CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

COMPARAÇÃO DE DESEMPENHO DE BANCOS DE DADOS SQL E NOSQL

EDER DOS SANTOS CUER

Marília, 2014

EDER DOS SANTOS CUER

COMPARAÇÃO DE DESEMPENHO DE BANCOS DE DADOS SQL E NOSQL

Monografia apresentada ao Centro Universitário Eurípides de Marília como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Elvis Fusco

Marília, 2014

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO – AVALIAÇÃO FINAL

Eder dos Santos Cuer

Comparação de desempenho de bancos de dados SQL e NoSQL

Banca examinadora da monografia apresentada ao Curso de Bacharelado em Ciência da Computação do UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Ciência da Computação.

Nota: 8,0 (oito)

Orientador: Elvis Fusco

1°. Examinador: Fabio Lucio Meira

2°. Examinador: Paulo Augusto Nardi

Marília, 09 de dezembro de 2014.

Dedico este trabalho aos meus pais, Valdenice e Edson, que me incentivaram desde o início.

AGRADECIMENTOS

Agradeço primeiramente a minha mãe, que diante de tantos problemas se mostra forte, servindo de exemplo para mim. Mulher guerreira que sempre me apoia, é uma honra poder dedicar esse trabalho à ela.

Ao meu pai que sempre me incentivou a focar nos estudos e que me apoia em minhas decisões, sempre me ajudando como pode.

A minha irmã, que mesmo diante de tantas brigas se mostrou uma ótima amiga, alguém em quem posso confiar e contar sempre.

A minha namorada Mariane, por me aturar nesses dias estressantes e difíceis, por estar ao meu lado, me ajudar a manter o foco. Companheira desde o início.

Aos meus amigos que direta ou indiretamente me deram forças e em especial para o Victor, que me ajudou muito com os problemas do CentOS e com os bancos de dados.

SUMÁRIO

INTRODUÇÃO 13
CAPÍTULO 1 - SGBD RELACIONAL 14
1.1 - História do modelo relacional 14
1.2 - Arquitetura de um SGBD 15
1.3 - Vantagens de um SGBD relacional 16
1.4 - Desvantagens de um SGBD relacional 19
1.5 - Desempenho 20
CAPÍTULO 2 - NOSQL 24
2.1 - História do NoSQL 24
2.2 - Big Data 24
2.3 - Características 25
2.4 - Teorema CAP 27
2.5 - Técnicas para implementação das funcionalidades do NoSQL 27
2.6 - Modelos NoSQL 28
CAPÍTULO 3 - TESTE DE PERFORMANCE EM BANCO DE DADOS 32
3.1 - Teste de Software 32
3.2 - Estratégias de Teste 35
3.3 - Técnicas de Teste 36
3.4 - Níveis de Teste 38
3.5 - Teste de Desempenho 39
CAPÍTULO 4 - METODOLOGIA 43
4.1 - Trabalhos Correlatos 43
4.2 - Ambiente de Testes 43
4.3 - Máquina Virtual 45
4.4 - Bancos de Dados Escolhidos 47

4.5 - Metodologia dos testes 53
CAPÍTULO 5 - TESTES E RESULTADOS 55
5.1 - Teste de inserção 56
5.2 - Teste de busca 59
5.3 - Teste de alteração 75
5.4 - Teste de remoção 82
CAPÍTULO 6 - CONCLUSÃO 90
REFERÊNCIAS BIBLIOGRÁFICAS 92

