UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE CIÊNCIAS APLICADAS A EDUCAÇÃO

DEPARTAMENTO DE CIÊNCIAS EXATAS

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Avaliação de Desempenho entre Bancos de Dados Relacionais e NoSQL

FRED AUGUSTO DE MELO FARIAS

Orientador: Prof. Ms. Renata Viegas de Figueiredo

Co-Orientador: Prof. Ms. Rodrigo de Almeida Vilar.

RIO TINTO - PB

2014

FRED AUGUSTO DE MELO FARIAS

Avaliação de Desempenho entre Bancos de Dados Relacionais e NoSQL

Monografia apresentada para obtenção do título de Bacharel à banca examinadora no Curso de Bacharelado em Sistemas de Informação do Centro de Ciências Aplicadas e Educação (CCAE), Campus IV da Universidade Federal da Paraíba.
Orientador: Prof. Ms. Renata Viegas de Figueiredo e Co-Orientador: Prof. Ms. Rodrigo de Almeida Vilar.

RIO TINTO - PB

2014

F224a Farias, Fred Augusto de Melo.

Avaliação de desempenho entre Bancos de Dados Relacionais e NoSQL. / Fred Augusto de Melo Farias. – Rio Tinto: [s.n.], 2014.

61 f. : il. –

Orientadora: Profa. Ms. Renata Viegas de Figueiredo.

Co-orientador: Prof. Ms. Rodrigo de Almeida Vilar.

Monografia (Graduação) – UFPB/CCAE.

FRED AUGUSTO DE MELO FARIAS

Título

Trabalho de Conclusão de Curso submetido ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal da Paraíba, Campus IV, como parte dos requisitos necessários para obtenção do grau de BACHAREL EM SISTEMAS DE INFORMAÇÃO.

Assinatura do autor: _____

APROVADO POR:

_____

Orientador: Prof. Ms. Renata Viegas de Figueiredo

Universidade Federal da Paraíba – Campus IV

_____

Co-Orientador: Prof. Ms. Rodrigo de Almeida Vilar

Universidade Federal da Paraíba – Campus IV

_____

Prof. Ms. Vanessa Farias Dantas

Universidade Federal da Paraíba – Campus IV

_____

Prof. Ms. Marcus Willians Aquino de Carvalho

Universidade Federal da Paraíba – Campus IV

RIO TINTO – PB

2014

Aos amigos, colegas e professores, minha eterna gratidão por compartilhar comigo seus conhecimentos.

AGRADECIMENTOS

A Deus, pois sem Ele nada disso seria possível!

Ao meu pai (In Memoriam), por seu inigualável incentivo;

A minha mãe, por depositar em mim todo o seu amor e sua fé;

A minha esposa, Meyka Almeida, por alegrar ainda mais os meus dias e por estar sempre ao meu lado;

Aos meus amigos, presentes ou distantes, por nunca deixarem de me apoiar e estimular;

Aos meus orientadores, Renata Viegas e Rodrigo Vilar, pela paciência e atenção dedicada a mim sempre que requisitados;

E a todos aqueles que, direta ou indiretamente, colaboraram para que este trabalho consiga atingir os objetivos propostos.

RESUMO

Com o grande volume de dados gerados pelas aplicações da *Web 2.0* e a necessidade constante de manter ao máximo os serviços disponíveis e de se possuir uma estrutura flexível, é eminente o uso de NoSQL(*Not Only SQL*) por essas aplicações. Porém, muito ainda se questiona sobre as vantagens e desvantagens das soluções NoSQL. O presente trabalho apresenta um estudo comparativo de desempenho entre bancos de dados relacionais e banco de dados NoSQL, em um cenário que utiliza Junção quando usado com modelo relacional. São realizados testes nas operações de INSERT, UPDATE, DELETE e SELECT e medido quantitativamente o tempo de execução em cada operação. Cada teste foi descrito e representado em forma de gráficos , assim como criteriosamente analisados e discutidos.

Palavras-chave: NoSQL. Banco de Dados. PostgreSQL. MongoDB Apache Cassandra.

ABSTRACT

With the large volume of data generated by Web 2.0 applications and the constant need to keep the most of the available services and to have a flexible structure, is the use of eminent NoSQL (Not Only SQL) for these applications. But much yet, questions remain about the advantages and disadvantages of NoSQL solutions. This paper presents a comparative performance study between relational databases and NoSQL database, in a scenario that uses Junction when used with relational model. Tests are performed in the INSERT, UPDATE, DELETE, and SELECT operations and quantitatively measured the execution time for each operation. Each test was described and represented in the form of graphs, as well as critically examined and discussed.

Palavras-chave: NoSQL. Database. PostgreSQL. MongoDB Apache Cassandra.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ACID	Atomicidade, Consistência, Isolamento, Durabilidade
BD	Banco de Dados
CAP	*Consistency, Availability e Partition Tolerance*
CMS	*Content Management System*
DW	*Data Warehouse*
MR	Modelo Relacional
NoSQL	*Not Only* SQL
OLAP	*Online Analytical Processing*
SGBD	Sistema de Gerenciamento de Banco de Dados
SQL	*Structured Query Language*
WWW	*World Wide Web*

SUMÁRIO

1. Introdução

Com o surgimento e a popularização da *World Wide Web* (WWW) 2.0 houve um aumento contínuo de requisições por parte dos usuários em diversas aplicações computacionais que concentram um grande volume de dados. Com isso, a necessidade de armazenar informações de grande escala de forma segura, operável e flexível está cada vez maior.

Para satisfazer a essas necessidades, desenvolvedores e arquitetos de *softwares* contam com os Bancos de Dados (BD). Dentre os modelos de BD existentes, o Relacional tem um destaque maior por ser utilizado desde sua concepção e por ser o mais utilizado quando se trata de armazenamento de dados, mas, por outro lado, não possui uma estrutura capaz de promover escalabilidade[1] suficiente para lidar com grandes aplicações que dispõem de um elevado volume de dados obtendo um desempenho satisfatório.

Segundo Leite (2010, p.15 apud Brito 2010), apesar das vantagens disponibilizadas pelo uso de Sistemas de Gerenciamento de Banco de Dados (SGBDs) relacionais em cenários nos quais é necessário o gerenciamento de grandes volumes de dados, conciliando-se com a demanda cada vez mais frequente por escalabilidade, a utilização de SGBDs relacionais já não é considerada tão eficiente.

A partir dessas considerações, é notória a importância de se utilizar um modelo de BD mais flexível que trabalhe com grande número de requisições e um elevado número de informações. Para solucionar esta problemática surge um novo modelo de armazenamento que trabalha com grande número de transações, compensando a crescente necessidade de obter um bom desempenho, conhecido como NoSQL (*Not Only* SQL).

Devido à necessidade de melhorar o desempenho das aplicações, muitas empresas já estão fazendo uso deste novo paradigma que, segundo Brito (2010) tem como propósito não substituir o Modelo Relacional como um todo, mas apenas em casos nos quais seja necessária uma maior flexibilidade da estruturação do banco de dados.

[1] Escalabilidade é a capacidade de um sistema crescer e continuar atendendo requisições, dado que a carga de acesso ao mesmo aumenta. Um sistema pode ser escalável horizontalmente ou verticalmente (MARQUES 2012, p.09).

Porém, muito ainda se questiona a respeito de soluções NoSQL e um dos primeiros questionamentos sobre a utilização do mesmo é o desempenho relacionado ao tempo de execução de operações dos dados. Ou seja, muitas dúvidas surgem em relação ao quão rápido os bancos de dados NoSQL são em relação aos bancos de dados relacionais. O presente trabalho, realizando uma análise comparativa de desempenho de bancos relacionais em relação a bancos de dados NoSQL, pretende fornecer uma premissa a fim de facilitar à tomada de decisão na escolha do banco de dados.

1.1. Objetivo Geral

Apresentar uma comparação de desempenho para medir e analisar o tempo de execução de instruções necessárias para inserir, atualizar, remover e consultar dados armazenados em duas tecnologias diferentes de bancos de dados: Reacional e NoSQL.

1.2. Objetivos Específicos

- Propor cenários de testes para avaliar o desempenho de bancos Relacionais e NoSQL, em relação ao tempo de execução.
- Mostrar como as soluções NoSQL modelam determinados cenários.
- Analisar o desempenho de soluções *NoSQL* em cenários que utilizam junções no BDR
- Apresentar os resultados obtidos em cada cenário.

1.3. Estrutura do Trabalho

Este trabalho foi organizado da seguinte maneira, no capítulo 1 foram apresentados a motivação, uma breve introdução do tema abordado e o objetivo do trabalho.

No capítulo 2 são abordadas as principais características do modelo relacional e das soluções NoSQL, assim como são citados trabalhos relacionados.

No capítulo 3, desenvolveu-se o estudo de caso, onde são definidos cenários de teste, realizadas as execuções dos testes e avaliação dos resultados obtidos. Por final, são apresentadas as considerações finais do trabalho.

2. Fundamentação Teórica

Neste capítulo serão apresentadas as principais características do modelo relacional e do NoSQL, assim como suas limitações e principais abordagens.

2.1. Modelo Relacional

"O Modelo Relacional (MR) é um modelo de dados representativo (ou de implementação) que foi proposto por Ted Codd, em 1970. O modelo fundamenta-se em conceitos da matemática – teoria dos conjuntos e lógica de predicado. Os primeiros sistemas comerciais baseados no MR foram disponibilizados em 1980 e desde então ele vem sendo implementado em muitos sistemas, tais como Access, Oracle, MySql, entre outros." (ELMASRI; NAVATHE, 2011, p. 38 apud COSTA 2011, p. 33).

Surgido em meados dos anos 70, o modelo de dados relacional foi muito bem aceito na área de banco de dados, tornando-se desde então a principal solução de persistência de dados que perdura até os dias de hoje, mesmo com o surgimento de novas tendências de *software*.

Delgado (2011, p. 17) afirma que esse modelo utiliza tabelas para armazenar dados e chaves para relacionar tabelas distintas. Os dados são manipulados através de linguagens de manipulação que oferecem algumas operações como consulta, inserção, exclusão e alteração de dados. Possui características que possibilitam que a integridade dos dados seja mantida, como as regras de normalização e o paradigma ACID (Atomicidade, Consistência, Isolamento, Durabilidade). Entretanto, apresenta algumas limitações que não permitem seguir uma estrutura flexível, um alto grau de escalabilidade e modelos de dados distintos.

2.1.1. Características

O Modelo Relacional (MR) consiste em uma ou mais tabelas que são utilizadas para representar os dados e as relações. As relações são compostas de linhas (tuplas) e colunas (atributos). Segundo Heuser (2009 apud Delgado 2011, p. 17) para que a integridade dos dados seja mantida, são utilizadas chaves. As chaves identificam unicamente linhas e estabelecem os relacionamentos entre tabelas distintas. Estas podem ser primárias ou estrangeiras, as chaves primárias identificam unicamente as tuplas da tabela e determinam sua ordem física, já as estrangeiras tornam os valores de um determinado atributo dependente de

outro atributo existente, na maioria das vezes, em outra tabela (DATE, 2004 apud DELGADO 2012, p.18).

Sabe-se que um BD Relacional é um BD que segue um MR e, para gerenciar seus dados além de utilizar um SGBD também pode fazer uso de um processo de Normalização. Segundo (CASANOVA, [S/d] apud DELGADO 2011, p. 18) esse processo consiste em um conjunto de regras que são aplicadas passo a passo, examinando atributos de determinada tabela. Seu objetivo é evitar que assuntos distintos se misturem em uma mesma tabela levando à redundância. Essa mistura pode gerar inconsistência de dados prejudicando o desempenho da aplicação (JÚNIOR, 2007 apud DELGADO 2011, p. 18).

2.1.2. Operações

Segundo Ramalho (1999 apud Delgado 2011, p. 18) o Modelo Relacional utiliza a *Structured Query Language* (SQL). Essa é uma linguagem de manipulação que possibilita que sejam realizadas operações sob os dados, como inserção, consulta, exclusão, atualização e algumas outras.

A seguir, serão apresentadas as operações básicas em SQL.

- *Insert*: é o comando responsável por inserir dados em uma tabela
- *Select*: é o comando responsável por selecionar um subconjunto de dados através de um comando de seleção.
- *Delete*: é o comando responsável por excluir dados de uma tabela;
- *Update*: é o comando responsável por atualizar os registros de uma tabela;

No *Select*, para buscar dados de mais de uma tabela do banco de dados são utilizadas as junções, que é um tipo de operação que permite realizar consultas dos dados de forma consistente entre as tabelas utilizadas em uma operação. Segundo Pereira e Moreira (2012), uma junção é caracterizada pelo produto de duas ou mais tabelas, e a relação entre essas tabelas é definida através da existência de uma ou mais condições na cláusula *where* própria da sintaxe SQL, ou cláusula *on*, relacionando as duas tabelas.

2.1.3. ACID (Atomicidade, Consistência, Isolamento, Durabilidade)

Segundo (PERKUSICH, [S.d] apud Delgado 2011, p. 23) os SGBDs relacionais são característicos por apresentar segurança nas suas transações, isso ocorre devido a quatro

propriedades estabelecidas pelo paradigma ACID (*Atomicity, Consistency, Isolation, Durability*). São elas: Atomicidade, Consistência, Isolamento e durabilidade.

A propriedade de Atomicidade garante que as transações sejam atômicas, ou seja, ou serão executadas totalmente ou não serão executadas. A propriedade de Consistência garante que uma transação deve respeitar as regras de integridade dos dados. A propriedade de Isolamento garante que, em sistemas multiusuários, várias transações podem estar acessando o mesmo registro (ou parte do registro) no banco de dados ao mesmo tempo, e a propriedade de Durabilidade garante que os efeitos de uma transação devem persistir no banco de dados mesmo em presença de falhas.

2.1.4. Limitações

Mesmo utilizando um conceito que garante a integridade dos dados, o Modelo Relacional apresenta algumas limitações, como não permitir que seja oferecida uma estrutura flexível, que favoreça a obtenção de um nível mais alto de escalabilidade, além de forçar as aplicações a utilizarem esquemas fixos, não possibilitando que possam estruturar seus dados de acordo com as necessidades individuais. Segundo Delgado (2011, p. 26) a estrutura do Modelo Relacional é complexa e pouco flexível, devido a este modelo possuir algumas características como a presença de chaves, que estabelecem relacionamentos entre tabelas, e seguir algumas regras, como a normalização de dados.

A fim de suprir essas limitações novas tecnologias foram desenvolvidas, entre elas, surgiu a chamada NoSQL, que é um termo genérico para soluções de armazenamento de dados que surgiram para suprir necessidades em demandas onde os bancos de dados tradicionais (relacionais) são ineficazes. Essas tecnologias serão discutidas na próxima sessão.

2.2. Tecnologia NoSQL

Sabe-se que existem vários modelos de Banco de Dados (BD), porém com o crescimento acelerado de informações geradas diariamente, percebeu-se a necessidade de desenvolver uma alternativa para resolver problemas vinculados à estruturação do Banco de Dados, permitindo uma maior flexibilidade dos dados armazenados. Segundo Soares (2012, p. 17) o BD NoSQL é uma alternativa para lidar com esses casos, por possuir uma arquitetura diferenciada, de forma a facilitar o tratamento com alta escalabilidade de dados.

