UNIVERSIDADE ESTADUAL DA PARAÍBA – UEPB
CENTRO DE CIÊNCIAS EXATAS E SOCIAIS APLICADAS – CCEA
CAMPUS VII – GOVERNADOR ANTÔNIO MARIZ
CURSO DE LICENCIATURA EM COMPUTAÇÃO

JAZIEL SILVA MOREIRA

ESTUDO COMPARATIVO ENTRE BANCOS DE DADOS RELACIONAIS E NOSQL: MySQL x CouchDB x MongoDB

PATOS-PB
2016

JAZIEL SILVA MOREIRA

ESTUDO COMPARATIVO ENTRE BANCOS DE DADOS RELACIONAIS E NOSQL: MySQL x CouchDB x MongoDB

Trabalho de Conclusão de Curso apresentado ao Curso de Licenciatura em Computação da Universidade Estadual da Paraíba, em cumprimento à exigência para obtenção do grau de Licenciado em Computação.

Orientador: Prof. MSc. Pablo Ribeiro Suárez

PATOS-PB
2016

M838e Moreira, Jaziel Silva
Estudo comparativo entre bancos de dados relacionais e NoSQL [manuscrito] : MySQL x CouchDB x MongoDB / Jaziel Silva Moreira. - 2016.
80 p. : il. color.

Digitado.
Trabalho de Conclusão de Curso (Graduação em Computação) - Universidade Estadual da Paraíba, Centro de Ciências Exatas e Sociais Aplicadas, 2016.
"Orientação: Prof. Me. Pablo Ribeiro Suárez, CCEA".

1. Desempenho de banco de dados. 2. Banco de dados relacionais. 3. Banco de dados NoSQL. I. Título.
21. ed. CDD 005.74

Jaziel Silva Moreira

ESTUDO COMPARATIVO ENTRE BANCOS DE DADOS RELACIONAIS E NOSQL: MySQL x CouchDB x MONGODB

Trabalho de Conclusão de Curso apresentado ao Curso de Licenciatura em Computação da Universidade Estadual da Paraíba, em cumprimento à exigência para obtenção do grau de Licenciado em Computação

Aprovado em 12 de julho de 2016

BANCA EXAMINADORA

Pablo Ribeiro Suárez
(Orientador)

Pablo Roberto Fernandes de Oliveira
(Examinador)

Wellington Candeia de Araujo
(Examinador)

Dedico este trabalho à minha mãe, Marineuza Moreira, ao meu pai, José Moreira, e à minha irmã, Jardeane Moreira, pelo grande incentivo tanto na minha jornada acadêmica, como na minha vida pessoal.

AGRADECIMENTOS

Inicialmente agradeço a Deus por mais uma fase da minha vida concluída.

A minha família, por todos os incentivos e dedicação, tanto na minha vida pessoal como acadêmica.

A meu orientador Pablo Ribeiro Suárez, por tantos anos de ensino e por todo suporte oferecido durante o desenvolvimento deste trabalho.

A turma 2012.1, em especial meus amigos Lyncoln Monteiro, Marianne Félix e Victor Fortunato, que em todos os momentos, bons e ruins, estiveram comigo compartilhando alegrias ou me dando forças pra superar os obstáculos da vida.

Aos meus amigos fora da UEPB, que sempre me apoiaram, em especial Micaelly Lucena, pela dicas durante a conclusão deste trabalho.

A todos os meus professores que me ajudaram a construir uma formação acadêmica.

Enfim, a todos que contribuíram direto ou indiretamente com meu percurso acadêmico.

RESUMO

Com o grande aumento do volume de dados existente, tornou-se necessário o desenvolvimento de alternativas de gerenciamento de dados, que possam suprir o aumento da demanda de desempenho e disponibilidade. Assim sendo, várias alternativas foram desenvolvidas em contraposição ao renomado modelo relacional. Essa nova categoria de bancos de dados ficou conhecida como NoSQL. Nesse sentido, este trabalho justifica-se pela necessidade de se apresentar as características desses sistemas e enfatizar as diferenças com o modelo relacional. A pesquisa tem como objetivo realizar um estudo comparativo entre sistemas de gerenciamento de banco de dados relacionais e NoSQL, realizando uma comparação de performance entre o MySQL, CouchDB e o MongoDB. O estudo foi desenvolvido a partir de uma análise quantitativa, tendo em vista que procura analisar o desempenho dos sistemas. Para a realização desta pesquisa foi adotado como procedimento técnico o estudo de caso, com a realização de simulações de operações de escrita e consulta, obtendo assim, dados relacionados especificamente a performance de cada SGBD. Deste modo, ao término deste trabalho foi percebida grande diferença entre os resultados obtidos pelos três softwares analisados, sendo o MongoDB o que apresentou melhor desempenho, seguido do CouchDB, e, por fim, o MySQL. Além disso, foi possível analisar como esses softwares trabalham de forma distribuída, e como reagem a falhas de conexão. Portanto, esta pesquisa contribuiu para melhor compreensão dessa nova categoria de bancos de dados, bem como seu funcionamento, mesmo os testes ocorrendo em um ambiente não distribuído.

Palavras-chave: Desempenho. Banco de dados relacional. Banco de dados NoSQL.

ABSTRACT

With the large increase in the existing data amount, it has become necessary to develop data management alternatives that can meet the increasing demand for performance and availability. Therefore, several alternatives have been developed in opposition to the renowned relational model. This new category of databases has become known as NoSQL. In this sense, this work is justified by the need to present the features of these systems and emphasize the differences with the relational model. The research aims to conduct a comparative study between relational and NoSQL database management systems, carrying out a performance comparison among MySQL, CouchDB and MongoDB. The study was developed from a quantitative analysis, since it seeks to analyze the performance of the systems. For this research was adopted as technical procedure the case study, performing simulations of writing and query operations, obtaining data specifically related to the performance of each DBMS. Thus, at the end of this work we could noticed big difference among the results obtained by the three softwares analyzed, with MongoDB presenting the best performance, followed by CouchDB, and finally MySQL. In addition, it was possible to analyze how these softwares work in a distributed manner, and how they react to connection failures. Therefore, this research has contributed to better understanding of this new category of databases, as well as its operation, even the tests taking place in a non-distributed environment.

Keywords: Performance. Relational database. NoSQL database.

LISTA DE GRÁFICOS

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE CÓDIGOS

SUMÁRIO

1 INTRODUÇÃO

1.1 CENÁRIO TÉCNICO-CIENTIFICO

A importância do armazenamento de dados vem intensificando-se ao longo do tempo no mundo da computação. Ao passo que cresce o volume de dados, cresce também a necessidade de alternativas de gerenciamento de dados, que possam suprir o aumento da demanda de performance e disponibilidade.

Os sistemas de armazenamento mais convencionais são os Bancos de dados relacionais, orientados a objetos e objeto relacional. Por décadas, os Bancos de dados relacionais vêm sendo utilizados com maior frequência em relação aos demais modelos (POLITOWSKI; MARAN, 2014). Através de *queries* SQL, que é a linguagem padrão na maioria dos Sistemas de Gerenciamento de Banco de Dados (SGBDs), é possível gravar e coletar dados de maneira simples.

Entretanto, embora os Bancos de Dados Relacionais (BDRs) ofereçam vários recursos, ainda existem limitações quanto à sua utilização, tais como complexidade de projeto, estrutura limitada, entre outras. Em consequência do crescimento da quantidade de dados, a escalabilidade está se tornando um fator de extrema importância nos bancos de dados.

Escalabilidade é uma característica que indica a capacidade que um sistema tem de manipular uma porção crescente de trabalho uniforme, ou estar preparado para crescer. A escalabilidade ocorre de duas maneiras: vertical (*scale up*) e horizontal (*scale out*). A escalabilidade vertical consiste em adicionar recursos a um nó do sistema enquanto a escalabilidade horizontal consiste em adicionar nós ao sistema.

Politowski e Maran (2014, p.3) afirmam que “Bancos de Dados Relacionais foram projetados para serem executados em uma máquina apenas [...]”, ou seja, para aumentar seu desempenho, é necessário realizar escalabilidade vertical. Contudo, apesar de ser possível escalar horizontalmente o banco de dados relacional em um sistema distribuído com particionamento de dados, tal tarefa não é realizada de maneira simples devido a sua natureza estruturada.

O teorema CAP fala sobre três propriedades esperadas de um sistema computacional distribuído. A sigla é formada pelas iniciais dessas propriedades, sendo elas **C**onsistancy (consistência), **A**vailability (disponibilidade) e **P**artition Tolerance

(tolerância a particionamento). Segundo o teorema, só é possível garantir duas dessas propriedades em um dado momento.

Assim sendo, Brito (2010) afirma que as soluções propostas para sanar as limitações dos SGBDs relacionais tinham como base a minimização de regras e estruturação presentes no modelo relacional, flexibilizando os sistemas de bancos de dados para cada organização.

A partir de 2004, grandes empresas como Google, Amazon, Facebook, entre outras, começaram a desenvolver seus próprios sistemas inteiramente não-relacionais. O termo NoSQL, que foi usado pioneiramente por Carlo Strozzi em 1998 para nomear um banco de dados relacional que não possuía interface SQL. Em 2009, o termo passou a representar uma categoria de bancos de dados que proviam uma alternativa ao Modelo Relacional.

Com o surgimento do movimento NoSQL, a sigla tornou-se uma abreviação de *Not Only SQL* (não apenas SQL). Carlo Strozzi (2015) alega que o movimento NoSQL deveria ser chamado de “NoREL” ou algo semelhante, uma vez que é completamente distinto do modelo relacional. Existem outros termos para esta categoria de banco de dados como NF^2, N1NF (*Nom Firt Normal Form*), *free-form*, MRNN (Modelo Relacional não normalizado), entre outros.

Diana e Gerosa (2010) afirmam em seu trabalho que os tipos mais conhecidos de bancos de dados dessa classe são: banco de dados orientados a documentos, orientados a colunas, armazéns de chave-valor e banco de dados baseados em grafos.

Na primeira categoria, os documentos são as unidades básicas da informação e não utilizam estruturação pré-definida. No segundo modelo, muda-se o foco em relação ao modelo relacional antes orientado às tuplas para orientado a atributos. Na terceira, existe uma coleção de chaves únicas e de valores que estão associados a estas chaves. No quarto grupo, os dados são armazenados em nós de grafos e as arestas representam as associações.

Iniciativas relacionadas a essas novas tecnologias já vem sendo observadas na literatura. Alguns estudos comparativos foram realizados buscando explicitar as diferenças existentes entre os BDRs convencionais com os SGBDs NoSQL. Almeida e Brito (2012) abordaram a utilização de bancos de dados não relacionais para a manipulação de diversas estruturas.

Diana e Gerosa (2010) realizaram um estudo comparativo entre ferramentas não relacionais e sua aplicação no armazenamento de dados na WEB 2.0. Politowski e Maran (2014) realizaram um comparativo de desempenho entre ferramentas de ambas as categorias, o PostgreSQL (relacional) e o MongoDB (não relacional).

Os resultados obtidos com os trabalhos realizados até então são importantes, e devem ser ampliados. Portanto, este trabalho objetiva realizar um estudo comparativo de desempenho entre representantes das duas categorias para verificar qual possui melhor desempenho no ambiente de teste definido. Ao mesmo tempo, explicitar as diferenças entre ambas as classes.

A escolha das ferramentas teve por base o teorema cap. Assim sendo, as ferramentas escolhidas foram o MySQL, que provê consistência e disponibilidade, o CouchDB que provê disponibilidade e tolerância a particionamento, e o MongoDB, que provê consistência e tolerância a particionamento. Tais ferramentas foram escolhidas devido ao seu alto nível de utilização, além de serem *Open Source*.

1.2 PROBLEMÁTICA DA PESQUISA

Tendo em vista o cenário previamente descrito, surgem as seguintes questões. Quais as principais características dos Sistemas NoSQL? Qual seu diferencial em relação aos SGBDs relacionais? Qual SGBD possui melhor desempenho de acordo com o cenário de experimento? Para obtenção das respostas desses questionamentos, o presente projeto será realizado tendo por base os seguintes objetivos.

1.3 OBJETIVOS

A seguir serão elencados os objetivos deste trabalho, tanto o de natureza geral quanto os de natureza específicas.

1.3.1 Objetivo geral

Realizar um estudo comparativo entre SGBDs relacionais e NoSQL realizando uma comparação de performance entre o MySQL, CouchDB e o MongoDB.

1.3.2 Objetivos específicos

O objetivo acima proposto foi alcançado através dos seguintes objetivos específicos.

- Realizar estudo sobre aplicação de SGBDs relacionais e NoSQL;
- Apresentar as ferramentas alvo da comparação;
- Definir ambientes de testes;
- Realizar simulações com operações de escrita e consulta;
- Analisar resultados obtidos;

1.4 JUSTIFICATIVA

Os SGBDs NoSQL estão se tornando cada vez mais populares entre os profissionais da área, porém, “Por se tratar de um conjunto de tecnologias e modelos relativamente novos, os bancos de dados NoSQL são vistos com desconfiança por parte da comunidade e descrença por parte de outros [...]” (POLITOWSKI; MARAN, 2014, p.2). Mesmo que grandes empresas como Google e Facebook tenham adotado a solução, parte da comunidade ainda duvida do potencial dessas alternativas.

Nesse sentido, este trabalho justifica-se pela necessidade de se apresentar as características desses sistemas e enfatizar as diferenças com o modelo relacional. Ao mesmo tempo, realizar um estudo comparativo de desempenho entre ferramentas de ambas as classes, expondo assim, qual se mostra mais o eficiente. Portanto, a realização deste projeto deu-se através da seguinte metodologia.

1.5 METODOLOGIA

A pesquisa é um processo metódico composto de atividades para encontrar soluções para um problema proposto. Em casos onde a informação é insuficiente, ou até mesmo inexistente, a pesquisa é realizada com o objetivo de proporcionar as informações necessárias. (SILVA; MENEZES, 2005).

De acordo com Silva e Menezes (2005), embora exista várias formas de se classificar pesquisas, as quatro mais clássicas são: quanto a sua natureza, quanto à forma de abordagem, quanto aos objetivos e quanto aos procedimentos técnicos.

Deste modo, a pesquisa realizada neste trabalho caracteriza-se como de natureza aplicada, pois de acordo com Sousa (2009, p. 15), tem por objetivo “[...] gerar conhecimentos para aplicação prática dirigidos à solução de problemas específicos. Envolve verdades e interesses locais.”

A forma de abordagem do problema dessa pesquisa é do tipo quantitativa, tendo em vista que procura analisar o desempenho das ferramentas, medindo quanto tempo gastam na execução de determinados comandos. Silva e Menezes (2005, p. 20) alega que a pesquisa quantitativa “[...] considera que tudo pode ser quantificável, o que significa traduzir em números opiniões e informações para classificá-las e analisá-las [...]”.

Em relação aos objetivos, esta pesquisa caracteriza-se como exploratória, que segundo Gil (2002), proporciona maior familiaridade com o problema estudado a fim de torna-lo mais explícito.

Para a realização desta pesquisa foi adotado como procedimento técnico o estudo de caso. Gil (2002, p. 54) define o estudo de caso como “[...] estudo profundo e exaustivo de um ou poucos objetos, de maneira que permita seu amplo e detalhado conhecimento [...]”.

Deste modo, esta pesquisa contempla as seguintes etapas:

1. Estudo sobre aspectos gerais e aplicações dos SGBDs relacionais e NoSQL através de fontes bibliográficas;
2. Apresentação de características das ferramentas que foram avaliadas neste estudo;
3. Definição do ambiente de teste, tendo em vista a compatibilidade com ambos os modelos;
4. Realização de simulações com operações de escrita e consulta, obtendo assim, dados relacionados a sua performance;
5. Análise dos resultados obtidos a fim de apresentar pontos positivos e negativos das ferramentas, identificando qual apresenta melhor desempenho.

1.6 ESTRUTURA DO TRABALHO

O presente trabalho foi dividido em capítulos para melhor organização das informações. O segundo capítulo fornece o conhecimento teórico necessário para compreensão da pesquisa realizada, explicando o funcionamento de bancos de dados relacionais, não relacionais, e suas propriedades.

O terceiro capítulo explica como foi realizado o experimento objeto de pesquisa deste trabalho. O quarto capítulo apresenta os resultados obtidos com os teste realizados além de compara-los. No quinto capítulo são apresentadas as conclusões do trabalho, além de sugestões de trabalhos futuros.

2 REFERENCIAL TEÓRICO

2.1 CONSIDERAÇÕES INICIAIS

Este capitulo objetiva fornecer conhecimentos teóricos acerca do tema abordado neste trabalho, servindo de embasamento para a realização da pesquisa e compreensão de seus resultados.

