UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

SQL X NOSQL: ANÁLISE DE DESEMPENHO DO USO DO MONGODB EM RELAÇÃO AO USO DO POSTGRESQL

Marcos André Pereira Martins Filho

Trabalho de Graduação

Recife

FEVEREIRO DE 2015

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

Marcos André Pereira Martins Filho

SQL X NOSQL: ANÁLISE DE DESEMPENHO DO USO DO MONGODB EM RELAÇÃO AO USO DO POSTGRESQL

Trabalho apresentado ao Programa de GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO do CENTRO DE INFORMÁTICA da UNIVERSIDADE FEDERAL DE PERNAMBUCO como requisito parcial para obtenção do grau de CIÊNCIA DA COMPUTAÇÃO.

Orientador(a): *Profº Drº Fernando da Fonseca de Souza*

Recife

FEVEREIRO DE 2015

AGRADECIMENTOS

Agradeço, primeiramente, a Deus, por ter me dado força, saúde e disposição para conseguir completar mais essa etapa na minha vida. Sem ele, com toda a certeza, eu já teria desistido. A ele todo o louvor e glória para sempre.

Aos meus pais, Marcos André Pereira Martins e Maria de Fátima Morais de Oliveira Martins, que batalharam a vida toda para que eu pudesse ter uma educação de qualidade, a quem eu devo tudo o que sou e conquistei. Vocês são meus heróis. Esta vitória também é de vocês.

Aos meus avós, Maria Anita Morais de Oliveira e Levy Morais de Mendonça, que sempre cuidaram de mim e foram meus segundos pais.

A minha namorada, Alessandra Penaforte Correia da Silva, que esteve ao meu lado, me apoiando, em todos os momentos, sempre me estimulando a seguir em frente e quem, se Deus quiser, estará comigo vencendo muitas outras etapas da minha vida. Te amo, amor.

Aos amigos que fiz na universidade, Tomaz, Filipe, Jonatas, Paulo, João Paulo, por serem o melhor grupo de todos. Só foi possível aguentar todo esse tempo porque vocês estavam lá. Podem ter certeza, que levarei para sempre a amizade de vocês.

Aos meus amigos de infância e adolescência, a quem, carinhosamente, chamo de tripé, pois se uma perna cair, todas caem. Especialmente nas pessoas de Vinícius Mateus, Júlio César, Levy Morais Neto e João Leonaldo, a quem considero verdadeiros irmãos e sempre estiveram comigo, me apoiando.

Aos professores do Centro de Informática, por contribuírem para a minha educação e desenvolvimento profissional, especialmente na pessoa do Professor Fernando Fonseca, que além de ser meu orientador, foi nas suas aulas que eu encontrei o rumo que eu tomarei na minha carreira.

E finalmente, a minha querida tia Chinha, que foi minha terceira mãe e que, infelizmente, não pode estar comigo neste momento. Mas sei que ela está em algum lugar lá em cima me olhando com orgulho.

RESUMO

Com a grande quantidade de dados que circula pela rede, começam a surgir limitações no uso dos Sistemas de Gerenciamento de Banco de Dados clássicos. Portanto, é necessário procurar outras formas mais eficientes de armazenamento. Dessa forma, sistemas de bancos de dados não relacionais começam a surgir como uma alternativa. Este trabalho visa fazer uma análise comparativa de desempenho entre a abordagem relacional e a não relacional.

Palavras-chave: Relacional, Não relacional, Banco de dados orientado a documentos

ABSTRACT

With the large amount of data flowing through the network,limitations on the use of the classic Database Management Systems has emerged. Therefore, it is necessary to search for more efficient ways of storage. Thus, non-relational databases are emerging as an alternative. This work aims at making a comparative analysis of performance between the relational and not relationalapproaches.

Key Words: Relational, Non-relational, Performance, Document-oriented database

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BSON	*Binary JSON*
DBRef	*Database References*
E/R	Entidade-Relacionamento
FTP	*File Transfer Protocol*
HTTP	*Hyper Text Transfer Protocol*
HTTPs	*Hyper Text Transfer Protocol Secure*
JDBC	*Java Database Connectivity*
JSON	*JavaScript Object Notation*
NoSQL	*Not SQL*
SGBD	Sistema de Gerenciamento de Banco de Dados
SMTP	*Simple Mail Transfer Protocol*
SOAP	*Simple Object Access Protocol*
SQL	*Structured Query Language*
TCP	*Transmission Control Protocol*
XML	*Extensible Markup Language*

SUMÁRIO

1. INTRODUÇÃO

Com o advento da web 2.0 e a enorme quantidade de dados circulando na rede, algumas empresas, que encontraram limitações na forma relacional do gerenciamento de dados, se viram obrigadas a procurar por alternativas não-relacionais.

Neste contexto surge o conceito do NoSQL (*NotOnly SQL*, que quer dizer “Não apenas SQL”) [1], uma abordagem não-estruturada de gerenciamento de dados que preza desempenho, escalabilidade e disponibilidade, fazendo com que premissas como, por exemplo, consistência, presente nos tradicionais bancos de dados relacionais fossem revistas.

1.1. Motivação

A motivação surgiu com a iniciativa de analisar o desempenho das duas abordagens de SGBD, visto as vantagens que a abordagem NOSQL oferece, como desempenho e escalabilidade.

O SGBD relacional escolhido para esse trabalho foi o PostgreSQL [2], o qual vem sendo utilizado nas atividades que o autor realiza no estágio. O MongoDB [3] foi o SGBD não-relacional escolhido, por ser um dos mais conhecidos, *open-source* e com uma abordagem orientada a documentos, que é algo bastante diferente da abordagem relacional convencional.

1.2. Objetivos

O objetivo deste trabalho é realizar uma análise comparativa entre SGBD relacional e não-relacional, utilizando uma base com um volume de dados considerável.

Como mencionado anteriormente, o SGBD relacional utilizado será o PostgreSQL e o não-relacional será o MongoDB.

A base de dados utilizada será o SoccerWiki [4], a qual é constituída por jogadores de futebol, times e campeonatos.

1.3. Estrutura do Trabalho

Além deste capítulo, a estrutura deste trabalho está organizada como segue. O Capítulo 2 faz uma revisão bibliográfica, contendo conceitos das abordagens relacionais e não relacionais. O Capítulo 3 aponta as tecnologias utilizadas, o plano de experimentos e as dificuldades para se chegar nele e a forma como os dados foram tratados. O Capítulo 4 aborda os resultados obtidos a partir da execução do plano de experimentos. Por fim, o Capítulo 5 apresenta as conclusões obtidas com o trabalho, as limitações encontradas e sugestões de trabalhos futuros.

2. REVISÃO BIBLIOGRÁFICA

Neste capítulo será feita uma revisão sobre os conceitos de SGBD relacional e não relacional este último terá um foco um pouco maior, devido ao fato de o modelo relacional ser bastante conhecido, tanto na academia quanto no mercado.

2.1. SGBD Relacional

Um SGBD relacional modela os dados de forma que eles sejam estruturados e percebidos pelo usuário em formato de tabela, ou de maneira mais formal, relações.

Bancos de dados relacionais surgiram mais ou menos na metade da década 70, mas só passaram a ser utilizados pelas empresasanos mais tarde, substituindo arquivos simples, bancos de dados hierárquicos e em rede.

No ano de 1985, Edgar Frank Codd, criador do modelo de relacional, escreveu um trabalho no qual definia 13 regras um Sistema de Gerenciamento de Banco de Dados possa ser considerado relacional [5]. São elas:

- **Regra Fundamental** - Um SGBD deve gerir seus dados utilizando unicamente suas capacidades de relacionamento.
- **Regra da Informação** - Toda informação deve ser representada unicamente como dados em uma tabela.
- **Regra da garantia de acesso** - Todo dado, que seja um valor atômico, pode ser acessado logicamente utilizando o nome da tabela, o valor da chave primária da linha e o nome da coluna no qual ele está inserido.
- **Tratamento sistemático de valores nulos** - Valores nulos (não considerar o zero, *string* vazia e outros valores não nulos) representam dados que não existem e são independentes do tipo de dado.
- **Catálogo dinâmico *on-line* baseado no modelo relacional** - No nível lógico, a descrição do banco de dados é representada como dados em tabelas. Isso permite que usuários autorizados apliquem as mesmas formas de manipular dados aplicada aos dados comuns ao consultá-las.

- **Regra da sub linguagem abrangente** - Um sistema relacional pode suportar várias linguagens e formas de uso. Entretanto, deve possuir ao menos uma linguagem com sintaxe bem definida e expressa por cadeia de caracteres e com habilidade de apoiar a definição de dados, a definição de visões, a manipulação de dados, as restrições de integridade, a autorização e a fronteira de transações.
- **Regra da atualização de visões** - Toda visão que for teoricamente atualizável deve ser também atualizável pelo sistema.
- **Inserção, atualização e eliminação de alto nível** - Qualquer conjunto de dados que pode ser manipulado com um único comando para retornar informações, também deve ser manipulado com um único comando para operações de inserção, atualização e exclusão. Ou seja, as operações de manipulação de dados devem poder ser aplicadas a várias linhas de uma vez, ao invés de somente uma por vez.
- **Independência dos dados físicos** - Programas de aplicação ou atividades de terminal permanecem logicamente inalteradas independente das modificações na representação de armazenagem ou métodos de acesso internos.
- **Independência lógica de dados** - Programas de aplicação ou atividades terminal permanecem logicamente inalteradas independente das mudanças de informação que permitam teoricamente a não alteração das tabelas base.
- **Independência de integridade** - As relações de integridade específicas de um banco de dados relacional devem ser definidas em uma sub-linguagem de dados e armazenadas no catálogo.
- **Independência de distribuição** - A linguagem de manipulação de dados deve possibilitar que as aplicações permaneçam inalteradas estejam os dados centralizados ou distribuídos fisicamente.
- **Regra da não-subversão** - Se o sistema relacional possui uma linguagem de baixo nível (um registro por vez), não deve ser possível subverter ou ignorar as regras de integridade e restrições definidas no alto nível (muitos registros por vez).

2.1.1. Normalização

Normalização é o processo que foi proposto pela primeira vez em 1972 por Codd [6] e trata-se de uma técnica baseada na teoria dos conjuntos que visa tornar o modelo de dados mais estável.

Através da normalização, é possível modificar o conjunto de entidades e relacionamentos de modo que o mesmo fique mais coeso e seja capaz de suportar atualizações dos dados (como inserção, alteração e exclusão), as quais podem acarretar problemas como redundância e perda de informação [7]. Este processo é baseado no conceito de forma normal. Forma normal trata-se de uma regra a ser obedecida por uma tabela. As tabelas que respeitam essas regras são consideradas bem projetadas. Neste trabalho serão consideradas cinco formas normais [8].

Embora evitando as anomalias de atualização, os sistemas normalizados em geral apresentam um *overhead* para execução de consultas uma vez que pode forçar a execução de um grande número de junções, a operação relacional de maior custo de execução. Esta condição é uma das principais diferenças com relação aos sistemas NoSQL abordados na próxima seção.