Segundo Lóscio et al. (2011), o Banco de Dados NoSQL foi proposto com o objetivo de atender aos requisitos de gerenciamento de grandes volumes de dados, semi-estruturados ou

não estruturados, que necessitam de alta disponibilidade e escalabilidade. A necessidade de uma nova tecnologia de BD surgiu como consequência da ineficiência dos bancos de dados relacionais em lidar com esta tarefa.

2.2.1. Histórico

Com o surgimento da Web 2.0, houve um aumento na produção contínua de volume de dados, gerando assim uma necessidade maior para serem armazenados e processados mais rapidamente. Esta manifestação popularizou a utilização de novas tecnologias permitindo uma interação social entre as pessoas através de ambientes que facilitam a colaboração e o compartilhamento de informações *online*.

Para exemplificar esta demanda, pode-se citar o *Facebook* que é um *site* de rede social onde os usuários podem publicar textos, fotos, vídeo, *chat*, sistema de geolocalização, linha do tempo, entre outras, se tornando um espaço de comunicação com um cenário que apresenta uma grande quantidade de dados, exigindo um maior gerenciamento e funcionamento. Segundo Diana & Gerosa (2010, p. 2 apud Almeida e Brito, 2012), para suprir essas necessidades surgiu uma nova categoria de banco de dados denominada NoSQL (abreviação para "*Not Only* SQL").

O termo *Not Only* SQL (NoSQL) foi utilizado pela primeira vez em 1998, por Carlo Strozzi, como nome de seu Sistema de Gerenciamento de Banco de Dados (SGBD), baseado no Modelo Relacional, sem interface *Structured Query Language* (SQL) (SUÍSSA [s/d] apud SOUSA e ROCHA 2010).

Em 2004, segundo Chang (2006 apud DELGADO 2011, p. 29) o Google lançou o *BigTable*, um banco de dados de alta performance que tinha o objetivo de oferecer escalabilidade e disponibilidade a aplicações. Essa foi a primeira implementação não-relacional[2] que recebeu destaque. Em 2007, a *Amazon* apresentou o *Dynamo*, um banco de dados não-relacional de alta disponibilidade utilizado por serviços *web* da empresa (DECANDIA 2007 apud DELGADO 2011, p. 29). Já em 2008, a rede social *Facebook* desenvolveu um banco de dados para manipular um grande volume de dados e o nomeou Cassandra, um sistema de banco de dados distribuído e de alta disponibilidade (BINGWEI 2010 apud DELGADO 2011, p. 29).

[2] Não-relacional: Este termo faz referência a SGBDs que não adotam o modelo relacional e são mais flexíveis quanto às propriedades ACID.

2.2.2. Banco de Dados Relacional x Banco de Dados NoSQL

Sabe-se que o BD Relacional é uma tecnologia muito cogitada quando se pensa em armazenar grande quantidade de dados, mas possui uma estrutura rígida, portanto não permite alta escalabilidade. Segundo Lóscio et al.,(2011) a tecnologia relacional foi proposta na década de 70, quando as aplicações de banco de dados caracterizavam-se por lidar com dados estruturados, ou seja, dados que possuem uma estrutura fixa e bem definida.

Para Suissa ([S/d] apud Sousa e Rocha 2010) os atuais bancos de dados relacionais são muito restritos em relação à escalabilidade, pois utilizam escalonamento vertical[3] nos servidores, quanto mais dados, maior necessidade de espaço no servidor e memória. A diferença do NoSQL é que utiliza escalonamento horizontal [4]e tem maior facilidade de distribuição do dados, ou seja, com um aumento de dados, aumenta-se o número de servidores, que podem ser ou não de alta performance, barateando e otimizando o armazenamento. É justamente neste ponto, escalabilidade, que o NoSQL destaca-se, pois este novo paradigma, segundo Lóscio et al., (2011, p. 3 apud Almeida e Brito 2012), atende "requisitos de alta escalabilidade necessários para gerenciar grandes quantidades de dados, bem como para garantir a alta disponibilidade dos mesmos". Para Delgado (2011, p. 28) a tecnologia NoSQL propõe um novo modelo de organização de dados, que permite que aplicações possuam uma estrutura mais flexível, possibilitando que sejam altamente escaláveis, além de possibilitar que os dados sejam manipulados livres de regras, podendo cada SGBD seguir um modelo de organização.

A Tabela 1 permite uma análise comparativa entre as classes de banco de dados, a partir do teorema CAP (*Consistency, Availability e Partition Tolerance*), que em português, significa: consistência, disponibilidade e tolerância ao particionamento (na Tabela 1 representada pelo escalonamento).

Essas três características são esperadas em um sistema computacional distribuído, porém segundo o teorema, no mundo real, um sistema computacional distribuído só pode garantir apenas duas dessas características simultaneamente (LEITE 2010 apud ALMEIDA e BRITO 2012). Essa abordagem pode ser observada em bases NoSQL, pois, apesar das bases de dados

[3] Escalabilidade vertical: consiste em aumentar o poder de processamento e armazenamento das máquinas. (LÓSCIO et al. 2011).

[4] Escalonamento horizontal ou *scale out* se trata do processo de dividir o banco de dados em vários servidores, sendo realizado assim o particionamento (DELGADO 2011, p. 30).

NoSQL não oferecerem a propriedade de consistência como em bases relacionais (que garantem a integridade dos dados), apresenta como pontos fortes o escalonamento e a disponibilidade (conforme visto na Tabela 1).

Tabela 1 - Análise Comparativa Modelo Relacional x NOSQL

	Modelo Relacional	**NoSQL**
Escalabilidade	Possível, mas complexo. Devido à natureza estruturada do modelo, a adição de forma dinâmica e transparente de novos nós no grid não é realizada de modo natural.	Uma das principais vantagens desse modelo. Por não possuir nenhum tipo de esquema pré-definido, o modelo possui maior flexibilidade o que favorece a inclusão transparente de outros elementos
Consistência	Ponto mais forte do modelo relacional. As regras de consistência presentes propiciam um maior grau de rigor quanto à consistência das informações.	Realizada de modo eventual no modelo: só garante que, se nenhuma atualização for realizada sobre o item de dados, todos os acessos a esse item devolverão o último valor atualizado.
Disponibilidade	Dada a dificuldade de se conseguir trabalhar de forma eficiente com a distribuição dos dados, esse modelo pode não suportar a demanda muito grande de informações do banco.	Outro fator fundamental do sucesso desse modelo. O alto grau de distribuição dos dados propicia que um maior número de solicitações aos dados seja atendido por parte do sistema e que o sistema fique menos tempo não disponível.

Fonte: (BRITO, 2010, p. 5)

Pode-se concluir que a classe de banco de dados NoSQL não tem por objetivo substituir o modelo relacional, mas atender uma nova demanda do mercado, em situações em que seja necessário de forma flexível trabalhar com cargas de dados semiestruturadas ou não

estruturadas e também em sistemas que apresentem grandes volumes de dados e necessitem de disponibilidade para seus usuários (ALMEIDA E BRITO, 2012).

2.2.3. Propriedades do NoSQL

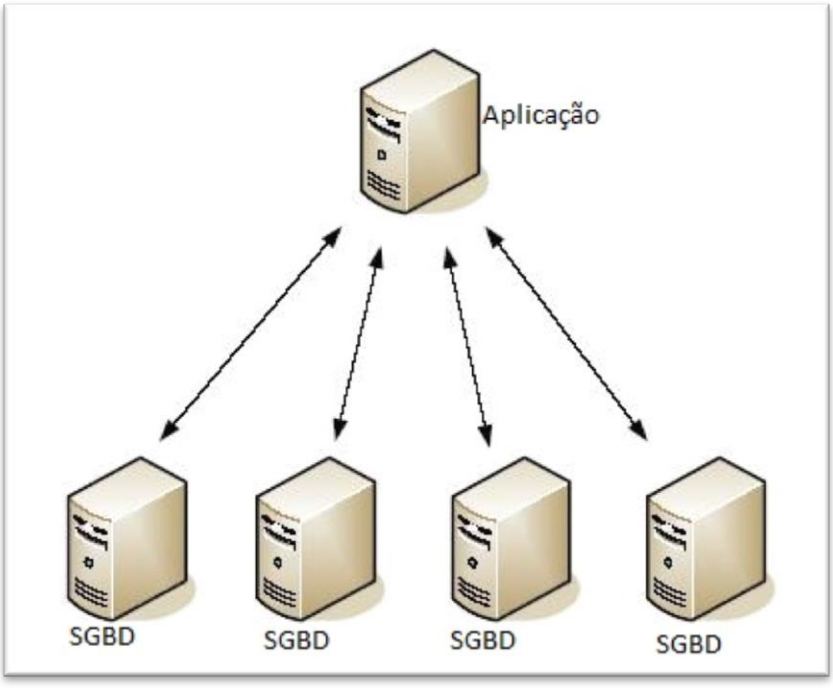
Os bancos de dados NoSQL apresentam algumas propriedades fundamentais que os diferenciam dos tradicionais sistemas de bancos de dados relacionais, tornando-os adequados para armazenamento de grandes volumes de dados não estruturados ou semiestruturados. A seguir, serão descritas alguma destas propriedades.

2.2.3.1. Escalabilidade

Segundo Delgado (2011, p. 30) a principal característica do NoSQL é a capacidade de oferecer a uma aplicação uma estrutura de dados flexível, possibilitando que seus dados sejam manipulados livres de regras e seu banco de dados seja distribuído em um nível maior do que o possibilitado se fosse seguida uma estrutura relacional ou pouco flexível.

A forma de divisão do banco de dados determina o potencial de escalabilidade, pois para cada partição existe um limite de dados a serem suportados (DELGADO, 2011, p.31). Para exemplificar, a Figura 1 ilustrará o aumento na quantidade de servidores, solução também conhecida como Escalonamento Horizontal (*scale out*) que tem sido mais utilizado nas camadas das aplicações, principalmente para a Web.

Figura 1 - Utilização do Escalonamento Horizontal



2.2.3.2. API simples de acesso aos dados

Segundo Lóscio et al. (2011) o objetivo da solução NoSQL é prover uma forma eficiente de acesso aos dados, oferecendo alta disponibilidade e escalabilidade, ou seja, o foco não está em como os dados são armazenados e sim em como poderão ser recuperados de forma eficiente. Para isto, é necessário que APIs sejam desenvolvidas para facilitar o acesso a estas informações, permitindo que qualquer aplicação possa utilizar os dados do banco de forma rápida e eficiente.

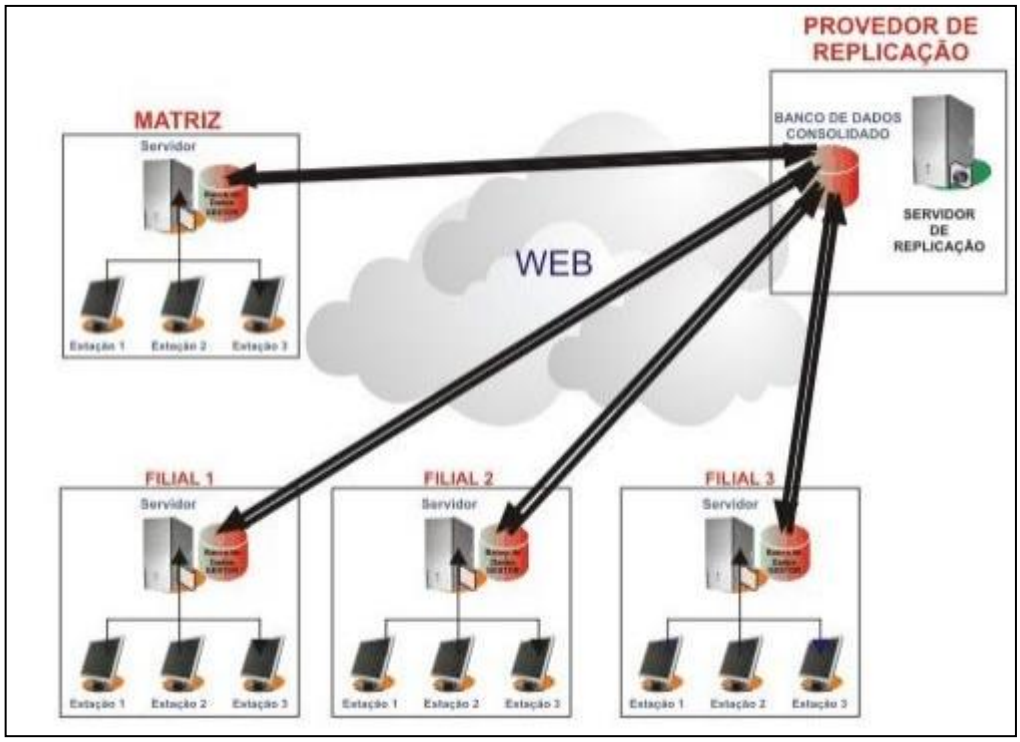
2.2.3.3. Ausência de esquema ou esquema flexível

Segundo Lóscio et al. (2011), uma característica evidente dos bancos de dados NoSQL é a ausência completa ou quase total do esquema que define a estrutura dos dados modelados. Esta ausência de esquema facilita tanto a escalabilidade quanto contribui para um maior aumento da disponibilidade. Em contrapartida, não há garantias da integridade dos dados, o que ocorre nos bancos relacionais, devido à sua estrutura rígida.

2.2.3.4. Suporte nativo à replicação

Segundo Lóscio et al. (2011), outra forma de prover escalabilidade é através da replicação. A replicação consiste em armazenar um mesmo conjunto de dados em vários servidores distintos de forma nativa, diminuindo o tempo gasto para recuperar informações. A Figura 2 ilustra uma empresa que possui seus dados replicados.

Figura 2 - Replicação de dados



Fonte: (DELGADO, 201, p.35)

Segundo Brito (2010 apud Delgado 2011, p.36), a utilização dessa técnica permite que a leitura seja potencializada, pois é realizado um balanceamento. Esse balanceamento distribui as requisições de leitura para todas as máquinas. Dessa maneira, a operação de leitura não é realizada de forma concentrada em um único servidor, mas distribuída entre todos os existentes na aplicação (OREND 2010 apud DELGADO, 2011, p.36). Esta técnica possibilita que o sistema esteja sempre disponível e, caso algum servidor falhe, o acesso aos dados seja desviado para outro servidor.

2.2.4. Taxonomia de Modelo de Dados no NoSQL

Esta seção descreve os tipos mais comuns de bancos de dados NoSQL: bancos de dados baseado em chave-valor, orientados a documentos, bancos de dados de famílias de colunas e bancos de dados de grafos.

Segundo Sadalage e Fowler (2013, p. 41), o termo “Modelo de Dados”, na maioria das vezes, serve para falar sobre o modelo pelo qual o gerenciador do banco de dados organiza seus dados – o que poderia ser mais formalmente chamado de metamodelo. Com o surgimento da tecnologia NoSQL, cada solução possui um modelo diferente, os quais foram divididos em quatro categorias amplamente utilizadas no ecossistema NoSQL: Chave-valor, Documento,

Família de Colunas e Grafos. Dessas, as três primeiras compartilham uma característica comum em seu modelo de dados – comumente chamada de "orientação agregada". O termo agregado é um conjunto de objetos relacionados que se deseja tratar como uma unidade (Sadalage e Fowler 2013, p. 42).