2.2 BANCOS DE DADOS RELACIONAIS

Em 1970, o matemático britânico Edgar Frank Codd propôs o modelo relacional como uma nova maneira de representação de dados. Uma descrição de maneira natural, sem estruturas adicionais, era possível através de uma visão relacional, tornando a camada de persistência mais independente da camada lógica (CODD, 1970). Para dar fundamentação a seu modelo, foi definida uma álgebra relacional que provou sua completude através de sua equivalência com o cálculo relacional.

O modelo relacional consiste basicamente na criação de tabelas capazes de representar entidades do mundo real, em que as colunas definem o esquema e as linhas são instâncias. O relacionamento entre as entidades dá-se através de chaves. Chaves primárias identificam de maneira única as instâncias presentes nas tabelas, já as chaves secundárias são responsáveis por garantir uma relação de dependência entre atributos de tabelas diferentes.

De acordo com Brito (2010, p.1), “Os SGBDs relacionais oferecem aos usuários processos de validação, verificação e garantias de integridade dos dados, controle de concorrência, recuperação de falhas, segurança, controle de transações, otimização de consultas, dentre outros”. Segundo o autor, tais recursos possibilitaram que os desenvolvedores de aplicações pudessem empenhar-se exclusivamente com o foco da aplicação.

Algumas características como, acesso concorrente de múltiplos usuários ao mesmo banco de dados e a possibilidade do sistema recuperar-se de possíveis falhas, fizeram com que os SGBDs relacionais se mantivessem em posição de destaque entre os diversos ambientes computacionais.

Porém, para o uso de um sistema relacional é necessário que seu projeto seja realizado de maneira adequada, uma vez que uma modelagem mal feita pode acarretar em inconsistências de integridade e prejudicar seu desempenho. Também é necessário atentar para as limitações de estruturas presentes nessa categoria de SGBD, ponto realmente impactante principalmente para sistemas multimídia que são semiestruturados ou não possui estrutura pré-definida, como é o caso de páginas web e arquivos multimídia.

Contudo, a maior dificuldade enfrentada ao se trabalhar com bancos de dados relacionais é realizar seu escalonamento. Com o crescimento exponencial do volume de dados, escalonar apenas os servidores não é mais tão eficiente, uma vez que o problema estaria no acesso à base de dados. Nesta perspectiva, escalar a base de dados torna-se a única solução.

2.3 BANCOS DE DADOS NOSQL

O modelo não-relacional surgiu como solução para empresas que lidavam com uma grande imensidade de dados, os quais o modelo relacional não era mais capaz de suprir. Várias alternativas de gerenciamento de dados surgiram nos últimos anos advindas da necessidade de minimizar a estrutura dos bancos de dados relacionais. Essas alternativas formaram uma nova categoria de banco de dados que ficou conhecida como NoSQL (DIANA; GEROSA, 2010).

Por terem sido criados para lidar com grande volume de dados, muitos dos quais não são estruturados ou são apenas semiestruturados, SGBDs NoSQL possuem características que os diferem dos relacionais. Para que o SGBD possa lidar com uma quantidade de dados cada vez maior é necessário escalonar o sistema. A escalabilidade vertical, além de economicamente inviável, está limitada a tecnologia atual, uma vez que para executá-la é necessária a aquisição de hardware superior ao presente no sistema.

A escalabilidade horizontal, por outro lado, ocorre através da inserção de máquinas no sistema, sendo possível escalona-lo quantas vezes forem necessárias. Um sistema escalonado horizontalmente divide uma tarefa em processos que são distribuídos entre suas máquinas. A estrutura flexível dos bancos de dados não-relacionais é uma de suas principais características, pois permite que sejam

facilmente escalonados horizontalmente. Além da escalabilidade, a minimização das regras trouxe também um aumento na disponibilidade das informações.

Outra característica importante dos SGBDs NoSQL é que permitem a replicação dos dados de forma nativa, provendo redução do tempo gasto na recuperação de informações (NOSQL, 2015). Há dois tipos de replicação, *Master-to-Master* e *Master-Slave.*

No modelo *Master-Slave* só é permitida a gravação na réplica *Master*, cujas modificações são transmitidas assim que possível para cópia *Slave.* Porém, operações de leitura podem ser executadas em ambas as réplicas. No modelo *Master-to-Master,* tanto operações de escrita quanto de leitura podem ser executadas em ambas as réplicas. As alterações realizadas em uma das cópias são automaticamente realizadas na outra.

A fim de prover maior disponibilidade de informações de forma eficiente, SGBDs desta categoria disponibilizam uma API simples, possibilitando o fácil acesso aos dados armazenados. Para que todas as funcionalidades de um sistema não-relacional possam ser implementadas, torna-se necessária a utilização de técnicas que garantam sua eficiência.

2.3.1 Técnicas implementadas

Para que um processo possa ser executado em várias máquinas, de forma paralela, é necessário adaptá-lo para tal finalidade. Entretanto, ao se realizar tal tarefa, tratando as possíveis falhas, o código, que originalmente é simples, tende a tornar-se cada vez mais complexo (DEAN; GHEMAWAT, 2004). Assim sendo, Jeffrey Dean e Sanjay Ghemawat (2004) desenvolveram uma nova abstração que permite a simplificação do código ocultando os detalhes da paralelização, distribuição de dados, carregamento balanceado, e tolerância a falha na biblioteca MapReduce.

O MapReduce consiste basicamente em duas funções, a função *Map* e a função *Reduce*. A função *Map,* desenvolvida pelo usuário, recebe uma entrada e produz um conjunto intermediário de pares chave/valor. Esses pares são distribuídos em outros nós da rede. A biblioteca MapReduce agrupa os valores intermediários com mesma chave e os passa para a função *Reduce*.

A função *Reduce*, também escrita pelo usuário, tem como entrada uma chave e um conjunto de valores. Os dados coletados são combinados e processados a fim

de produzir uma nova saída que pode ser coletada e passada para outra invocação da função *Reduce.* Os dados vão sendo processados e repassados nó a nó até chegar ao nó raiz do problema.

Outro problema ao se trabalhar com grandes volumes de dados é o tempo gasto para localizar a informação desejada. Deste modo, várias técnicas foram desenvolvidas objetivando prover maior velocidade na recuperação dos dados. Tabelas *hash* são amplamente utilizadas nesse cenário pois permitem a localização precisa (ou pelo menos aproximada) do dado procurado. Para este fim, o endereçamento dos dados na tabela é definido através de uma função chamada *hash,* que também é utilizada na recuperação da informação.

No modelo cliente-servidor é comum a inserção de uma camada de cache a fim de minimizar o processamento no servidor. Assim sendo, uma função *hash* pode ser utilizada para mapear qual cache é responsável por determinada informação. Porém, ao se alterar o número de máquinas disponíveis a inconsistência gerada em uma distribuição com uma função *hash* clássica seria catastrófica (KARGER; *et al*, 1997)

Para resolver problemas de inconsistência, Karger *et al* (1997) desenvolveram uma técnica chamada *Consistent hashing,* que consiste na definição de um *view* para ser o conjunto de caches para um cliente em particular. Para obtenção de dados, o cliente utiliza uma função *hash* consistente para mapear o objeto para uma cache em seu *view*.

A fim de prover maior disponibilidade dos dados armazenados, alguns sistemas NoSQL utilizam o método *Multiversion concurrency control* (MVCC) que provê acesso concorrente aos objetos armazenados no banco de dados (LÓSCIO; OLIVEIRA; PONTES, 2011). Este método consiste na criação de múltiplas versões dos objetos para que comandos de escrita e leitura possam ser efetuados simultaneamente. Ao se atualizar um dado, a nova versão, ao invés de sobrescrever a antiga, será escrita em outro local.

Deste modo, se um cliente desejar obter um dado que esteja sendo reescrito por outro cliente, este receberá a versão mais antiga da informação. Para controle das versões existentes na base dados, este método utiliza um *timestamp* para determinar qual a versão mais atual de cada objeto. Isso permite que versões obsoletas sejam descartadas, disponibilizando sempre a versão mais recente disponível.

2.3.2 Modelos de Dados

Apesar dos sistemas não-relacionais possuírem várias características em comum como as descritas anteriormente, existem muitas especificidades apresentadas pelas ferramentas desta categoria, especialmente referente à forma como os dados são armazenados. Como dito anteriormente, os tipos mais conhecidos de SGBDs não-relacionais são: armazéns de chave-valor, banco de dados orientados a colunas, orientados a documentos e banco de dados baseados em grafos.

Armazéns de chave-valor é o mais simples dos quatro. Este modelo é composto por pares chave/valor, que permite a visualização do banco como uma grande tabela *hash* altamente escalável. Para a manipulação dos dados são necessárias apenas duas funções: ***put*()** que insere um novo registro na base de dados, e ***get*()** para recuperação dados através de uma chave.

Em bancos de dados orientados a colunas, os dados são armazenados contiguamente no disco, ou seja, são orientados a atributos, diferente do que ocorre no modelo relacional que é orientado a registros (DIANA; GEROSA, 2010). Deste modo, perde-se desempenho ao se armazenar uma nova informação, entretanto, para buscas onde se é necessário verificar várias linhas em poucas colunas, este mostra-se extremamente eficaz.

Neste modelo, os dados são referenciados por colunas e linhas, que representam respectivamente os atributos e os registros armazenados. Além destes, os dados também são referenciados por um *timestamp* para identificar a versão mais recente dos dados armazenados.

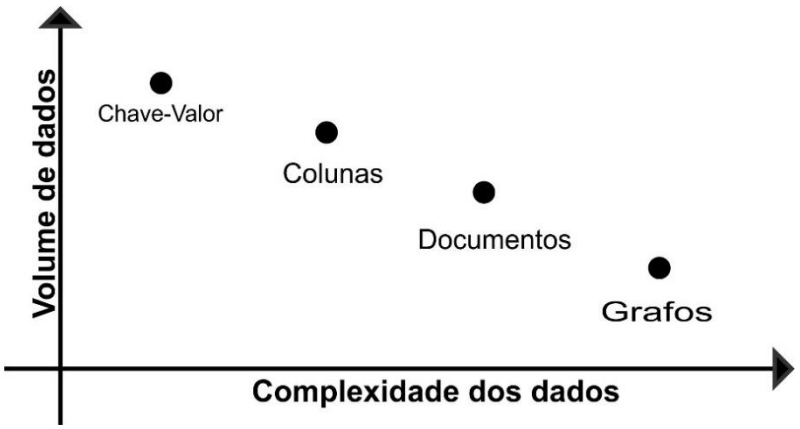
Bancos de dados orientado a documentos são compostos por coleções de documentos que são sua unidade básica de armazenamento. Os documentos são formados por um conjunto de campos responsáveis por armazenar as informações que podem ser simples ou multivaloradas, sendo capazes de armazenar listas ou mesmo outros documentos. Esses campos são referenciados por chaves e não possuem esquemas pré-definidos, assim sendo, para alterar o esquema de um documento não é necessário que os demais sigam o mesmo modelo.

No modelo orientado a grafos os registros são armazenados em nós que se conectam através de arestas responsáveis pela representação de seus relacionamentos. Nesse modelo podem ser aplicados conceitos de grafos tais como transformação, caminhos, vizinhos e sub-grafos. Deste modo, fica evidente sua

performance na execução de buscas complexas, fazendo deste modelo o mais indicado quando os relacionamentos dos dados são de complexidade elevada.

A seguir, temos a representação gráfica da relação entre o volume e a complexidade dos dados.

Gráfico 1 – Relação Volume x Complexidade.



Fonte: NASCIMENTO, 2010, adaptado pelo autor.

Tomando por base o gráfico apresentado, nota-se que a relação entre o volume e a complexidade dos dados são inversamente proporcionais. Ou seja, quanto maior a complexidade dos dados, menor o volume de dados possível de ser trabalhado de forma eficiente.

2.4 BASE VS ACID

Para que fosse possível ambas as categorias serem capazes de sanar a demanda vigente para a qual foram criadas, foi necessário a definição de propriedades que satisfizessem seus requisitos operacionais. Na década de 1970, quando o modelo relacional foi proposto, a maior preocupação relacionada ao armazenamento de dados era que este fosse realizado de forma consistente e duradoura.

2.4.1 Propriedades ACID

Assim sendo, sistemas desenvolvidos nesse modelo devem possuir um conjunto de características necessárias para alcançar este objetivo. As propriedades ACID consistem em um conjunto de características que visam preservar a

consistência e a durabilidade dos dados. A sigla é formada pelas iniciais das propriedades que abrange, sendo elas ***A****tomicity* (Atomicidade)*,* ***C****onsistency* (Consistência)*,* ***I****solation* (Isolamento) e ***D****urability* (Durabilidade). Segundo Chapple (2016), essas propriedades possuem as seguintes definições.

- **Atomicidade:** define que todas as operações devem ser atômicas, ou seja, não podem ser divididas em sub operações. Deste modo, a operação será executada completamente, ou nada será executado.
- **Consistência:** esta propriedade é responsável por garantir a integridade dos dados através de um conjunto de regras, evitando redundância, ambiguidade, entre outros problemas. Assim sendo, uma transação só é efetivada caso obedeça todas as regras definidas, do contrário, nenhuma modificação será armazenada.
- **Isolamento:** determina que toda e qualquer operação seja executada por completo sem interferência de outras operações. Entretanto, mais de uma operação pode ocorrer ao mesmo tempo sobre mesmo dado, desde que uma não interfira na outra.
- **Durabilidade:** garante a perpetuação dos dados armazenados até que alguma operação de atualização ou exclusão ocorra sobre o mesmo. Ou seja, o sistema deve se prevenir contra eventuais falhas que possam acarretar na corrupção dos dados.

Por outro lado, com o passar dos anos, o volume de dados cresceu de forma exorbitante, assim como sua complexidade. Logo, consistência não era mais a principal característica no armazenamento de dados, cedendo lugar para particionamento e disponibilidade dos mesmos

2.4.2 Propriedades BASE

Ao contrário das propriedades ACID que buscam a consistência dos dados, as propriedades BASE objetivam sua disponibilidade e particionamento através de mecanismos previamente citados. As propriedades representadas por esta sigla são ***B****asically* ***A****vailable* (basicamente disponível)*,* ***S****oft-State* (estado leve) e ***E****ventually*

Consistent (eventualmente consistente). De acordo com Chapple (2016), estas propriedades definem que:

- **Basicamente Disponível:** o sistema está basicamente disponível o tempo todo, não havendo restrições em operações simultâneas, independentemente de suas naturezas.
- **Estado leve:** o sistema não precisa ser consistente o tempo todo, possibilitando a realização de operações de escrita simultaneamente a outras operações.
- **Eventualmente consistente:** o sistema sempre retornará a versão atualizada dos dados, obtendo-se consistência enquanto não houver acesso simultâneo de escrita ao mesmo dado.

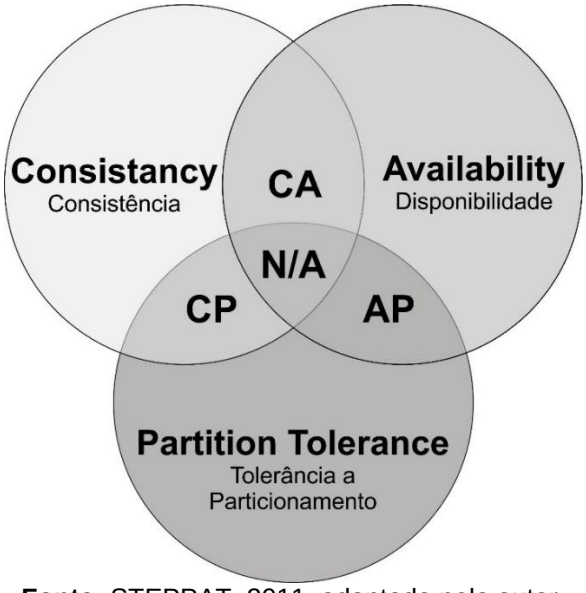
Para escolha de uma ferramenta de determinada categoria, deve-se primeiro analisar as necessidades do sistema no qual será utilizada, uma vez que cada categoria tem suas vantagens e desvantagens. A impossibilidade da existência de uma ferramenta que possua as vantagens de ambas as categorias é definida pelo teorema CAP.

2.4.3 Teorema CAP

Ao analisar as características obtidas através das propriedades BASE e ACID, Eric Brewer propôs o teorema CAP. Este teorema define que é impossível para um sistema distribuído prover disponibilidade, consistência e tolerância a particionamento ao mesmo tempo, sendo possível alcançar apenas duas destas características. (BREWER, 2000). A prova formal deste teorema foi realizada por Gilbert e Lynch (2002). A Figura 1 mostra as propriedades citadas e as combinações possíveis de serem obtidas.

De acordo com Gilbert e Lynch (2002), as propriedades consistência, disponibilidade e tolerância a particionamento garantem respectivamente que, as informações estejam atualizadas em todos os nós do sistema, todas as requisições recebidas por um nó funcional do sistema gere uma resposta, e que sejam capazes de operar de forma assíncrona caso a conexão entre os nós do sistema seja perdida.

