do sistema, o aplicativo móvel, ficou com um desempenho ruim em relação a sua implementação original.

Migração de tecnologia de banco de dados relacional para orientado a documentos é um processo complicado e que requer uma reestruturação do modelo de dados para que seja possível extrair as mesmas informações que são possíveis através do uso de banco de dados relacional. Além disso, a forma como esses dados serão consultados tem influência direta de como será feita a reestruturação.

A forma de consultar dados no CouchDB é completamente diferente da utilizada pelos bancos de dados relacionais. Isso faz com que se tenha uma dificuldade adicional para planejar como seriam feitas as consultas e como seria a estrutura dos documentos, pois não há mesma flexibilidade existente no SQL.

Esse gerenciador de banco de dados possui alto desempenho na hora de gravar dados e de mantê-los consistentes pois utiliza um formato de arquivo que só permite acrescentar dados. Porém, isso faz com que ele não seja adequado em situações onde a constantes alterações nos dados ou onde o espaço de armazenamento seja reduzido.

Em relação a sincronização de dados, o CouchDB já fornece essa funcionalidade pronta para usar, exigindo do programador apenas a configuração. Apesar desse recurso funcionar perfeitamente, caso haja uma grande quantidade de dados a serem sincronizados, poderá levar mais tempo que com outras tecnologias existentes.

Depois desse estudo conclui-se que não vale a pena migrar o sistema para o CouchDB e que este é um gerenciador de banco de dados mais adequado para adotar em projetos mais novos e onde não seja necessário um relacionamento complexo entre os dados armazenados.

7 Trabalhos Futuros

O objetivo desse trabalho é estudar a viabilidade de se utilizar o banco de dados CouchDB em um sistema já em funcionamento. Por se tratar apenas de um experimento, não foi incluído nenhum recurso autenticação e este deve ser implementado antes o sistema possa ser realmente utilizado.

O desempenho do aplicativo móvel ficou muito abaixo da expectativa, portanto é necessário encontrar meios de melhorar seu desempenho, permitindo ao aplicativo ter um desempenho similar a versão usando banco de dados relacional.

Conflitos de sincronização são eventuais no sistema, mas poderia ser implementado uma funcionalidade que permitisse ao supervisor verificar e resolver esses conflitos.

8 Referências bibliográficas

ANDERSON, J. C.; LEHNARDT, J.; SLATER, N. **CouchDB:** The Definitive Guide. 1ª. ed. [S.l.]: O'Reilly Media, 2010.

APACHE SOFTWARE FOUNDATION. CouchDB Technical Overview. **Apache CouchDB 1.6 Documentation**, 2015. Disponivel em: <http://docs.couchdb.org/en/1.6.1/intro/overview.html>. Acesso em: 11 julho 2015.

BROWN, M. C. **Getting Started with CouchDB**. 1ª. ed. [S.l.]: O'Reilly Media, 2012.

CORRÊA, P. R. L.; FRANÇAB, E.; BOGUTCHIC, T. F. Infestação pelo Aedes aegypti e ocorrência da dengue em Belo Horizonte, Minas Gerais. **Revista Saúde Pública**, Belo Horizonte, 2005. Disponivel em: <http://www.scielo.br/pdf/rsp/v39n1/05.pdf>. Acesso em: 08 jul. 2015.

DEAN, J.; GHEMAWAT, S. MapReduce: Simplified Data Processing on Large Clusters. **Sixth Symposium on Operating System Design and Implementation**, San Francisco, Dezembro 2004. Disponivel em: <http://static.googleusercontent.com/media/research.google.com/pt-BR//archive/mapreduce-osdi04.pdf>. Acesso em: 25 julho 2015.

HOLT, B. **Writing and Querying MapReduce Views in CouchDB**. [S.l.]: [s.n.], 2011.

KATSOV, I. NoSQL data modeling techniques. **Highly Scalable Blog**, 2012. Disponivel em: <https://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/>. Acesso em: 2 agosto 2015.

LIMA, C. D.; MELLO, R. S. **Um Estudo sobre Modelagem Lógica para Bancos de Dados NoSQL**. Departamento de Informática e Estatística, Universidade Federal de Santa Catarina. Florianópolis, SC. 2015.

REDMOND, E.; WILSON, J. R. **Seven Databases in Seven Weeks**. [S.l.]: Pragmatic Programmers, 2012.

SADALAGE, P. J.; FOWLER, M. **NoSQL Essencial**. São Paulo: Novatec, 2013.

SECRETARIA DE VIGILÂNCIA EM SAÚDE. Monitoramento dos casos de dengue e febre de chikungunya até a Semana Epidemiológica 23, 2015. **Boletim Epidemiológico**, 2015.

SILVA, L. A.; BRETERNITZ, V. J. Big Data: Um novo conceito gerando opotunidades e desafios. **Revista Eletrônica de Tecnologia e Cultura**, n. 13ª, outubro 2013. Disponivel em: <http://revista-fatecjd.com.br/retc/index.php/RETC/article/view/74/pdf>.

WANZELLER, D. A. P. **Investigando o Uso de Banco de Dados Orientados a Documentos para Gerenciar Informações da Administração Pública**. Universidade de Brasília. Brasilia. 2013.

WASCHKE, M. Introduction to Big Data. **CA Technology Exchange**, v. 3, outubro 2012. Disponivel em: <http://www.ca.com/es/~/media/Files/About%20Us/CATX/catx-big-data-oct2012.pdf>.

WIKIPEDIA. Erlang (linguagem de programação). **Wikipédia, a enciclopédia livre**, 2015. Disponivel em: <http://pt.wikipedia.org/wiki/Erlang_(linguagem_de_programa%C3%A7%C3%A3o)>. Acesso em: 11 julho 2015.

WIKIPEDIA. MapReduce. **Wikipédia, a enciclopédia livre**, 2015. Disponivel em: <http://pt.wikipedia.org/wiki/MapReduce>. Acesso em: 11 julho 2015.

XÉXEO, G. Big Data - Computação para uma sociedade conectada e digitalizada. **Revista Ciência Hoje**, agosto 2013. Disponivel em: <http://cienciahoje.uol.com.br/revista-ch/2013/306/pdf_aberto/bigdata306.pdf>. Acesso em: 4 setembro 2015.