2.2.4.1. Sistema baseado em Chave – valor

Segundo Lóscio et al. (2011), este modelo é considerado bastante simples e permite a visualização do banco de dados como uma grande tabela *hash*. De maneira bem simples, o banco de dados é composto por um conjunto de chaves, as quais estão associadas a um único valor, que pode ser uma *string* ou um arquivo binário.

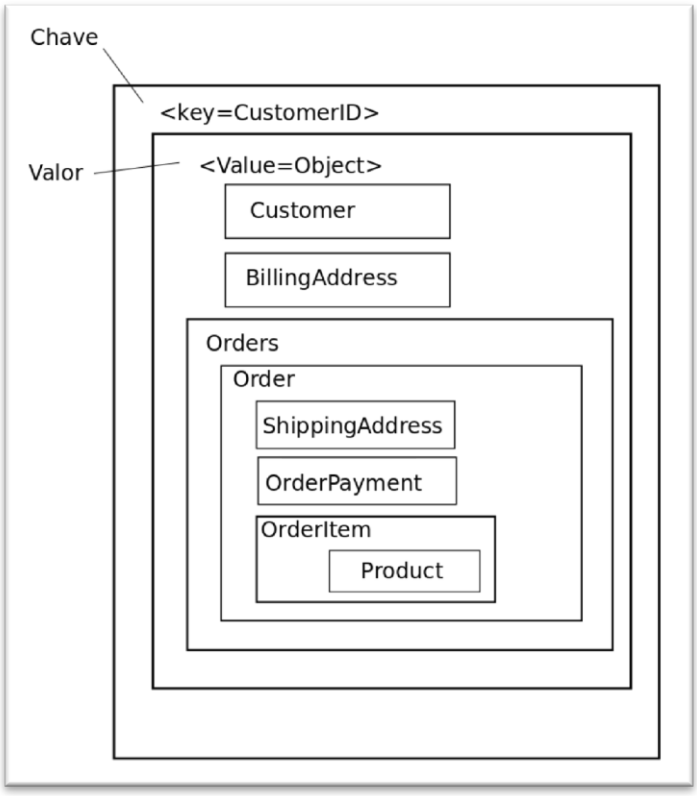
Delgado (2013, p.40) afirma que, embora os sistemas chave-valor sejam os mais simples, são os que suportam maior volume de dados e potenciais de escalabilidade. Segundo Toth (2011 apud Almeida e Brito 2012), o modelo chave-valor na maioria dos casos apresenta uma interface composta principalmente pelos seguintes métodos:

- Put(chave, valor) – representa o método que insere um registro e garante maior velocidade a operação;
- Get(chave) – responsável por recuperar um registro, não oferece opções para buscas mais complexas;

Existem vários tipos de banco de dados do tipo chave- valor, mas os mais populares segundo Sadalage e Fowler (2013, p.124) são o *Riak*, Redis (muitas vezes chamado de servidor *Data Structure*), *Memcached DB* e suas variáveis, *Amazon DynamoDB*, entre outros. Para exemplificar um sistema baseado em chave-valor, tem-se o *Riak* que permite adicionar metadados e agregados para indexação e conexões entre interagregados, pois possui um recurso de consulta que utiliza pesquisas do tipo *Solr*[5], buscando em quaisquer agregados que estejam armazenados como estruturas XML ou JSON. A Figura 3 demostra como é a modelagem de um sistema baseado em chave valor.

[5] Solr: Solr é uma plataforma de busca NoSQL. Suas principais características incluem pesquisa completa em textos estruturados e não estruturados, destaques das informações, exibição de resultados de uma busca, integração com banco de dados, analise em documentos ricos como, por exemplo: DOC, XLS, PDF e MP3, além de permitir a realização de buscas geoespaciais. (COSTA, Marcelo. 2013)

Figura 3- Dados armazenados em chave-valor



Fonte: (SADALAGE E FOWLER, 201, p.67)

- **Recursos dos depósitos chave-valor**:
  - ***Consistência***: é aplicável apenas às operações em uma única chave; gravações otimistas podem ser executadas, mas são muito caras para implementar.
  - ***Transações***: Diferentes produtos no armazenamento de chave-valor têm diferentes especificações de transação; Não há garantia nas transações.
  - ***Recursos de consultas***: Todos os armazenamentos de chave-valor podem ser consultados pela chave; Realizar consultas por chave otimiza muito o tempo de consulta.
  - ***Estrutura de dados***: banco de dados chave-valor não se importa com o que é armazenado na parte do valor do par chave-valor; o valor pode ser um *blob*, texto, JSON, XML e etc.
  - ***Escalabilidade***: muitos armazenamentos de chave-valor podem ser escalados utilizando a fragmentação.

- Soluções apropriadas utilizando chave-valor:

Sabe-se que existem alguns cenários para os quais os armazenamentos de chave-valor são uma solução apropriada (Sadalage e Fowler 2013, p.131), a citar:

- **Perfis de usuário, preferências:** Todos os usuários possuem um único *userId* (ID do usuário) e um único *usernam*e (nome do usuário), ou algum outro atributo único, além de preferências, como idioma, cor etc. Tudo isso pode ser colocado em um objeto, de modo que as preferências de um usuário sejam obtidas por meio de uma única operação *GET*.
- **Dados de carrinho de compras:** *Websites* de comércio eletrônico possuem carrinhos de compras associados ao usuário. Para os carrinhos de compras estarem sempre disponível em vários navegadores, máquinas e sessões, todas as informações de compras podem ser colocadas no valor onde a chave é *userid.*

2.2.4.2. Sistemas Orientados a Documentos

Como o próprio nome diz baseia-se em documentos. Segundo Lóscio et al., (2011) um documento, em geral, é um objeto com um identificador único e um conjunto de campos, que podem ser *strings*, lista ou documentos aninhados. No modelo orientado a documentos tem-se um conjunto de documentos e em cada documento um conjunto de campos (chaves) e os valores deste campo. Outra característica importante é que este modelo não depende de um esquema rígido, ou seja, não exige uma estrutura fixa como ocorre nos bancos de dados relacionais. Assim, é possível que ocorra uma atualização na estrutura do documento, com a adição de novos campos, por exemplo, sem causar problemas ao banco de dados.

Como principais aplicações que adotam o modelo orientado a documentos destacamos *CouchDB, MongoDB, RavenDB, Scalaris, Jackrabbit* (KUBACKI, 2010 apud Delgado 2011, p. 39).

- Soluções apropriadas utilizando Documentos: Sabe-se que existem alguns cenários para os quais os armazenamentos de documentos são uma solução apropriada (Sadalage e Fowler 2013, p.145), a citar:
  - **Sistema de Gerenciamento de Conteúdo (CMS), plataformas de *blog*:** Como os bancos de dados de documentos não têm esquemas predefinidos e geralmente entendem documentos JSON, eles funcionam bem em sistemas de gerenciamento de

conteúdo ou aplicativos para publicações de *websites*, gerenciando os comentários dos usuários, seus registros, perfis e documentos visualizados pela web.

- **Análise web ou em tempo real (*analytics*):** bancos de dados de documentos podem armazenar dados para análises em tempo real; uma vez que partes do documento podem ser atualizadas, é muito fácil armazenar visualizações de páginas ou visitantes únicos. Além disso, novas métricas podem ser adicionadas com facilidade, sem alterações no esquema.

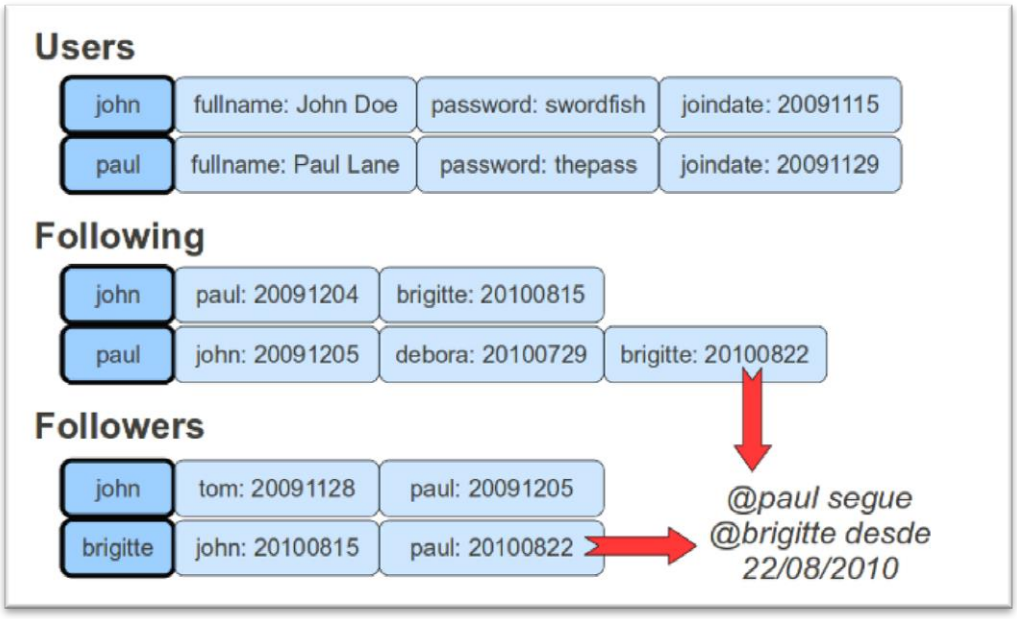
2.2.4.3. Sistemas Orientados a Colunas

Este modelo é um pouco mais complexo do que o modelo chave-valor e, neste caso, muda-se o paradigma de orientação a registro ou tuplas (como no modelo relacional) para orientação a atributos ou colunas (modelo NoSQL) (LÓSCIO et al., 2011). Este modelo se contrapõe ao paradigma do modelo relacional de armazenamento em tuplas (instâncias dos atributos em formato de linha), pois no modelo baseado em família de colunas "os dados são indexados por uma tripla (linha, coluna e timestamp), onde linhas e colunas são identificadas por chaves e o timestamp permite diferenciar múltiplas versões de um mesmo dado" (LÓSCIO et al., 2011, p. 6 apud Almeida e Brito 2012).

Como principais soluções que adotam o modelo orientado a colunas destaca-se o *BigTable* da *Google*, pois permite particionamento dos dados, além de oferecer forte consistência, e foi pioneiro para esse modelo de dados. Outro exemplo é o Cassandra, que foi desenvolvido pelo *Facebook* e é baseado no *Dynamo*.

Para ilustrar a modelagem de dados orientada a colunas, um grande exemplo muito comum utilizado são as redes sociais, tais como Twitter. Na Figura 4 pode-se ver um exemplo de modelagem em famílias de colunas.

Figura 4 - Modelagem Família de Colunas



Fonte: Página do Demoiselle Framework

- Soluções apropriadas utilizando Família de Colunas: Sabe-se que existem alguns cenários para os quais os bancos de dados de família de colunas são soluções apropriadas (Sadalage e Fowler 2013, p.159), a citar:
  - **Sistema de Gerenciamento de Conteúdo (CMS), plataformas de *blog*:** utilizando famílias de colunas, pode armazenar entradas de *blogs* com *tags*, categorias, *links* e *trackbacks* em diferentes colunas. Comentários podem ser armazenados na mesma ou movidos para um *keyspace* diferente; de forma semelhante, os usuários dos blogs e os próprios blogs podem ser colocados em diferentes famílias de colunas.
  - **Registro de eventos (log):** É uma ótima escolha para armazenar informações sobre eventos, como, por exemplo, os estados do aplicativo ou erros encontrados por ele.

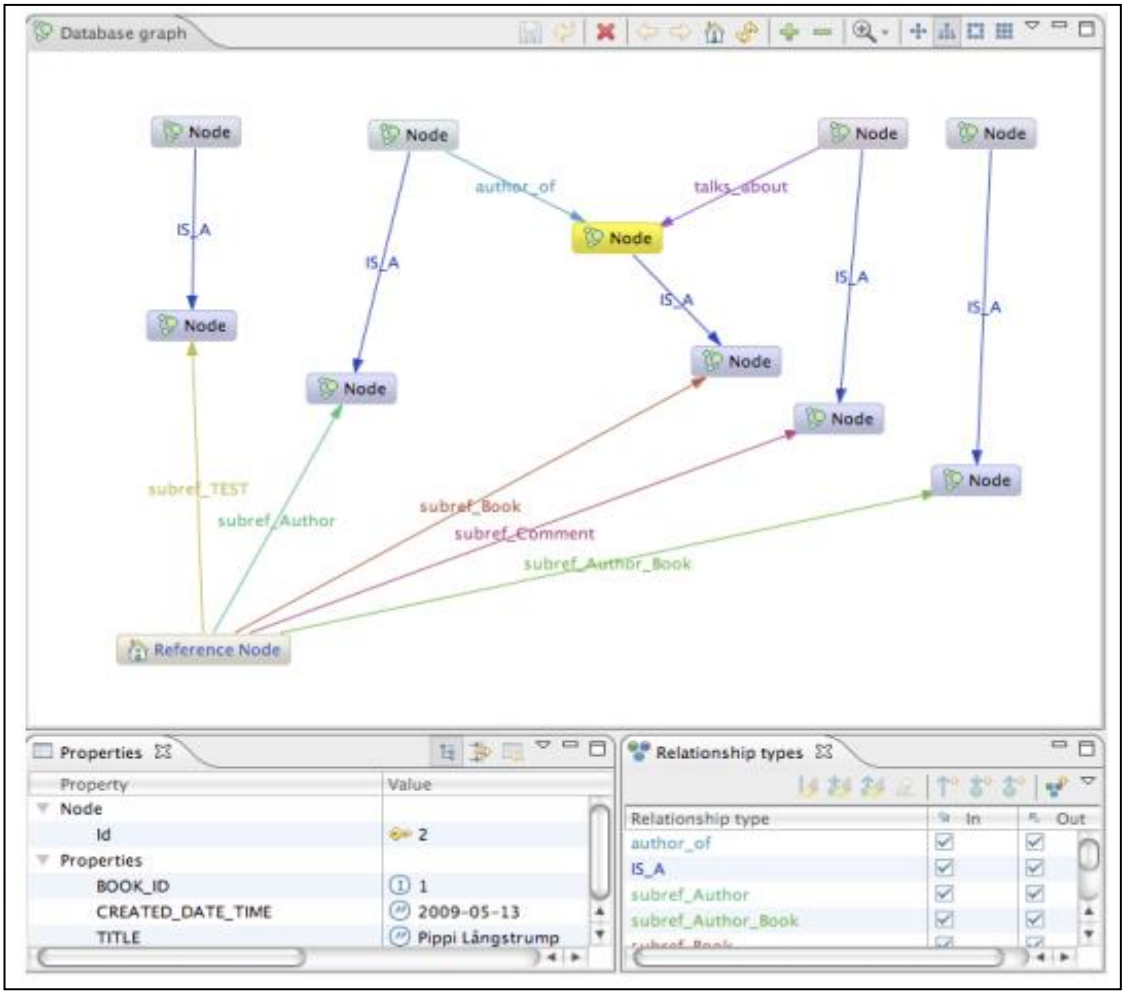
2.2.4.4. Sistema Orientado a Grafos

O modelo orientado a grafos possui três componentes básicos: os nós (são os vértices do grafo), os relacionamentos (são as arestas) e as propriedades (ou atributos) dos nós e relacionamentos. Neste caso, o banco de dados pode ser visto como um multigrafo rotulado e direcionado, onde cada par de nós pode ser conectado por mais de uma aresta.