Figura 1 – Propriedades teorema CAP.



Fonte: STEPPAT, 2011, adaptado pelo autor.

Deste modo, torna-se possível que um sistema seja apenas consistente e disponível, disponível e tolerante a particionamento, ou consistente e tolerante a particionamento. Isso ocorre devido a impossibilidade de atualização de dados em todos os nós de um sistema assíncrono, ou seja, caso haja falhas na conexão, o sistema não será capaz de garantir a consistência dos dados.

Neste caso, para que a consistência seja mantida, o sistema deverá suspender seu funcionamento, quebrando assim, a propriedade disponibilidade. Por outro lado, é possível que o sistema continue disponível mesmo com falhas de conexão entre os nós, porém, um estado de consistência só será possível após as falhas de conexão terem sido resolvidas. Outra solução é o não particionamento, o que torna possível que o sistema seja consistente e disponível.

Entretanto, com o passar dos anos, desenvolvedores e pesquisadores desenvolveram sistemas distribuídos com diferentes níveis de consistência e disponibilidade, capazes de optar por preservar uma dessas propriedades na presença de particionamento, tomando por base a operação solicitada, os dados acessados, ou até mesmo o usuário solicitante.

Após doze anos do lançamento do teorema, Eric Brewer publicou um novo trabalho intitulado “CAP Twelve Years Later: How the ‘Rules’ Have Changed” no qual ele faz uma reanálise da aplicação do teorema em sistemas distribuídos. Em seu trabalho, Brewer (2012) afirma que é possível que um sistema possua todas as três propriedades, porém, nunca em sua totalidade. Segundo o autor, a restrição descrita

pelo teorema CAP está relacionada a presença integral das três propriedades. Ou seja, é possível aplicar uma visão contínua destas propriedades, maximizando o potencial do sistema.

2.5 CONSIDERAÇÕES FINAIS

A partir do conhecimento teórico fornecido neste capítulo, tornou-se possível a realização do estudo de caso descrito neste trabalho, bem como a compreensão e análise dos resultados obtidos. Deste modo, o seguinte capítulo descreve com detalhes como o estudo de caso foi realizado.

3 ESTUDO DE CASO

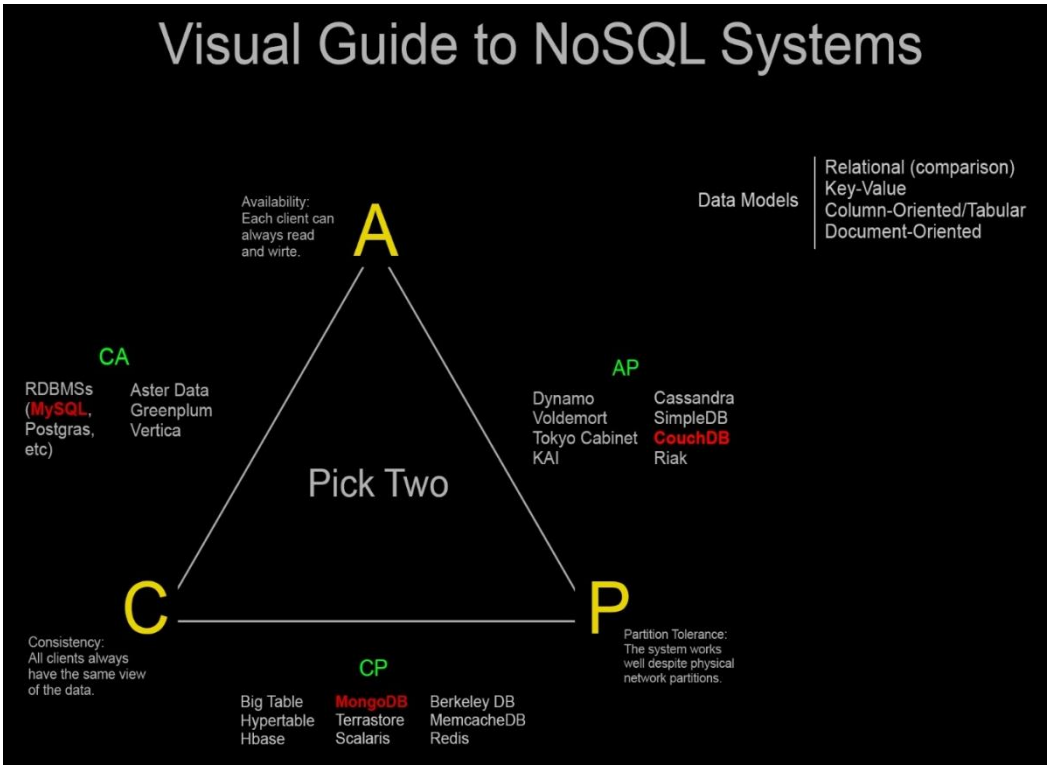
3.1 CONSIDERAÇÕES INICIAIS

Com base no referencial teórico estudado, foi desenvolvido o seguinte experimento, objetivando a obtenção de resultados capazes solucionar a problemática proposta. Assim sendo, o estudo de caso realizado é descrito nos tópicos a seguir.

3.2 FERRAMENTAS UTILIZADAS

Para a realização dos testes de performance foram escolhidas três ferramentas para realização das simulações. O principal critério de seleção foi o teorema CAP, uma vez que buscou-se ferramentas que apresentassem as possíveis combinações das propriedades descritas pelo teorema. A Figura 2 mostra a classificação de alguns SGBDs de acordo com o teorema.

Figura 2 – Classificação de SGBDs de acordo com o teorema CAP.



Fonte: HURST, 2010, adaptado pelo autor.

Deste modo, a ferramenta escolhida para representar o modelo relacional e a aresta CA foi o MySQL, que atualmente encontra-se na versão 5.7.12. Para representar a aresta AP foi escolhido o CouchDB que se encontra na versão 1.6.1. E por fim, para representar a aresta CP foi escolhido o MongoDB, que encontra-se na versão 3.2.5. A seguir serão descritas as ferramentas utilizadas no comparativo.

3.2.1 MySQL

O MySQL é um SGBD relacional capaz de operar em diversas plataformas, possui licença *open-source*, e é escrito em C++. Foi desenvolvido para ser completamente *multi-thread*, sendo capaz de fazer uso de múltiplos processadores, caso disponíveis. (MYSQL, 2016). O MySQL tem como mecanismo padrão de armazenamento o *InnoDB*, que adere aos princípios ACID.

Tratando-se dos tipos de dados, o MySQL suporta múltiplas categorias de dados SQL tais como: tipos numéricos, *strings*, data, tempo, tipos espaciais e até documentos JSON (MYSQL, 2016). Além da vasta gama de dados suportados, o MySQL também oferece várias funções de consulta, agregações e *joins* entre as tabelas. A conectividade entre programas clientes e o SGBD se dá através de conectores e APIs que possibilitam a execução de comandos a partir de diversos ambientes e linguagens de programação.

3.2.2 CouchDB

O CouchDB é um SGBD NoSQL de esquema livre orientado a documentos capaz de operar em diversas plataformas, porém com limitações. Possui licença *open-source* e é completamente escrito em Erlang[1] (APACHE, 2016b). O CouchDB provê uma API HTTP para manipulação dos dados e uma interface web administrativa chamada Futon.

Os dados são armazenados em documentos de até 4GB em formato JSON identificados por um atributo “_id”, e são capazes de armazenar diversos tipos de dados, incluindo listas e até documentos aninhados. Para lidar com o acesso

[1]Linguagem orientada a concorrência desenvolvida pela Ericsson para desenvolvimento de sistemas de tempo real de alta disponibilidade (Ericsson, 2016).

concorrente dos dados, o CouchDB utiliza o modelo de controle de concorrência multiversionado (MVCC) (OREND, 2010). Deste modo, os documentos salvos possuem um atributo especial “_rev” responsável pelo versionamento dos dados.

A consulta aos dados são realizadas através de *views*, que por sua vez, é um método que possibilita diferentes representações para os dados armazenados, independentemente da estrutura dos documentos salvos. *Views* são definidos através de funções *map-reduce* escritas em *JavaScript*.

3.2.3 MongoDB

Assim como o CouchDB, o MongoDB é um SGBD NoSQL *open-source*, de esquema livre orientado a documentos, capaz de operar em diversas plataformas, além de ser escrito em C++. (OREND, 2010). O MongoDB armazena os dados em documentos de até 16MB no formato JSON binário (BSON), e assim como no CouchDB, são identificados unicamente por um atributo “_id” e suportam diversos tipos de dados como listas e documentos aninhados.

O MongoDB não faz uso do modelo MVCC (OREND, 2010). Assim sendo, atualizações podem ocorrer em campos individuais, ao contrário do que ocorre no CouchDB em que um novo documento inteiro é armazenado.

Consultas tradicionais são realizadas através de *queries* que são expressas na forma de documentos JSON e enviadas para o SGBD na forma de BSON através de seu *driver*. Já agregações complexas podem ser realizadas através de uma variação do *map-reduce*.

Para a realização dos testes foi definido o seguinte ambiente de testes.

3.3 AMBIENTE DE TESTES

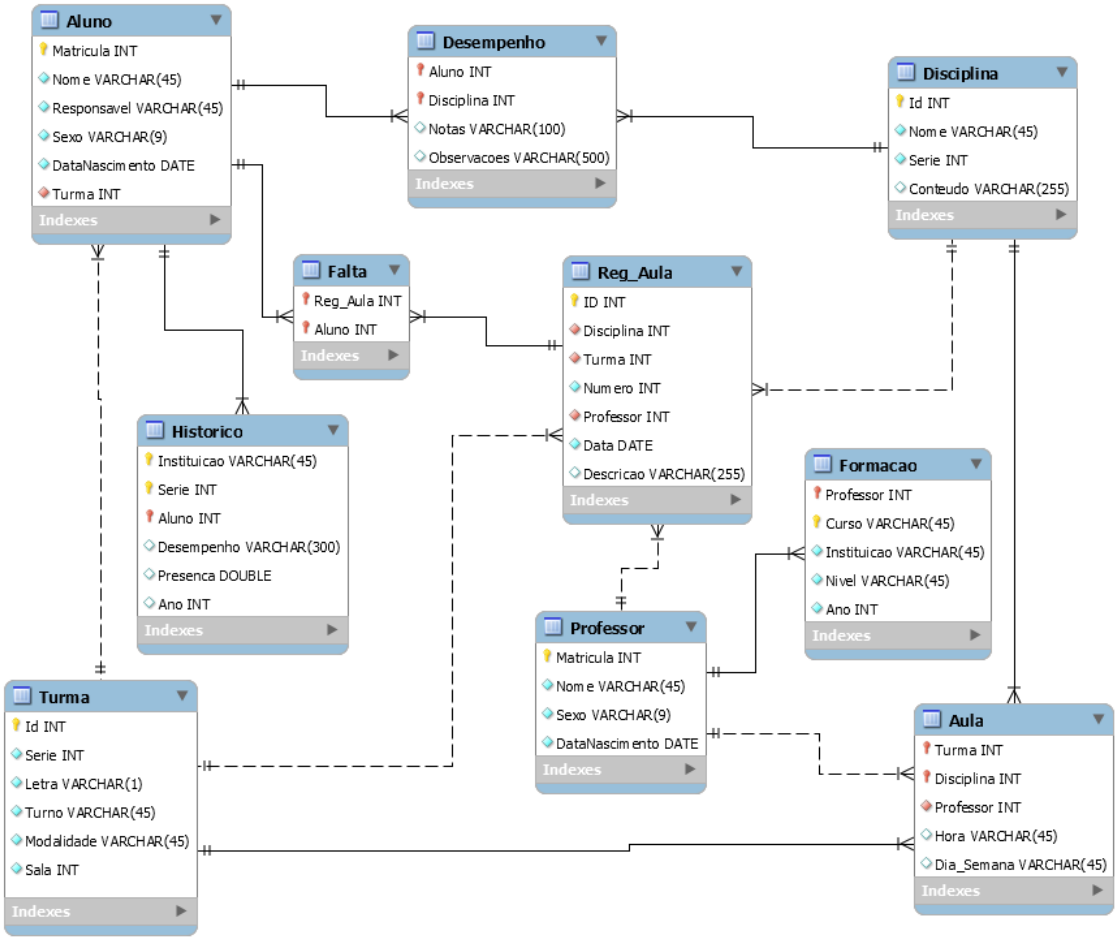
Para a realização dos testes, foi utilizado um computador com processador Intel Core I7-5500U CPU de 2.4 GHz com 8 GB de memória RAM e 500 GB de disco rígido com sistema operacional Ubuntu em sua versão 14.04 de 64bits (x64). Quanto as ferramentas, a versão utilizada do MySQL foi a 5.7.12, a do CouchDB foi a 1.6.1, e a do MongoDB foi a 3.2.5.

A implementação dos testes foi realizada na linguagem Java (JAVA, 2016) utilizando o JDK 8u77 juntamente com os *drivers* necessários para conexão com os

SGBDs. Para conexão com o MySQL foi utilizado o MySQL *Connector* Java 5.1.38 (MYSQL, 2016). Para conexão com o CouchDB foi utilizado o CouchDB4J 0.1.2, juntamente com suas dependências Apache commons[2] e JSON-lib[3] (APACHE, 2016a). Para conexão com o MongoDB foi utilizado o Mongo Java *Driver* 3.0.4 (MONGODB, 2016).

Referente ao modelo de dados, a Figura 3 mostra o modelo relacional da estrutura dos dados utilizada na realização dos testes. Este modelo pode ser entendido como a modelagem de um sistema acadêmico genérico e simplificado de nível fundamental e médio, para fins de utilização no experimento descrito neste trabalho.

Figura 3 – Modelo Relacional dos Dados.



Fonte: Elaborado pelo autor.

[2] Bibliotecas: apache-commons-lang; commons-beanutils-1.8.3; commons-codec-1.9; commons-collections-3.2.1; commons-httpclient-3.1; commons-logging-1.2; httpclient-4.4.1; httpcore-4.4.1.
[3] Bibliotecas: ezmorph-1.0.6; json-lib-2.4-jdk15.

Também foi definida a estrutura dos documentos a serem armazenados nos demais SGBDs de modo que nenhuma informação fosse perdida. Mesmo que os documentos possuam esquema livre, não significa que uma modelagem eficiente seja desnecessária, uma vez que uma modelagem mal feita pode acarretar na alta complexidade e inconsistência dos dados.

A modelagem de um banco de dados orientado a documentos consiste na agregação dos dados que são utilizados em conjunto, deste modo, evitando múltiplas consultas e junções entre entidades diferentes. Entretanto, há situações em que a agregação não é a melhor opção, sendo necessário criar relações entre documentos. Tendo isso em mente, CrawCour (2016) oferece algumas sugestões de quando realizar agregação dos dados ou referenciar os mesmo. Contudo, afirma que a melhor modelagem deve levar em consideração a lógica do aplicativo.

Segundo CrawCour (2016), os dados devem ser agregados quando apresentarem relações de um para um e um para poucos, tendo em mente que os documentos possuem tamanho limitado. Além disso, documentos muito grandes prejudicam o desempenho do banco, já que todo o documento é enviado para o cliente quando requisitado. Assim sendo, entidades que apresentem relações de um para muitos e muitos para muitos devem ser referenciadas, evitando assim, o crescimento exagerado.

Em casos onde a mesma informação é referenciada por mais de um registro, deve-se considerar a frequência em que esta informação é atualizada. Dados frequentemente atualizados geram uma grande carga de trabalho quando agregados a mais de um documento, pois é necessário atualizar todos os documentos que os contém. Portanto, dados frequentemente atualizados devem ser referenciados ao invés de agregados (CRAWCOUR, 2016).

Levando em consideração as sugestões propostas por CrawCour, o modelo relacional previamente definido foi adaptado para dois documentos: Disciplina e Turma (Apêndice A). A grande maioria dos dados foram agregados ao documento turma, pois são os dados que geralmente são unidos durante consultas. O documento Disciplina armazena o conteúdo programado da disciplina, que dificilmente é consultado juntamente com os demais dados. A seguir, a estrutura do documento disciplina.

Quadro 1 – Estrutura do documento disciplina.

```
Disciplina {
    _id          ← Identificador
    Type         ← Variável responsável pela identificação da tipagem do documento
    Nome         ← Nome da disciplina
    Série        ← Série a qual a disciplina é aplicada
    Conteúdo     ← Conteúdo programado para ser ministrado
}
```

Fonte: Elaborado pelo autor.

Para medir a performance dos SGBDs foram realizados testes de escrita, consulta simples e consulta complexa. A fim de obter dados referentes a diferentes volumes de dados processados, os testes foram executados em *loops* de 1, 10, 100 e 1000 iterações. Para obtenção de resultados mais genéricos, cada teste foi executado três vezes para verificação da variação de tempo nas execuções.