2.2. NoSQL

A grande quantidade de dados gerada atualmente, juntamente com requisitos de aplicações relacionados à escalabilidade sob demanda e disponibilidade em alto grau, contribui para que surjam novos paradigmas de armazenamento. Assim, surge o NoSQL (*Not Only SQL*) [9], uma proposta com o intuito de atender aos requisitos de gerência de volume de dados de grande porte, semi-estruturados ou mesmo não-estruturados.

Os sistemas NoSQL não utilizam álgebra relacional, bem como não possuem uma linguagem nativa, como o SQL [9], tornando o desenvolvimento mais complexo. No geral, os sistemas NoSQL apresentam uma estrutura simplificada, não possuem uma estrutura de relacionamentos e têm suporte natural à replicação. Devido ao bom desempenho que apresenta lidando com um grande volume de dados, também são conhecidos como banco de dados escaláveis e direcionados para aplicações web.

Os sistemas NoSQL possuem características importantes que os distinguem dos SGBD relacionais. Dessa forma, eles se tornam adequados ao armazenamento de dados não-estruturados e semi-estruturados, de grande volume. Algumas dessas características são descritas a seguir [9].

- **Escalabilidade horizontal:** A escalabilidade horizontal consiste em aumentar a quantidade de máquinas que processam e armazenam os dados. A escalabilidade horizontal aparenta ser uma solução viável para o problema do aumento do volume de dados. Entretanto, ela necessita que várias *threads* e processos de uma dada tarefa sejam criadas e distribuídas. Para um banco de dados relacional, isso pode ser inviável, visto que vários processos conectando ao mesmo tempo um mesmo conjunto de dados pode causar alta concorrência, isso aumenta o tempo de acesso às tabelas envolvidas. A ausência de bloqueios é um dos fundamentos dos bancos de dados NoSQL, isso permite a escalabilidade horizontal, fazendo com que esses tipos de SGBD estejam aptos a solucionar problemas relacionados ao gerenciamento de grandes volumes de dados.

- **Ausência de esquema ou esquema flexível:** Esta característica é notória nos sistemas NoSQL. A falta de um esquema define a estrutura dos dados facilita a escalabilidade, bem como contribui para o aumento da disponibilidade. Entretanto, a integridade dos dados não é garantida, diferentemente do que acontece nos bancos de dados relacionais, por conta de sua estrutura inflexível.

- **Suporte nativo a replicação:** Mais uma maneira de aumentar a escalabilidade. Ao permitir a replicação nativamente, é possível diminuir o tempo que é gasto na recuperação de informação.

- **API simples para acesso aos dados:** O foco de uma solução NoSQL não está na forma como os dados são armazenados, mas na forma como eles são recuperados de forma eficiente. Para isso, as APIs precisam ser desenvolvidas afim de facilitar o acesso às informações, permitindo, assim,

que as aplicações possam fazer uso dos dados do banco com rapidez e eficiência.

- **Eventual consistência:** Esta característica é relacionada ao fato de que nem sempre é possível manter a consistência entre vários pontos de distribuição de dados. Ela é baseada no teorema CAP (*Consistency, Availability, Partition tolerance*), a qual diz que, só é possível garantir duas entre as três propriedades entre consistência, disponibilidade e tolerância à partição, em um dado momento.

Bancos de dados NoSQL surgem com grande variedade de formas e funcionalidades. De fato, a única característica comum em todas as abordagens é que elas não são relacionais. Fora isso, elas podem diferir, de forma bastante acentuada, entre si. Apesar dessas diferenças, a indústria de software tentou categorizar os bancos de dados NoSQL em pequenos conjuntos de áreas funcionais [10]. Os tópicos seguintes falam um pouco sobre algumas dessas abordagens.

2.2.1. Base de Dados Orientada a Chave/Valor

Esta abordagem implementa, ao menos no nível conceitual, o mais simples mecanismo de armazenamento NoSQL. Essencialmente, a abordagem chave/valor é uma grande tabela *hash*.

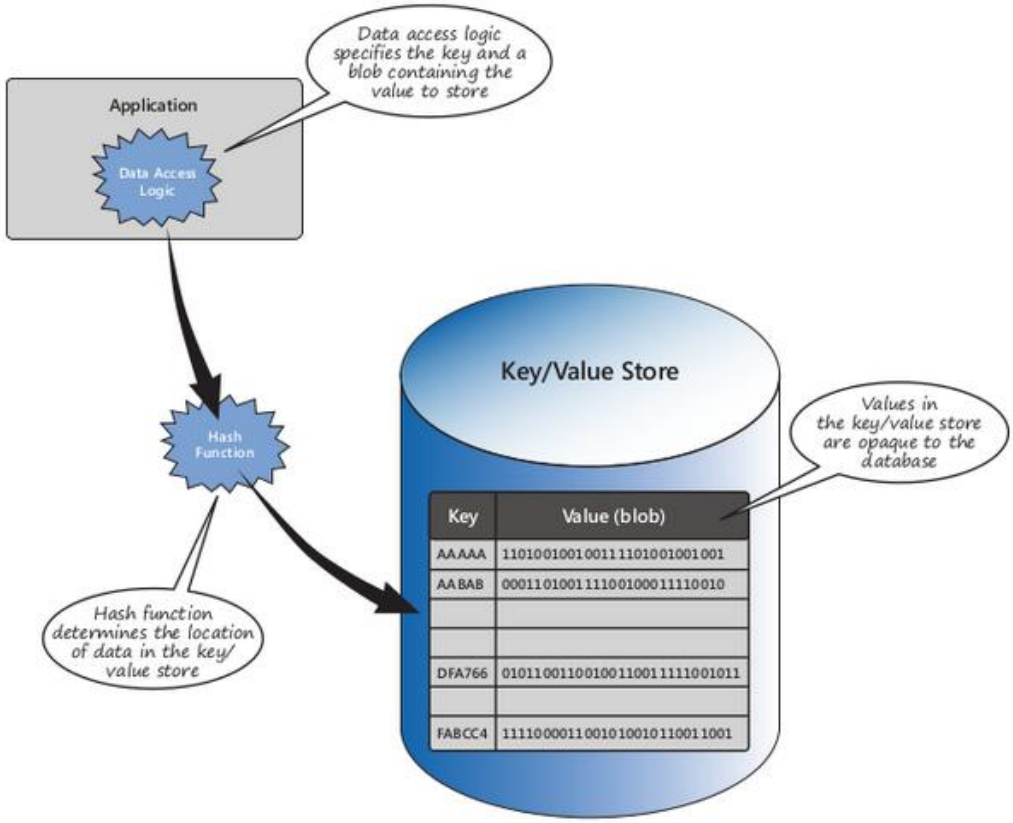
Uma tabela *hash* é a estrutura de dados mais simples que pode suportar um conjunto de pares chave/valor. Essa estrutura de dados é extremamente eficiente.

Cada valor é associado a uma chave de valor único e o SGBD utiliza essa chave para determinar onde armazenar os dados, usando uma função *hash* apropriada [11]. A função *hash* é selecionada para proporcionar uma distribuição uniforme de chaves *hash* através de armazenamento de dados. Os valores depositados no armazenamento chave/valor são opacos para o SGBD. Essencialmente, os valores são do tipo BLOB e a abordagem simplesmente retorna ou armazena um valor utilizando a chave fornecida. Por esta razão, a maioria das bases de dados chave/valor suporta apenas operações de consultas simples, inserções e deleções. Para atualizar um dado, é necessário substituir um valor já existente. É possível armazenar qualquer tipo de dados, entretanto, algumas

implementações da abordagem chave/valor impõem limites de tamanho máximo dos valores. Geralmente, na maioria das implementações, a leitura e escrita de um único valor é uma operação atômica.

A Figura 1mostra uma estrutura conceitual do armazenamento chave/valor.

Figura 1 - Estrutura de um banco de dados orientado a chave/valor



Fonte: MCMURTRY, 2014

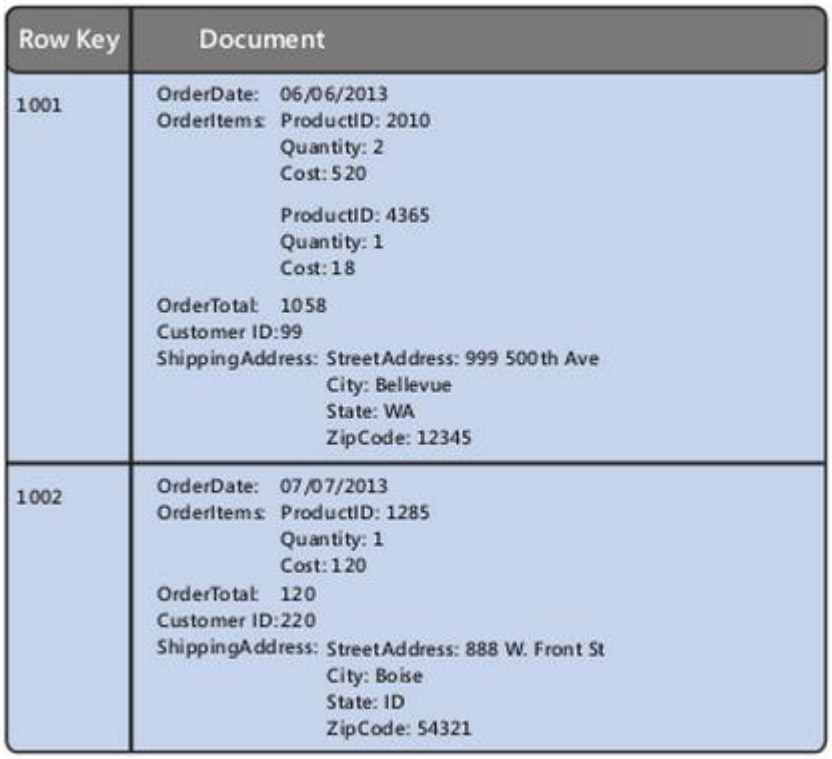
2.2.2. Base de Dados orientada a Documento

Um banco de dados orientado a documento é similar, em seu conceito, ao orientado à chave/valor, com exceção de que os dados são armazenados em documentos. Um documento é uma coleção de campos nomeados e valores associados, onde cada um desses campos pode ser um simples valor escalar ou elementos compostos, como uma lista de documentos filhos. Os dados podem ser codificados de várias formas, entre elas estão XML e JSON [10]. É importante frisar

que o termo documento, neste contexto, é o nome dado à coleção de itens de dados relatados, que constituem uma entidade, e não uma estrutura de texto [11].

Os campos no documento são expostos para o SGBD, permitindo buscar e filtrar dados utilizando valores nesses campos. A Figura 2 ilustra um exemplo de dois documentos referentes a informações de pedidos de vendas.