A vantagem de utilização do modelo baseado em grafos fica bastante clara quando consultas complexas são exigidas pelo usuário. Comparado ao modelo relacional, que para estas situações pode ser muito custoso, o modelo orientado a grafos tem um ganho de performance permitindo um melhor desempenho das aplicações.

Exemplos de bancos de dados baseados em grafos são: Neo4j14, AllegroGraph15 e Virtuoso16. O Neo4j é um exemplo de banco que segue esse modelo (KUBACKI, 2010 apud Delgado 2011, p. 43). Um exemplo da estrutura de nós ou *node* desse Banco de Dados está sendo mostrado na Figura 5. Através desse exemplo, pode-se visualizar de forma mais clara como os nós de um Banco de Dados com grafos se interligam.

Figura 5- Sistema baseado em grafos, Neo4j



Fonte: (DELGADO, 2012, p.44)

- Soluções apropriadas utilizando grafos:

Sabe-se que existem alguns cenários para os quais os bancos de dados de grafos apresentam soluções apropriadas (Sadalage e Fowler 2013, p.173), a citar:

- **Dados Conectados:** É nas redes sociais que os bancos de dados de grafos podem ser instalados de maneira muito eficaz. Esses grafos sociais não têm de ser apenas do tipo que relaciona amigos; eles podem, por exemplo, representar funcionários, seu conhecimento e onde trabalham com outros funcionários em diferentes projetos. Qualquer domínio rico em *links* é apropriado para banco de dados orientados a grafos. Os relacionamentos entre entidades de diferentes domínios (como social, espacial, comercial) em um único banco de dados, podem tornar esses relacionamentos mais valiosos, fornecendo a eles a capacidade de percorrer o domínio.
- **Roteamento, envio e serviços baseados em localização:** Todo local ou endereço que possui uma entrega é considerado um nodo, e todos os nodos onde a entrega tiver de ser efetuada pelo entregador pode ser modelado na forma de um grafo de nodos. Os relacionamentos entre os nodos podem ter a propriedade de distancia, permitindo, assim que a entrega das encomendas sejam realizadas de um modo eficiente.

2.2.5. Sistemas de Gerenciamento de Banco de Dados NoSQL

Como as grandes empresas hoje dispõem de um grande fluxo de informações, passaram a desenvolver seus próprios SGBDs baseados na filosofia NoSQL. A seguir a Tabela 2 descreve alguns exemplos de BD:

Tabela 2- Características dos SGBDs NoSQL e casos de sucesso

SGBDs	Características	Casos de sucesso
BigTable	Foi desenvolvido para distribuir dados por centenas de servidores e escalar por conjuntos de dados de até 1 petabyte O *BigTable* é proprietário, mas o modelo de dados existe em implementações de código aberto. Ele pode ser usado como *input* ou *output* para o *MapReduce*, que ativa o processo de distribuição de arquivos ou banco de	Google: A Google desenvolveu sua própria solução NoSQL, chamada de *BigTable* que é um sistema de armazenamento distribuído para gerenciar dados estruturados em larga escala. Esta solução permite a escalabilidade de recursos, bem como alta performance no processamento das consultas, dos

	dados usando funções de mapeamento e redução.	processos e dos serviços.
Dynamo	Desenvolvido em 2007, foi criado para oferecer armazenamento de valores-chaves de dados de alta disponibilidade, permitindo atualizações para sobreviver às falhas de servidor e rede.	Amazon: Um dos grandes desafios enfrentados pela Amazon.com diz respeito à confiabilidade do grande volume de dados gerenciado por suas aplicações, não apenas por questões financeiras e devido aos gastos com soluções convencionais, mas também por causa do impacto da confiança de seus cliente em seus produtos. Em 2007, com o intuito de garantir alta disponibilidade dos dados de seus serviços "*always-on*", a *Amazon* desenvolveu uma solução NoSQL, o *Dynamo*. Como resultado da adoção desta nova tecnologia, diversos serviços da *Amazon* tem se mantido disponíveis em 99,9995% das requisições realizadas.
Apache Cassandra	É um sistema de banco de dados distribuído, altamente escalável, que foi desenvolvido na plataforma Java, reuni a arquitetura do *Dynamo*, da *Amazon* e o modelo de dados do *BigTable*, do *Google*. O Cassandra exerce com excelência a função de repositório de dados. Teve seu código-fonte aberto à comunidade em 2008. Agora é mantido por desenvolvedores da fundação Apache e colaboradores de outras empresas	*Facebook*: Após seis anos de sua criação, o *Facebook* possui, atualmente, cerca de 3,5 bilhões de conteúdos (*links*, posts, etc) compartilhados por semana. Para evitar problemas com a escalabilidade e disponibilidade dos dados, a empresa desenvolveu o Cassandra, uma solução NoSQL. Inicialmente criado para otimização do sistema de busca do Facebook, atualmente o Cassandra é utilizado para dar suporte à replicação, detecção de falhas, armazenamento em cache dentre outras funcionalidades.
MongoDB	MongoDB é uma aplicação de código aberto, de alta performance, sem esquemas, orientado a documentos. Foi	*CartolaFC*: É um fantasy game da Globo, é a maior aplicação dinâmica do portal, com mais de 2 milhões de

	escrito na linguagem de programação C++. Seu desenvolvimento começou em outubro de 2007 pela 10gen. E teve sua primeira versão pública lançada em fevereiro de 2009.	usuários cadastrados e quase 90 milhões de pageviews somente no mês de Junho de 2011 (Amorin, 2011). Com o crescimento do game, a equipe de desenvolvimento decidiu por migrar o banco de dados do Mysql para MongoDB e com isso obtiveram as seguintes vantagens: - Velocidade (2x mais rápido que o MySQL) - Sem necessidade de um ORM (não tem abstração de tabela, por exemplo) - Acesso mais natural aos dados (não tem que escrever query) - Sem schema / sem migrations (com exceções, mas em geral não há problema de executar migration)

2.3. Trabalhos Relacionados

Existem vários aspectos para serem mensurados sobre desempenho de Banco de Dados (BD). Atualmente os cenários da *web* 2.0 tem se mostrado problemáticos devido ao gerenciamento exacerbado de volume de dados, pois não se mostram mais tão eficientes.

Para verificar a problemática que os Sistemas de Gerenciadores de Banco de Dados (SGBDs) Relacionais enfrentam para gerenciar sistemas que dispõem de um grande volume de dados, tem-se o trabalho de Brito (2013) que mostra uma análise comparativa fazendo uso de uma nova geração de Banco de Dados conhecida como NoSQL versus SGBDs Relacionais. Esta análise foi motivada principalmente pela questão da escalabilidade do sistema como uma alternativa para aplicar em sistemas que tem o crescimento cada vez mais intensificado de volume de dados.

Esse trabalho se difere dos demais, pois faz uma análise comparativa de desempenho com as principais operações de banco de dados, usando um cenário que usa de muita junção no modelo relacional, realizando testes em variados volumes de dados.

Brito (2013) afirma que devido ao intenso crescimento do número de aplicações, soluções, recursos e tudo o que se refere a sistemas computacionais nas últimas décadas, o volume de dados associados a tais tipos de sistemas também teve um ritmo de crescimento acelerado.

Outro projeto que faz alusão ao trabalho citado anteriormente apresentando um diferencial como solução para manipulação de grande quantidade de volume de dados é o projeto de Soares (2012), que apresenta os BDs NoSQL em especial o Modelo Orientado a Colunas, destacando as principais características e diferenças experimentais entre alguns SGBDs de Modelo Relacional e Colunar em diferentes situações de escalabilidade. Nesta análise foi verificada que o Modelo Colunar é mais estável em termos de escalabilidade do que o Modelo Relacional.

Para Soares (2012, p. 17) com o crescimento e diversidade de informações geradas a todo tempo, os modelos relacionais nem sempre são a melhor alternativa para manipulação de dados. Isso ocorre por diversos fatores, como a forma de armazenamento dos dados em disco, a forma com que o executor de consultas recupera as informações e monta as tabelas e o método de compressão utilizado. Principalmente em aplicações com grandes quantidades de dados, onde consultas complexas são executadas, o modelo relacional enfrenta redução significativa em seu desempenho na leitura de informações.

Han et al. (2011 apud Soares 2012, p. 8) afirmam que os Bancos de Dados NoSQL surgiram para satisfazer necessidades como: (i) Armazenamento de grandes volumes de dados de forma eficiente e requerimentos de acesso; (ii) Alta escalabilidade e disponibilidade; e, (iii) Menor custo operacional e de gestão. Para exemplificar a questão da disponibilidade pode ser citado o caso do *Twitter*. Em 2008, enquanto ainda utilizava o *PostgreSQL*, o site que funciona como rede social ficou 84 horas for do ar. Em 2009, após a utilização do Cassandra, esse tempo foi sensivelmente reduzido para 23 horas e 45 minutos (LAKSHMAN e MALIK 2008 apud BRITO 2013).

O NoSQL pode ser utilizado como uma alternativa aos Bancos de Dados Relacionais objetivando o desempenho no processamento de consultas de implementações Data *Warehouse* (DW) utilizando OLAP (*Online Analytical Processing*) para analisar um grande volume de dados sobre varias perspectivas. Esta atividade é mostrada no trabalho de Carnil et al. (2012) que investigam e comparam implementações de DW usando banco de dados

relacionais e NoSQL avaliando o tempo de resposta no processamento de consultas, o uso de memória e o uso porcentual de CPU, considerando as consultas do Star Schema Benchmark.

2.4. Considerações Finais

Nesse capítulo foram apresentadas as principais propriedades dos modelos relacionais, suas características e limitações. Além de uma introdução a NoSQL, o que é, os principais modelos de dados adotados, as principais diferenças em relação ao modelo relacional e suas principais características.

No próximo capitulo serão apresentados vários testes de desempenho em três banco de dados distintos, que utilizam modelo de dados diferentes. Os testes são quantitativos e medem o tempo de resposta de operações como INSERT, DELETE, UPDATE e SELECT.

3. Desenvolvimento

No processo de desenvolvimento de *software* vários aspectos são considerados a fim de adquirir o máximo de qualidade possível como: disponibilidade, segurança, usabilidade, desempenho, dentre outros.

O desempenho de um sistema na engenharia de *software* é um atributo que requer muito cuidado, pois fatores como *hardware*, linguagem de programação, algoritmo, banco de dados, podem influenciar na performance do sistema. Por exemplo, um sistema que possui um banco de dados relacional, com várias tabelas relacionadas, provavelmente exigirá muito dos recursos de junções. Esse recurso por sua vez exige um alto processamento, dado que o produto de uma junção entre duas tabelas A e B é igual a quantidade de tuplas da tabela A vezes a quantidade de tuplas da tabela B, quando não usado filtros.

Visto isto, a proposta desse trabalho é apresentar um estudo comparativo de desempenho entre três soluções de persistências de dados: usando o Modelo Relacional, com o SGBD PostgreSQL, e usando a tecnologia NoSQL, através de um Modelo Orientado a Documentos, com o *MongoDB*, e um Modelo de Famílias de Colunas, com o Cassandra. Foram escolhidos esses modelos de dados, pois acredita-se que estes são os mais viáveis para modelar o caso de uso proposto.

Foi criado um cenário fictício que serviu como base para todo o desenvolvimento. A partir desse cenário foram feitas as modelagens dos dados para cada banco de dados, *PostgreSQL, MongoDB e Cassandra*, e realizados testes de inserção, alteração, pesquisa e remoção de dados. O objetivo dos testes foi comparar o desempenho de cada operação em cada BD.

Salienta-se que o cenário foi propositalmente criado para forçar junções entre tabelas no modelo relacional a fim de obter o objetivo específico de analisar o desempenho de soluções *NoSQL* em cenários que utilizam junções no BDR.

3.1. Especificação do Cenário

O cenário criado foi fictício e serviu como base para criação do modelo de dados das três soluções de persistência. A seguir pode-se ver uma descrição do cenário.

"A Lig Corretora é uma empresa especializada em intermediar a aquisição de linhas de telefonia móvel ou fixa. Os clientes da Lig são pessoas físicas ou jurídicas que procuram o

melhor plano de telefonia dado as suas necessidades. A Lig é responsável por intermediar o negócio entre o cliente (pessoa) e uma das operadoras de telefonia credenciadas a Lig.

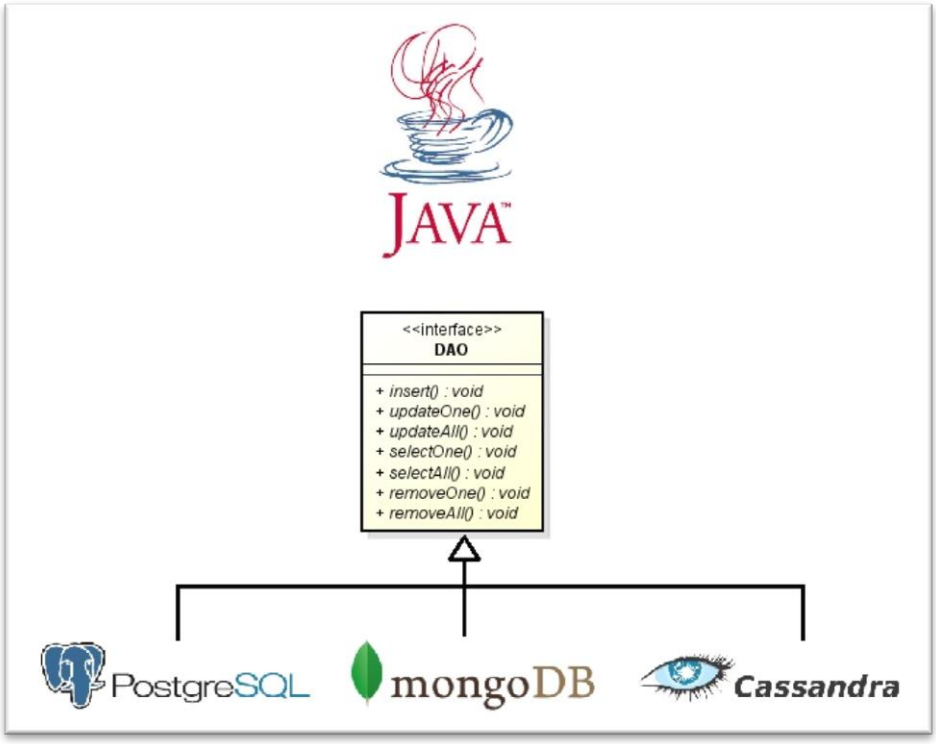
A Lig precisa de um sistema de gerenciamento de seus clientes onde possa realizar cadastros tanto das pessoas quanto das operadoras credenciadas. Assim como intermediar a venda de linhas telefônicas.

Uma operação de venda, caracteriza-se por uma associação entre uma pessoa e uma operadora através de um telefone."