Deste modo, os testes de escrita consistem na inserção de informações em todos os campos de dados, tanto das tabelas quanto dos documentos. A cada iteração foi inserido no banco de dados um professor com uma formação, uma disciplina, uma turma, trinta alunos pertencentes a esta turma, uma aula, um registro de aula, um registro de histórico e desempenho para cada aluno inserido, e um registro de falta para o vigésimo aluno da turma.

Os testes de consulta simples consistem na busca pela matrícula e nome de todos os alunos cadastrados juntamente com o id da turma, a série e a letra. Todos os testes de busca simples retornaram um total de 99990 (noventa e nove mil, novecentos e noventa) resultados.

A consulta complexa consiste na busca pelo nome, notas e respectivas disciplinas, id de turma, série e letra de todos os alunos não faltosos que possuem pelo menos um professor com formação de nível superior. Todos os testes de busca complexa retornaram 96657 (noventa e seis mil, seiscentos e cinquenta e sete) resultados.

3.4 SIMULAÇÕES DE ESCRITA E CONSULTA

Para a realização dos testes, foi desenvolvido um *software* multithread em três diferentes versões, uma para cada SGBD. Uma das *threads* é incumbida de executar

as simulações enquanto a outra monitora o progresso dos testes. A medição do tempo gasto para execução dos testes foi feita através do método ***System.nanoTime()*** da biblioteca padrão java.lang.

Deste modo, para maior precisão nas medições, os dados inseridos, assim como as *queries* das consultas, foram previamente preparados para que fosse possível mensurar apenas o tempo gasto pelos *drivers* das ferramentas para executar as operações.

A estrutura básica das três versões do software consiste na definição de uma classe que possui métodos necessários para execução e monitoramento dos testes. Os construtores dessas classes recebem um valor inteiro responsável pela inicialização das variáveis de controle. Estas, por sua vez, são responsáveis pela automatização na definição de identificadores, tanto de tabelas quanto de documentos, bem como nomes, que são incrementados numericamente (exemplo: “Professor 1”, “Professor 2”, ...).

Todas as versões possuem um método chamado **defineThread()**. Este método é encarregado de criar a *Thread* que executa as simulações em *loop*. Para isso, este recebe um parâmetro indicando qual operação deve ser executada: “inserir”, “buscaSimples” ou “buscaComplexa”. Este método também contabiliza o tempo gasto nas operações (retornado pelos respectivos métodos a cada iteração) e os armazena em um arquivo intitulado “log.txt”.

O método **teste()**, também presente em todas as versões, recebe como parâmetros uma ação e um total de repetições. Este método é responsável por chamar o método **defineThread()**, por carregar o *driver* da ferramenta quando necessário, por iniciar a conexão com o banco de dados, por criar a interface de manipulação, por iniciar a *Thread* criada anteriormente, e, por fim, monitorar o progresso dos testes executados.

O método ***main()*** cria uma instância das classes, passando como parâmetro o total de registros presentes no banco de dados e, então, executa o método teste dentro de um *loop* de três repetições. Os dados passados tanto para o construtor quanto para o método **teste()** são informados diretamente no código a cada execução. A seguir, serão descritos trechos de códigos específicos do uso de cada uma das ferramentas anteriormente mencionadas.

3.4.1 Simulações MySQL

As simulações executadas no MySQL foram executadas através do código presente no Apêndice C. As variáveis de controle inicializadas pelo construtor são contDisciplina, contProfessor, contTurma, contRegistro e contAluno. Cada variável é responsável pela definição das chaves primárias de suas respectivas tabelas e pela automatização nos nomes necessários.

O método **inserir()** (linhas 89 à 208) é responsável pelas simulações de escrita. O driver utilizado para conexão recebe uma *string* que, por sua vez, é uma *query* SQL. Assim sendo, as *query* SQL foram elaboradas e armazenadas em variáveis para só então serem enviadas para o SGBD através do *driver* (linhas 99, 109, 120, 132, 143, 156, 172, 183, 195 e 204). O trecho de Código 1 foi utilizado para a inserção de um registro na tabela disciplina.

Código 1 – Inserção de registro disciplina no MySQL.

```
 93   String disciplina="insert into disciplina values (";
 94   disciplina+=++contDisciplina;
 95   disciplina+=",\"disciplina "+contDisciplina+"\",";
 96   disciplina+="2";
 97   disciplina+=",\"conteúdos a serem estudados na disciplina\")";
 98   ini=System.nanoTime();
 99   statement.execute(disciplina);
100   fim=System.nanoTime();
101   soma+=fim-ini;
```

Fonte: Elaborado pelo autor.

As primeiras linha (linhas 93 à 97) montam a *query* com os dados de disciplina. A linha 98 captura a marcação atual da máquina virtual Java em nano segundos e armazena na variável ini. Na linha 99, o driver da ferramenta envia a *query* formada para o SGBD.

A linha 100 captura uma nova marcação de tempo que é armazenada na variável fim. Por fim, na linha 101, é calculada a diferença entre as variáveis fim e ini, a fim de obter o tempo gasto durante a operação. O resultado obtido é então contabilizado na variável soma.

O código utilizado para a consulta simples possui estrutura apresentada no trecho de Código 2.

Código 2 – Consulta simples no MySQL.

```
210 private long buscaSimples() throws SQLException{
211     long ini;
212     long fim;
213     ini=System.nanoTime();
214     statement.executeQuery("select a.matricula, a.nome, t.id, t.serie,"
215        + "t. letra from aluno a inner join turma t on a.turma = t.id");
216     fim=System.nanoTime();
217     return fim-ini;
218 }
```

Fonte: Elaborado pelo autor.

O método **buscaSimples()** (linhas 210 à 218) possui apenas a execução da *query* de consulta, uma vez que não é necessária a modificação dos dados presentes na *query*. A consulta foi realizada selecionando os dados desejados e realizando um *join* entre as tabelas necessárias.

A execução da consulta complexa foi realizada através do trecho de Código 3.

Código 3 – Consulta complexa no MySQL.

```
220 private long buscaComplexa() throws SQLException{
221   long ini;
222   long fim;
223   ini=System.nanoTime();
224   statement.executeQuery("select t.serie, t.letra, al.nome, de.notas, "
225           + "di.nome from professor p inner join formacao f on "
226           + "p.matricula = f.professor inner join aula au on "
227           + "p.matricula = au.professor inner join disciplina di on "
228           + "au.disciplina = di.id inner join desempenho de on "
229           + "di.id = de.disciplina inner join aluno al on "
230           + "de.aluno = al.matricula left join falta fa on "
231           + "al.matricula = fa.aluno inner join turma t on "
232           + "al.turma = t.id where f.nivel = \"superior\" and "
233           + "fa.aluno is null");
234   fim=System.nanoTime();
235   return fim-ini;
236   }
```

Fonte: Elaborado pelo autor.

O método **buscaComplexa()** (linhas 220 à 236) também possui apenas a execução da *query* de consulta. A busca complexa foi realizada através de *joins* entre quase todas as tabelas do modelo e na cláusula *where* foi definido o critério de seleção dos alunos. Além disso, foi definido no início da *query* os valores que se deseja obter.

3.4.2 Simulações CouchDB

O código utilizado para a realização das simulações no CouchDB está presente no Apêndice D. As variáveis de controle iniciadas pelo construtor são: contDocumentos, responsável pela identificação única dos documentos armazenados no banco, contAluno, responsável pela identificação únicas dos alunos, e contIteracao, utilizada na automatização dos nomes necessários.

O driver utilizado para a conexão com o banco recebe como parâmetro um documento no formato JSON e o envia para ser armazenado no banco de dados. O trecho de Código 4 a seguir é responsável pela definição e inserção do documento disciplina.

Código 4 – Inserção de registro disciplina no CouchDB.

```
 96   Document disciplina= new Document();
 97   disciplina.setId(++contDocumentos+"");
 98   disciplina.put("type","disciplina");
 99   disciplina.put("nome", "disciplina "+contIteracao);
100   disciplina.put("serie", 2);
101   disciplina.put("conteudo","conteúdos a serem estudados na disciplina");

178   db.saveDocument(disciplina);
```

Fonte: Elaborado pelo autor.

A primeira linha (linha 96) cria o documento. Na segunda linha (linha 97) é definido o id do documento. Caso não seja definido um id, o CouchDB o define automaticamente, entretanto, documentos aninhados não recebem ids automaticamente, já que estes não são obrigatórios. As linhas 98 à 101 inserem os demais dados a serem armazenados neste documento. E, por fim, na linha 178 o documento é enviado ao banco através do *driver*.

Como dito anteriormente, as consultas no CouchDB são realizadas através de *views* que são definidos através de funções *map-reduce* e armazenadas no banco. O código das funções *map* e da função *reduce* utilizadas neste experimento estão presentes no Apêndice B.

Para a consulta simples, foi necessária apenas a implementação da função *map*. A função *map* implementada no CouchDB recebe como parâmetro um documento e verifica se o mesmo é do tipo turma. Em caso positivo, é emitido um *array* contendo matrícula e nome dos alunos como chave. Como valor, é emitido um

documento contendo o id, a série e a letra da turma. O código da função *map* implementada é mostrado a seguir no trecho de Código 5.

Código 5 – Função *map* da consulta simples CouchDB.

```
1 function(doc){
2  if(doc.type=="turma"){
3   for(var aluno in doc.alunos){
4    emit({aluno:doc.alunos[aluno].matricula,nome:doc.alunos[aluno].nome},
     {turma:doc._id,serie:doc.serie,letra:doc.letra});
5   }
6  }
7 }
```

Fonte: Elaborado pelo autor.

Para a consulta complexa foi necessária a definição de uma *view* utilizando tanto a função *map* quanto a função *reduce*. A função *map* verifica inicialmente se o documento recebido como parâmetro é do tipo turma. Em caso positivo, verifica se algum professor dessa turma possui nível superior para só então verificar quais alunos não possuem falta em nenhuma disciplina. Por fim, para todos os alunos que não possuem faltas são emitidos o id, série, letra e nome do aluno como chave e o documento completo do aluno como valor.

Para a sintetização dos dados emitidos foi definida a função *reduce,* que recebe como parâmetros uma chave e uma lista de valores. O retorno produzido por esta função consiste em um *array* com os nomes e as notas do aluno. A consulta destes dados se dá através de uma requisição HTTP. Esta consulta é mostrada no trecho de Código 6.

Código 6 – Consulta simples e complexa no CouchDB.

```
187   HttpClient httpclient = HttpClientBuilder.create().build();
188   HttpGet get = new HttpGet("http://localhost:5984/escola/_design/"
189         + "Consulta/_view/simples");
194   httpclient.execute(get);


199   HttpClient httpclient = HttpClientBuilder.create().build();
200   HttpGet get = new HttpGet("http://localhost:5984/escola/_design/"
201         + "Consulta/_view/complexa?group_level=1");
205   httpclient.execute(get);
```

Fonte: Elaborado pelo autor.

Para a realização da consulta foi inicialmente criado um objeto do tipo cliente HTTP, responsável pela execução das requisições. Em seguida, foi criado um objeto

do tipo HTTP *get*, que recebe o endereço dos dados a serem executados. Por fim, o método ***execute***() do objeto cliente HTTP é chamado e recebe como parâmetro o objeto do tipo HTTP *get*.

A única diferença no acesso aos dados da consulta simples e da consulta complexa é o endereço fornecido para a criação do objeto HTTP *get*. Este endereço corresponde ao endereço da *view* no banco de dados.

3.4.3 Simulações MongoDB

O código utilizado para a realização das simulações no MongoDB está presente no Apêndice E. As variáveis de controle iniciadas pelo construtor são as mesmas utilizadas no CouchDB, sendo elas: contDocumentos, responsável pela identificação única dos documentos armazenados no banco, contAluno, responsável pela identificação únicas dos alunos, e contIteracao, utilizada na automatização dos nomes necessários.

O driver utilizado para a conexão com o banco recebe como parâmetro um documento no formato BSON e o envia para ser armazenado no banco de dados. A definição do documento a ser enviado para o MongoDB ocorreu de forma diferente do CouchDB. O trecho de Código 7 é responsável pela definição e inserção do documento disciplina no MongoDB.

Código 7 – Inserção de registro disciplina no MongoDB.

```
 91   Document disciplina= new Document("_id",++contDocumentos)
 92         .append("type","disciplina")
 93         .append("nome","disciplina "+contIteracao)
 94         .append("serie", 2)
 95         .append("conteudo","conteúdos a serem estudados na disciplina");

164   colecao.insertOne(disciplina);
```

Fonte: Elaborado pelo autor.

Na primeira linha (linha 91), o documento é criado e recebe como atributo um par chave-valor, que neste caso, foi o "_id". A inserção dos demais dados no documento ocorreu através do método ***append*()** (linhas 92 à 95) que retorna um novo documento com os dados inseridos. Por fim, o documento é enviado ao banco através do driver (linha 164).

As queries de consulta consultas no MongoDB são expressas também na forma de documentos. Tanto no método **buscaSimples()** quanto no método **buscaComplexa()** foi utilizado o método ***aggregate*()** que recebe uma lista de documentos com as especificações dos dados a serem retornados. O trecho de Código 8 refere-se a busca simples.

Código 8 - Consulta simples no MongoDB.

```
171   List query=new ArrayList<Document>();
172   query.add(new Document("$unwind","$alunos"));
173
174   Document match=new Document("type","turma");
175   query.add(new Document("$match",match));
176
177   Document push=new Document("turma","$_id")
178           .append("serie","$serie")
179           .append("letra","$letra");
180
181   Document group=new Document(" id","$alunos.matricula")
182           .append("nome", new Document("$push","$alunos.nome"))
183           .append("turma", new Document ("$push",push));
184   query.add(new Document("$group",group));
189   colecao.aggregate(query);
```

Fonte: Elaborado pelo autor.

Inicialmente, é criada uma lista que armazenará os dados de consulta (linha 171). Em seguida, são inseridos documentos com especificações dos dados a serem retornados pela busca linhas (172 à 184). Por fim, o método ***aggregate*()** é chamado e passado como parâmetro a *query* recém criada. O método **buscaComplexa()** (linhas 194 à 244) possui a mesma estrutura, diferenciando apenas nas especificações que são inseridas na *query*.

3.5 CONSIDERAÇÕES FINAIS

Este capitulo objetivou descrever como se deu a realização do estudo de caso proposto. Deste modo, foi possível obter detalhes de implementação que possibilitam seu entendimento e sua replicação, caso necessário. Além disso, também foi possível obter dados que permitem analisar como as ferramentas trabalham no cenário definido. A análise desses resultados está presente no capítulo seguinte.

4 RESULTADOS

4.1 CONSIDERAÇÕES INICIAIS

A partir das simulações realizadas foi possível a obtenção de dados referentes a performance de cada uma das ferramentas. Deste modo, este capítulo objetiva apresentar e analisar o resultados obtidos com o experimento realizado. Os dados a seguir estão apresentados em milissegundos com três casas decimais, uma vez que algumas operações foram executadas em microssegundos.

4.2 MYSQL

O Quadro 2 apresenta os resultados obtidos através dos testes realizados com o MySQL.

Quadro 2 – Resultados MySQL.

	1 Iteração	10 Iterações	100 Iterações	1000 Iterações
Inserir	4943,613ms	51353,553ms	520478,623ms	5282916,389ms
Busca Simples	152,259ms	976,067ms	9184,550ms	88973,140ms
Busca Complexa	427,206ms	3536,860ms	34548,704ms	345654,451ms

Fonte: Elaborado pelo autor.

Os testes de inserção apresentaram um acréscimo de tempo com o aumento da carga de trabalho. Foram necessários quase 5 segundos para realização de uma iteração do teste de inserção, que consiste na inserção de 97 registros[4] entre as tabelas. Logicamente, 10 iterações do mesmo processo deveriam ser executadas em quase 50 segundos, porém, o SGBD necessitou de pouco mais de 51 segundos, tendo um gasto extra de aproximadamente 3,8% em cada iteração.

[4] 30 registros de alunos, 30 registros de desempenho, 30 registros de histórico, 1 registro de falta, 1 registro de turma, 1 registro de aula, 1 registro de disciplina, 1 registro de reg_aula, 1 registro de professor, e 1 registro de formação.

A execução de 100 iterações teve duração de quase 9 minutos, tempo este que representa um gasto adicional de 5,28% por iteração e 1,35% a mais em cada conjunto de 10 iterações. A execução de 1000 iterações ocorreu em um período de pouco mais de 1h18min. Este valor em comparação com os conjuntos de 1, 10 e 100 iterações representa respectivamente um acréscimo de 6,86%; 2,87% e 1,5% no tempo gasto para execução da mesma operação.