Figura 2 - Exemplo de documentos



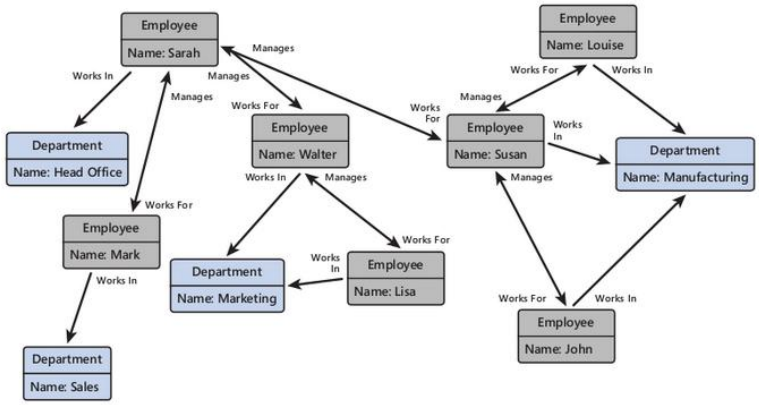
Fonte: MCMURTRY, 2014

O objetivo do banco de dados da Figura 2 é otimizar o acesso à busca de informações relativas aos pedidos de venda, visto que, estas são as operações mais comuns utilizadas pela aplicação que utiliza essa base de dados. Um problema que um modelo relacional clássico poderia ter com este exemplo é relacionado com endereço de entrega. Esses dados, em um modelo relacional tradicional, deveriam ser separados em outra tabela, de forma que eles possam ser modificados sem que seja necessário buscar todos os pedidos que façam referência a ele.

2.2.3. Base de Dados orientada a Grafo

Assim como outras categorias de bancos de dados NoSQL, uma base de dados orientada a grafo permite armazenar entidades, porém o seu foco principal é nos relacionamentos que essas entidades têm umas com as outras [10]. Um banco de dados orientado a grafos armazena dois tipos de informação: Nós, os quais podem ser pensados como instâncias de entidades, e arestas, as quais especificam a relação entre os nós. Nós e arestas podem ter tanto propriedades que fornecem informações sobre aquele nó, quanto de arestas. Além disso, arestas podem ter uma direção indicando a origem do relacionamento. A proposta de um banco orientado a grafos é permitir que uma aplicação execute, de forma eficiente, consultas que percorram a rede de grafos e arestas e analisem a relação entre as entidades. A Figura 3 mostra um exemplo de uma base de dados estruturada com um grafo. As entidades são os empregados e departamentos na organização e as arestas indicam a relação entre os empregados e o departamento ao qual pertencem. Além disso, as setas nas arestas indicam a direção dos relacionamentos.

Figura 3 - Exemplo de estrutura de uma base de dados orientada a grafos



Fonte: MCMURTRY, 2014

2.2.4. Base de Dados orientada à Família de Colunas

Um banco de dados orientado à família de colunas organiza os seus dados dentro de linhas e colunas e, em sua forma mais simples, pode parecer muito similar a um banco de dados relacional, pelo menos conceitualmente [10]. Porém, o real poder de um banco de dados orientado à família de colunas está em sua abordagem desnormalizada para estruturar dados esparsos. A Figura 4 mostra uma forma de estruturação de informação, utilizando uma base de dados orientada à família de colunas que agrupa a informação em duas famílias de colunas, mantendo o nome do cliente e as informações do endereço. Outros esquemas são possíveis, porém, deve-se implementar as famílias de colunas de modo a otimizar a consulta mais comum realizada pela aplicação.

Figura 4 - Exemplo de esquema de família de colunas

Row Key	Column Families	
CustomerID	CustomerInfo	AddressInfo
1	CustomerInfo:Title Mr CustomerInfo:FirstName Mark CustomerInfo:LastName Hanson	AddressInfo:StreetAddress 999 500th Ave AddressInfo:City Bellevue AddressInfo:State WA AddressInfo:ZipCode 12345
2	CustomerInfo:Title Ms CustomerInfo:FirstName Lisa CustomerInfo:LastName Andrews	AddressInfo:StreetAddress 888 W. Front St AddressInfo:City Boise AddressInfo:State ID AddressInfo:ZipCode 54321
3	CustomerInfo:Title Mr CustomerInfo:FirstName Walter CustomerInfo:LastName Harp	AddressInfo:StreetAddress 999 500th Ave AddressInfo:City Bellevue AddressInfo:State WA AddressInfo:ZipCode 12345

Fonte: MCMURTRY, 2014

2.3. Considerações Finais

Este capítulo teve como objetivo fazer uma revisão sobre os conceitos presentes em SGBD relacionais, destacando normalização. Também foram vistos os conceitos de sistemas não relacionais, bem como foram abordados os seus tipos, como bancos de dados orientados a documentos, grafo,chave/valor e família de colunas.

No próximo capítulo serão apresentados os sistemas utilizados nos experimentos, além da ferramenta de análise de desempenho. Além disso, serão introduzidos o conjunto de dados e o seu tratamento para a elaboração da modelagem e o plano de experimento do trabalho.

3. TECNOLOGIAS, DADOS E PLANO DE EXPERIMENTO

Este capítulo é dedicado a apresentar as tecnologias que serão utilizadas no trabalho. Inicialmente, serão introduzidos os SGBD PostgreSQL[2] e MongoDB [3]. Em seguida, será apresentada a ferramenta de análise de desempenho JMeter [12]. Também neste capítulo será apresentada a base de dados utilizada e como ela foi capturada e modelada para os dois SGBD. Por fim, é detalhado o plano de experimento do trabalho.

3.1. PostgreSQL

O PostgreSQL é um poderoso e *open source* SGBD, que tem mais de 15 anos de desenvolvimento ativo e uma arquitetura robusta, a qual adquiriu uma forte reputação em confiabilidade, integridade de dados e corretude [2]. Ele pode executar em todos os grandes sistemas operacionais, incluindo Linux [13], Sistemas UNIX [14] e Windows [15] e é compatível com a linguagem SQL ANSI, uma linguagem de consulta declarativa, que é a padrão para SGBD relacionais.

O PostgreSQL apresenta suporte a chave estrangeira, operações *join*, *views*, triggers, procedures, entre outros. Além disso, ele aceita a maioria dos tipos de dados da especificação SQL:2008, dentre eles estão *INTEGER*, *VARCHAR*, *BOOLEAN*, *NUMERIC*, *CHAR*, *DATE*, *INTERVAL* e*TIMESTAMP*.

A plataforma de desenvolvimento utilizada para manusear o PostegreSQL foi o pgAdmin [16], uma ferramenta *open source* bastante popular no mercado. Ela pode ser utilizada em sistemas como o Linux [13], Mac OSX [17], Windows [15], entre outros e gerencia as versões do PostgreSQL acima da 7.3.

3.2. MongoDB

O MongoDB é um sistema banco de dados orientado a documento livre de *schema* e escrito em C++[18]. Uma base de dados contém uma ou mais coleções de

documentos. Por ser livre de esquema, os documentos dentro de uma coleção podem ser heterogêneos [3].

Uma vez que o primeiro documento é inserido no banco, uma coleção é criada automaticamente e o documento é, então, adicionado a essa coleção, que é configurada com parâmetros *default* pelo próprio MongoDB.

Como já mencionado, o MongoDB é orientado a documentos. Um documento é a unidade de dados armazenável. Possui uma estrutura comparável a um documento XML [19] ou JSON [20]. De fato, o MongoDB baseia os documentos em um formato chamado BSON [21], o qual é muito similar ao JSON. A Figura 5 mostra um exemplo de documento do MongoDB.

Figura 5 - Exemplo de documento do MongoDB

```
{
  title: "MongoDB",
  last_editor: "172.5.123.91",
  last_modified: new Date("9/23/2010"),
  body: "MongoDB is a...",
  categories: ["Database", "NoSQL", "Document Database"] ,
  reviewed: false
}
```

Fonte: STRAUCH, 2012

Para adicionar um documento em uma coleção, é utilizada a função de inserção mostrada na Figura 6.

Figura 6 – Inserção no MongoDB

```
db.<collection>.insert( { title: "MongoDB", last_editor: ... } );
```

Fonte: STRAUCH, 2012

Após o documento ser inserido, ele pode ser recuperado por meio de consultas correspondentes, utilizando a operação *find()* e atualizado pela operação *save()*, como mostrado na Figura 7.

Figura 7 - Busca e atualização no MongoDB

```
db.<collection>.find( { categories: [ "NoSQL", "Document Databases" ] } );
db.<collection>.save( { ... } );
```

Fonte: STRAUCH, 2012

O MongoDB não dá suporte à chave estrangeira, então, as referências entre os documentos devem ser resolvidas por consultas adicionais emitidas pela aplicação cliente. As referências podem ser definidas manualmente atribuindo a algum campo o valor do campo _id do documento referenciado. Além disso, o MongoDB fornece uma forma mais formal de especificar as referências, chamada de *DBRef* (*Database Reference*). Na Figura 8 é mostrado um exemplo do uso de *DBRef*.

Figura 8 - Uso de DBref

```
{ $ref : <collectionname>, $id : <documentid>[, $db : <dbname>] }
```

Fonte: STRAUCH, 2012

Uma alternativa ao *DBRef* é aninhar documentos dentro de documentos. Esta abordagem é chamada de *Embedded Documents* e é muito mais eficiente [3]. Quando se utilizar referências, cada uma delas é uma consulta no banco de dados, diferente da abordagem de *embedded documents*, onde essas consultas são eliminadas. A Figura9 mostra uma tabela que dá uma orientação de quando utilizar a abordagem de referência e quando utilizar a abordagem de *embedded documents*.

Figura 9 - DBrefs x Embedded documents

Critérios para *Object References*	Critérios para *Object Embedddings*
• Objetos de domínio de primeira classe (tipicamente residentes em coleções separadas) • Referências do tipo muitos para muitos entre os objetos • Objetos deste tipo são frequentemente consultados em grandes números (retornar tudo/ os primeiros n objetos de um certo tipo) • O objeto é grande (vários megabytes)	• Objetos com a característica de "detalhe de item de linha" • Relacionamento de agregação entre o objeto e o objeto hospedeiro • O objeto não é referenciado por outro objeto • Desempenho para requisições e operações no objeto e seu hospedeiro é crucial

Fonte: Adaptada de STRAUCH, 2012

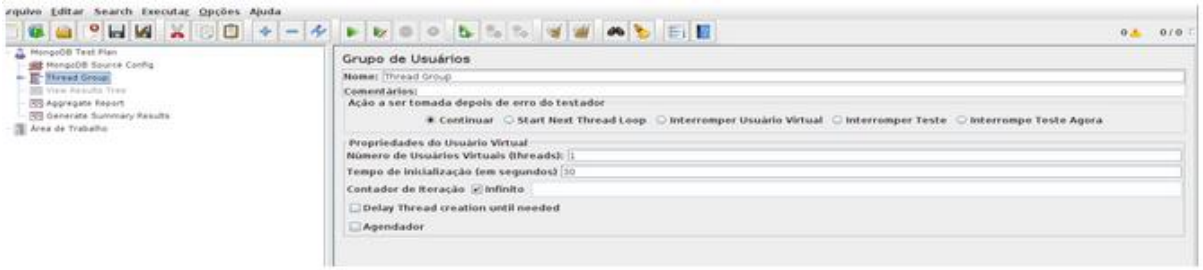
3.3. JMeter

O JMeter [12] é uma ferramenta *open source* para desktop, desenvolvida pela Apache, feita totalmente em Java [22] analisar o comportamento de testes funcionais e medir o desempenho. Para a realização desses testes, a ferramenta suporta vários tipos de requisiçõescomoSOAP [23], FTP [24], JDBC [25], MongoDB [3], entre outros. Além disso, existem controladores lógicos e controles condicionais que são usados para construir planos de teste.