3.2. Arquitetura do sistema

Assim como o cenário, os dados que alimentaram as bases de dados também foram fictícios. Para facilitar as operações realizadas aos bancos de dados, criou-se uma aplicação para realizar os testes. A Figura 6 representa como ficou a arquitetura adotada.

Figura 6 - Arquitetura do Sistema



A linguagem de programação adotada foi Java, por possuir API's para todos os BD's utilizados, além do amplo suporte dado pela comunidade de desenvolvedores da mesma.

Foi criada uma interface DAO[6] com as operações a serem realizadas, pelo menos três classes implementaram essa interface, um para a conexão e lógica aplicada junto com o

[6] DAO: Objeto de acesso a dados, um acrônimo de Data Access Object.

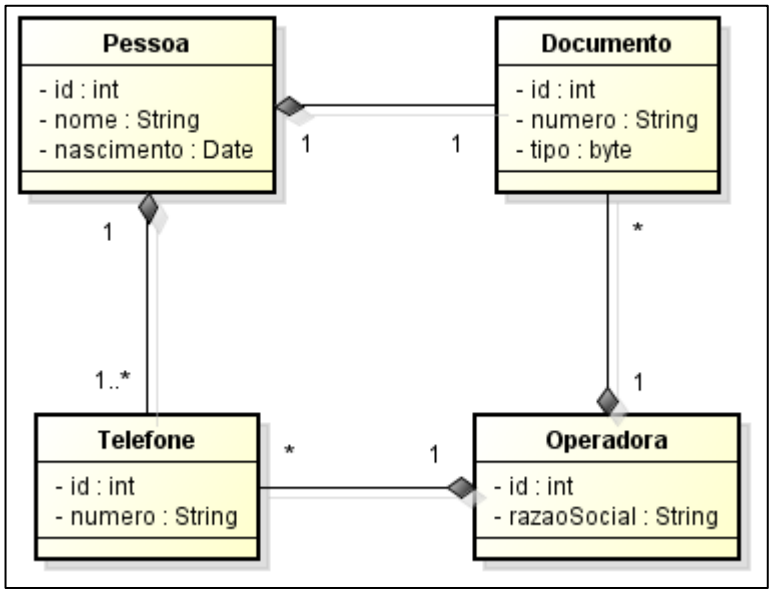
PostgreSQL, outra para o MongoDB e uma terceira para o Cassandra. Para fazer a comunicação entre o banco e a aplicação, foram usadas as API's java fornecidas por cada BD, sem o uso de nenhum *framework*. Dessa forma, o cenário fica o mais próximo possível do tempo gasto por cada operação do Banco de Dados. Toda a lógica de geração de dados para operações de inserção e atualização foi feita em uma camada superior, não influenciando no tempo.

Portanto, o tempo decorrido para cada operação incluirá a camada de aplicação java, pois os teste não foram feitos diretamente no bancos de dados. Seguindo a arquitetura exposta na Figura 6, criou-se uma lógica que é comum a todos os BD's, sendo o processamento igualmente distribuído, inibindo dessa forma o risco de beneficiar um ou outro banco de dados, pois a camada de persistência ficou totalmente isolada.

3.3. O que persistir?

De acordo com o cenário descrito no item 3.1 foi elaborado um diagrama de classes (ver Figura 7) que representa as entidades da aplicação. As propriedades dessas entidades representam o que foi persistido por cada BD.

Figura 7- Diagrama classes



Um cadastro de um cliente dá-se pelo registro dos dados da pessoa, associado a um documento e a pelo menos um telefone. Um telefone por sua vez, pertence a uma operadora e uma operadora pode ter vários documentos.

As demais classes da aplicação foram inibidas, pois modelá-las vai além do escopo proposto pelo trabalho em questão. Apenas a diagramação das classes de entidades é suficiente para o nosso contexto.

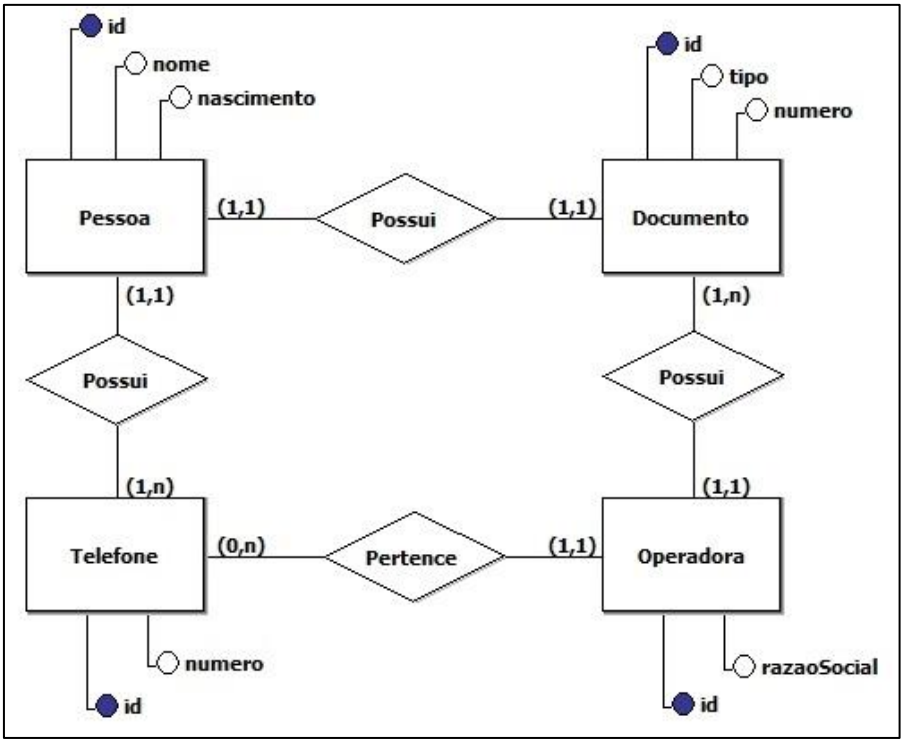
3.4. Modelando a aplicação

Nessa seção será apresentado como ficou a modelagem dos dados de cada solução de persistência.

3.4.1. Modelo relacional

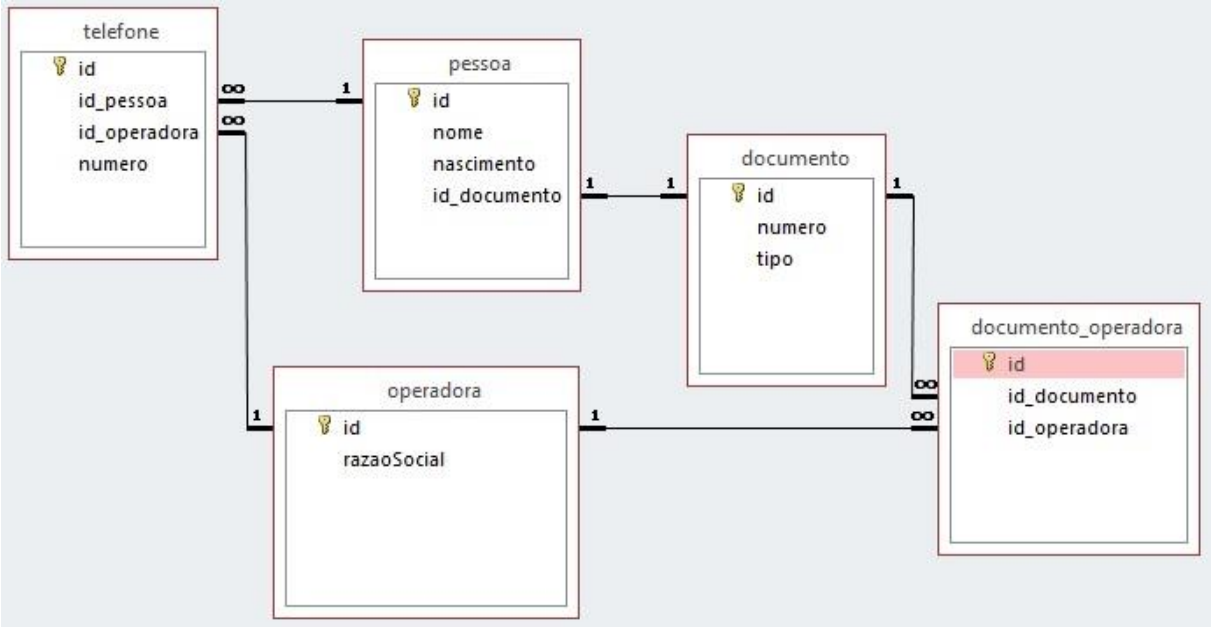
Na Figura 8 é apresentado o diagrama entidade relacionamento (ER) que descreve os requisitos de dados da aplicação, onde cada pessoa possui um id único, um nome e uma data de nascimento; cada documento possui um id, um número e um tipo (Tipo é um inteiro que representa o tipo do documento, por exemplo, 1 – CPF, 2 – CNPJ); cada telefone possui um id e um número, e cada operadora possui seu identificador (id) e uma razão social. Uma pessoa possui um Documento e pelo menos um telefone, cada telefone pertence a uma operadora que por sua vez tem pelo menos um documento.

Figura 8- Modelo Entidade Relacionamento - ER



A partir desse modelo conceitual, desenvolveu-se o diagrama lógico de dados relacional, conforme a Figura 9.

Figura 9 - Diagrama Lógico



O modelo lógico, é uma representação fiel de como ficou a base de dados no *PostgreSQL* que implementa o modelo relacional.

3.4.2. Modelo orientado a documentos

Como visto anteriormente, o paradigma *NoSQL* é livre de esquema, ele não exige uma estrutura prévia para utilização da aplicação e a estrutura pode ser alterada livremente em tempo de execução. Para o cenário em questão, optou-se por agregar todas as informações em um único documento. Assim, quando uma consulta for feita, em apenas uma consulta ao banco obtêm-se todos os dados necessários e prontos para uso.

Se o cenário utilizado fosse real, os dados de operadora e de telefone raramente seriam alterados, esse fator incentiva o uso de apenas um documento para reunir todos os dados do cliente.

Segundo Lennon (2013), o MongoDB armazena dados dentro de documentos semelhantes a JSON (usando BSON — uma versão binária de JSON), que retém os dados usando pares de chave/valor. Para exemplificar melhor, o Quadro 1 representa um documento Json que demonstra como ficou o modelo.

```
{ "_id" : ObjectId("5336ef18e4b0a99de4f8f0ef"),
"nome" : "Maria Joaquina de Amaral",
"documento" : { "tipo" : 1, "numero" : "00745678234" },
"nascimento" : ISODate("1979-11-01T16:04:40.884Z"),
"telefones" : [ { "operadora" : {
                        "documentos" : [ {"tipo" : 2,"numero" : "04164616000159" },
                            {"tipo" : 3, "numero" : "77115080" } ],
                            "razaoSocial" : "Oi" },
                            "numero" : "558391342734" }]}
```

Quadro 1- Modelo Orientado a Documento

O documento Json demonstra um cadastro de uma cliente de nome Maria Joaquina, que possui um telefone da operadora Oi; a operadora Oi possui dois documentos cadastrados.

3.4.3. Modelo orientado a colunas

Para o contexto da aplicação, seria possível fazer uma modelagem utilizando família de colunas muito similar a modelagem relacional, porém sem a consistência que o MR possui, por exemplo, no modelo de colunas não existe o conceito de chave estrangeira. Porém existe o de chave secundária, que consiste em usar o mesmo identificador único para relacionar tabelas distintas, mas não garante a integridade entre as tabelas.

NoSQL não se preocupa com normalização, deve-se pensar na modelagem de acordo com as consultas que serão realizadas, mesmo que isso ocasione a duplicação de dados. Foi pensando nas consultas à realizar que foi criado o esquema exposto na Figura 10. Onde se tem duas famílias de colunas, uma documento_telefone e uma operadora. É fácil perceber que seguindo a forma como foi modelado os dados, caso uma pessoa tenha mais de um telefone, resultará em dois registros na família de coluna documento_telefone praticamente idênticos, mudando apenas o Telefone e, dependendo, a Operadora_id.

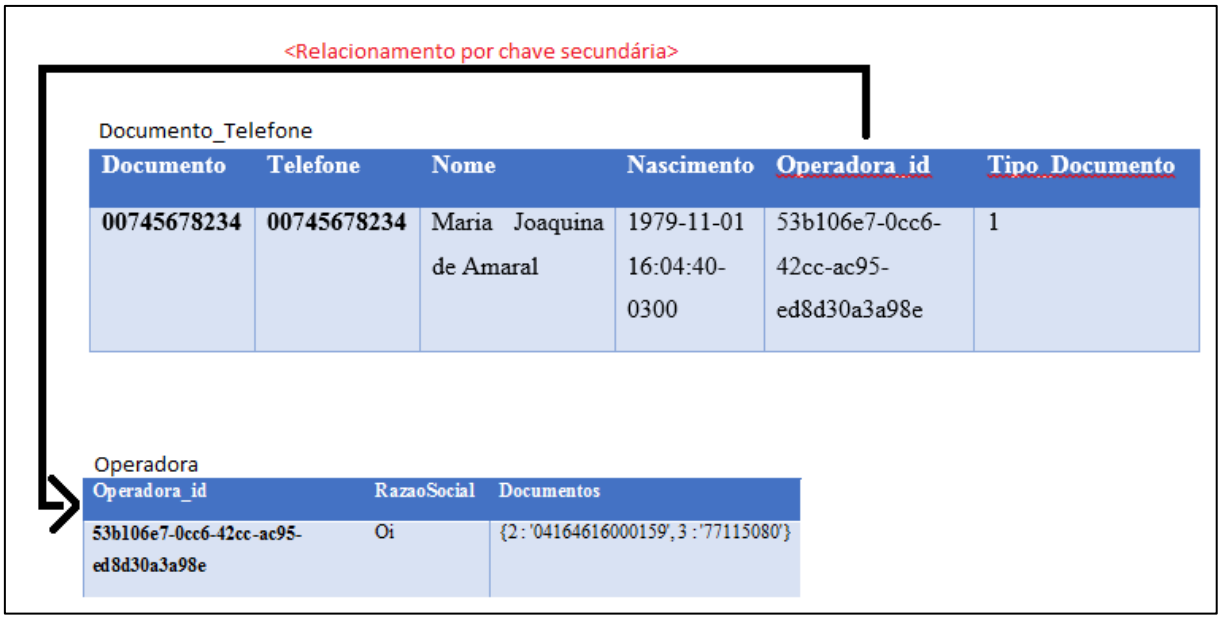


Figura 10 - Modelo Orientado a Colunas

Pode-se observar que a tabela documento_telefone possui uma coluna operadora_id, a qual o valor é um identificador único, que servirá como chave para a tabela operadora.

Outro fator que chama atenção é o tipo de dado armazenado na coluna [documentos] da tabela operadora, o tipo de dados é um map. Onde a chave do map representa o tipo do documento e o valor, o número.

3.5. Ambiente de Testes

O ambiente de testes usado para comparar os tempos de execução de instruções SQL com os comandos dos bancos NoSQL contou com um notebook equipado com as configurações descritas na tabela 3.