Os testes de consulta obtiveram um melhor desempenho, necessitando de pouco mais de 152 milissegundos para a execução de uma busca. Como base no teste de inserção, a execução do mesmo processo 10 vezes deveria resultar em um gasto maior de tempo por iteração, entretanto, foi necessário menos de 1 segundo para execução repetida 10 vezes, o que representa uma queda de 35,89% do tempo gasto por operação.

Para a execução do mesmo processo 100 vezes foi necessário um pouco mais de 9 segundos. Este tempo representa uma redução de 39,68% do tempo gasto por operação e 5,9% a cada conjunto de 10 iterações. A execução de 1000 consultas simples necessitou de quase um minuto e meio para seu término. Este tempo corresponde a uma redução de 41,56% referente a 1 iteração, 8,84% referente a 10 iterações, e, por fim, 3,13% referente a 100 iterações.

Como esperado, a buscas complexas levaram mais tempo para serem finalizadas do que buscas simples, sendo necessário quase meio segundo para execução de 1 iteração. Da mesma forma que as buscas simples, as buscas complexas também obtiveram uma redução do tempo gasto de acordo com o aumento no número de iterações, com exceção da relação entre 1000 e 100 iterações, onde houve um acréscimo de 0,05% no tempo gasto.

A execução de 10 iterações do processo de busca complexa obteve uma queda de 17,21% em relação a 1 iteração do mesmo processo. A execução de 100 iterações obteve redução de 19,13% referente a 1 iteração e 2,32% referente a 10 iterações. A execução de 1000 iterações, por sua vez, obteve um redução de 19,09% referente a 1 iteração e 2,27% referente a 10 iterações.

4.3 COUCHDB

O Quadro 3 apresenta os resultados obtidos a partir dos teste executados com a ferramenta CouchDB.

Quadro 3 – Resultados CouchDB.

	1 Iteração	10 Iterações	100 Iterações	1000 Iterações
Inserir	78,188ms	877,228ms	9442,379ms	93866,004ms
Busca Simples	11,571ms	40,909ms	207,110ms	1200,577ms
Busca Complexa	29,834ms	69,751ms	233,853ms	1271,354ms

Fonte: Elaborado pelo autor.

Os resultados obtidos pelo CouchDB são bem mais satisfatórios em comparação com o MySQL, uma vez que necessitou de pouco mais de 78 milissegundos para inserir dois documentos com a mesma quantidade de dados. Assim como o MySQL, o CouchDB também teve um gasto extra de tempo nos testes de inserção com o aumento do número de iterações, com exceção da relação 1000 para 100, onde houve uma redução de 0,59% do tempo gasto.

A execução de 10 iterações ocorreu em um período de pouco mais de 877 milissegundos, que representa cerca de 12,19% a mais em cada iteração. Para a execução de 100 iterações foi necessário um pouco mais de 9 segundos, que corresponde a um aumento de 20,76% referente a uma iteração e 7,64% referente ao conjunto de 10 operações. A execução de 1000 iterações obteve um aumento de 20,05% em relação a uma iteração e 7% em relação a 10 iterações.

As buscas simples, por outro lado, obtiveram redução do tempo gasto com o aumento do número de iterações, onde 1 iteração necessitou de pouco mais de 11 milissegundos para ser finalizada enquanto 10 iterações necessitaram de quase 41 milissegundos, representando 64,65% a menos do tempo gasto.

Para execução de 100 iterações houve uma redução de 82,1% em relação a 1 iteração do mesmo processo e 49,38% em relação a um conjunto de 10 iterações. Para a execução de 1000 iterações o SGBD necessitou de apenas 1,2 segundo, o que representa 89,62% a menos em relação a 1 iteração, 70,65% em relação a 10 iterações e 42,38% em relação a 100 iterações.

As buscas complexas necessitaram mais tempo para serem executadas do que as buscas simples, porém, também obtiveram ótimo desempenho. A execução

de 1 iteração ocorreu em quase 30 milissegundos, enquanto 10 iterações ocorreram em quase 70 milissegundos, apresentando uma redução de 76,62% do tempo gasto por iteração.

Para execução de 100 iterações foram gastos quase 234 milissegundos, esse total corresponde a uma queda de 92,16% do tempo gasto por iteração e 66,47 do tempo gasto a cada conjunto de 10 iterações. A execução de 1000 buscas complexas obteve um queda ainda maior do tempo gasto, apresentando uma redução de 95,74% por iteração, 81,77% em relação a um conjunto de 10 iterações, e, por fim, 45,63% em relação em relação a um conjunto de 100 iterações.

4.4 MONGODB

O Quadro 4 apresenta os resultados obtidos através da execução de testes no MongoDB.

Quadro 4 – Resultados MongoDB.

	1 Iteração	10 Iterações	100 Iterações	1000 Iterações
Inserir	31,512ms	106,362ms	343,180ms	1039,820ms
Busca Simples	0,154ms	0,182ms	0,261ms	0,420ms
Busca Complexa	0,179ms	0,186ms	0,271ms	0,587ms

Fonte: Elaborado pelo autor.

O MongoDB obteve resultados ainda mais satisfatórios, sendo necessário pouco mais de 31 milissegundos para a execução de uma iteração do processo de inserção. Além disso, foi o único que obteve melhor desempenho no teste de inserção com o aumento do número de iterações. A execução de 10 iterações ocorreu em pouco mais de 106 milissegundos, que representa uma redução de 66,25% do tempo gasto em relação a 1 iteração do mesmo processo.

A execução de 100 iterações obteve uma redução ainda maior, sendo realizada em pouco mais de 343 milissegundos, este tempo corresponde a uma redução de 89,11% por iteração e de 67,73% em relação ao tempo obtido com 10

iterações. A execução de 1000 iterações o correu em pouco mais de 1 segundo, obtendo uma redução de 96,7% por iteração, 90,22% em relação a 10 iterações e 69,7% em relação ao tempo obtido com 100 iterações.

Os testes de busca simples foram os que obtiveram os melhores resultados. Uma busca simples necessitou de apenas 0,154 milissegundo para ser realizada. A execução de 10 iterações ocorreu em 0,182 milissegundo, o que representa 88,18% a menos no tempo gasto por iteração.

A execução de 100 iterações obteve uma redução de 98,30% por iteração e 85,66% em relação a execução de 10 iterações. A execução de 1000 iterações foi realizada em 0,420 milissegundo. Este valor representa uma redução de 99,73% do tempo gasto por iteração, 97,69% em relação a 10 iterações e 83,91% em relação a execução de 100 iterações.

As buscas complexas também obtiveram excelentes resultados, onde uma iteração ocorreu em 0,179 milissegundo. A execução de 10 iterações ocorreu em 0,186 milissegundo, que representa uma redução de 89,61% por iteração. 100 iterações obtiveram uma redução de 98,49% por iteração e 85,43% em relação a 10 iterações. A execução de 1000 iterações obteve uma redução surpreendente de 99,67% por iteração, 96,84% em relação a 10 iterações e 78,34% em relação ao tempo gasto com 100 iterações.

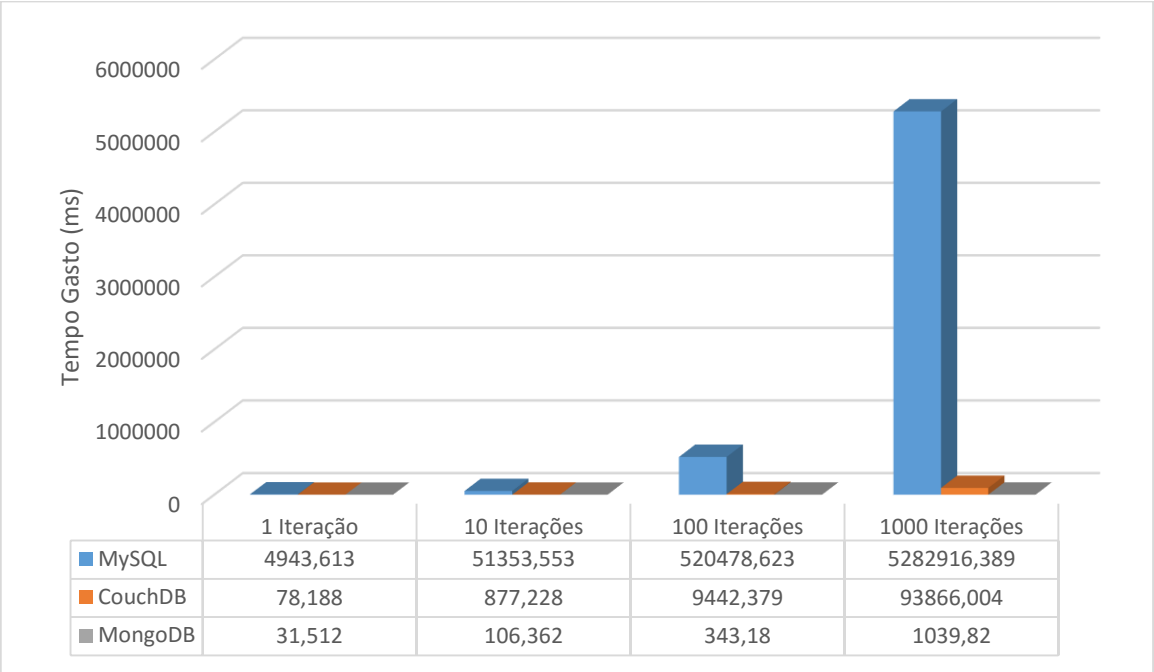
4.5 COMPARAÇÃO ENTRE AS FERRAMENTAS

A diferença entre os resultados obtidos pelas ferramentas é imensa, sendo o MySQL a ferramenta que apresentou pior desempenho, necessitando de pelo menos 10 vezes mais tempo que as demais ferramentas para execução da mesma operação. Por outro lado, o MongoDB apresentou o melhor desempenho, necessitando de menos da metade do tempo utilizado pelas demais ferramentas.

Na maioria dos casos, com o aumento do número de iterações cresce também a diferença entre os resultados apresentados por todas as ferramentas analisadas neste comparativo, havendo apenas poucas exceções.

Para melhor visualização da diferença entre os resultados obtidos foram criados gráficos que comparam os resultados de cada ferramenta. Os gráficos foram divididos por operação sendo o Gráfico 2 responsável por apresentar os dados obtidos através dos testes de inserção.

Gráfico 2 – Testes de inserção.



	1 Iteração	10 Iterações	100 Iterações	1000 Iterações
MySQL	4943,613	51353,553	520478,623	5282916,389
CouchDB	78,188	877,228	9442,379	93866,004
MongoDB	31,512	106,362	343,18	1039,82

Fonte: Elaborado pelo autor.

Na execução de uma iteração do teste de inserção o CouchDB obteve uma redução de 98,42% em comparação com o MySQL, enquanto o MongoDB obteve 99,36%. Já em comparação com o CouchDB, o MongoDB obteve uma redução de 59,7%. Com o aumento no número de iterações o CouchDB apresentou reduções no tempo gasto de 98,29%, 98,18%, e 98,22% respectivamente para 10, 100 e 1000 iterações em comparação com o MySQL.

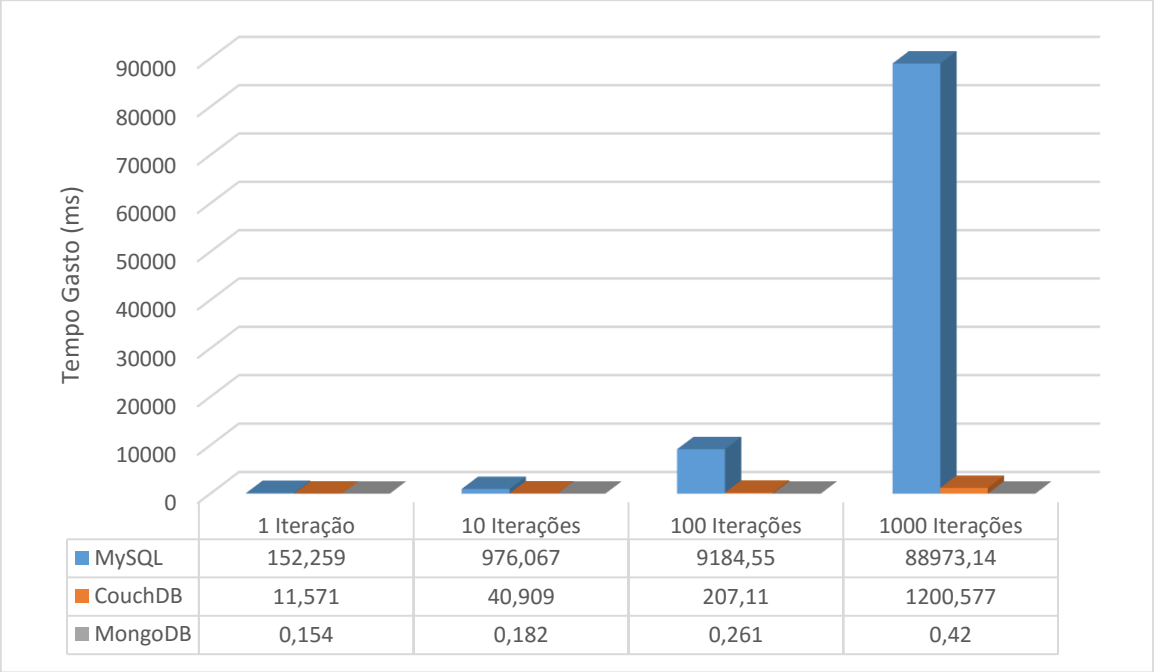
Os resultados obtidos pelo MongoDB quando comparados com os obtidos pelo MySQL apresentaram reduções de 99,79%, 99,93% e 99,98% respectivamente para a execução de 10, 100 e 1000 iterações. Em comparação com o CouchDB, os Resultados obtidos pelo MongoDB apresentaram reduções de 87,88%, 96,36% e 98,89% respectivamente na execução de 10, 100 e 1000 iterações.

O Gráfico 3 mostra os dados obtidos através dos testes de consulta simples. Através dele é possível notar também uma grande diferença nos resultados obtidos. Ao se comparar os resultados obtidos pelo CouchDB com o resultados obtidos pelo MySQL notasse uma diferença quase tão grande quanto a apresentada nos testes de inserção, obtendo reduções de 92,4%, 95,8%,97,74% e 98,65% respectivamente para a execução de 1, 10, 100 e 1000 iterações.

Por outro lado, ao se comparar os resultados obtidos pelo MongoDB com os obtidos pelo MySQL notasse que a diferença obtida foi ainda maior que os obtidos nos testes de inserção, apresentando reduções de 99,9%, 99,98%, 99,997% e 99,9995% respectivamente para a execução de 1, 10, 100 e 1000 iterações.

Na realização dos testes de consultas simples, a diferença entre os resultados obtidos pelo CouchDB e o MongoDB foi bem maior que a apresentada nos testes de inserção, atingindo reduções de 98,76%, 99,56%, 99,87% 4 99,96% respectivamente na execução de 1, 10, 100 e 1000 iterações.

Gráfico 3 – Testes de consulta simples.



	1 Iteração	10 Iterações	100 Iterações	1000 Iterações
MySQL	152,259	976,067	9184,55	88973,14
CouchDB	11,571	40,909	207,11	1200,577
MongoDB	0,154	0,182	0,261	0,42

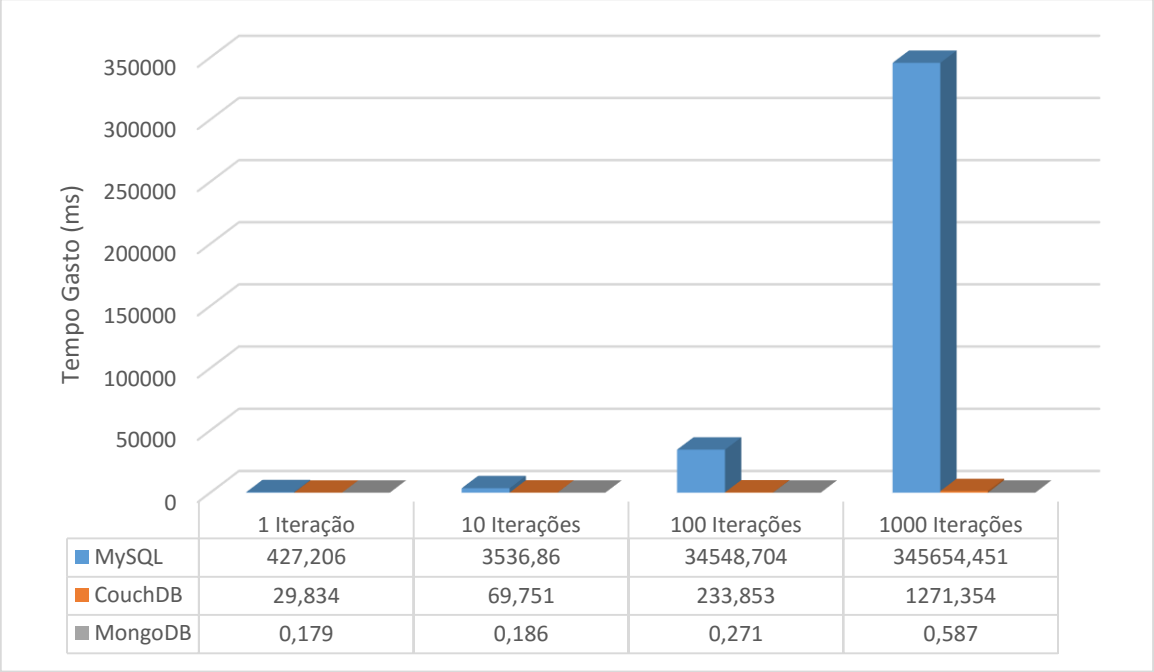
Fonte: Elaborado pelo autor.