O JMeter fornece um controle de *threads*, o qual é chamado de *Thread Group*. Nele é possível configurar a quantidade de *threads* que irão executar o teste, quantas vezes cada uma das *threads* será executada e, também, o intervalo de tempo entre cada execução. Além disso, existem diversos *listeners* que utilizam os resultados obtidos das requisições para poderem gerar gráficos e tabelas.

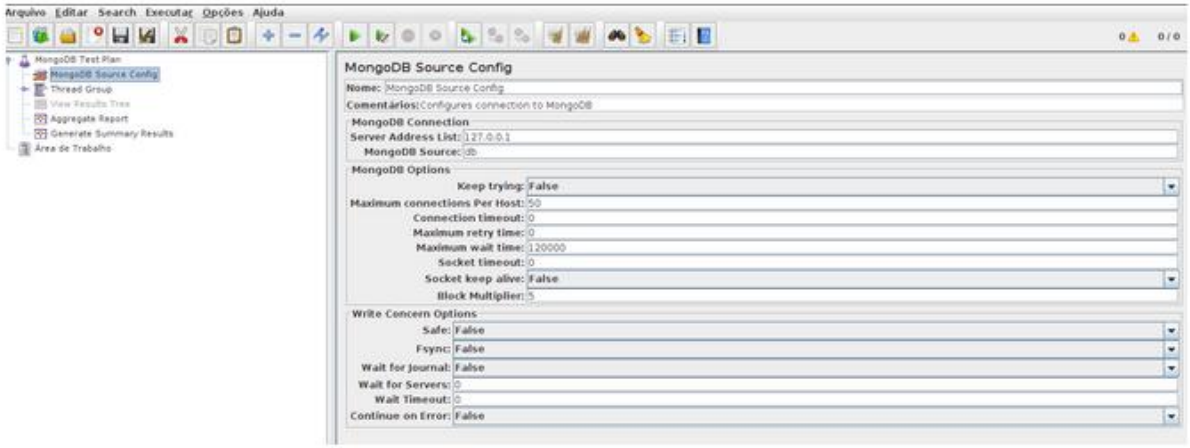
As figuras 10 e 11 mostram duas telas do JMeter. A primeira é a tela de configuração de um *Thread Group*, enquanto a segunda trata-se da janela de configuração de conexão com o MongoDB.

Figura 10 - Tela de configuração do Thread Group



Fonte: *print screen* do JMeter

Figura 11 - Tela de configuração de conexão com o MongoDB



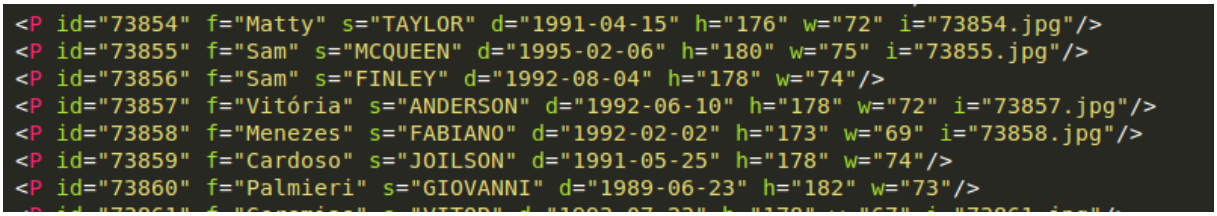
Fonte: *print screen* do JMeter

3.4. Conjunto de Dados

Os dados utilizados neste trabalho foram coletados do site SoccerWiki [4], um portal que contém informações sobre jogadores de futebol, clubes, estádios, treinadores, entre outras informações. O site disponibiliza quatro documentos em formato XML, contendo 59922 instâncias de jogador, 3327 estádios, 3237 clubes e 5857 treinadores.

A Figura 12 mostra um trecho do documento referente aos jogadores. É possível perceber que um jogador possui um atributo de identificação (atributo id), primeiro nome (atributo f), segundo nome (atributo s), data de nascimento (atributo d), altura (atributo h), peso (w) e um nome de um arquivo com formato de imagem (atributo i).

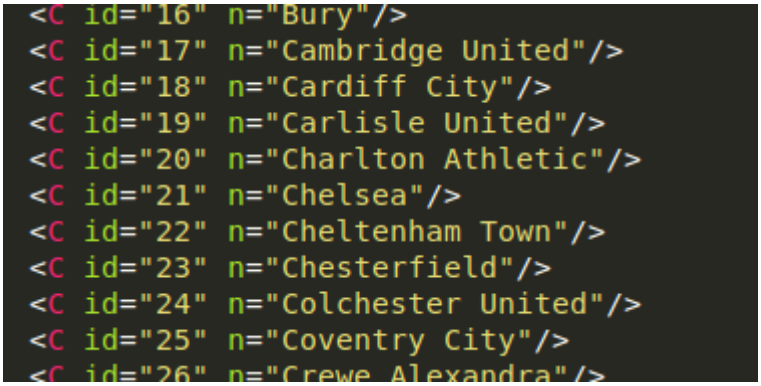
Figura 12 - Trecho do documento XML de jogadores



Fonte: http://pt-br.soccerwiki.org/wiki.php

Em seguida, é mostrado na Figura 13 um trecho do documento que contém as informações dos clubes. Um clube possui um código de identificação (id) e um nome (n).

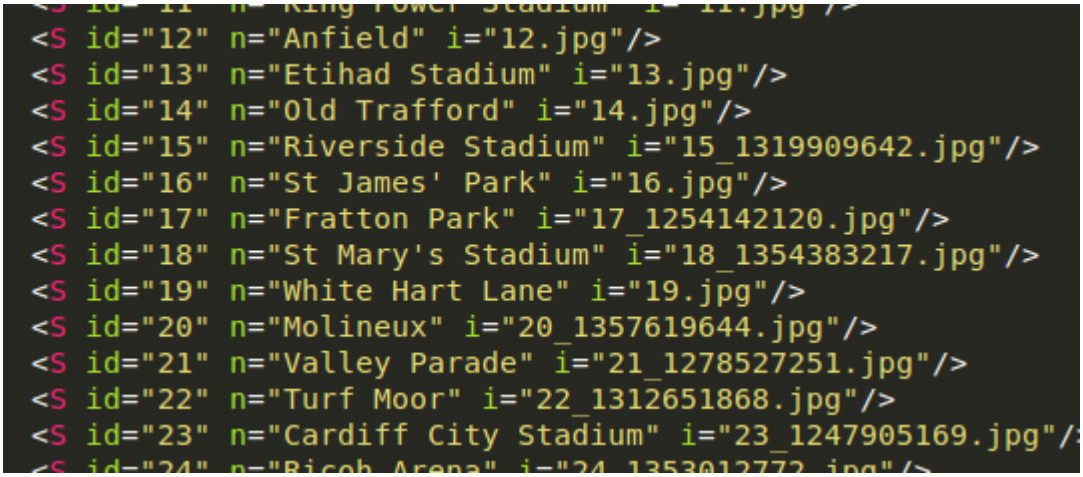
Figura 13 - Trecho do documento XML de clube



Fonte: http://pt-br.soccerwiki.org/wiki.php

Na Figura 14, é mostrada um trecho do documento com as informações dos estádios, os quais possuem uma identificação (id), nome (n) e um nome de uma imagem (i).

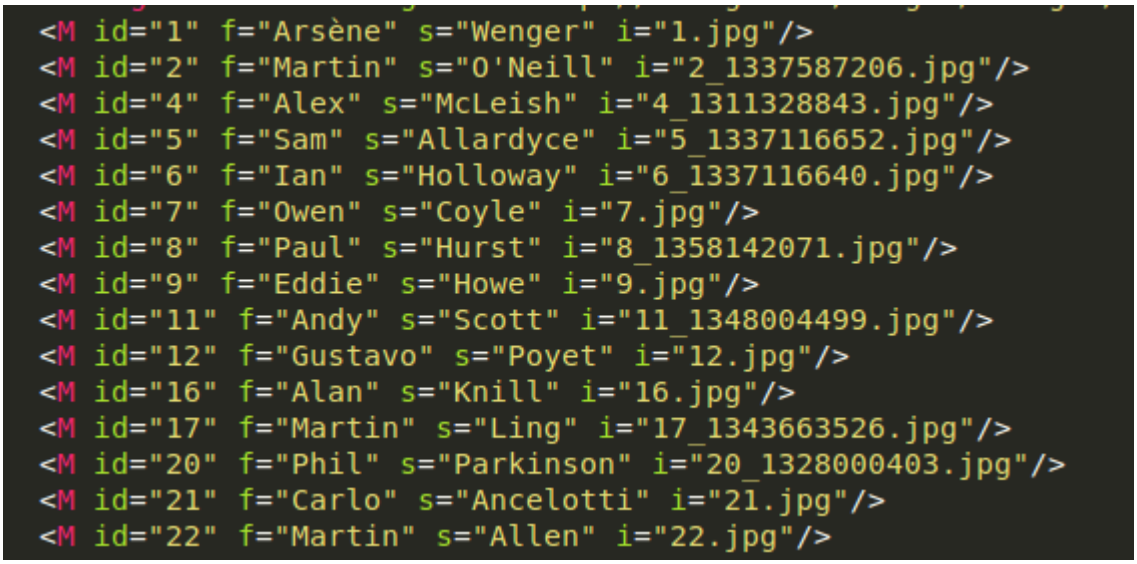
Figura 14 - Trecho do documento XML de estádio



Fonte: http://pt-br.soccerwiki.org/wiki.php

O documento referente aos dados dos treinadores é ilustrado na Figura 15. Um treinador possui um id, primeiro nome (f) e segundo nome (s).

Figura 15 - Trecho do documento XML de treinador



Fonte: http://pt-br.soccerwiki.org/wiki.php

A Tabela 1 ilustra a quantidade de dados que foi coletada do SoccerWiki:

Tabela 1 - Quantidade de dados coletada

Documento	Quantidade de Dados
Jogador	59922
Clube	3237
Estádio	3327
Treinador	5857

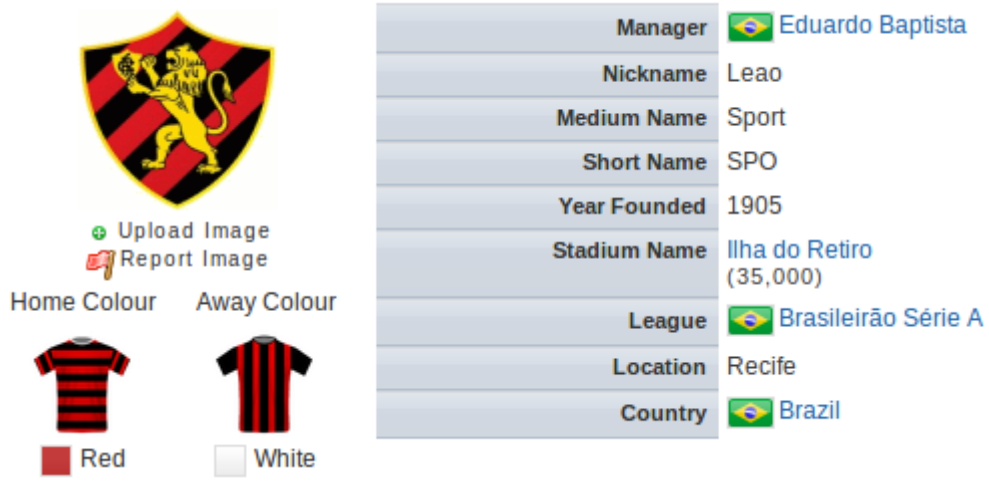
Fonte: O autor

Apesar dos dados terem sido disponíveis, ainda existem informações relevantes que não estão presentes nestes documentos. Informações como o país de cada jogador e de cada clube, as ligas às quais cada time pertence e o país de cada uma delas e a relação entre todos esses conjuntos de dados estão disponíveis no próprio site do SoccerWiki. As figuras 16, 17 e 18 mostram como estão dispostas as informações dos jogadores, clubes e ligas respectivamente. É possível perceber na Figura 16 que o jogador apresenta informações sobre a sua nacionalidade e o clube no qual joga, possuindo um link para suas respectivas páginas. Na Figura 17 o clube possui informações sobre o treinador, o campeonato do qual participa, o país ao qual pertence e o estádio que possui, os quais também possuem um link associado.