Lista de Figuras

Fig. 1. Arquitetura das camadas de um SGBD..16
Fig. 2. Exemplo de generalização.20
Fig. 3. Processo de consulta no modelo relacional..22
Fig. 4. Modelo NoSQL chave valor..28
Fig. 5. Exemplo de modelo baseado em colunas..29
Fig. 6. Exemplo de modelo orientado a documentos..30
Fig. 7. Exemplo de modelo orientado a grafos..30
Fig. 8. Regra 10 de Myers..33
Fig. 10. Planos necessários para o gerenciamento de projetos.34
Fig. 11. Estratégia de teste..35
Fig. 12. Etapas do teste de software.36
Fig. 13. Modelo V..38
Fig. 14. Características e funcionalidade das ferramentas de teste de desempenho..42
Fig. 15. Arquitetura do ambiente de testes..44
Fig. 16. Configuração de memória RAM..45
Fig. 17. Configuração do processador.46
Fig. 18. Configuração da capacidade de disco .47
Fig. 19. Ranking dos bancos de dados mais populares atualmente..48
Fig. 20. MySQL versão 5.1.73 instalado no CentOS 6.5.49
Fig. 21. PostgreSQL versão 9.3.5 instalado no CentOS 6.5.50
Fig. 22. MongoDB versão 2.6.4 instalado no CentOS..52
Fig. 23. Modelagem da estrutura das tabelas.55
Fig. 24. Estrutura dos registros das tabelas X e Y..55
Fig. 25. Problema de memória na consulta do PostgreSQL..61
Fig. 26. Exemplo de agregação avançada no MongoDB.64
Fig. 27. Por falta de memória a consulta falha no MySQL.65

Lista de Tabelas

Tabela 1. Resultados obtidos da inserção de registros da collection Y no MongoDB. 57
Tabela 2. Resultados obtidos da inserção de registros da collection X no MongoDB. 57
Tabela 3. Resultados obtidos da inserção de registros da tabela Y no MySQL. 57
Tabela 4. Resultados obtidos da inserção de registros da tabela X no MySQL. 58
Tabela 5. Resultados obtidos da inserção de registros da tabela Y no PostgreSQL. 58
Tabela 6. Resultados obtidos da inserção de registros da tabela X no PostgreSQL. 59
Tabela 7. Resultados obtidos da busca de todos os registros da collection Y no MongoDB. .. 61
Tabela 8. Resultados obtidos da busca de todos os registros da collection X no MongoDB. .. 61
Tabela 9. Resultados obtidos da busca de todos os registros da tabela Y no MySQL. 62
Tabela 10. Resultados obtidos da busca de todos os registros da tabela X no MySQL. 62
Tabela 11. Resultados obtidos da busca de todos os registros da tabela Y no PostgreSQL. ... 63
Tabela 12. Resultados obtidos da busca de todos os registros da tabela X no PostgreSQL. ... 63
Tabela 13. Resultados obtidos da busca de todos os registros com agregação avançada das collections X e Y no MongoDB. 66
Tabela 14. Resultados obtidos da busca de todos os registros com INNER JOIN das tabelas X e Y no MySQL. 66
Tabela 15. Resultados obtidos da busca de todos os registros com INNER JOIN das tabelas X e Y no PostgreSQL. 67
Tabela 16. Resultados obtidos da busca de um registro pelo identificador de documentos com agregação avançada das collections X e Y no MongoDB. 68
Tabela 17. Resultados obtidos da busca de um registro pela chave primária com INNER JOIN das tabelas X e Y no MySQL. 69
Tabela 18. Resultados obtidos da busca de um registro pela chave primária com INNER JOIN das tabelas X e Y no PostgreSQL. 69
Tabela 19. Resultados obtidos da busca de um registro pela descrição com agregação avançada das collections X e Y no MongoDB. 71
Tabela 20. Resultados obtidos da busca de um registro pela descrição com INNER JOIN das tabelas X e Y no MySQL. 71
Tabela 21. Resultados obtidos da busca de um registro pela descrição com INNER JOIN das tabelas X e Y no PostgreSQL. 72