Dispositivo	Descrição
Processador	Intel(R) Core(TM) i7-2640M CPU @ 2.80GHz
Memória cache	4 MB
Memória RAM	6,00 GB
HD	1 TB SATA 2
Sistema operacional	Windows 7 64 bits

Tabela 3 - Configurações do Computador

Contou ainda com um sistema de virtualização Oracle VM VirtualBox, onde uma máquina virtual foi criada com 1 processador virtual, 3 GB de memória RAM e 25 GB de espaço em disco.

Esta configuração da VM foi suficiente para suportar os testes utilizando o sistema operacional Elementary Luna 64 bits. As versões dos bancos de dados são descritas na tabela 4, a seguir:

Software	Versão
PostgreSQL	9.1.12
MongoDB	2.4.9
Cassandra	2.0,6 com Cql 3.1.1
Java	1.7.0_5

Tabela 4 - Lista de Softwares Utilizados

3.6. Realização dos testes e resultados obtidos

Nessa seção é descrito como foram feitos os testes, onde, primeiramente se povoou o banco de dados através da operação de INSERT, utilizando as modelagens descritas na seção anterior, e também os testes de SELECT, UPDATE e DELETE, os testes medem a diferença entre os tempos de execução de todas essas operações nos banco de dados PostgreSQL, MongoDB e Cassandra.

Como visto na seção 3.3, foram utilizados diversos tipos de dados, como: String (Varchar), inteiro e date. Com a finalidade de evidenciar que é possível a cada banco persistir os mais variados tipos de dados.

Para medir o tempo de execução de cada comando, foi utilizado o método currentTimeMillis() da classe java.lang.System, o qual retorna o horário atual do sistema em milissegundos. Sendo assim, a aplicação armazena em uma variável o tempo inicial em seguida executa a operação, captura o tempo final e enão realiza a diferença entre o tempo final e o inicial obtendo assim o tempo de execução. Como pode ser visto no quadro a seguir.

```
1  long inicio = System.currentTimeMillis();
2
3  metodoDao();
4
5  long fim = System.currentTimeMillis();
6
7  System.out.println(fim - inicio);
```

Quadro 2- Calculo utilizado para medir o tempo gasto.

3.6.1. INSERT

Para os testes de inserção criou-se um método de controle responsável por criar dados fictícios e inseri-los à base de dados, a declaração do método é ilustrada no quadro 3.

```
public static void createBase(int qtdPessoas, int qtdTelefones,
                int qtdOperadoras, int qtdDocOperadora){
```

Quadro 3 - Cabeçalho do método responsável por criar a base.

Onde, os parâmetros são referentes à:

- qtdPessoas: Quantidade de pessoas (clientes) a serem criados na base.
- qtdTelefones: Quantidade de telefones criados para cada pessoa.
- qtdOperadoras: Quantidade de operadoras a serem criadas na base.
- qtdDocOperadora: Quantidade de documentos criados para cada operadora.

Sendo assim, se o método createBase for chamado da seguinte forma:

createBase(1000, 2, 5, 2);

Significa que durante a execução do método serão criados na aplicação, mil instâncias de pessoas, 2 mil instâncias de telefones (2 telefones por Pessoa), 5 instâncias de operadoras e 1010 instâncias de documentos (1 documento por pessoa, totalizando mil, mais 10 documentos de operadoras (2 para cada uma das 5 operadoras).

O método createBase foi executado quatro vezes, conforme a modelagem definida na seção 3.4, a quantidade de registros gerados para cada execução é demonstrada na tabela 5. Para fins de avaliação, adotou-se a quantidade de registro de Pessoas como referência de comparação.

Para inserção no *PostgreSQL* foi utilizado a instrução INSERT INTO da linguagem *SQL*.

<INSERT INTO nome_tabela VALUES (valor1,valor2,valor3,...);>

Para o *MongoDB* usou-se o comando save.

<db.<nome_coleção>.save(<documento>)>

Para o Cassandra a instrução INSERT INTO do CQL foi responsável por inserir os dados.

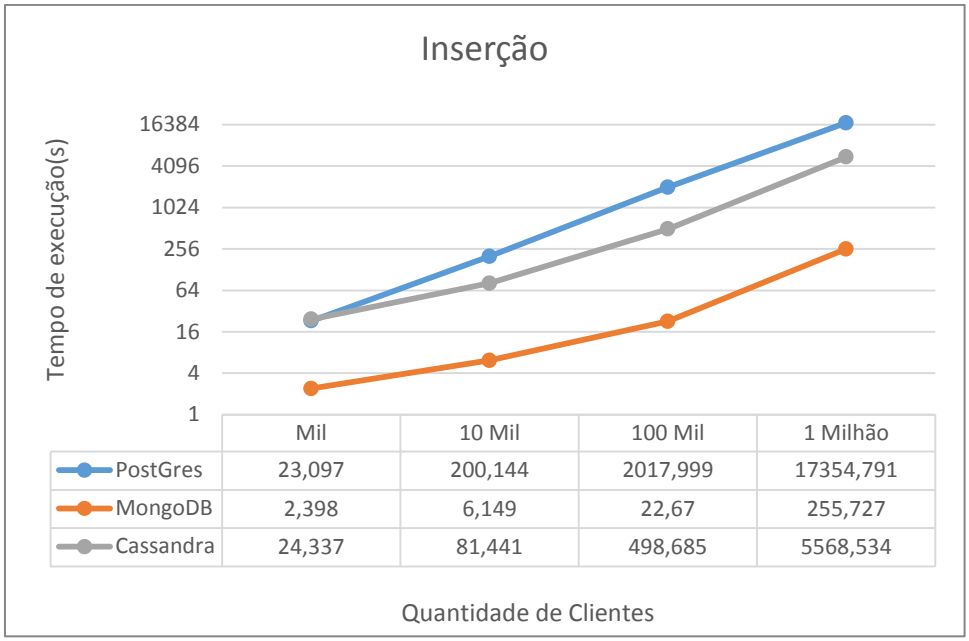
<INSERT INTO nome_tabela VALUES (valor1,valor2,valor3,...);>

Chamada ao método	Quantidade de Registros gerados no PostgreSQL		Quantidade de Coleções Geradas do MongoDB	Quantidade de Registros Gerados no Cassandra	
createBase(1000, 2, 5, 2)	Pessoa	1000	1000	Doc_Tel	2000
	Telefone	2000			
	Operadora	5		Operadora	5
	Documento	1010			
	Doc_Ope	10			
createBase(10000, 2, 10, 2)	Pessoa	10000	10000	Doc_Tel	20000
	Telefone	20000			
	Operadora	10		Operadora	10

	Documento	1020			
	Doc_Ope	20			
createBase(100000, 2, 15, 2)	Pessoa	100000	100000	Doc_Tel	200000
	Telefone	200000			
	Operadora	15		Operadora	15
	Documento	1030			
	Doc_Ope	30			
createBase(1000000, 2, 20, 2)	Pessoa	1000000	1000000	Doc_Tel	2000000
	Telefone	2000000			
	Operadora	20		Operadora	20
	Documento	1040			
	Doc_Ope	40			

Tabela 5- Quantidade de Registro Gerados

O gráfico da Figura 11 mostra a comparação do tempo de execução entre as instruções de inserção para o PostgreSQL, MongoDB e Cassandra. Salientando que o eixo-y está em escala de Log na base 2, essa escala será adotada para todos os gráficos no decorrer do texto.



	Mil	10 Mil	100 Mil	1 Milhão
PostGres	23,097	200,144	2017,999	17354,791
MongoDB	2,398	6,149	22,67	255,727
Cassandra	24,337	81,441	498,685	5568,534

Figura 11 - Gráfico Insert

O principal ponto que se destaca no gráfico da Figura 9 é que o MongoDB possui um desempenho superior, ou seja, gasta menos tempo para inserir os dados do que o PostgreSQL e o Cassandra. Uma possível explicação para este resultado é que definiu-se o esquema do MongoDB em apenas um documento agregado, isso significa que todos os dados de um cadastro de cliente estão modelados em um único documento. Já no PostgreSQL, o de menor desempenho, estão em cinco tabelas diferentes.

O MongoDB apresentou o melhor resultado e a maior diferença em relação ao PostgreSQL com N=1 milhão (17099,064s). A maior diferença entre o MongoDB e o Cassandra também foi quando N=1.000.000 (5312,807s). Na média, o tempo de inserção do MongoDB foi 95,99% melhor que o PostgreSQL e 93,37% melhor que o Cassandra.

3.6.2. SELECT

Para o testes de consulta adotou-se o seguinte cenário.

"Um corretor da Lig Corretora pode em determinado momento consultar os dados cadastrais de um cliente, para isso, ele deve fornecer o número de documento do cliente e o telefone do cliente que deseja consultar. Em seguida, uma tela com todos os dados do cliente e do telefone deve ser exibida, assim como os dados da operadora do telefone."

Dado o cenário, em uma aplicação real, seria necessário recuperar os dados da base conforme critérios de buscas informados e mapear todas os dados em objetos. Para isso, utilizou-se o seguinte método do DAO.

```
public Pessoa findCliente (String documento, String telefone) throws TccException;
```

Quadro 4 - Método DAO para pesquisar cliente a partir do número de documento e de telefone.

Para implementação do método utilizando o PostgreSQL, a consulta utilizou o recurso de junções entre as tabelas Documento, Pessoa e Telefone, em seguida fez uma série de consultas simples às tabelas Operadora, Documento_Operadora e Documento, afim de recuperar os dados e documentos da Operadora. O quadro 5 apresenta as consultas realizadas.

```
1   SELECT  d.id as d_id,
2           d.numero as d_numero,
3           tipo, p.id as p_id,
4           id_documento,
5           nome,
6           nascimento, t.id as t_id, id_pessoa,
7           id_operadora,
8           t.numero as t_numero
9           FROM documento d
10          inner join pessoa p
11          on d.id = p.id_documento
12          inner join telefone t
13          on p.id = t.id_pessoa
14          WHERE d.numero = ? AND t.numero = ?
15
16  SELECT * FROM operadora WHERE id = ?
17
18  SELECT  * FROM documento_operadora WHERE id_operadora = ?
19
20  SELECT  * FROM documento WHERE id = ?
```

Quadro 5- SELECTs realizados no PostgreSQL

Já no MongoDB, como definiu-se a chave (_id) do documento sendo o documento da Pessoa, a consulta ficou como mostrado no quadro 6.

```
1   db.cliente.find({"_id" : ?})
```

Quadro 6 - Comando de pesquisa no MongoDB.

Porém, dessa forma todos os dados do cliente são carregados, inclusive de outros telefones, então o filtro deve ser feito programaticamente. Para o cenário utilizado isso não foi problema, mas em casos em que o documento pesquisado seja muito grande, não deve ser apropriado recuperar todo o documento, devido ao alto consumo de recursos como o consumo de memória, por exemplo. Para esses casos, o MongoDB fornece alternativas para recuperar apenas partes do documento.

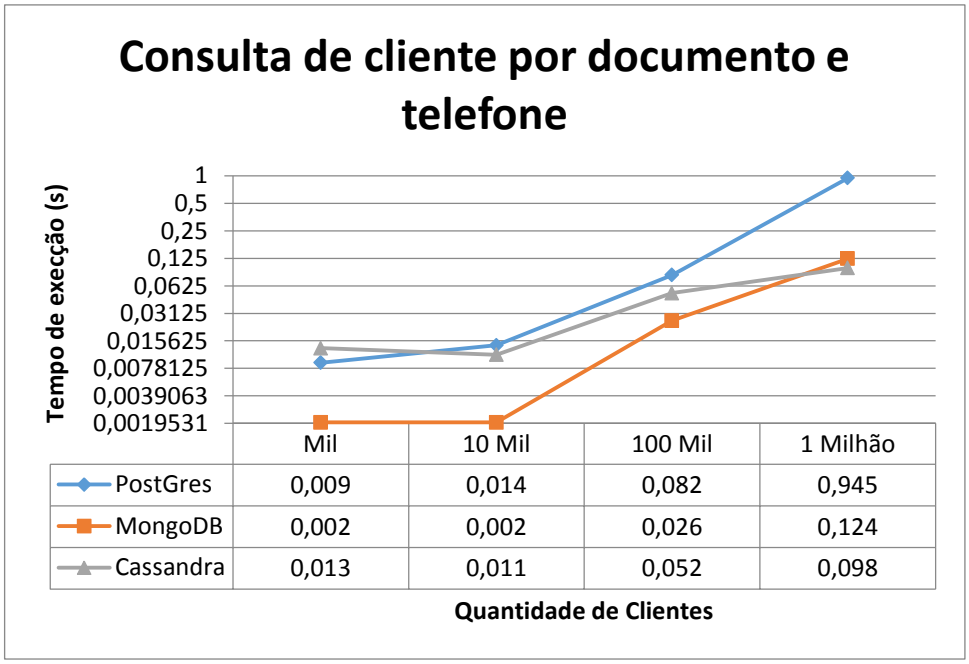
Para a consulta do cliente no Cassandra, como ele não suporta o uso de instrução por junção, foram feitas duas consultas serialmente, primeiro consultou-se os dados da Pessoa e do Telefone e em seguida recuperou-se os dados da Operadora. O quadro 7 demonstra como foram feitas as consultas.

```
1  SELECT * FROM documento_telefone WHERE documento = ? and telefone = ?
2
3  SELECT * FROM operadora WHERE operadora_id = ?
```

Quadro 7 - SELECTs realizados no Cassandra

Em todos os casos as consultas resultaram na mesma informação, comprovando que a comparação foi realizada através de comandos que produziram o mesmo resultado e que as bases continham os mesmos dados.

Para cada quantidade de dados (mil, 10 mil, 100 mil e 1 milhão) as consultas foram realizadas 10 vezes, cada vez com dados de entrada diferentes, afim de forçar a consulta a base, evitando um possível uso de cache do banco de dados. Os dados apresentados na Figura 12 são as médias das 10 execuções.



	Mil	10 Mil	100 Mil	1 Milhão
PostGres	0,009	0,014	0,082	0,945
MongoDB	0,002	0,002	0,026	0,124
Cassandra	0,013	0,011	0,052	0,098

Figura 12- Gráfico Select

Assim como na instrução de inserção, nota-se uma clara diferença entre o MongoDB e PostgreSQL. Um fator que chamou atenção é que nos testes com N= 1.000, o Cassandra foi o BD que apresentou menor desempenho, porém no com N= 1.000.000, foi o BD que apresentou o melhor tempo de execução. Provavelmente, por ele usar um esquema de chave-

valor, o desempenho numa consulta por chave não sofre muito com a quantidade de dados armazenados.

Já no PostgreSQL nota-se um crescimento exponencial, o que era esperado, pois com o aumento do volume de dados, o produto cartesiano entre as tabelas resulta um número exorbitante de registros.

Apesar da melhoria apresentada pelo Cassandra no decorrer dos testes, o MongoDB apresentou um desempenho de 49% de superioridade em relação ao Cassandra e uma média de 79% mais eficiência que o PostgreSQL.

3.6.3. SELECT TOP 50

Para analisar o desempenho de uma consulta baseada a um campo qualquer do conjunto de dados e não apenas da consulta via índice (chave) como no exemplo anterior, realizou-se uma consulta que baseou-se no seguinte cenário.