Figura 16 - Exemplo de visualização dos dados de jogador no Soccerwiki

Fonte: http://pt-br.soccerwiki.org/wiki.php

Figura 17 - Exemplo de visualização dos dados de clube no Soccerwiki



Fonte: http://pt-br.soccerwiki.org/wiki.php

Figura 18 - Exemplo de visualização dos dados de liga no Soccerwiki

Fonte: http://pt-br.soccerwiki.org/wiki.php

3.4.1. Modelo Relacional de Dados

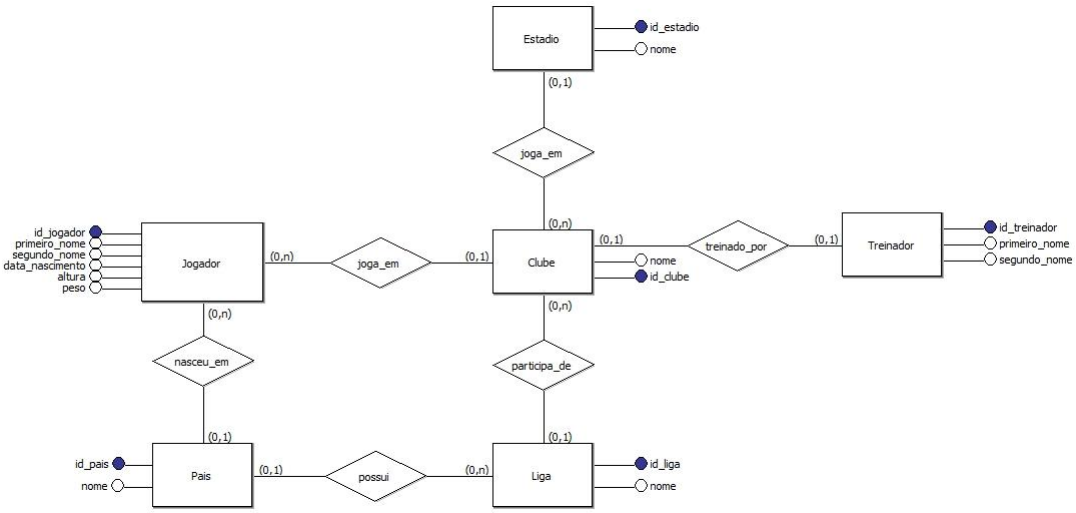
Acima foram apresentados os documentos que foram disponibilizados pelo site do SoccerWiki. É possível perceber que cada documento pode ser modelado como uma entidade. Além disso, a partir das informações que estão contidas no site e que não foram fornecidas para download, é possível extrair mais duas entidades: País e Liga. Portanto, o modelo conceitual possui seis entidades: Jogador, Clube, Estádio, Treinador, País e Liga.

Os atributos presentes nos elementos de cada documento são atributos de suas respectivas entidades. Então, a entidade jogador terá os seguintes atributos: id de jogador, primeiro nome, segundo nome, data de nascimento, altura e peso. Clube possui um id de clube e um nome. Treinador possui id de treinador, primeiro nome e segundo nome. Estádio possui id de estádio e nome. Os atributos de Liga e País foram coletados do site. A primeira possui id de liga e nome, enquanto a segunda, id de país e nome.

Os relacionamentos foram identificados por meio dos links presentes nas páginas referentes a cada instância das entidades. Então, como mostrado na Figura 16, a entidade Jogador se relaciona com as entidades Clube e País. Na Figura 17, é possível perceber que Clube se relaciona com Treinador, Estádio, Liga e País e na Figura 18 Ligas se relaciona com País.

A Figura 19 mostra o modelo conceitual, que ilustra o que foi explicado acima. Apesar de Clube se relacionar com País, essa relação não foi utilizada no modelo conceitual pelo fato de que, se um país possui uma liga e dessa liga participa um time, esse time consequentemente pertencerá ao país da liga.

Figura 19 - Proposta de modelo conceitual do SoccerWiki



Fonte: O autor

O passo seguinte, ao construir o modelo conceitual, é construir o modelo lógico (ou esquema relacional). Os relacionamentos são implementados fazendo com que a chave primária de uma entidade seja chave estrangeira na outra, com a qual se relaciona. Portanto, a seguir é apresentado um esquema relacional e em seguida, o mesmo em forma tabular para melhor visualização (Figura 20).

País(<u>id_pais</u>, nome)

Treinador(<u>id_treinador</u>, primeiro_nome, segundo_nome)

Estádio(id_estadio, nome)

Liga(id_liga, nome, id_pais)

id_pais referencia País

Clube(id_clube, nome, id_treinador, id_estadio, id_liga)

id_treinador referencia Treinador

id_estadio referencia Estadio

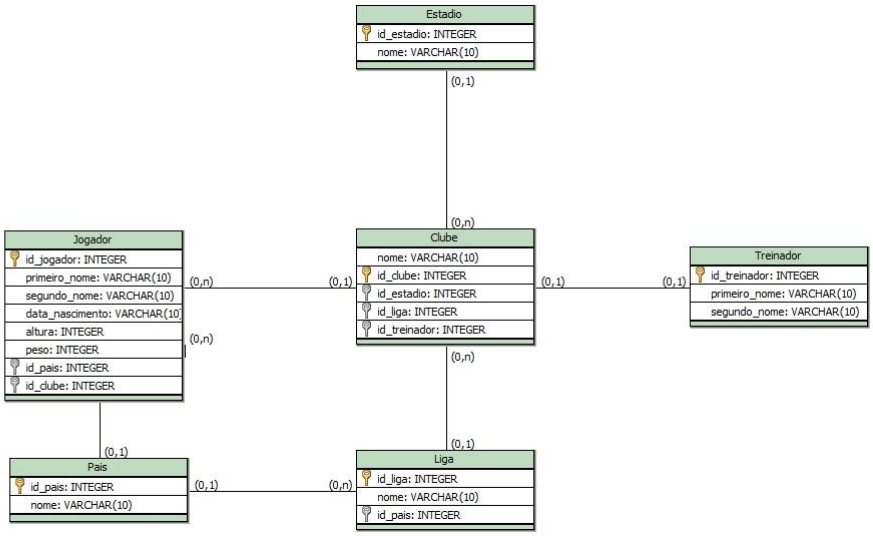
id_liga referencia Liga

Jogador(id_jogador, primeiro_nome, segundo_nome, data_nascimento, altura, peso, id_pais, id_clube)

id_pais referencia Pais

id_clube referencia Clube

Figura 20 - Proposta de modelo lógico em forma tabular do Soccerwiki

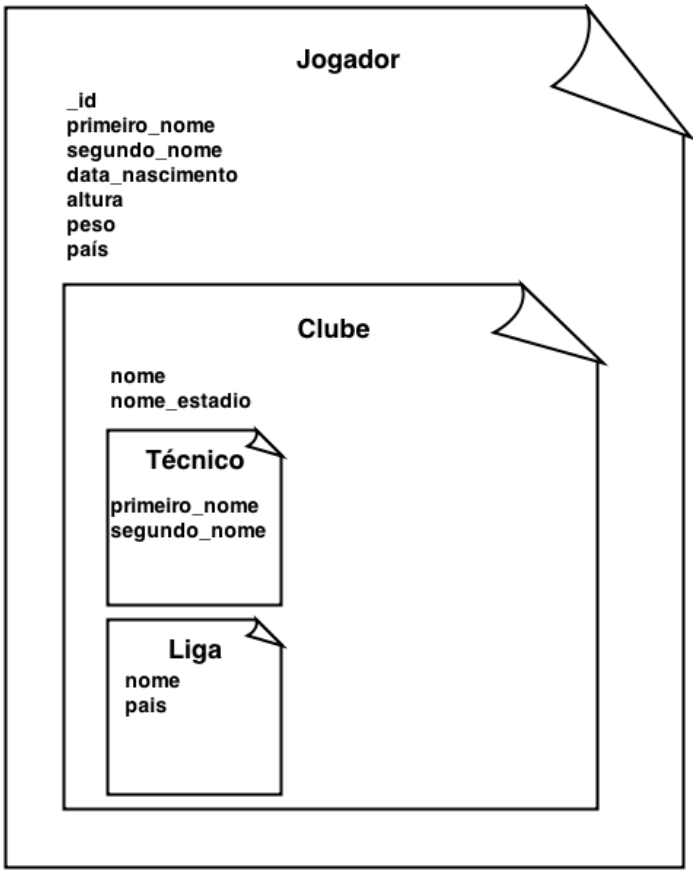


Fonte: O autor

3.4.2. Modelo de Documentos

O sistema não relacional utilizado neste trabalho é o MongoDB, que utiliza abordagem orientada a documentos. Como dito anteriormente, existem duas formas de implementar um relacionamento, pela abordagem *DBRefs* ou pela *Embedded Documents*, esta última é mais eficiente, vide Seção 3.2. Portanto, a abordagem utilizada neste trabalho será a *Embedded Documents*. A Figura 21 mostra a modelagem do documento equivalente à entidade Jogador. É possível perceber que, dentro do documento Jogador, existe o documento Clube, que por sua vez, dentro dele existem outros dois Técnico e Liga. Com isso, é possível perceber que haverá uma grande repetição de dados, visto que, vários jogadores jogam em um mesmo clube, isso faz com os dados do clube sejam replicados em cada documento jogador. Porém, essa repetição acarreta em um aumento do desempenho, já que não haverá a necessidade de buscar a informação do clube em outra coleção de documentos [11].

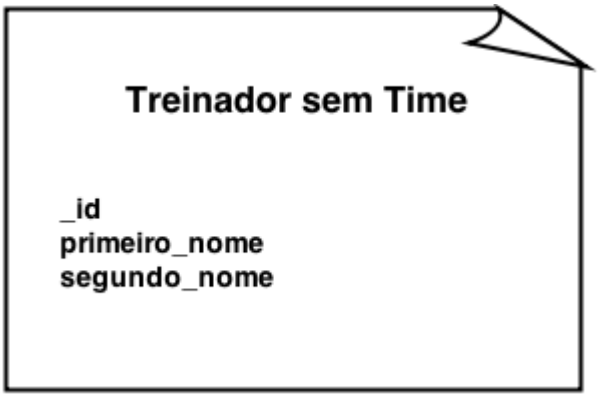
Figura 21 - Proposta de estrutura de documento de jogador para a base do Soccerwiki



Fonte: O autor

Como foi possível perceber, todo o modelo relacional pôde ser modelado em apenas um documento no MongoDB. Porém, no conjunto de dados existem exceções que não compatíveis com o documento da Figura 21. Neste caso, existe um treinador que não está relacionado com nenhum clube. Como o treinador é um sub documento de clube, não é possível inserir esse treinador no modelo de dados. Portanto, para atender a essa exceção, foi criada mais uma coleção chamada Treinador sem Time, na qual todos os treinadores que não estão relacionados com nenhum clube serão inseridos nela. A Figura 22 mostra a estrutura de um documento de treinador sem time.