O Gráfico 4 apresenta os dados obtidos pelas ferramentas através dos testes de consulta complexa. Assim sendo, ao se comparar os resultados obtidos pelo CouchDB com os obtidos pelo MySQL nota-se reduções de 93,02%, 98,03%, 99,32% e 99,63% respectivamente nas execuções de 1, 10, 100 e 1000 iterações da mesma operação.

A comparação dos resultados obtidos pelo MongoDB com os resultados obtidos pelo MySQL nos testes de consulta complexa apresentaram a maior disparidade dentre todas as demais comparações, apresentando reduções de 99,96%, 99,994%, 99,9992% e 99,9998% respectivamente na execução de 1, 10, 100 e 1000 iterações.

A comparação entre os resultados obtidos pelo MongoDB e o CouchDB também apresentou grande disparidade, atingindo reduções de 99,4% na execução de 1 iteração, 99,73% na execução de 10 iterações, 99,88% na execução de 100 iterações e 99,95% na execução de 1000 iterações

Gráfico 4 – Testes de consulta complexa.



	1 Iteração	10 Iterações	100 Iterações	1000 Iterações
MySQL	427,206	3536,86	34548,704	345654,451
CouchDB	29,834	69,751	233,853	1271,354
MongoDB	0,179	0,186	0,271	0,587

Fonte: Elaborado pelo autor.

Com estes resultados, fica evidente que quando o critério de seleção de qual SGBD adotar no desenvolvimento de um sistema é a velocidade em que as operações são executadas, a melhor opção é o MongoDB. Porém, este não deve ser o único critério, uma vez que também é necessário atentar para questões como modelo de dados adotado, como o sistema se adequa ao teorema CAP, entre outras.

Como dito anteriormente, o MySQL foi desenvolvido para trabalhar de forma centralizada, não havendo, portanto, particionamento de rede, tornando-se capaz de prover grande disponibilidade de forma consistente. O CouchDB e o MongoDB, por outro lado, foram desenvolvidos objetivando trabalhar de forma distribuída, estando sujeitos a particionamento de rede. A diferença entre estes sistemas é que na presença do particionamento o MongoDB preza por consistência enquanto o CouchDB por disponibilidade.

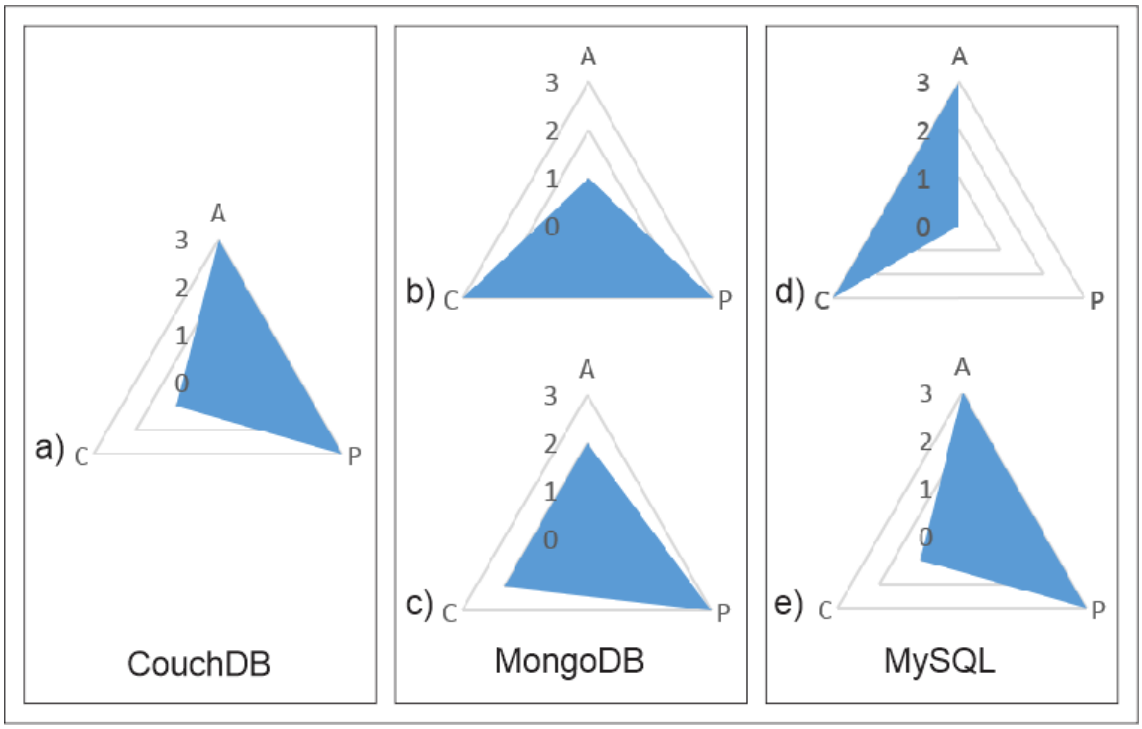
Entretanto, de acordo Brewer (2012), o sistema pode oscilar entre as três propriedades, provendo diferentes níveis de consistência, disponibilidade e tolerância

a particionamento. Deste modo, os SGBDs evoluíram e se tornaram mais adaptáveis as situações para as quais são configurados.

Assim sendo, o MySQL pode se tornar tolerante a particionamento com ajuda do MySQL Cluster, sacrificando consistência. O MongoDB pode optar entre consistência imediata e consistência eventual, provendo assim, mais disponibilidade ou consistência, dependendo da opção escolhida. O CouchDB, por outro lado, disponibiliza apenas consistência eventual, uma vez que trabalha com controle de concorrência multiversionado.

A Figura 4 ilustra o nível de consistência, disponibilidade e tolerância a particionamento das ferramentas para as configurações citadas.

Figura 4 – Propriedades CAP: CouchDB, MongoDB e MySQL.



Fonte: Elaborado pelo autor.

A possibilidade de inserir dados ou realizar atualizações em qualquer nó do sistema, mesmo na presença de falhas de conexão entre os nós, torna o CouchDB tolerante a particionamento e altamente disponível. Entretanto, o mesmo provê baixa consistência, pois para que um estado consistente seja obtido, é necessário que a conexão entre os nós seja reestabelecida. Uma vez reestabelecida, o CouchDB mantém a cópia mais recente dos documentos armazenados, atingindo assim, um estado de consistência.

A replicação de dados no MongoDB é realizada adotando o modelo *Master-Slave* (DB-ENGINES, 2016). Deste modo, todas as operações de escrita são realizadas no nó *master* e repassadas para as cópias *slave*. No cenário 4.b, o MongoDB realiza a escrita no modo seguro, garantindo que a informação seja gravada em todos os nós do sistema.

Portanto, na presença de particionamento, o sistema não será capaz de realizar operações de escrita, um vez que não será possível repassar os dados para as réplicas *slaves*. Assim sendo, o sistema é altamente consistente e tolerante a particionamento, porém, possui baixa disponibilidade.

No cenário 4.c, as operações são realizadas sem obrigatoriedade das réplicas slaves obterem as devidas atualizações. Esta configuração aumenta a disponibilidade e enfraquece a consistência, passando a ser tolerante a particionamento com média consistência e média disponibilidade. O sistema não se torna altamente disponível devido a impossibilidade de operações de escritas serem realizadas em réplicas *slaves*, deste modo, a consistência na réplica *master* é garantida.

O cenário apresentado na ilustração 4.d mostra as propriedades oferecidas pelo MySQL quando executado em modo centralizado, não estando sujeito a particionamento de rede, provendo assim, alta consistência e alta disponibilidade. Para que o MySQL seja capaz de trabalhar em modo distribuído é necessária a utilização da ferramenta MySQL *Cluster*.

Segundo Clement (2012), o MySQL *Cluster* oferece dois modos de operação, o modo único e o modo assíncrono. No primeiro caso, é definido um conjunto de réplicas *slaves* para o banco de dados, entretanto, todas as operações, sejam elas de escrita ou consulta, são realizadas na réplica *master* (MYSQL, 2016). Assim sendo, o sistema opera de forma similar a um sistema centralizado, provendo os mesmo recursos. Neste caso, o único diferencial é em caso de falha da réplica *master*, onde uma réplica *slave* toma seu lugar.

No modo assíncrono (cenário 4.e), o sistema está apto a realizar operações de escrita e consulta em todos os nós do sistema, mesmo na presença de falhas de conexão. Deste modo, o sistema além de se tornar tolerante a particionamento, uma vez que passou a trabalhar de forma distribuída, manteve seu alto nível de disponibilidade. Entretanto, neste modo de operação a consistência é drasticamente reduzida, fazendo com que o sistema se torne semelhante ao CouchDB.

4.6 CONSIDERAÇÕES FINAIS

Este capítulo objetivou apresentar e analisar os resultados obtidos com esta pesquisa, tornando evidente como as ferramentas se comportaram no cenário definido e qual apresentou maior desempenho. Além disso, também foi possível verificar algumas propriedade das mesmas.

5 CONCLUSÃO

Este capítulo tem por objetivo apresentar as considerações finais da pesquisa, bem como ressaltar suas contribuições. Além disso, salientar as limitações do trabalho a fim de fornecer propostas de continuidade para futuras pesquisas.

5.1 CONSIDERAÇÕES FINAIS

O volume de dados tem crescido exponencialmente com o passar do tempo, bem como a necessidade por performance, disponibilidade e flexibilidade dos dados. Assim sendo, este trabalho objetivou apresentar os SGBDs NoSQL como alternativa ao modelo relacional e realizar uma comparação de desempenho entre representantes de ambas as classes.

Com o desenvolver deste trabalho, ficou claro que a principal diferença entre o bancos de dados relacionais e NoSQL é a flexibilidade apresentada pelo novo paradigma. Ou seja, tanto o modelo de dados adotado quanto as propriedades oferecidas variam de acordo com a ferramenta utilizada e as configurações adotadas, provendo assim, mais opções de desenvolvimento.

No que diz respeito a performance, as ferramentas escolhidas para o experimento apresentaram grande diferença nos resultados obtidos. Deste modo, fica evidente que, mesmo apresentando tempos diferentes entre si, as ferramentas NoSQL apresentaram resultados consideravelmente melhores no ambiente de testes definido, sendo o MongoDB a ferramenta que necessitou de menos tempo para execução tanto das operações de escrita quanto de consulta.

Entretanto, percebe-se também que, para escolha de um SGBD, é necessário atentar para outros quesitos além de performance, tais como consistência, disponibilidade e tolerância a particionamento. Também foi explicado que, de acordo com o teorema CAP, para um sistema ser tolerante a particionamento, o mesmo deveria optar por oferecer maior disponibilidade ou maior consistência, sendo impossível prover ambos em sua totalidade.

Deste modo, esta pesquisa proveu grandes contribuições tanto para o âmbito acadêmico quanto profissional. Estas contribuições estão descritas a seguir.

5.2 CONTRIBUIÇÕES DA PESQUISA

Este trabalho foi de grande importância, pois contribuiu para melhor compreensão dessa nova categoria de bancos de dados bem como seu funcionamento. Com isso, espera-se que profissionais da área vejam estas ferramentas como alternativas promissoras para persistência de dados nos mais diversos sistemas, especialmente quando o principal critério de seleção for desempenho.

Entretanto, mesmo com significantes contribuições, esta pesquisa ainda apresenta limitações quanto a sua abrangência. Estas limitações estão descritas a seguir.

5.3 LIMITAÇÕES DO TRABALHO

A realização do estudo comparativo descrito neste trabalho foi desenvolvido com base apenas no critério de desempenho, além de ser realizado em um único computador, deixando em aberto lacunas sobre outras questões também importantes na escolha de um SGBD. Características como acesso concorrente, tolerância a particionamento, consistência, disponibilidade, entre outras, não foram devidamente exploradas na realização deste trabalho, tornando necessária a realização de trabalhos futuros.

5.4 PROPOSTAS DE CONTINUIDADE

Dada as limitações previamente citadas, abre-se um grande leque de possíveis trabalhos futuros, tais como uma análise detalhada de como estas ferramentas se adequam ao teorema CAP, realização de testes de performance quando escalado horizontalmente, bem como adição de outras ferramentas ao comparativo.

REFERÊNCIAS

ALMEIDA, Ricardo Cardoso de; BRITO, Parcilene Fernandes de. Utilização da Classe de Banco de Dados NOSQL como Solução para Manipulação de Diversas Estruturas de Dados*.* In: Encontro de Computação e Informática de Tocantins – ENCOINFO, 14, 2012, Palmas, **Anais...** Palmas: ULBRA, 2012. Disponível em <http://ulbra-to.br/encoinfo/artigos/2012/Utilizacao_da_Classe_de_Banco_de_%20Dados_NOSQL_como_Solucao_para_Manipulacao_de_Diversas_Estruturas_de_Dados.pdf>. Acesso em: 27 out. 2014

APACHE. ***Getting_started_with_Java:*** *Couchdb Wiki*. Disponível em: <https://wiki.apache.org/couchdb/Getting_started_with_Java>. Acesso em: 18 abr. 2016.

_____. **Overview:** *Apache CouchDB 2.0.0 Documentation*. Disponível em: <http://docs.couchdb.org/en/1.6.1/>. Acesso em: 13 abr. 2016.

BREWER, Eric Allen. *Towards Robust Towards Robust Distributed Systems Distributed Systems.* In*: ACM Symposium on Principles of Distributed Computing*, 19, 2000, Portland. **Anais…** Nova York: ACM, 2000. Disponível em: <https://people.eecs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf>. Acesso em: 04 fev. 2016.

_____. *CAP twelve years later: How the "rules" have changed*. ***Computer***, [S.l.], v. 45, n. 2, p. 23-29, fev. 2012. Disponível em: <https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed>. Acesso em: 05 fev. 2016.

BRITO, Ricardo W. Bancos de Dados NoSQL x SGBDs Relacionais: Análise Comparativa. In: InfoBrasil TI & Telecom, 36, 2010, Fortaleza. **Anais...** Fortaleza: SEBRAE, 2010. Disponível em <http://infobrasil.inf.br/userfiles/27-05-S4-1-68840-Bancos%20de%20Dados%20NoSQL(1).pdf>. Acesso em: 27 out. 2014.

CHAPPLE, Mike. ***Abandoning ACID in Favor of BASE***. Disponível em: <http://databases.about.com/od/sqlserver/a/isolationmodels.htm>. Acesso em: 04 fev. 2016

CODD, Edgar Frank. *A Relational Model of Data for Large Shared Data Banks*. ***Communications of the ACM***, Nova York, v. 13, n. 6, p. 377-387, jun. 1970. Disponível em <https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf>. Acesso em: 17 abr. 2015.

CRAWCOUR, Ryan. ***Modeling data in Azure DocumentDB*****:** Microsoft Azure. Disponível em: <https://azure.microsoft.com/en-us/documentation/articles/documentdb-modeling-data/>. Acesso em: 20 abr. 2016.

DB-ENGINES. ***CouchDB vs. MongoDB vs. MySQL Comparison***. Disponível em: <http://db-engines.com/en/system/CouchDB%3BMongoDB%3BMySQL>. Acesso em: 28 jun. 2016.

DEAN, Jeffrey; GHEMAWAT, Sanjay. *MapReduce: Simplied Data Processing on Large Clusters.* In: *Symposium on Operating Systems Design and Implementation*, 6, 2004, San Francisco. **Anais…** São Francisco: USENIX, 2004. Disponível em <http://static.googleusercontent.com/media/research.google.com/pt-BR//archive/mapreduce-osdi04.pdf>. Acesso em: 18 nov. 2015.

DIANA, Mauricio De; GEROSA, Marco Aurélio. NOSQL na Web 2.0: Um Estudo Comparativo de Bancos Não-Relacionais para Armazenamento de Dados na Web 2.0. In: Workshop de Teses e Dissertações em Banco de Dados, 9, 2010, Belo Horizonte. **Anais...** Belo Horizonte, 2010. Disponível em <http://www.lbd.dcc.ufmg.br/colecoes/wtdbd/2010/sbbd_wtd_12.pdf>. Acesso em: 27 out. 2014.