"*Espera-se obter uma lista de 50 clientes que possuam telefone de uma Operadora X dado a sua razão social. A lista deve ser ordenada pelo nome da pessoa em ordem alfabética. Os dados da operadora, inclusive com seus documentos também devem ser carregados."*

No banco de dados relacional optou-se por fazer a consulta em duas etapas, a primeira etapa consiste em recuperar a Operadora com o filtro da razão social junto com seus documentos, em caso de retorno, realizou-se a segunda etapa a qual consiste em uma nova consulta trazendo os dados das pessoas e seus telefones. A sequência de consultas pode ser visto no Quadro 8.

```
--Etapa 1
SELECT * FROM operadora WHERE razaoSocial = ?

SELECT  * FROM documento_operadora WHERE id_operadora = ?

SELECT * FROM documento WHERE id = ?

--Etapa 2
SELECT d.id as d_id,
        d.numero as d_numero,
        tipo,
        p.id as p_id,
        id_documento,
        nome, nascimento,
        t.id as t_id,
        id_pessoa,
        id_operadora,
        t.numero as t_numero
        FROM documento d
        inner join pessoa p
        on d.id = p.id_documento
        inner join telefone t
        on p.id = t.id_pessoa
        WHERE id_operadora = ?
        ORDER BY nome
        LIMIT 50
```

Quadro 8 - Select Top 50 no PostgreSQL .

Em segundo plano foram realizados testes para verificar o desempenho dessa consulta em uma única junção, ou seja, fazendo apenas uma consulta com todas as tabelas envolvidas ou mais de uma consulta, conforme foi feito. Os resultados foram muito semelhantes, com uma diferença de poucos milissegundos para mais ou menos, dependendo da vez que foi executada. Dado a semelhança, optou-se por realizar a consulta dessa forma, pois, para o desenvolvimento da aplicação, o código é mais simples para povoar os objetos em java.

No MongoDB a consulta foi realizada conforme o Quadro 9.

```
db.cliente.find({"telefones.operadora.razaoSocial": <razaoSocial>})
    .sort({"nome": 1}).limit(50);
```

Quadro 9 - Select Top 50 no MongoDB

No Cassandra foi encontrada uma limitação para executar tais operações, pois o mesmo não permite fazer a Ordenação por um campo da família de coluna, apenas pelas suas chaves. Para realizar tal ordenação, seria necessário fazer via programação. Como esse trabalho avalia o

desempenho do banco de dados, realizar essa tarefa na aplicação está além do nosso contexto e por isso não foi implementado.

A Figura 13 exibe o tempo de execução da consulta.

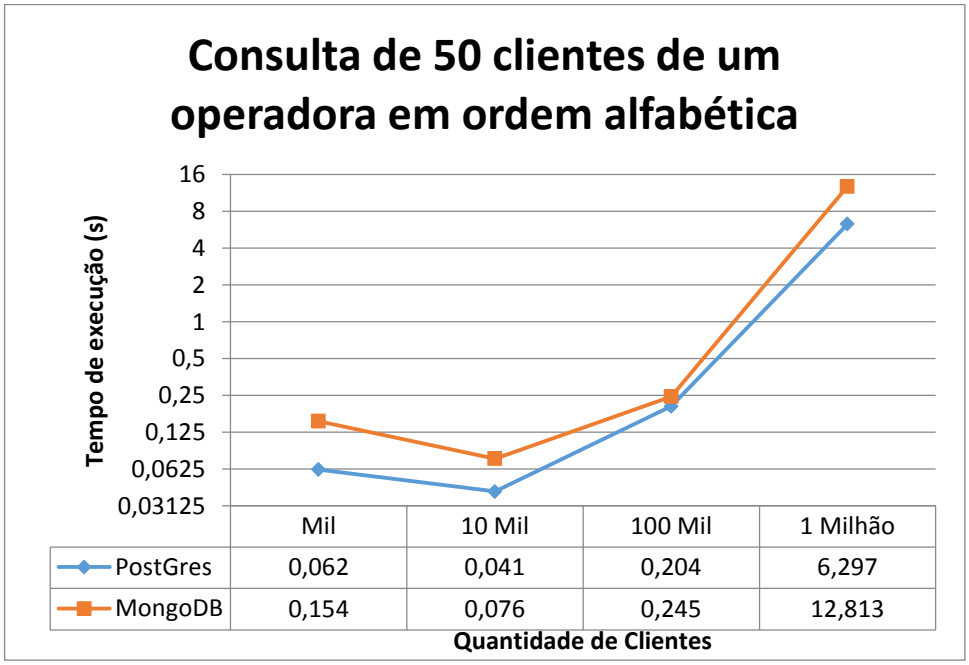


Figura 13- Gráfico Select To 50

Nota-se que o PostgreSQL é superior ao MongoDB em todos os testes, sendo a maior diferença de tempo com N= 1.000.000 de 6,516 segundos, para os demais volumes a diferença é singela e não ultrapassou 1 segundo. O PostgreSQL foi superior ao MongoDB em média 43,34%.

3.6.4. UPDATE PESSOA

Em um cenário real para realizar uma operação de update, na maioria dos casos primeiramente seleciona-se o que se deseja atualizar, fazem-se as alterações em memória e só então salva-se no disco (banco de dados). Foi seguindo esse fluxo de caso de uso que se implementou a atualização dos dados.

Para essa operação, realizou-se a alteração do nome do cliente, primeiramente foi feita a consulta a base usando o método findCliente do DAO detalhado no item 3.6.2, depois alterado o nome (novo nome também foi gerado aleatoriamente) e em seguida executado o método updatePessoa(Pessoa pessoa) do DAO.

Na implementação utilizando o PostgreSQL do método updatePessoa, a instrução SQL, que pode ser vista no Quadro 10, foi utilizada:

```
1    UPDATE pessoa SET id_documento = ?, nome = ?, nascimento = ? WHERE id = ?
```

Quadro 10 - Update no PostgreSQL

Para o MongoDB foi executado o comando "*save*", o mesmo utilizado no *insert*, esse comando ao ser executado verifica através da chave (id) se o documento passado como parâmetro já existe, se existir, ele atualiza o documento, senão, ele cria o documento. Dessa forma o comando *save* pode ser utilizado tanto para o *insert* como para o *update*.

O Quadro 11 detalha a instrução de update realizada pelo Cassandra.

```
1    UPDATE documento_telefone SET nome = ? WHERE documento = ? and telefone = ?
```

Quadro 11- Update Cassandra

Pode-se notar através da Figura 14 que a operação de atualização é extremamente rápida em todos os bancos de dados, pois o tempo de consulta está incluído nos testes, e comparando aos testes realizado na seção 3.6.2 a diferença é muito sutil.

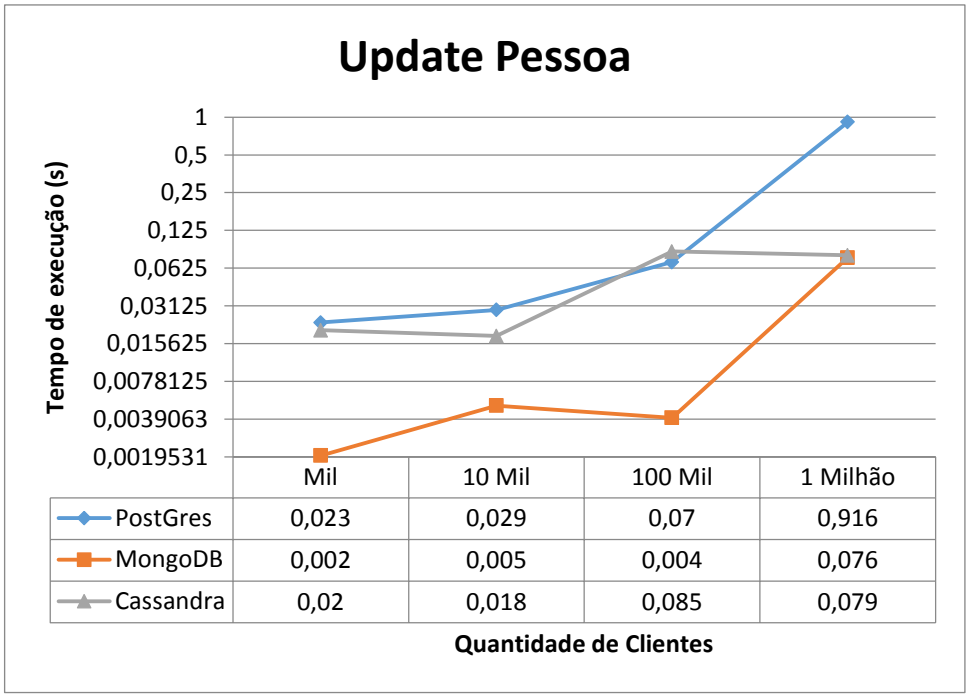


Figura 14 - Gráfico Update

Provavelmente esse alto desempenho na atualização, dá-se pelo bom uso das chaves. A avaliação do update em muito se assemelha ao select do item 3.6.2, por essa forte semelhança foi inibida essa avaliação no item, podendo o leitor tomar o item 3.6.2 como referência.

3.6.5. UPDATE OPERADORA

Para um outro teste de atualização, pretende-se atualizar a razão social de uma operadora. Esse caso de uso se assemelha ao update pessoa descrito no item anterior. A diferença no caso do PostgreSQL e do Cassandra é apenas a tabela alvo que nesse caso será a tabela Operadora.

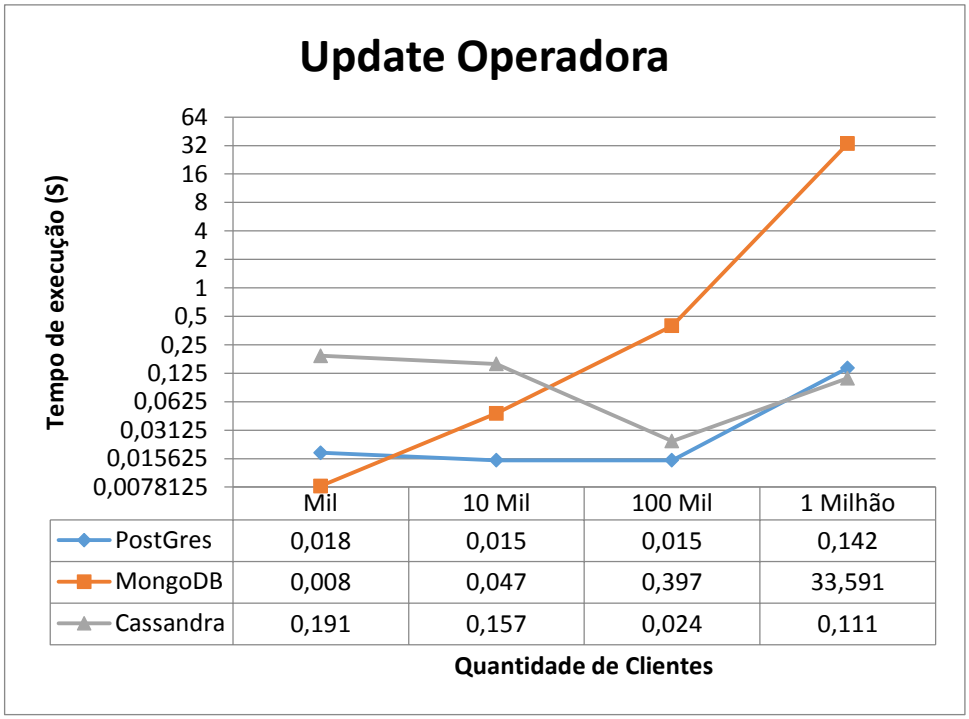
Já para o MongoDB há uma diferença maior, o comando utilizado é apresentado no Quadro 12, o valor *false* informa ao comando que caso o documento procurado não seja encontrado, ele não deve ser criado. O valor *true*, passado por parâmetro, informa ao comando que todos os registros que correspondem ao critério de busca devem ser atualizados.

```
1  this.cliente.update({"telefones.operadora.razaoSocial": <razaoSocial>},
2      {"$set": {"telefones.$.operadora.razaoSocial": <novaRazaoSocial>}, false, true);
```

Quadro 12 - Update Operadora utilizando MongoDB

Para a atualização da razão social de uma operadora é fácil notar que de acordo com a modelagem adotada no banco de dados relacional (PostgreSQL) e no orientado a coluna (Cassandra) apenas um registro da tabela operadora será alterado. Já no MongoDB, como adotou-se um modelo representado por um único agregado, existe a duplicação dos dados das operadoras em milhares de documentos.

O gráfico da Figura 15 mostra o tempo de execução para cada volume de dados.



	Mil	10 Mil	100 Mil	1 Milhão
PostGres	0,018	0,015	0,015	0,142
MongoDB	0,008	0,047	0,397	33,591
Cassandra	0,191	0,157	0,024	0,111

Figura 15- Gráfico Update Operadora

Surpreendentemente o MongoDB apresentou o melhor desempenho para o volume de dados de N=1.000, o que é impressionante, pois para cada documento o MongoDB irá verificar se existe Operadora com a razão Social igual a passada por parâmetro e caso verdadeiro, atualizá-la.

Porém, como esperado, o MongoDB apresentou um crescimento exponencial no tempo de execução, enquanto que o PostgreSQL e Cassandra mantiveram um alto desempenho na atualização. O PostgreSQL teve um desempenho de 52% maior que o MongoDB e 49% maior que o Cassandra.

3.6.6. DELETE

Para o testes de Delete pretende-se remover um cadastro de cliente, ou seja, o registro da pessoa, de seu documento e de seus telefones.

No PostgreSQL a chave estrangeira de documento na tabela pessoa foi criada com referência de delete em cascata, assim como a chave estrangeira de pessoa na tabela telefone. Ou seja, se um documento referenciado por uma pessoa é deletado a pessoa também será e consequentemente seus telefones. Contudo, para o teste de delete no PostgreSQL usou-se a instrução, demonstrada no Quadro 13.

```
1   DELETE FROM documento where numero = ?
```

Quadro 13 - DELETE utilizado no PostgreSQL

O quadro 14 apresenta o comando utilizado pelo MongoDB para exclusão do documento.

```
1   this.cliente.remove({"_id": <numeroDocumento>});
```

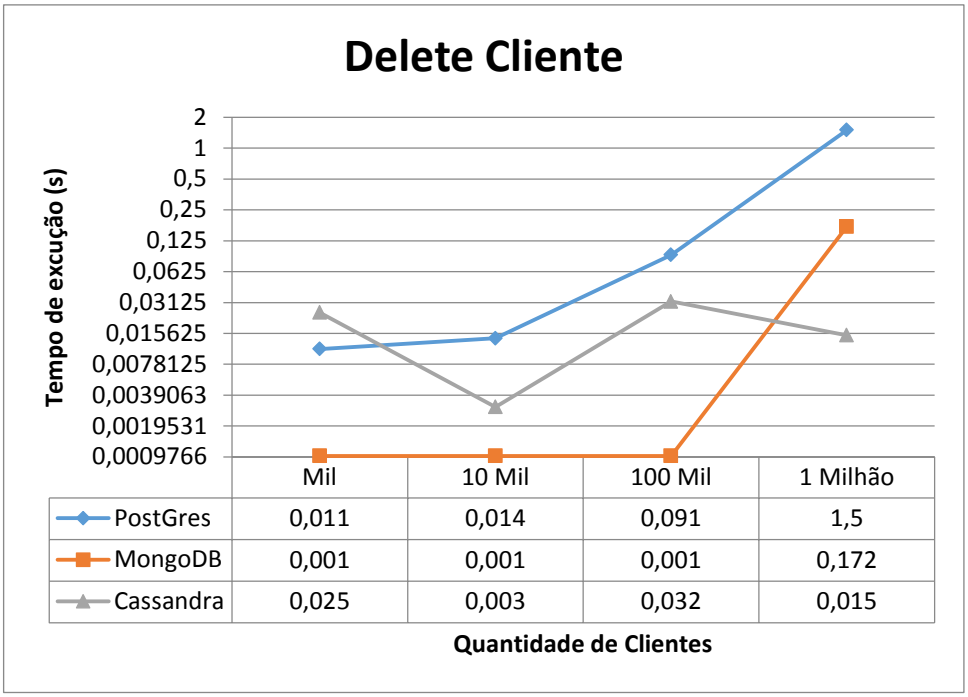
Quadro 14 - DELETE utilizado no MongoDB

Para o Cassandra a instrução foi detalhada no Quadro 15.:

```
1   DELETE FROM documento_telefone WHERE documento = ?
```

Quadro 15 - DELETE utilizado no Cassandra.