Tabela 22. Resultados obtidos da busca com $GROUP, $SUM, e agregação avançada das collections X e Y no MongoDB. 73
Tabela 23. Resultados obtidos da busca com INNER JOIN, GROUP BY e SUM das tabelas Y e X no MySQL. 74
Tabela 24. Resultados obtidos da busca com INNER JOIN, GROUP BY e SUM das tabelas Y e X no PostgreSQL. 74
Tabela 25. Resultados obtidos da alteração por identificador de documento da collection Y no MongoDB. 76
Tabela 26. Resultados obtidos da alteração por identificador de documento da collection X no MongoDB. 77
Tabela 27. Resultados obtidos da alteração por descrição da collection Y no MongoDB. 77
Tabela 28. Resultados obtidos da alteração por valor numérico da collection X no MongoDB. 78
Tabela 29. Resultados obtidos da alteração por chave primária da tabela Y no MySQL. 78
Tabela 30. Resultados obtidos da alteração por chave primária da tabela X no MySQL. 78
Tabela 31. Resultados obtidos da alteração por descrição da tabela Y no MySQL. 79
Tabela 32. Resultados obtidos da alteração por valor numérico da tabela X no MySQL. 79
Tabela 33. Resultados obtidos da alteração por chave primária da tabela Y no PostgreSQL. . 80
Tabela 34. Resultados obtidos da alteração por chave primária da tabela X no PostgreSQL. . 80
Tabela 35. Resultados obtidos da alteração por descrição da tabela Y no PostgreSQL. 80
Tabela 36. Resultados obtidos da alteração por valor numérico da tabela X no PostgreSQL. 81
Tabela 37. Resultados obtidos da remoção de um registro por identificador de documento da collection Y no MongoDB. 83
Tabela 38. Resultados obtidos da remoção de um registro por identificador de documento da collection X no MongoDB. 84
Tabela 39. Resultados obtidos da remoção de um registro por descrição da collection Y no MongoDB. 84
Tabela 40. Resultados obtidos da remoção de um registro por valor numérico da collection X no MongoDB. 85
Tabela 41. Resultados obtidos da remoção de um registro por chave primária da tabela Y no MySQL. 85
Tabela 42. Resultados obtidos da remoção de um registro por chave primária da tabela X no MySQL. 85

Tabela 43. Resultados obtidos da remoção de um registro por descrição da tabela Y no MySQL. 86
Tabela 44. Resultados obtidos da remoção de um registro por valor numérico da tabela X no MySQL. 86
Tabela 45. Resultados obtidos da remoção de um registro por chave primária da tabela Y no PostgreSQL. 87
Tabela 46. Resultados obtidos da remoção de um registro por chave primária da tabela X no PostgreSQL. 87
Tabela 47. Resultados obtidos da remoção de um registro por descrição da tabela Y no PostgreSQL. 87
Tabela 48. Resultados obtidos da remoção de um registro por valor numérico da tabela X no PostgreSQL. 88

Lista de Gráficos

Gráfico 1. Resultados dos testes de inserção das tabelas Y e X. 56
Gráfico 2. Resultados dos testes de busca de todos os registros das tabelas Y e X. 60
Gráfico 3. Resultado dos testes de INNER JOIN com todos os registros. 65
Gráfico 4. Resultado dos testes de busca de um registro pela chave primária e identificador de documentos utilizando INNER JOIN e agregação avançada. 68
Gráfico 5. Resultado dos testes de busca de um registro pela descrição utilizando INNER JOIN e agregação avançada. 70
Gráfico 6. Resultado dos testes de busca com GROUP BY, SUM, INNER JOIN e agregação avançada. 73
Gráfico 7. Resultado dos testes de alteração por chave primária e identificador de documento. 75
Gráfico 8. Resultado dos testes de alteração descrição e valor numérico. 76
Gráfico 9. Resultado dos testes de remoção pela chave primária e identificador de documento. 82
Gráfico 10. Resultado dos testes de remoção pela descrição e valor numérico. 83

Resumo

O novo conceito de banco de dados, o modelo não relacional conhecido como NoSQL vem ganhando espaço na área tecnológica. Este trabalho abordou o conceito de bancos de dados relacional e não relacional, história, vantagens, desvantagens e características de cada um. Apresentou também trabalhos relacionados com o tema e os resultados dos testes realizados com o desempenho dos bancos de dados selecionados. O principal foco do trabalho é a análise comparativa entre o desempenho de operações de inserção, busca, edição e exclusão no MySQL, PostgreSQL e MongoDB. Para que os ambientes fossem idênticos para os testes, foi necessário virtualizar um ambiente para cada banco de dados, o software utilizado foi o VMware.

Palavras-Chave: NoSQL; performance de banco de dados; banco de dados relacional.

Abstract

The new concept of the database, the model non-relational, known as NoSQL has been gaining ground in the technology area. This work addressed the concepts of both relational and non-relational databases, the story, the advantages and disadvantages, and the characteristics of each one. In addition, presented several works related to the theme and tests results conducted with the performance of the selected databases work. The focus of this paper is the comparative analysis among the performance of insertion operations, search, edit, and delete operations on MySQL, PostgreSQL and MongoDB. In order to equal the environments to the tests it was necessary to virtualize an environment for each database, the software used was VMware.