ERICSSON. ***Erlang Programming Language***. Disponível em: <http://www.erlang.org/>. Acesso em: 13 abr. 2016.

GIL, Antônio Carlos. **Como Elaborar Projetos de Pesquisa**. 4. Ed. São Paulo: Atlas. 2002. Disponível em: <http://www.academia.edu/4405328/GIL_Antonio_Carlos_COMO_ELABORAR_PROJETOS_DE_PESQUISA_Copia>. Acesso em: 16 nov. 2014

GILBERT, Seth; LYNCH, Nancy. *Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services.* ***ACM SIGACT News***, Neva York, v. 33, n. 2, p. 51-59, jun. 2002. Disponível em: <http://dl.acm.org/citation.cfm?id=564601>. Acesso em: 04 fev. 2016.

HURST, Nathan. ***Visual Guide to NoSQL Systems***. [s.l.:s.n]. 2010. Disponível em: <http://blog.nahurst.com/visual-guide-to-nosql-systems>. Acesso em: 10 mar. 2016.

KARGER, David *et. al. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web.* In*: Annual ACM symposium on Theory of computing,* 29, 1997, El Paso. **Anais…** Nova York: ACM, 1997. p. 654-663. Disponível em: <http://www.cs.princeton.edu/courses/archive/fall09/cos518/papers/chash.pdf>. Acesso em: 18 nov. 2015.

LÓSCIO, Bernadette Farias; OLIVEIRA, Hélio Rodrigues de; PONTES, Jonas César de Sousa. NoSQL no desenvolvimento de aplicações Web colaborativas. In: SIMPÓSIO BRASILEIRO DE SISTEMAS COLABORATIVOS, 8, 2011, Paraty. **Anais...** Paraty: SBC, 2007. Disponível em <http://www.addlabs.uff.br/sbsc_site/SBSC2011_NoSQL.pdf>. Acesso em: 19 nov. 2015

MONGODB. ***MongoDB Java Driver***. Disponível em: <http://mongodb.github.io/mongo-java-driver/>. Acesso em 18 abr. 2016.

MYSQL. **MySQL**: MySQL 5.7 Reference Manual. Disponível em: <http://dev.mysql.com/doc/refman/5.7/en/>. Acesso em: 13 abr. 2016.

NASCIMENTO, Jean. **NoSQL:** você realmente sabe do que estamos falando? [s.l.:s.n]. 2010. Disponível em: <http://imasters.com.br/artigo/17043/banco-de-dados/nosql-voce-realmente-sabe-do-que-estamos-falando?trace=1519021197&source=author-archive>. Acesso em: 20 nov. 2015.

NOSQL. **NOSQL Databases.** Disponível em: <http://nosql-database.org/>. Acesso em: 16 nov. 2015.

JAVA. **Java SE Development Kit 8:** Downloads. Disponível em: <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>. Acesso em: 18 abr. 2016.

OREND, Kai. **Analysis and Classification of NoSQL Databases and Evaluation of their Ability to Replace an Object-relational Persistence Layer**. 2010. 100 f. Dissertação (Mestrado em Informática) - Technische Universität München, Munique 2010. Disponível em: <https://wwwmatthes.in.tum.de/file/ikcuitkq8cpm/2010/Or10/Or10.pdf>. Acesso em: 13 abr. 2016.

POLITOWSKI, Cristiano; MARAN, Vinícius. Comparação de Performance entre PostgreSQL e MongoDB. In: Escola Regional de Banco de Dados, 10, 2014, São Francisco do Sul. **Anais...** São Francisco do Sul: IFSC, 2014. Disponivel em: <http://www.lbd.dcc.ufmg.br/colecoes/erbd/2014/003.pdf>. Acesso em: 27 out. 2014.

SILVA, Edna Lúcia da; MENEZES, Estera Muszkat. **Metodologia da pesquisa e elaboração de dissertação**. 4. Ed. Florianópolis: UFSC, 2005. Disponível em: <https://projetos.inf.ufsc.br/arquivos/Metodologia_de_pesquisa_e_elaboracao_de_teses_e_dissertacoes_4ed.pdf>. Acesso em: 16 nov. 2014.

SOUSA, Clóvis**. Metodologia da Pesquisa Científica**. Rio de Janeiro, 2009. Disponível em: <http://sis.posugf.com.br/AreaProfessor/Materiais/Arquivos_1/19908.pdf>. Acesso em: 16 nov. 2014.

STEPPAT, Nico. **NoSQL:** Do teorema CAP para P?(A|C):(C|L). [s.l.:s.n]. 2011. Disponível em: <http://blog.caelum.com.br/nosql-do-teorema-cap-para-paccl/>. Acesso em: 04 fev. 2016.

STROZZI, Carlo. **NoSQL Relational Database Management System**. Disponível em: <http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/NoSQL/Home%20Page>. Acesso em: 16 jun. 2015.

APÊNDICE A: Estrutura dos Documentos

```
        { } = Documento
        [ ] = Lista

Disciplina {
        _id
        Type
        Nome
        Série
        Conteúdo
}

Turma {
        _id
        Type
        Série
        Letra
        Turno
        Modalidade
        Sala
        Aulas [
                {
                        Disciplina (Referência a um registro de Disciplina)
                        Professor {
                                _id
                                Nome
                                Sexo
                                Nascimento
                                Formação [
                                        {
                                                Curso
                                                Instituição
                                                Nível
                                                Ano
                                        }
                                ]
                        }
                        Hora
                        Dia_da_semana
                        Registros [
                                {
                                        Professor {
                                                _id
```

```
                    Nome
                }
                Número
                Data
                Descrição
                Alunos_faltosos [ (alunos que faltarão) ]
            }
        ]
    }
]
Alunos [
    {
        Matrícula
        Nome
        Responsável
        Sexo
        Nascimento
        Histórico [
            {
                Instituição
                Série
                Desempenho
                Presença
                Ano
            }
        ]
        Desempenho [
            {
                Disciplina (Referência a um registro de Disciplina)
                Notas
                Observações
            }
        ]
    }
]
}
```

APÊNDICE B: Funções MapReduce utilizadas no CouchDB

CONSULTA SIMPLES

FUNÇÃO *MAP:*

```
function(doc){
   if(doc.type=="turma"){
        for(var aluno in doc.alunos){
            emit({aluno:doc.alunos[aluno].matricula,nome:doc.alunos[aluno].nome},
           {turma:doc._id,serie:doc.serie,letra:doc.letra});
        }
   }
}
```

CONSULTA COMPLEXA

FUNÇÃO *MAP*:

```
function(doc) {
   var aux=false;
   if(doc.type=="turma"){
        for(var aula in doc.aulas){
           for(var formacao in doc.aulas[aula].professor.formacao){
                if(doc.aulas[aula].professor.formacao[formacao].nivel=="superior")
                   aux=true
                   break;
           }
           if(aux) break;
        }
        if(aux){
           for(var aluno in doc.alunos){
```

```
                aux=false;
                for(var aula in doc.aulas){
                   for(var reg in doc.aulas[aula].registros){
                        for(var falta in doc.aulas[aula].registros[reg].alunos_faltosos){
                           if(doc.aulas[aula].registros[reg].alunos_faltosos[falta]==
                           doc.alunos[aluno]._id){
                                aux=true;
                                break;
                           }
                        }
                        if(aux) break;
                   }
                   if(aux) break;
                }
                if(!aux)
                        emit({turma:doc._id,serie:doc.serie,letra:doc.letra,aluno:doc.
                        alunos[aluno].nome},doc.alunos[aluno].desempenho);
            }
        }
    }
}
```

FUNÇÃO *REDUCE*:

```
function(key,values){
    var aux=[];
    for(var i in values){
        aux=aux.concat([values[i][0].disciplina,values[i][0].notas]);
    }
    return aux;
}
```

APÊNDICE C: Código para simulações no MySQL

```
1 package testemysql;
2
3 import java.io.FileWriter;
4 import java.sql.Connection;
5 import java.sql.DriverManager;
6 import java.sql.Statement;
7 import java.sql.Date;
8 import java.sql.SQLException;
9
10 /**
11  *
12  * @author jaziel
13  */
14 public class TesteMySQL {
15
16     public TesteMySQL(int atual){
17         contDisciplina=atual;
18         contProfessor=atual;
19         contTurma=atual;
20         contRegistro=atual;
21         contAluno=atual*30;
22     }
23
24     private void teste(String acao,int total){
25         int aux=0;
26         this.total=total;
27         defineThread(acao);
28         try {
29             Class.forName("com.mysql.jdbc.Driver");
30             Connection connect = DriverManager.getConnection("jdbc:mysql://"
31                     + "localhost/escola?" + "user=root&password=root");
32             statement= connect.createStatement();
33             trabalho.start();
34             iteracao=0;
35             while(iteracao<total){
36                 Thread.sleep(100);
37                 if(iteracao!=aux){
38                     System.out.println((iteracao/this.total)*100 + "%");
39                     aux=iteracao;
40                 }
41             }
42         } catch (Exception ex) {
```

```
43          ex.printStackTrace();
44      }
45  }
46
47  private void defineThread(String op){
48      trabalho= new Thread(() -> {
49          long tempoGasto=0;
50          try {
51              FileWriter fw=new FileWriter("log.txt",true);
52              switch (op) {
53                  case "inserir":
54                      System.out.println("Operação Inserir");
55                      System.out.println("Repetições: "+total);
56                      for(iteracao=0;iteracao<total;iteracao++)
57                          tempoGasto+=inserir();
58                      fw.write("Operação Inserir\n");
59                      fw.write("Repetições: "+total+"\n");
60                      fw.write("Tempo gasto: "+tempoGasto+"ns\n\n");
61                      fw.flush();
62                      break;
63                  case "buscaSimples":
64                      System.out.println("Operação Busca Simples");
65                      System.out.println("Repetições: "+total);
66                      for(iteracao=0;iteracao<total;iteracao++)
67                          tempoGasto+=buscaSimples();
68                      fw.write("Operação Busca Simples \n");
69                      fw.write("Repetições: "+total+"\n");
70                      fw.write("Tempo gasto: "+tempoGasto+"ns\n\n");
71                      fw.flush();
72                      break;
73                  case "buscaComplexa":
74                      System.out.println("Operação Busca Complexa");
75                      System.out.println("Repetições: "+total);
76                      for(iteracao=0;iteracao<total;iteracao++)
77                          tempoGasto+=buscaComplexa();
78                      fw.write("Operação Busca Complexa \n");
79                      fw.write("Repetições: "+total+"\n");
80                      fw.write("Tempo gasto: "+tempoGasto+"ns\n\n");
81                      fw.flush();
82              }
83          } catch (Exception ex) {
84              ex.printStackTrace();
85          }
86      },op+", repetiçoes: "+total);
87  }
```

```
88
89    private long inserir() throws Exception{
90        long ini;
91        long fim;
92        long soma=0;
93        String disciplina="insert into disciplina values (";
94        disciplina+=++contDisciplina;
95        disciplina+=",\"disciplina "+contDisciplina+"\",";
96        disciplina+="2";
97        disciplina+=",\"conteúdos a serem estudados na disciplina\")";
98        ini=System.nanoTime();
99        statement.execute(disciplina);
100        fim=System.nanoTime();
101        soma+=fim-ini;
102
103        String professor="insert into professor values (";
104        professor+=++contProfessor;
105        professor+=",\"professor "+ contProfessor+"\"";
106        professor+=",\"masculino\"";
107        professor+=",\""+new Date(0)+"\")";
108        ini=System.nanoTime();
109        statement.execute(professor);
110        fim=System.nanoTime();
111        soma+=fim-ini;
112
113        String formacao="insert into formacao values (";
114        formacao+= contProfessor;
115        formacao+=",\"pedagogia\"";
116        formacao+=",\"UEPB\"";
117        formacao+=",\"superior\"";
118        formacao+=",2015)";
119        ini=System.nanoTime();
120        statement.execute(formacao);
121        fim=System.nanoTime();
122        soma+=fim-ini;
123
124        String turma="insert into turma values (";
125        turma+=++contTurma;
126        turma+=",2";
127        turma+=",\"A\"";
128        turma+=",\"noite\"";
129        turma+=",\"regular\",";
130        turma+=contTurma+")";
131        ini=System.nanoTime();
132        statement.execute(turma);
```

```
133         fim=System.nanoTime();
134         soma+=fim-ini;
135
136         String aula="insert into aula values (";
137         aula+=contTurma;
138         aula+=","+contDisciplina;
139         aula+=","+contProfessor;
140         aula+=",\"7-11\"";
141         aula+=",\"seg/ter/qua/qui/sex\")";
142         ini=System.nanoTime();
143         statement.execute(aula);
144         fim=System.nanoTime();
145         soma+=fim-ini;
146
147         String reg="insert into reg_aula values (";
148         reg+=++contRegistro;
149         reg+=","+contDisciplina;
150         reg+=","+contTurma;
151         reg+=",1";
152         reg+=","+contProfessor;
153         reg+=",\""+new Date(System.currentTimeMillis())+"\"";
154         reg+=",\"aula 1\")";
155         ini=System.nanoTime();
156         statement.execute(reg);
157         fim=System.nanoTime();
158         soma+=fim-ini;
159
160         String aluno;
161         Date dataNascimento=new Date((System
162                 .currentTimeMillis()/1000-220903200)*1000);//data de 7 anos atrás
163         for(int i=0;i<30;i++){
164            aluno="insert into aluno values (";
165            aluno+=++contAluno;
166            aluno+=",\"Aluno "+contAluno+"\"";
167            aluno+=",\"Responsavel "+contAluno+"\"";
168            aluno+=",\"masculino\"";
169            aluno+=",\""+dataNascimento+"\"";
170            aluno+=","+contTurma+")";
171            ini=System.nanoTime();
172            statement.execute(aluno);
173            fim=System.nanoTime();
174            soma+=fim-ini;
175
176            String desempenho="insert into desempenho values (";
177            desempenho+=contAluno;
```

```
178             desempenho+=","+contDisciplina;
179             desempenho+=",\"10/10\"";
180             desempenho+=",\"Observações referente ao comportamento do aluno"
181                     + " em sala de aula\")";
182             ini=System.nanoTime();
183             statement.execute(desempenho);
184             fim=System.nanoTime();
185             soma+=fim-ini;
186 
187             String historico="insert into historico values (";
188             historico+="\"Escola\"";
189             historico+=",1";
190             historico+=","+contAluno;
191             historico+=",\"portugues:10/matematica:10\"";
192             historico+=",100";
193             historico+=",2015)";
194             ini=System.nanoTime();
195             statement.execute(historico);
196             fim=System.nanoTime();
197             soma+=fim-ini;
198         }
199 
200         String falta="insert into falta values (";
201         falta+=contRegistro;
202         falta+=","+(contAluno-10)+")";
203         ini=System.nanoTime();
204         statement.execute(falta);
205         fim=System.nanoTime();
206         soma+=fim-ini;
207         return soma;
208     }
209 
210     private long buscaSimples() throws SQLException{
211         long ini;
212         long fim;
213         ini=System.nanoTime();
214         statement.executeQuery("select a.matricula, a.nome, t.id, t.serie,"
215                 + "t. letra from aluno a inner join turma t on a.turma = t.id");
216         fim=System.nanoTime();
217         return fim-ini;
218     }
219 
220     private long buscaComplexa() throws SQLException{
221         long ini;
222         long fim;
```

```
223         ini=System.nanoTime();
224         statement.executeQuery("select t.serie, t.letra, al.nome, de.notas, "
225                 + "di.nome from professor p inner join formacao f on "
226                 + "p.matricula = f.professor inner join aula au on "
227                 + "p.matricula = au.professor inner join disciplina di on "
228                 + "au.disciplina = di.id inner join desempenho de on "
229                 + "di.id = de.disciplina inner join aluno al on "
230                 + "de.aluno = al.matricula left join falta fa on "
231                 + "al.matricula = fa.aluno inner join turma t on "
232                 + "al.turma = t.id where f.nivel = \"superior\" and "
233                 + "fa.aluno is null order by al.matricula desc");
234         fim=System.nanoTime();
235         return fim-ini;
236     }
237 
238     /**
239      * @param args the command line arguments
240      */
241     public static void main(String[] args) {
242         TesteMySQL testeDB= new TesteMySQL(3333);
243         for(int i=0;i<3;i++)
244             testeDB.teste("buscaComplexa", 1000);
245     }
246 
247     private Thread trabalho;
248     private Statement statement;
249     private int contDisciplina;
250     private int contProfessor;
251     private int contTurma;
252     private int contRegistro;
253     private int contAluno;
254     private int iteracao;
255     private double total;
256 }
```