Figura 22 - Proposta de estrutura de documento para treinador sem time do Soccerwiki



Fonte: O autor

3.4.3. Tratamento dos Dados

Para povoar os bancos de dados, foi necessário coletar os dados presentes apenas no site do SoccerWiki. Para isso, foi construído um programa Java [22], que varre as páginas do site e complementa as informações que não foram disponibilizadas nos documentos XML [19]. A varredura foi realizada inspecionando o código HTML [26] das páginas. Para isso, foi utilizado o Jsoup [27], uma biblioteca em Java que trabalha com HTML e possui uma API muito conveniente para extração e manipulação de dados. Em seguida, os dados são inseridos no banco de dados relacional.

Após a conclusão do povoamento do PostgreSQL, foi realizado o povoamento do MongoDB. Neste processo de povoamento foi utilizado outro programa escrito em Java e os dados foram extraídos do PostgreSQL. Dessa forma, não houve necessidade de coletar os dados do site do SoccerWiki novamente.

Para preencher a coleção de documentos referentes a jogador, o programa faz uma consulta ao PostgreSQL solicitando todos os dados pessoais de todos osjogadores, para cada jogador, o time no qual ele joga, o estádio no qual o time joga, o treinador do time e a liga da qual o time participa. Para a coleção de treinador sem time, o programa utiliza uma consulta que retorna todos os treinadores que não treinam nenhum time. Com os resultados das consultas, o programa estrutura o documento e o insere na coleção a qual pertence.

3.5. Plano de Experimento

O objetivo deste trabalho é analisar e comparar o desempenho dos dois SGBD, utilizando a base de dados coletada no SoccerWiki. O experimento consiste em analisar o desempenho de dois SGBD, definindo o tempo de inicialização dos usuários, variando o número de usuários que farão uma requisição. Assim, será possível analisar como os SGBD se comportam com o aumento do número de usuários, analisando três situações diferentes. Para o experimento, foi utilizado o tempo de inicialização igual a 10. Isso quer dizer que todos os usuários serão inicializados em 10 segundos. Os valores para o número de usuários são 2000, 4000 e 8000 e cada usuário faz uma requisição. A Tabela 2 resume os valores que foram utilizados. Ao fim deste trabalho, será possível apontar qual dos dois SGBD teve o melhor desempenho.

Tabela 2 - Valores utilizados nos experimentos

# Usuários	**Tempo de Inicialização**	**# Repetições/Usuário**
2000	10	1
4000	10	1
8000	10	1

Fonte: O autor

A consulta deve retornar todas as informações dos jogadores, o time no qual jogam, o estádio no qual o time joga, o treinador do time e a liga da qual o time participa de todos os jogadores que nasceram no Brasil.

A consulta formulada para o PostgreSQL, utilizando a linguagem SQL é mostrada na Figura 23:

Figura 23 - Consulta formulada para retornar todos os jogadores brasileiros no PostgreSQL

```
SELECT j.primeiro_nome AS jogador_primeiro_nome,
j.segundo_nome AS jogador_segundo_nome,
j.data_nascimento AS jogador_data_nascimento,
j.peso AS jogador_peso,
j.altura AS jogador_altura,
c.nome AS clube,
e.nome AS estadio,
t.primeiro_nome AS treinador_primeiro_nome,
t.segundo_nome AS treinador_segundo_nome,
l.nome AS liga,
p.nome AS pais_liga
FROM jogador j, clube c, estadio e, treinador t, liga l, pais p
WHERE j.id_time = c.id
AND c.estadio_id = e.id
AND c.treinador_id = t.id
AND c.liga_id = l.id
AND l.id_pais = p.id
AND j.id_pais = (select p.id from pais p where p.nome like '%Brasil%')
```

Fonte: O autor

A mesma consulta convertida para o MongoDB fica como mostrado na Figura 24.

Figura 24 - Consulta formulada para retornar todos os jogadores brasileiros no MongoDB

```
>
> db.jogador.find({pais: "  Brasil"})
```

Fonte: O autor

3.5.1. Dificuldades

Para se chegar ao plano de experimento utilizado neste trabalho, foram formulados outros planos, que por motivos de qualidade e limitações de hardware, foram sendo modificados até chegar ao experimento atual.

O primeiro plano de experimentos tinha a ideia de variar os três valores do *Thread Group*: o número de *threads*, o tempo de inicialização e a quantidade de repetições da requisição a ser feita. Dessa forma, esperava-se obter 27 configurações diferentes para analisar os dois SGBD. A Tabela 3 ilustra os valores para quantidade de usuários e repetições de requisição por usuário que seriam utilizados, enquanto a Tabela 4 mostra os valores de tempo de inicialização, em segundos, para cada valor de quantidade de usuários. Estes experimentos não puderam ser completados por limitações de hardware.

Tabela 3 – Valores para usuários e quantidade de repetições do primeiro plano de experimentos

#Usuários	**#Repetições/Usuário**
100, 1.000, 10.000	50, 500, 5.000

Fonte: O autor

Tabela 4 - Valores para usuários e tempo de inicialização do primeiro plano de experimentos

#Usuários	**Tempo de Inicialização (em segundos)**
100	50, 100, 200
1.000	500, 1.000, 2.000
10.000	5.000, 10.000, 20.000

Fonte: O autor

Foi possível executar todos os experimentos para 100 usuários e 50 repetições por usuário. Entretanto, ao variar um desses parâmetros, a tarefa não pôde ser finalizada para o PostgreSQL devido a limitações de hardware, travando o JMeter.

O segundo plano de experimento proposto visava contornar o problema encontrado no plano anterior. Para isso, decidiu-se fixar o número de usuários em 1000, fixar o tempo de inicialização de usuários em 250 segundos e variar apenas a quantidade de repetições de usuários entre 50, 500 e 5000. A Tabela 5 ilustra o resumo dos valores utilizados neste experimento.

Tabela 5 – Valores para o segundo plano de experimentos

# Usuários	Tempo de Inicialização	# Repetições/Usuário
1000	250	50
1000	250	500
1000	250	5000

Fonte: O autor

Apesar das modificações, que diminuiriam a quantidade de requisições nos experimentos, encontrou-se o mesmo problema: logo na primeira tupla de parâmetros, a tarefa não pôde ser completada para o SGBD relacional.

O terceiro plano de experimento foi o último antes do plano que foi escolhido para este trabalho. Para este plano de experimento, fixou-se a quantidade de repetições de requisições por usuário em 1 e variou-se a quantidade de usuários e tempo de inicialização conforme a Tabela 6.

Tabela 6 – Valores para o terceiro plano de experimentos

# Usuários	Tempo de Inicialização	# Repetições/Usuário
1000	1000	1
2000	500	1
4000	250	1

Fonte: O autor

Com esse conjunto de parâmetros, foi possível resolver o problema de limitações de hardware. Porém, os testes foram bastante demorados em ambos os sistemas, chegando a levar mais de 12h para a tarefa ser finalizada. Analisando este plano foi possível perceber que os tempos de inicialização dos usuários utilizados até então eram demasiadamente grandes em relação ao que acontece na realidade. Então, foi pensado um plano de experimento que inicializasse uma quantidade considerável de usuários em um intervalo de tempo pequeno, levando em consideração o hardware disponível para a realização dos testes.

3.6. Considerações Finais

Neste capítulo foram apresentados o PostgreSQL e o MongoDB, que são os SGBD utilizados no experimento. Além disso, foi feita uma breve introdução à ferramenta JMeter, utilizada para testes de desempenho. Também foi visto que a base de dados utilizada é disponibilizada pelo SoccerWiki e como os dados coletados foram tratados. Por fim, foi descrito o plano de experimentos e apontadas as dificuldades encontradas para se chegar nele.

O capítulo seguinte mostra os resultados que foram obtidos para os dois sistemas para o plano descrito acima.

4. RESULTADOS

Esta seção é dedicada a analisar os resultados obtidos nos experimentos. Como dito na seção anterior, os dois SGBD foram comparados com três valores de usuários diferentes em um ambiente em que todos os usuários devem ser inicializados em 10 segundos e cada usuário faz uma requisição. Para a quantidade de usuários foram utilizados os valores 2000, 4000 e 8000.

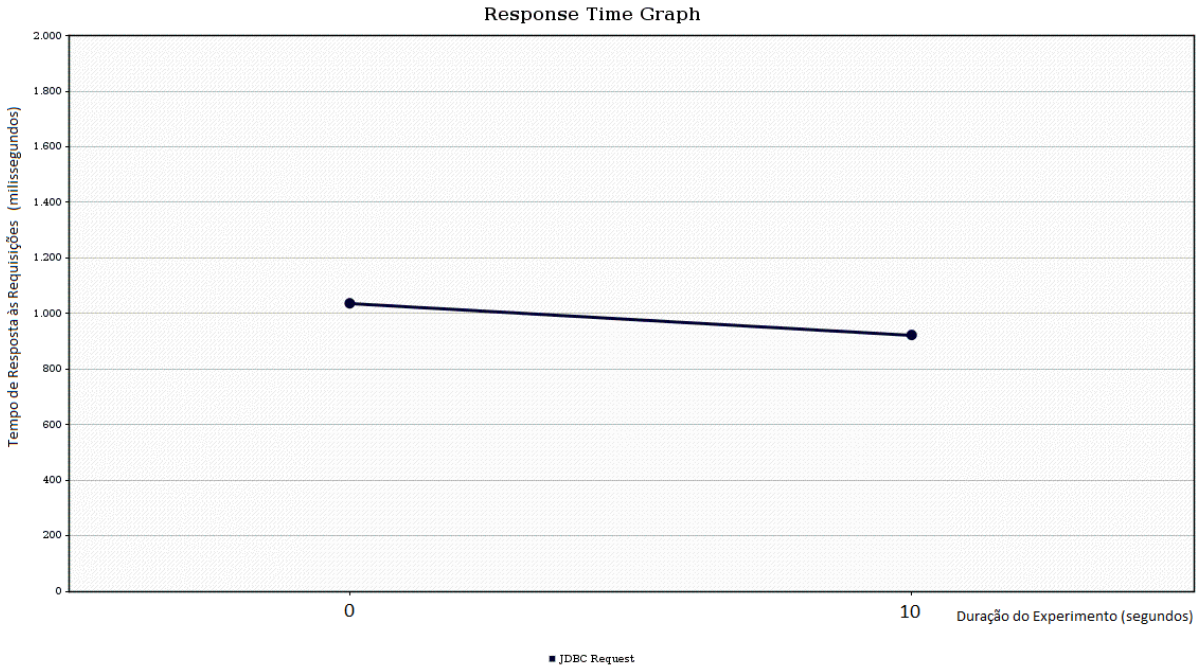
Os testes foram realizados em um computador com processador Intel(R) Core(TM) i5-3570 CPU @ 3.40GHz e 8GB de memória RAM. As evoluções do tempo de resposta dos SGBD durante os testes são apresentadas a seguir.