Keywords- performance: NoSQL; database performance; relational database.

INTRODUÇÃO

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 (BRITO, 2010).

Esse grande crescimento de dados está relacionado ao Big Data, que pode ser definido como o processamento analítico de grande volume de dados produzido por várias aplicações (CUZZOCREA; DAVIS; SONG, 2011) apud (FIGUEIREDO et al., 2012). Todos os dados oriundos da população, cada vez mais conectada, formam um grande banco de dados não estruturado. Estima-se que cerca de 2,5 quinquilhões de bytes são gerados a cada dia (CISCO, 2013) apud (AMORIM; MOURA, 2014).

Pensando em solucionar as limitações dos bancos de dados relacionais, desenvolvedores criaram o NoSQL, uma alternativa de alto armazenamento com velocidade e grande disponibilidade, se livrando de algumas regras do modelo relacional. Por um lado existe a ruptura das regras do modelo relacional, mas por outro lado houve um ganho significativo de desempenho e flexibilidade (KOKAY, 2012).

A realização deste trabalho é justificada por esse grande crescimento do volume de dados, e com a questão de analisar os bancos de dados NoSQL e SQL no gerenciamento das operações de inserção, busca, alteração e exclusão de dados.

O objetivo do trabalho é comparar o desempenho dos processos transacionais e da recuperação da informação dos bancos de dados SQL e NoSQL, analisar os resultados e mostrar os bancos de dados mais eficientes.

No capítulo 1 e 2 serão apresentados os conceitos, história, arquitetura, vantagens e desvantagens e características dos bancos de dados SQL e NoSQL.

No capítulo 3 é apresentada a história do teste de software, quem tem como um tipo de teste o teste de desempenho ou performance, além de tratar de algumas ferramentas utilizadas nesses tipos de teste.

A preparação do ambiente de teste, sua arquitetura, configurações das máquinas virtuais, os trabalhos relacionados ao tema, realização dos testes, parâmetros utilizados, método de extração dos dados para análise são apresentados no capítulo 4.

No capítulo 6 são apresentados os resultados dos testes realizados.

CAPÍTULO 1 - SGBD RELACIONAL

Como será visto mais a frente, o modelo relacional revolucionou a área de banco de dados. Segundo Silberschatz, Korth e Sudarshan (1999):

> Um Sistema Gerenciador de Banco de Dados (SGBD) é constituído por um conjunto de dados associados a um conjunto de programas para acesso a esses dados. O conjunto de dados, comumente chamado banco de dados, contém informações sobre uma empresa em particular. O principal objetivo de um SGBD é proporcionar um ambiente tanto conveniente quanto eficiente para a recuperação e armazenamento das informações do banco de dados.

Segundo Date (2004) o modelo relacional é constantemente descrito tendo três aspectos: aspecto estrutural, aspecto de integridade e aspecto manipulador. No aspecto estrutural os dados são percebidos pelo usuário como tabelas, e nada além disso. No aspecto de integridade essas tabelas satisfazem restrições de integridade. E no aspecto manipulador os operadores disponíveis para que o usuário possa manipular tabelas são operadores que derivam tabelas a partir de outras tabelas. Três operadores particularmente importantes são os de restrição (extrai linhas específicas de uma tabela), projeção (extrai colunas específicas de uma tabela) e junção (une duas tabelas com base em valores comuns em uma coluna comum).

Um grande equívoco que ainda existe nos dias de hoje é chamar um SGBD de Banco de Dados. Como já foi definido, um SGBD é composto por um conjunto de softwares que gerenciam o banco de dados, e o "banco de dados é o conjunto de dados inter-relacionados, representando informações sobre um domínio". (KORTH, 2001) apud REZENDE (2006, p. 1). Pode-se dizer que tudo o que é feito em um banco de dados passa pelo SGBD.

Os SGBD's são projetados para gerenciar uma grande massa de informações. Esse gerenciamento de informações implica na definição de estruturas de armazenamento das informações e a definição dos mecanismos para a manipulação dessas informações. Um sistema de banco de dados também deve garantir a segurança das informações, além de não permitir acessos não autorizados e resultados anormais (SILBERSCHATZ; KORTH; SUDARSHAN, 1999).