APÊNDICE D: Código para simulações no CouchDB

```
1 package testecouch;
2
3 import com.fourspaces.couchdb.Database;
4 import com.fourspaces.couchdb.Document;
5 import com.fourspaces.couchdb.Session;
6 import java.io.FileWriter;
7 import java.io.IOException;
8 import java.util.ArrayList;
9 import static java.util.Arrays.asList;
10 import java.util.Date;
11 import java.util.List;
12 import org.apache.http.client.HttpClient;
13 import org.apache.http.client.methods.HttpGet;
14 import org.apache.http.impl.client.HttpClientBuilder;
15
16 /**
17  *
18  * @author jaziel
19  */
20 public class TesteCouch {
21
22     /**
23      * @param atual total de documanetos salvos no banco
24      */
25     public TesteCouch(int atual){
26         contDocumentos=atual;
27         contAluno=atual*15;
28         contIteracao=atual/2;
29     }
30
31     private void teste(String acao,int total){
32         int aux=0;
33         this.total=total;
34         Session dbSession = new Session("localhost", 5984);
35         db = dbSession.getDatabase("escola");
36         defineThread(acao);
37         trabalho.start();
38         iteracao=0;
39         try {
40             while(iteracao<total){
41                 Thread.sleep(500);
42                 if(iteracao!=aux){
```

```
43                  System.out.println((iteracao/this.total)*100 + "%");
44                  aux=iteracao;
45              }
46          }
47      } catch (Exception ex) {
48          ex.printStackTrace();
49      }
50  }
51
52  private void defineThread(String op){
53      trabalho= new Thread(() -> {
54          long tempoGasto=0;
55          try {
56              FileWriter fw=new FileWriter("log.txt",true);
57              switch (op) {
58                  case "inserir":
59                      System.out.println("Operação Inserir");
60                      System.out.println("Repetições: "+total);
61                      for(iteracao=0;iteracao<total;iteracao++)
62                          tempoGasto+=inserir();
63                      fw.write("Operação Inserir\n");
64                      fw.write("Repetições: "+total+"\n");
65                      fw.write("Tempo gasto: "+tempoGasto+"ns\n\n");
66                      fw.flush();
67                      break;
68                  case "buscaSimples":
69                      System.out.println("Operação Busca Simples");
70                      System.out.println("Repetições: "+total);
71                      for(iteracao=0;iteracao<total;iteracao++)
72                          tempoGasto+=buscaSimples();
73                      fw.write("Operação Busca Simples \n");
74                      fw.write("Repetições: "+total+"\n");
75                      fw.write("Tempo gasto: "+tempoGasto+"ns\n\n");
76                      fw.flush();
77                      break;
78                  case "buscaComplexa":
79                      System.out.println("Operação Busca Complexa");
80                      System.out.println("Repetições: "+total);
81                      for(iteracao=0;iteracao<total;iteracao++)
82                          tempoGasto+=buscaComplexa();
83                      fw.write("Operação Busca Complexa \n");
84                      fw.write("Repetições: "+total+"\n");
85                      fw.write("Tempo gasto: "+tempoGasto+"ns\n\n");
86                      fw.flush();
87              }
```

```
88            } catch (Exception ex) {
89                ex.printStackTrace();
90            }
91        },op+", repetiçoes: "+total);
92    }
93
94    private long inserir(){
95        contIteracao++;
96        Document disciplina= new Document();
97        disciplina.setId(++contDocumentos+"");
98        disciplina.put("type","disciplina");
99        disciplina.put("nome", "disciplina "+contIteracao);
100       disciplina.put("serie", 2);
101       disciplina.put("conteudo", "conteúdos a serem estudados na disciplina");
102
103       Document formacao= new Document();
104       formacao.put("curso","pedagogia");
105       formacao.put("instituicao","UEPB");
106       formacao.put("nivel", "superior");
107       formacao.put("ano",2015);
108
109       Document professor= new Document();
110       professor.setId(++contIteracao+"");
111       professor.put("nome", "professor "+contIteracao);
112       professor.put("sexo", "masculino");
113       professor.put("nascimento",new Date(0));
114       professor.put("formacao", asList(formacao));
115
116       List alunos=new ArrayList<>();
117       Document aluno;
118       Document historico;
119       Document desempenho;
120       Date dataNascimento=new Date((System
121               .currentTimeMillis()/1000-220903200)*1000);//data de 7 anos atrás
122       for(int i=0;i<30;i++){
123           historico=new Document();
124           historico.put("instituicao","Escola");
125           historico.put("serie", 1);
126           historico.put("desempenho", "portugues:10/matematica:10");
127           historico.put("presenca", 100);
128           historico.put("ano", 2015);
129
130           desempenho=new Document();
131           desempenho.put("disciplina","disciplina "+contIteracao);
132           desempenho.put("notas", "10/10");
```

```
133         desempenho.put("observacoes", "Observações referente ao "
134                 + "comportamento do aluno em sala de aula");
135
136         aluno=new Document();
137         aluno.setId(++contAluno+"");
138         aluno.put("nome", "Aluno "+ contAluno);
139         aluno.put("responsavel", "responsavel "+contAluno);
140         aluno.put("sexo", "masculino");
141         aluno.put("nascimento",dataNascimento);
142         aluno.put("historico", asList(historico));
143         aluno.put("desempenho", asList(desempenho));
144         alunos.add(aluno);
145     }
146
147     Document prof_aux = new Document();
148     prof_aux.setId(contIteracao +"");
149     prof_aux.put("nome", "professor "+contIteracao);
150
151     Document registro=new Document();
152     registro.put("data", new Date(System.currentTimeMillis()));
153     registro.put("descricao", "aula 1");
154     registro.put("professor",prof_aux);
155     registro.put("alunos_faltosos", asList(contAluno-10));
156
157     Document aula = new Document();
158     aula.put("disciplina",contDocumentos);
159     aula.put("professor", professor);
160     aula.put("hora", "7-11");
161     aula.put("dia_da_semana","seg/ter/qua/qui/sex");
162     aula.put("registros", asList(registro));
163
164     Document turma= new Document();
165     turma.setId(++contDocumentos+"");
166     turma.put("type", "turma");
167     turma.put("serie", 2);
168     turma.put("letra", "A");
169     turma.put("turno", "noite");
170     turma.put("modalidade", "regular");
171     turma.put("sala", contIteracao);
172     turma.put("aulas", asList(aula));
173     turma.put("alunos", alunos);
174
175     long ini;
176     long fim;
177     ini=System.nanoTime();
```

```
178         db.saveDocument(disciplina);
179         db.saveDocument(turma);
180         fim=System.nanoTime();
181         return fim-ini;
182     }
183 
184 
185     private long buscaSimples() throws IOException{
186 
187         HttpClient httpclient = HttpClientBuilder.create().build();
188         HttpGet get = new HttpGet("http://localhost:5984/escola/_design/"
189                 + "Consulta/_view/simples");
190 
191         long ini;
192         long fim;
193         ini=System.nanoTime();
194         httpclient.execute(get);
195         fim=System.nanoTime();
196         return fim-ini;
197     }
198     private long buscaComplexa() throws IOException{
199         HttpClient httpclient = HttpClientBuilder.create().build();
200         HttpGet get = new HttpGet("http://localhost:5984/escola/_design/"
201                 + "Consulta/_view/complexa?group_level=1");
202         long ini;
203         long fim;
204         ini=System.nanoTime();
205         httpclient.execute(get);
206         fim=System.nanoTime();
207         return fim-ini;
208     }
209 
210     /**
211      * @param args the command line arguments
212      */
213     public static void main(String[] args) {
214         TesteCouch testeDB=new TesteCouch(6666);
215         for(int i=0;i<3;i++)
216             testeDB.teste("buscaComplexa", 1000);
217     }
218 
219     private Thread trabalho;
220     private Database db;
221     private int contDocumentos;
222     private int contAluno;
```

```
223     private double total;
224     private int iteracao;
225     private int contIteracao;
226 }
```

APÊNDICE E: Código para simulações no MongoDB

```
1 package testemongo;
2
3 import com.mongodb.MongoClient;
4 import com.mongodb.client.MongoCollection;
5 import com.mongodb.client.MongoDatabase;
6 import java.io.FileWriter;
7 import java.util.ArrayList;
8 import static java.util.Arrays.asList;
9 import java.util.Date;
10 import java.util.List;
11 import org.bson.Document;
12
13 /**
14  *
15  * @author jaziel
16  */
17 public class TesteMongo {
18
19     public TesteMongo(int atual){
20         contDocumentos=atual;
21         contAluno=atual*15;
22         contIteracao=atual/2;
23     }
24
25     private void teste(String acao,int total){
26         int aux=0;
27         this.total=total;
28         MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
29         MongoDatabase db = mongoClient.getDatabase("escola");
30         colecao = db.getCollection("escola");
31         defineThread(acao);
32         trabalho.start();
33         iteracao=0;
34         try {
35             while(iteracao<total){
36                 Thread.sleep(100);
37                 if(iteracao!=aux){
38                     System.out.println((iteracao/this.total)*100 + "%");
39                     aux=iteracao;
40                 }
41             }
42         } catch (Exception ex) {
```

```
43          ex.printStackTrace();
44      }
45  }
46
47  private void defineThread(String op){
48      trabalho= new Thread(() -> {
49          long tempoGasto=0;
50          try {
51              FileWriter fw=new FileWriter("log.txt",true);
52              switch (op) {
53                  case "inserir":
54                      System.out.println("Operação Inserir");
55                      System.out.println("Repetições: "+total);
56                      for(iteracao=0;iteracao<total;iteracao++)
57                          tempoGasto+=inserir();
58                      fw.write("Operação Inserir\n");
59                      fw.write("Repetições: "+total+"\n");
60                      fw.write("Tempo gasto: "+tempoGasto+"ns\n\n");
61                      fw.flush();
62                      break;
63                  case "buscaSimples":
64                      System.out.println("Operação Busca Simples");
65                      System.out.println("Repetições: "+total);
66                      for(iteracao=0;iteracao<total;iteracao++)
67                          tempoGasto+=buscaSimples();
68                      fw.write("Operação Busca Simples \n");
69                      fw.write("Repetições: "+total+"\n");
70                      fw.write("Tempo gasto: "+tempoGasto+"ns\n\n");
71                      fw.flush();
72                      break;
73                  case "buscaComplexa":
74                      System.out.println("Operação Busca Complexa");
75                      System.out.println("Repetições: "+total);
76                      for(iteracao=0;iteracao<total;iteracao++)
77                          tempoGasto+=buscaComplexa();
78                      fw.write("Operação Busca Complexa \n");
79                      fw.write("Repetições: "+total+"\n");
80                      fw.write("Tempo gasto: "+tempoGasto+"ns\n\n");
81                      fw.flush();
82              }
83          } catch (Exception ex) {
84              ex.printStackTrace();
85          }
86      },op+", repetiçoes: "+total);
87  }
```

```
88
89  private long inserir(){
90      contIteracao++;
91      Document disciplina= new Document("_id",++contDocumentos)
92              .append("type","disciplina")
93              .append("nome","disciplina "+contIteracao)
94              .append("serie", 2)
95              .append("conteudo","conteúdos a serem estudados na disciplina");
96
97      List alunos=new ArrayList<Document>();
98      Date dataNascimento=new Date((System
99              .currentTimeMillis()/1000-220903200)*1000);//data de 7 anos atrás
100     Document aluno;
101     for(int i=0;i<30;i++){
102        Document historico=new Document("instituicao","Escola")
103                .append("serie", 1)
104                .append("desempenho", "portugues:10/matematica:10")
105                .append("presenca", 100)
106                .append("ano", 2015);
107
108        Document desempenho=new Document("disciplina","disciplina "
109                +contIteracao)
110                .append("notas", "10/10")
111                .append("observacoes", "Observações referente ao "
112                        + "comportamento do aluno em sala de aula");
113
114        aluno=new Document("matricula",++contAluno)
115                .append("nome", "Aluno "+ contAluno)
116                .append("responsavel", "responsavel "+contAluno)
117                .append("sexo", "masculino")
118                .append("nascimento",dataNascimento)
119                .append("historico", asList(historico))
120                .append("desempenho", asList(desempenho));
121
122        alunos.add(aluno);
123     }
124
125     Document formacao=new Document("curso","pedagogia")
126             .append("instituicao","UEPB")
127             .append("nivel", "superior")
128             .append("ano",2015);
129
130     Document professor= new Document("_id",contIteracao)
131             .append("nome", "professor "+contIteracao)
132             .append("sexo", "masculino")
```

```
133             .append("nascimento",new Date(0))
134             .append("formacao", asList(formacao));
135
136     Document prof_aux=new Document("_id",contIteracao)
137             .append("nome", "professor "+contIteracao);
138
139     Document registro=new Document("numero",1)
140             .append("data", new Date(System.currentTimeMillis()))
141             .append("descricao", "aula 1")
142             .append("professor", prof_aux)
143             .append("alunos_faltosos", asList(contAluno-10));
144
145     Document aula=new Document("Disciplina",contDocumentos - 2)
146             .append("professor", professor)
147             .append("hora", "7-11")
148             .append("dia_da_semana","seg/ter/qua/qui/sex")
149             .append("registros", asList(registro));
150
151     Document turma= new Document("_id",++contDocumentos)
152             .append("type", "turma")
153             .append("serie", 2)
154             .append("letra", "A")
155             .append("turno", "noite")
156             .append("modalidade", "regular")
157             .append("sala", contIteracao)
158             .append("aulas", asList(aula))
159             .append("alunos", alunos);
160
161     long ini;
162     long fim;
163     ini=System.nanoTime();
164     colecao.insertOne(disciplina);
165     colecao.insertOne(turma);
166     fim=System.nanoTime();
167     return fim-ini;
168 }
169
170 private long buscaSimples(){
171     List query=new ArrayList<Document>();
172     query.add(new Document("$unwind","$alunos"));
173
174     Document match=new Document("type","turma");
175     query.add(new Document("$match",match));
176
177     Document push=new Document("turma","$_id")
```

```
178             .append("serie","$serie")
179             .append("letra","$letra");
180
181     Document group=new Document("_id","$alunos.matricula")
182             .append("nome", new Document("$push","$alunos.nome"))
183             .append("turma", new Document ("$push",push));
184     query.add(new Document("$group",group));
185
186     long ini;
187     long fim;
188     ini=System.nanoTime();
189     colecao.aggregate(query);
190     fim=System.nanoTime();
191     return fim-ini;
192 }
193
194 private long buscaComplexa(){
195     List query=new ArrayList<Document>();
196     query.add(new Document("$unwind","$aulas"));
197     query.add(new Document("$unwind","$aulas.professor.formacao"));
198     query.add(new Document("$unwind","$alunos"));
199     query.add(new Document("$unwind","$alunos.desempenho"));
200     query.add(new Document("$unwind","$aulas.registros"));
201     query.add(new Document("$unwind","$aulas.registros.alunos_faltosos"));
202
203     Document eq=new Document("$eq",asList(
204             "$aulas.registros.alunos_faltosos","$alunos.matricula"));
205     Document cond=new Document("$cond",asList(eq,true,false));
206
207     Document project=new Document("falta",cond)
208             .append("alunos",true)
209             .append("aulas",true)
210             .append("serie",true)
211             .append("letra",true)
212             .append("type", "turma");
213     query.add(new Document("$project",project));
214
215     Document match=new Document("type","turma");
216     query.add(new Document("$match",match));
217
218     match= new Document("aulas.professor.formacao.nivel","superior");
219     query.add(new Document("$match",match));
220
221     match=new Document("falta",false);
222     query.add(new Document("$match",match));
```

```
223
224         Document push=new Document("turma","$_id")
225                 .append("serie","$serie")
226                 .append("letra","$letra");
227         Document turma=new Document("$push",push);
228
229         push=new Document("disciplina","$alunos.desempenho.disciplina")
230                 .append("notas","$alunos.desempenho.notas");
231         Document desempenho=new Document("$push",push);
232
233         Document group=new Document("_id","$alunos.nome")
234                 .append("turma", turma)
235                 .append("desempenho", desempenho);
236         query.add(new Document("$group",new
Document("_id",null).append("total", new Document("$sum",1))));
237
238         long ini;
239         long fim;
240         ini=System.nanoTime();
241         colecao.aggregate(query);
242         fim=System.nanoTime();
243         return fim-ini;
244     }
245
246     /**
247      * @param args the command line arguments
248      */
249     public static void main(String[] args) {
250         TesteMongo testeDB=new TesteMongo(6666);
251         for(int i=0;i<3;i++){
252             testeDB.teste("buscaComplexa", 1000);
253
254         }
255     }
256
257     private Thread trabalho;
258     private MongoCollection colecao;
259     private int contDocumentos;
260     private int contAluno;
261     private double total;
262     private int contIteracao;
263     private int iteracao;
264
265 }
```