Em cada *Response Time Graph* mostrado nos resultados é ilustrado um gráfico em linha,representando a evolução do tempo de resposta do sistema para cada requisição. No caso em que houver várias requisições ao mesmo tempo, é mostrado um valor médio.

4.1. Experimento 1

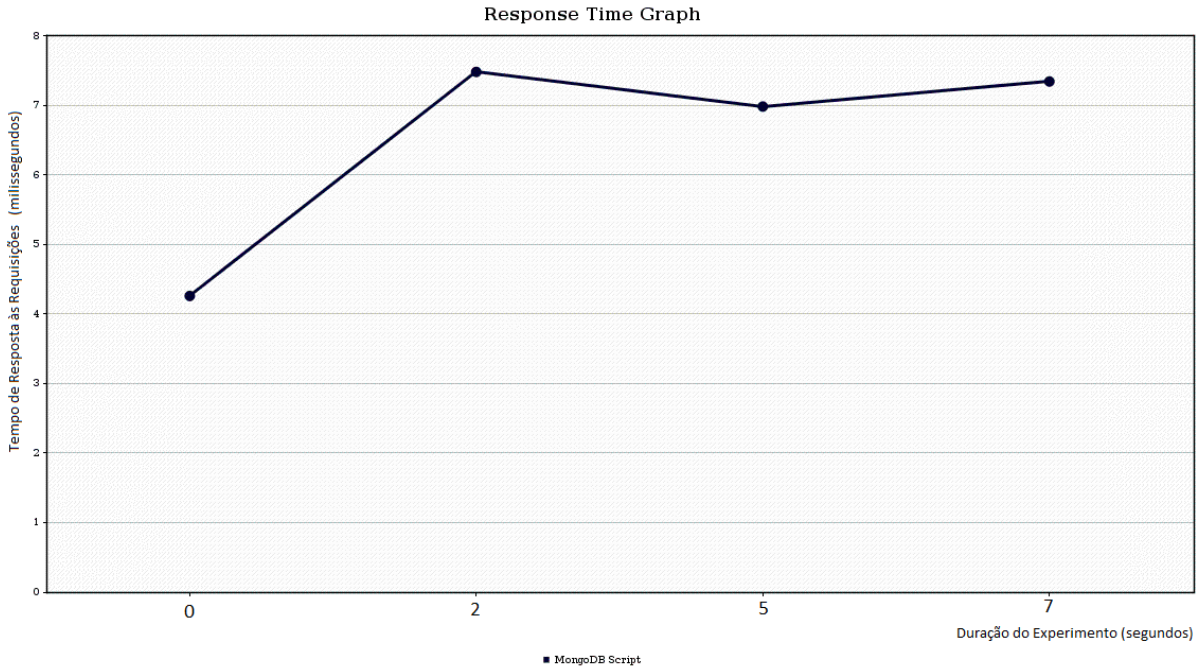
A consulta foi testada com 2000 usuários. A Figura 25 mostra a evolução do tempo de resposta do PostgreSQL, enquanto a Figura 26 mostra como o MongoDB se comportou com a mesma configuração.

Figura 25 - Resultado do experimento 1 para o PostgreSQL



Fonte: O autor

Figura 26 - Resultado do experimento 1 para o MongoDB



Fonte: O autor

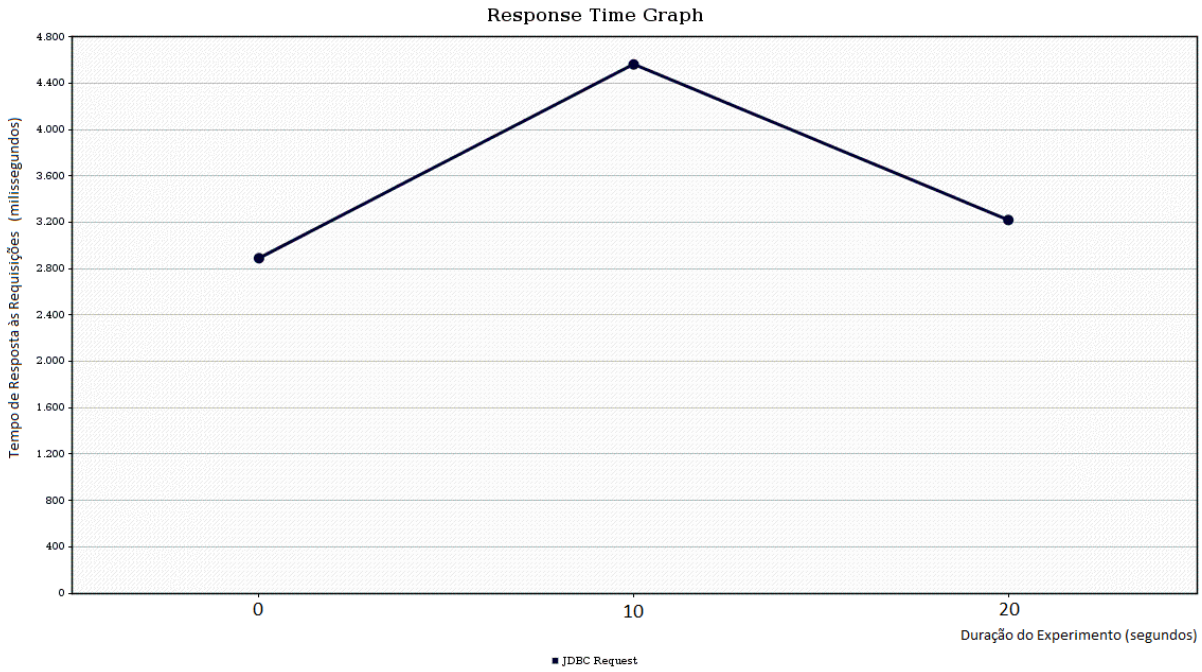
É possível perceber a diferença de tempo de resposta entre os dois sistemas logo neste primeiro experimento. Enquanto o tempo de resposta do PostgreSQL

teve uma média de 1 segundo (1000 milissegundos), o MongoDB realizou a mesma tarefa com o tempo de resposta abaixo de 10 milissegundos. Além disso, o experimento com o PostgreSQL durou 10 segundos, enquanto o MongoDB durou apenas 7 segundos.

4.2. Experimento 2

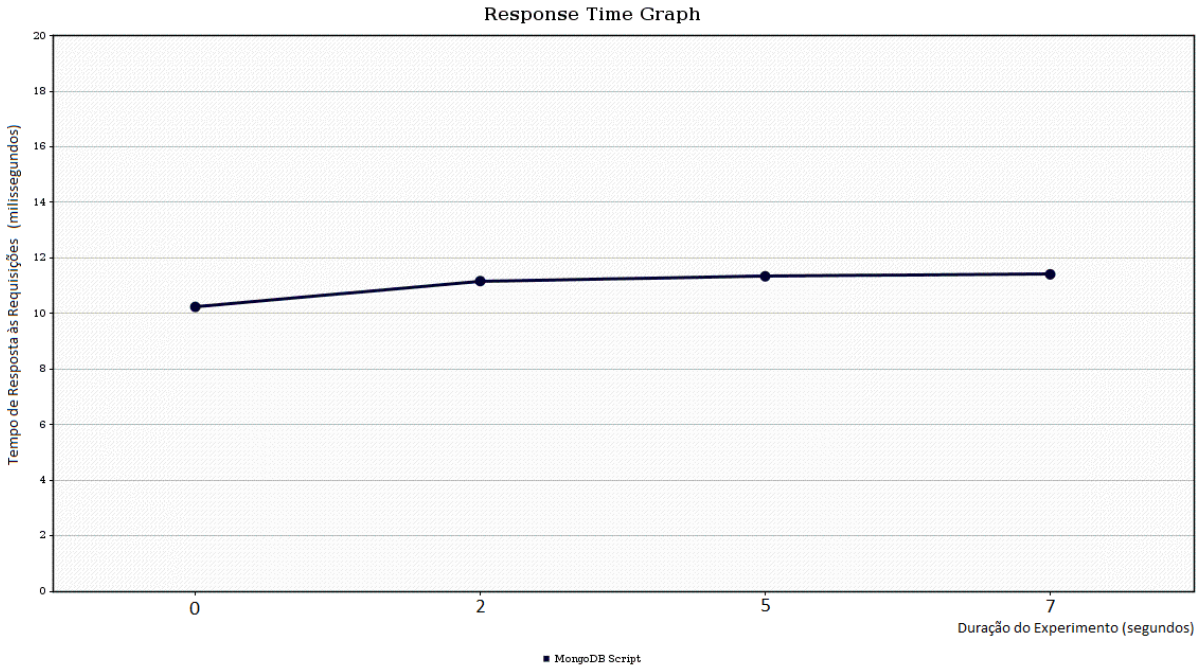
No experimento 2, o número de *threads* foi dobrado, ou seja o teste foi realizado com 4000 usuários. As figuras 27 e 28 mostram o desempenho do PostgreSQL e MongoDB, respectivamente.

Figura 27 - Resultado do experimento 2 para o PostgreSQL



Fonte: O autor

Figura 28 - Resultado do experimento 2 para o MongoDB



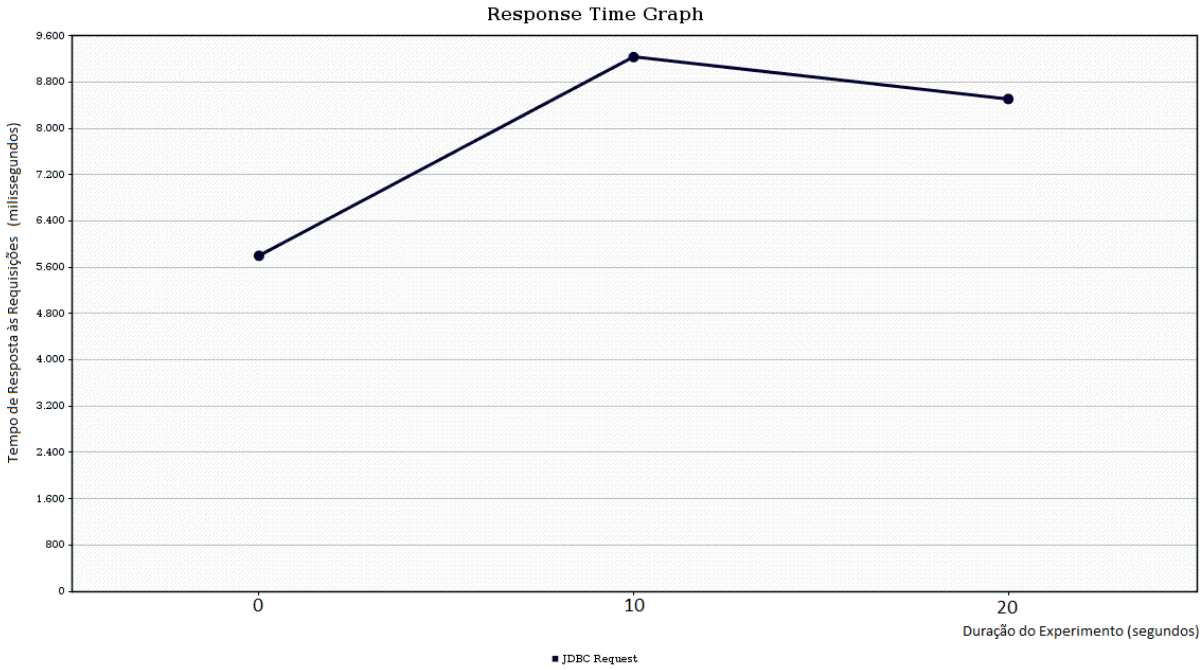
Fonte: O autor

Assim como no experimento 1, o MongoDB apresentou um desempenho melhor do que o PostgreSQL. Com 4000 usuários, o PostgreSQL teve um tempo de resposta entre 2800 e 4800 milissegundos e o experimento durou 20 segundos. Já o MongoDB estabilizou o seu tempo de resposta entre 10 e 12 milissegundos aumentando muito pouco a partir do experimento anterior, ainda mais se comparado ao aumento sofrido pelo PostgreSQL. Além disso, o MongoDB manteve os 7 segundos como tempo de duração do experimento.