1.1 - História do modelo relacional

Antes do surgimento do modelo relacional as empresas usavam sistemas de processamento de arquivos, mas este modelo apresentava muitas desvantagens tais como: inconsistência e redundância de dados, dificuldade de acesso aos dados, isolamento de dados, problemas de integridade, problemas de atomicidade, anomalias no acesso concorrente, problemas de segurança. O modelo relacional surgiu em 1970, proposto por Edgar Codd. Uma

proposta de grande valor, que revolucionou a área de banco de dados, até então dominada por modelos de rede e hierárquico. De acordo com Macário e Baldo (2005, p. 1):

> Neste seu trabalho Codd mostrou que uma visão relacional dos dados permite a sua descrição em uma maneira natural, sem que sejam necessárias estruturas adicionais para sua representação, provendo uma maior independência dos dados em relação aos programas. Em complementação, apresentou bases para tratar problemas como redundância e consistência. Mais tarde, em outro trabalho, Codd definiu uma álgebra relacional e provou, por meio de sua equivalência com o cálculo relacional, que ela era relacionalmente completa, dando fundamentação teórica ao modelo relacional.

Nos anos 70 a IBM começou a desenvolver os primeiros sistemas relacionais, como o Sistema-R e o Ingres. O uso desses sistemas de bancos de dados intensificou-se e passou a ser dominante na área (MACÁRIO; BALDO, 2005, p. 2).

1.2 - Arquitetura de um SGBD

Os SGBD's relacionais permitem a abstração de dados. Segundo ELMASRI e NAVATHE (2011, p. 19) apud COSTA (2011, p. 13), a abstração de dados "refere-se à supressão de detalhes da organização e armazenamento de dados, destacando recursos essenciais para um melhor conhecimento desses dados". Isso quer dizer que, pode-se descrever um banco de dados sem prender-se as especificações de hardware e software.

E para descrever um banco, não se prendendo à maneira como ele será implementado utiliza-se a modelos de dados. Na definição de ELMASRI e NAVATHE (2011, p. 19) apud COSTA (2011, p. 13), um modelo de dados é "uma coleção de conceitos que podem ser usados para descrever a estrutura de um banco de dados". A arquitetura de um SGBD se divide em três níveis: físico, conceitual e externo.

- Nível Físico: o mais baixo nível de abstração, neste nível é descrito como os dados estão realmente armazenados.
- Nível Conceitual: neste nível temos o modelo entidade-relacionamento, mostra quais os relacionamentos existentes entre os dados, e o que realmente está contido no banco. Este nível é utilizado por administradores de banco de dados.
- Nível Externo: nível onde os dados são apresentados aos usuários finais.

Na Figura 1 é apresentada a arquitetura das camadas de um SGBD.

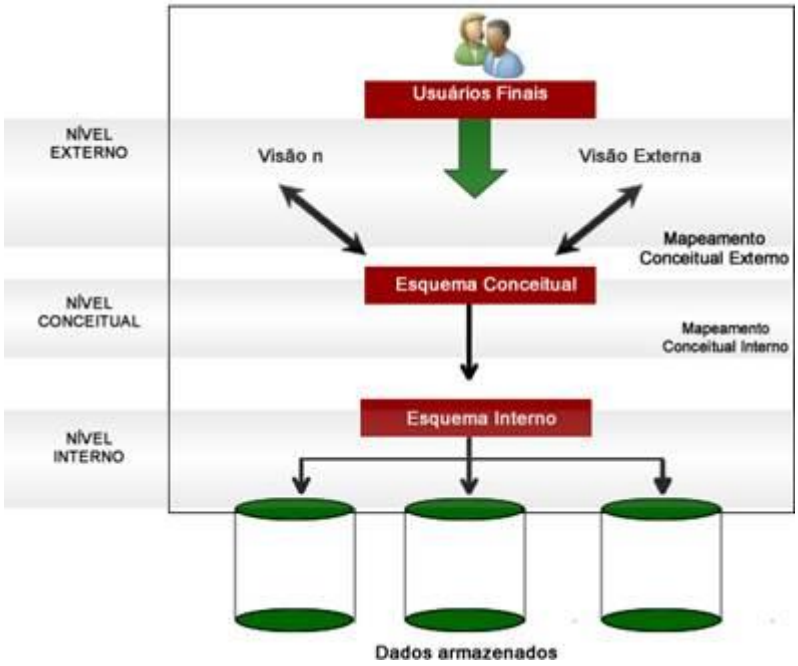


Fig. 1. Arquitetura das camadas de um SGBD. (http://www.devmedia.com.br/arquitetura-de-um-sgbd/25007).