Como nos outros casos de uso onde se fez operações utilizando a chave do registro, o MongoDB teve um desempenho superior aos demais bancos. O gráfico da Figura 16 detalha o tempo de execução para cada volume de dados.



	Mil	10 Mil	100 Mil	1 Milhão
PostGres	0,011	0,014	0,091	1,5
MongoDB	0,001	0,001	0,001	0,172
Cassandra	0,025	0,003	0,032	0,015

Figura 16 - Gráfico Delete

O MongoDB se mostrou 93% mais eficiente que o PostgreSQL e 42% superior ao Cassandra. Mais uma vez o Cassandra se destacou por ser o mais lento para o volume de dados com N=1.000 e o mais rápido para N=1.000.000, mostrando que o desempenho não sofre muito com o aumento da quantidade de dados na base.

3.6.7. DELETE MENOR 18

A fim de testar um delete em massa criou-se o seguinte cenário.

"Uma nova lei surgiu e a corretora de telefonia não pode ter clientes menores de 18 anos. Deve-se excluir todos os registros de clientes que sejam menores de 18 anos."

Para tal tarefa executou-se a partir da aplicação a instrução SQL apresentada no Quadro 16 para excluir os registros.

```
1  DELETE FROM documento where id IN
2  (select d.id from pessoa p
3      inner join documento d
4      on p.id_documento = d.id
5      WHERE nascimento > ?)
```

Quadro 16 - DELETE Menor no PostgreSQL

Nota-se que foi realizado o delete a partir de uma subconsulta, que retorna todos os registros que possuem data de nascimento maior que a passada por parâmetro. No exemplo corrente, a data passada foi a de exatamente 18 anos atrás da data de execução.

O Quadro 17 demonstra o comando utilizado para remoção dos documentos no MongoDB.

```
1   db.cliente.remove({"nascimento": {$gt : IsoDate(<date>)}})
```

Quadro 17 - Delete Menor MongoDB.

Mais uma vez foi encontrada uma limitação do Cassandra para executar a operação do caso de uso, pois o mesmo não permite a comparação de “maior” ou “menor” a campos que não sejam chaves.

Diferente das demais operações que não foram realizadas diretamente com a chaves do registro, para a operação de delete o MongoDB se mostrou com desempenho superior ao PostgreSQL. Como pode ser visto no gráfico da Figura 17.

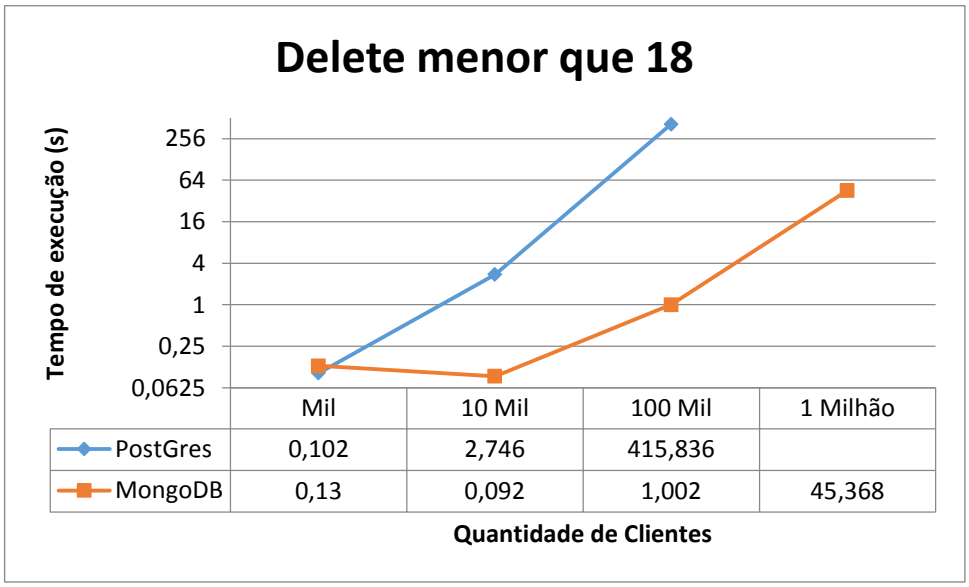


Figura 17 - Gráfico Delete Menor 18

Nota-se que a partir do volume de dados N=100.000 a diferença é imensa, provavelmente por que o PostgreSQL precisa atualizar uma série de registros e propagar a remoção dos dados por três tabelas (Documento, Pessoa e Telefone), enquanto que o MongoDB só precisa excluir o documento que satisfaça a condição.

A quantidade de menores de 18 anos encontrados e deletados para cada volume de dados é detalhado na tabela a seguir:

	Mil	10 Mil	100 Mil	1 Milhão
Menores 18	114	903	9378	92094

Tabela 6 - Quantidade de Registros Deletados

O tempo de execução para N=1.000.000 do PostgreSQL está em branco pois passados 6 horas ainda não havia concluído, então cancelou-se a operação.

3.7. Análise dos resultados

Analisando os dados apresentados na seção anterior é possível tirar algumas conclusões em relação à inserção, exclusão, atualização e consulta dos dados de acordo com o escopo e ambiente dos testes. Contudo, é importante frisar que os indícios apresentados pelos dados são apenas uma base do desempenho geral e devem ser analisados sempre no contexto real da aplicação.

A primeira conclusão é que, na média, em 71% dos testes, o MongoDB obteve um desempenho melhor que o PostgreSQL e o Cassandra. Os principais motivos para justificar tal desempenho foi o uso de um único documento para representação de todo cadastro de clientes e a chave do documento que facilitou a busca.

Outro fator que deve-se observar foi a limitação que o Cassandra impôs diante do modelo criado inicialmente, isso enfatiza que o NoSQL, apesar da flexibilidade para alterar o modelo dos dados, não é totalmente livre de esquemas, o modelo inicial do projeto deve ser bem elaborado de forma a satisfazer as necessidades da aplicação. Por exemplo, na atualização de operadora, o MongoDB certamente teria um desempenho melhor se uma coleção de operadoras tivesse sido criada a parte e no documento do cliente apenas uma referência ao id da operadora, porém, essa metodologia poderia afetar nas demais consultas, que trazem todas as informações em um único agregado. Tudo isso deve ser levado em consideração ao montar o esquema dos dados.

Vale salientar também que todos os teste foram realizados em um único cluster e a proposta dos bancos de dados NoSQL é fornecer a possibilidade e simplicidade de distribuí-los, o que pode aumentar ainda mais o seu desempenho.

4. Considerações Finais

Os diversos produtos existentes no mercado para o armazenamento de dados, conhecidos como bancos de dados, trabalham com diversas tecnologias. Dentre os mais populares estão os produtos que implementam o modelo relacional, como o SQL Server, o Oracle, o DB2, o MySQL e o PostgreSQL. Recentemente novas abordagens para o armazenamento e manipulação de dados foram apresentadas e agrupadas pela sigla NoSQL, incluindo produtos como o MongoDB, Cassandra, Redis, CouchDB, Neo4J, entre outros.

O presente trabalho apresentou uma comparação de desempenho do tempo de execução de instruções que incluem, removem, alteram e consultam dados nos bancos de dados SQL e NoSQL. O sistema operacional utilizado para a realização dos testes foi o Elementary Luna. Os bancos de dados comparados foram o PostgreSQL, MongoDB e Cassandra.

Os resultados mostraram que o MongoDB possui melhor desempenho em 5 dos 7 testes realizados. O Cassandra apresentou-se sempre na média entre os dois demais bancos de dados.

Contudo, é preciso sempre levar em consideração diversos fatores quando se decide adotar ou não um software, e não apenas o desempenho no tempo de execução de instruções que manipulam dados. Portanto, os resultados obtidos não são suficientes para influenciar a escolha das tecnologias a adotar. Mas, os profissionais que trabalham com bancos de dados e programação possuem um ponto de partida para justificar futuras pesquisas e testes que avaliem o desempenho dos bancos de dados NoSQL e SQL.

Como trabalhos futuros, sugere-se a realização dos testes com outros bancos de dados NoSQL, que utilizam outro modelo de dados. Sugere-se também realizar teste com outras operações como funções agregadas, muito usadas em banco de dados relacionais.

REFERÊNCIAS BIBLIOGRÁFICAS

ALMEIDA, R. C.; BRITO, P. F. de..(2012). **Utilização da Classe de Banco de Dados NOSQL como Solução para Manipulação de Diversas Estruturas de Dados.** In:. XIV Encoinfo, CEULP/ULBRA, Palmas, TO. 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: 18 de Dezembro de 2013, às 13h.

BRITO, R. W.. (2013). **Bancos de Dados NoSQL x SGBDs Relacionais:Análise Comparativa*.** In:. Infobrasil TI & TELECOM , Fortaleza – CE. Disponível em: < http://www.infobrasil.inf.br/userfiles/27-05-S4-1-68840-Bancos%20de%20Dados%20NoSQL.pdf >. Acesso em: 13 de Janeiro de 2014, às 10h 47min.

CARNIL, A. C.; SÁ, A. de A.; RIBEIRO, M. X.; BUENO, R.; CIFERRI, C. D. de A.; CIFERRI, R. R.. (2012). **Análise Experimental de Bases de Dados Relacionais e NoSQL no Processamento de Consultas sobre Data Warehouse.** In:. SBBD 2012, São Paulo, SP. Disponível em: < http://data.ime.usp.br/sbbd2012/artigos/pdfs/sbbd_shp_15.pdf>. Acesso em: 13 de Janeiro de 2014, às11h.

COSTA, E. R. de.. **Banco de Dados Relacional.** São Paulo, 2011. 64p. Trabalho de conclusão de curso (Tecnólogo em Processamento de Dados) - Faculdade de Tecnologia de São Paulo, São Paulo, SP. Disponível em: < http://www.fatecsp.br/dti/tcc/tcc0025.pdf >. Acesso em: 22 de Janeiro de 2013, às 10h.

DELGADO, A. G.. **Apresentação da Tecnologia NoSQL através de um estudo de caso do Banco de Dados Apache Cassandra**. João Pessoa, 2011. 82p. Trabalho de conclusão de curso (Graduação em Bacharelado em Ciência da Computação) – Centro Universitário de João Pessoa – UNIPE, João Pessoa – PB. Disponível em: Acesso em: 18 de Dezembro de 2013, às 13h 27min.

LEITE, G.S. **Análise Comparativa do Teorema CAP Entre Bancos de Dados NoSQL e Bancos de Dados Relacionais.** Fortaleza, 2010. 49p. Trabalho de conclusão de curso (Graduação em Ciência da Computação) – Faculdade Farias Brito, Fortaleza – CE.

Disponível em: < http://www.ffb.edu.br/sites/default/files/tcc-20102-gleidson-sobreira-leite.pdf >. Acesso em: 10 de Fevereiro

LÓSCIO, B. F.; OLIVEIRA, H. R.; PONTES, J. C. de S..(2011). **NoSQL no Desenvolvimento de Aplicações Web Colaborativas.** In:. VIII SBSC, Paraty, RJ. Disponivel em: < http://www.addlabs.uff.br/sbsc_site/SBSC2011_NoSQL.pdf>. Acesso em 18 de Dezembro de 2013, às 12h54min.

MACÁRIO, C. G. do N.; BALDO, S. M.. **O Modelo Relacional.** Trabalho como parte da avaliação do curso MO-410 (curso: Introdução a Banco de Dados) - Instituto de Computação da Unicamp, Campinas, SP, 2005. Disponível em: < http://www.ic.unicamp.br/~geovane/mo410-091/Ch03-RM-Resumo.pdf >. Acesso em: 21 de Janeiro de 2013, às 13h.

MARQUES, H. S. **Aumento da Escalabilidade com o uso de Service Oriented Arquitecture (SOA)**. Belo Horizonte, 2012. 27p. Trabalho para obtenção do título de Especialista (curso: Estratégia em Arquitetura de Software) – Instituto de Gestão em Tecnologia da Informação, Belo Horizonte, MG. Disponível em: < http://pt.slideshare.net/HugoMarques16/tcc-igti-hugo-marques-v11>. Acesso em: 04 de Fevereiro de 2014, às 16h.

PEREIRA, G. B. M.; MOREIRA, M. P. (2012). **Otimização da Junção por Laço Aninhado Usando Expressões da Cláusula Where.** In:. .... PUC (Minas Gerais). Belo Horizonte – MG. Disponível em: < http://seer.ufrgs.br/reic/article/download/21961/21685%E2%80%8E >. Acesso em: 22 de Janeiro de 2013, às 11h.

SADALAGE, J.; FOWLER, M.. **NoSQL: Um Guia Conciso para o Mundo Emergente da Persistência Poliglota**, [tradução Acauan Fernandes]. Novatec Editora Ltda, 2013. 216p. ISBN 978-85-7522-338-3.

SOARES, B. E.. **Uma avaliação experimental de desempenho entre Sistemas Gerenciadores de Banco de Dados Colunares e Relacionais**. Cascavel, 2012. 102p. Trabalho de conclusão de curso (Graduação em Bacharelado em Ciência da Computação) – Universidade Estadual do Oeste do Paraná – UNOPA, Cascavel, PA. Disponível em: <

http://www.inf.unioeste.br/~tcc/2012/TCC_Bruno.pdf >. Acesso em: 06 de Janeiro de 2014, às 12h.

SOUSA, T. R. P. de; ROCHA, A. L. de S. S.. (2010). **NoSQL: Princípios e Características.** Trabalho de disciplina - Disciplina: Banco de Dados Avançado. Faculdade de Tecnologia da Paraíba, Cabedelo – PB. Disponível em: < http://pt.slideshare.net/andrerochajp/artigo-NoSQL >. Acesso em: 02 de Fevereiro de 2014, às 15h.

LENNON, Joe (2013). MongoDB. Disponível em: http://alfavillecode.blogspot.com.br/2013/01/MongoDB.html. Acesso em: 06 de abril de 2014, às 18h.