4.3. Experimento 3

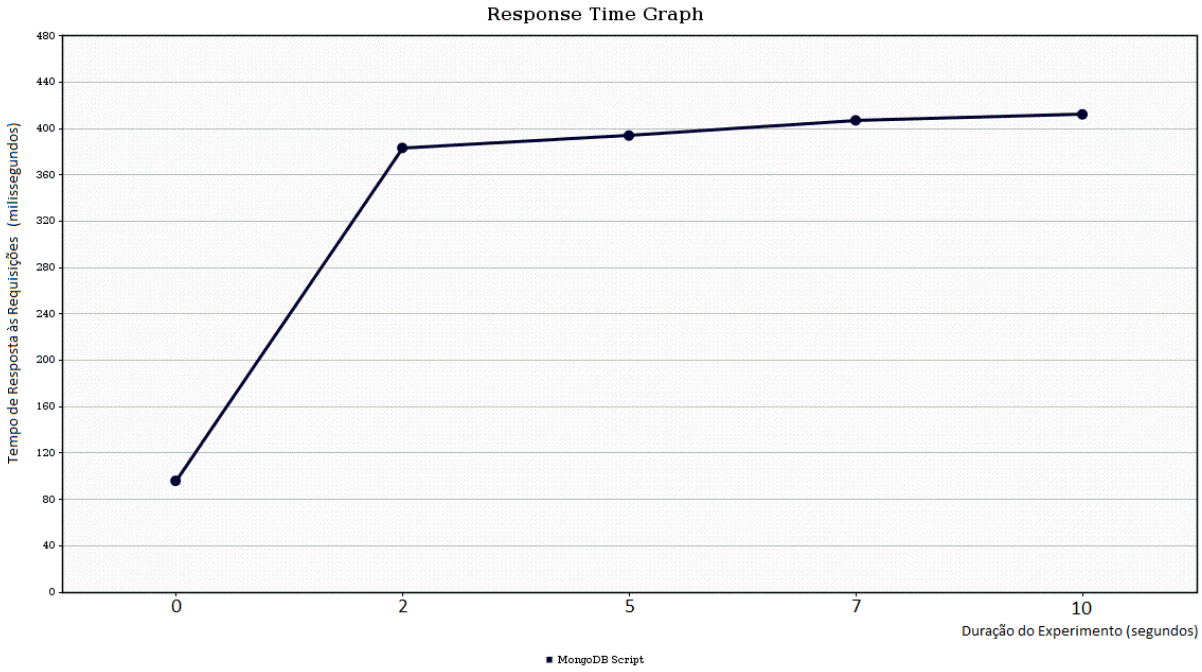
Para este último teste foram utilizados 8000 *threads*, ou seja, 8000 usuários, inicializados em 10 segundos. A Figura 29 mostra o comportamento do PostgreSQL, enquanto a Figura 30 ilustra o comportamento do MongoDB.

Figura 29 - Resultado do experimento 3 para o PostgreSQL



Fonte: O autor

Figura 30 - Resultado do experimento 3 para o MongoDB



Fonte: O autor

Assim como nos outros experimentos, é possível perceber que o tempo de resposta do MongoDB foi muito melhor do que o do PostgreSQL. O PostgreSQL teve o tempo de resposta entre 5600 e 9600 milissegundos e o experimento durou20

segundos. Já o tempo de resposta do MongoDB ficou entre 80 e 440 milissegundos e o experimento teve duração de 10 segundos.

4.4. Considerações Finais

Este capítulo apresentou os resultados obtidos para o PostgreSQL e o MongoDB, mostrando o tempo de resposta e em quanto tempo a tarefa é finalizada. Foram testadas três situações com quantidade de usuários diferentes e foi possível notar que, em todas, o MongoDB teve um desempenho melhor do que o PostgreSQL.

No próximo capítulo, serão feitas as considerações finais, bem como serão apontadas as contribuições e limitações encontradas no trabalho.

5. CONCLUSÃO

O objetivo deste trabalho foi analisar o desempenho de dois sistemas, um relacional e outro não relacional, com a mesma base de dados. O SGBD relacional escolhido foi o PostgreSQL, por ser o mais completo que se encontra no mercado sem custos. Para o sistema não relacional, foi escolhido o MongoDB por ser um dos mais conhecidos e por ser orientado a documentos, que é uma abordagem bastante diferente da abordagem relacional. A ferramenta escolhida foi o JMeter, devido à variedade de opções de configuração e compatibilidade com os dois sistemas.

A partir dos resultados, é possível afirmar que o MongoDB, utilizando a abordagem *Embedded Documents*, cuja consequência é uma ocupação maior de espaço em disco, apresenta o tempo de resposta consideravelmente menor do que o PostgreSQL. Também foi analisando o tempo de resposta do PostgreSQL, que os primeiros experimentos não puderam ser finalizados. Dessa forma, é possível afirmar que o tempo de resposta do mesmo foi determinante para o insucesso do experimento.

5.1. Contribuições e limitações

A principal contribuição deste trabalho foi mostrar o alto desempenho de um sistema não relacional, em particular, o MongoDB, que foi alvo deste experimento, em relação a um SGBD relacional, tendo um tempo de resposta às requisições muito menor do que o PostgreSQL e finalizando os experimentos em menor tempo.

Devido às configurações do computador utilizado para realizar os experimentos, o JMeter não conseguiu finalizar com sucesso as primeiras versões do plano de testes e, por essa razão, foi necessário fazer várias modificações no mesmo até chegar na versão final, que foi utilizada no trabalho.

5.2. Direções futuras

O uso do MongoDB foi bastante motivador. Portanto, como trabalhos futuros pretende-se replicar os testes descritos neste documento, com o MongoDB utilizando a abordagem de *BDRef*, já que foi utilizada apenas a *Embedded Documents*. Além disso, existe a pretensão de realizar outros testes de desempenho envolvendo outros tipos de SGBD não relacionais, como o orientado a grafo e chave/valor.

6. REFERÊNCIAS

[1] DIANA, M.de; GEROSA. M.A. NOSQL na Web 2.0: Um Estudo Comparativo de Bancos Não-Relacionais para Armazenamento de Dados na Web 2.0. IX Workshop de Teses e Dissertações em Banco de Dados. São Paulo, 2010.

[2] PostgreSQL. Disponível em:<http://www.postgresql.org> Acesso em 2 de julho de 2013.

[3] MongoDB. Disponível em:<http://www.mongodb.org> Acesso em 2 julho de 2013.

[4] SoccerWiki. Disponpivel em <http://pt-br.soccerwiki.org/wiki.php> Acesso em 2 de julho de 2014.

[5] CERÍCOLA, Osvaldo Vicente. *Banco de Dados Relacional e Distribuído*. Rio de Janeiro: LTC, 1991. p. 115.

[6] TAKAI, Osvaldo Kotaro; ITALIANO, Isabel Cristina; FERREIRA, João Eduardo. Projeto de Banco de Dados. São Paulo: Campus, 2000.

[7] MACHADO, Felipe; ABREU, Maurício. Projeto de Banco de Dados: Uma Visão Prática. 11. ed. São Paulo: Érica, 1996.

[8] ELMASRI, Ramez; NAVARTHE, ShamkantB..Sistemas de banco de dados. 6. ed. São Paulo: Pearson, 2011.

[9] Lóscio, B. F., Oliveira, H. R. e Pontes, J. C. S. (2011) “NoSQL no desenvolvimento de aplicações Web colaborativas”, VIII Simpósio Brasileiro de Sistemas Colaborativos.

[10] MCMURTRY, Douglas et al. Data Access for Highly- Scalable Solutions: Using SQL, NoSQL, and Polyglot Persistence. Disponível em: <http://www.microsoft.com/en-us/download/details.aspx?id=40327>. Acessoem: 06 out. 2014.

[11] TIWARI, Shashank. PROFESSIONAL NoSQL. Indianapolis: John Wiley & Sons, Inc., 2011.

[12] JMeter. Disponível em <https://jmeter.apache.org> Acesso em 2 de julho de 2014.

[13] Whatis Linux**.** Disponível em: <http://www.linuxfoundation.org/what-is-linux>. Acesso em: 23 jan. 2015

[14] Whatis UNIX ®? Disponível em: <http://www.unix.org/what_is_unix.html>. Acesso em: 23 jan. 2015.

[15] A historyof Windows**.** Disponível em: <http://windows.microsoft.com/en-us/windows/history#T1=era0>. Acesso em: 23 jan. 2015.

[16] pgAdmin. Disponível em <http://www.pgadmin.org/> Acesso em 14 de outubro de 2014.

[17] OS X. It's what makes a Mac a Mac. Disponível em: <https://www.apple.com//osx/what-is/>. Acesso em: 23 jan. 2015.

[18] A BriefDescription. Disponível em: <http://www.cplusplus.com/info/description/>. Acesso em: 23 jan. 2015.

[19] ExtensibleMarkupLanguage (XML). Disponível em: <http://www.w3.org/XML/>. Acesso em: 28 jan. 2015.

[20] Introducing JSON. Disponível em: <http://json.org/>. Acesso em: 28 jan. 2015.

[21] BSON. Disponível em: <http://bsonspec.org/>. Acesso em: 28 jan. 2015.

[22] Java. Disponível em <https://www.java.com/pt_BR> Acesso em 2 de julho de 2014.

[23] SOAP Introduction. Disponível em: <http://www.w3schools.com/webservices/ws_soap_intro.asp>. Acesso em: 31 jan. 2015.

[24] FILE TRANSFER PROTOCOL (FTP). Disponível em: <http://www.w3.org/Protocols/rfc959/>. Acesso em: 31 jan. 2015.

[25] JDBC. Disponível em: <http://www.oracle.com/technetwork/java/javase/jdbc/index.html>. Acesso em: 31 jan. 2015.

[26]HTMLIntroduction. Disponível em: <http://www.w3schools.com/html/html_intro.asp>. Acesso em: 31 jan. 2015.

[27] Jsoup. Disponível em <https://www.jsoup.org> Acesso em 3 de novembro de 2014.