1.2.1 - Esquema

A descrição do banco de dados denomina-se esquema. Esse é definido na fase de projeto do banco de dados e, geralmente, sofre pouca mudança (ELMASRI; NAVATHE, 2011, p. 21) apud COSTA (2011, p. 14).

1.2.2 - Instância

"O conjunto de informações contidas em determinado banco de dados, em um dado momento, é chamado instância do banco de dados" (SILBERSCHATZ; KORTH; SUDARSHAN, 1999, p. 6).

A instância é como uma fotografia atual da base de dados, ela tende a mudar com frequência, pois a cada atualização do banco de dados obtém-se um novo estado (COSTA, 2011, p. 14).

1.3 - Vantagens de um SGBD relacional

As vantagens desses tipos de sistemas de banco de dados são inúmeras. Pode-se citar a sua rapidez, a diminuição do volume e acumulo de papeis e arquivos, diminuição do trabalho manual, dados atualizados. Mas essas vantagens são as mais básicas quando se fala em banco de dados, e quais as melhorias que esse tipo de sistema proporcionou.

O que se pode citar como uma grande vantagem para as organizações devido ao uso de sistemas de bancos de dados é que, esses sistemas proporcionam as empresas o controle centralizado dos seus dados operacionais. Algumas vantagens são resultantes da noção de controle centralizado, como: reduzir a redundância, evitar a inconsistência, compartilhar dados, reforçar padrões, aplicar restrições de segurança, manter a integridade, equilibrar as necessidades conflitantes.

1.3.1 - Redução de redundância

Em um dicionário qualquer, é possível encontrar a definição para a palavra redundância, que nada mais é que, uma ideia contida em um termo expresso anteriormente. Em um banco de dados, a redundância é quando um dado se repete desnecessariamente entre duas entidades, o que pode ocupar mais espaço do que o esperado.

Em um SGBD a redundância pode ser reduzida, mas nunca eliminada já que em alguns casos existem razões, técnicas ou comerciais para se manter cópias múltiplas de um dado (KAMADA, 2004).

1.3.2 - Evitar a inconsistência

É uma consequência natural da redundância. Ocorre quando uma redundância não é controlada, e então apenas um dado é atualizado. E com isso o banco de dados pode fornecer dados incorretos (KAMADA, 2004).

Ainda, segundo Kamada (2004), se a redundância for removida a inconsistência não poderá acontecer. Mas se a redundância for controlada, o DBA (Administrador de Banco de Dados) pode garantir que o banco de dados não estará inconsistente, garantindo que uma alteração em um dado seja automaticamente feita no outro. Mas poucos sistemas são capazes de propagar automaticamente as atualizações, exceto em poucos casos especiais.

1.3.3 - Compartilhar dados

O compartilhamento não acontece somente em aplicações existentes, mas também em novas aplicações. As necessidades de dados de novas aplicações podem ser satisfeitas sem a criação de mais dados (KAMADA, 2004).

1.3.4 - Reforçar padrões

Garante que as informações armazenadas sigam um padrão, facilitando a utilização dos dados por vários sistemas.

O DBA, pelo controle central do banco de dados, pode assegurar que todos os padrões aplicáveis serão observados na representação dos dados (KAMADA, 2004).

1.3.5 - Restrições de segurança

O DBA tendo toda a autoridade sobre os dados operacionais pode certificar-se que os acessos ao banco de dados sejam realizados através de certos canais, e definir os controles de segurança a adotar ao acesso a determinados dados. Pode estabelecer controles para cada tipo de acesso e informação. Mas os dados sem os controles de segurança podem trazer um risco maior do que um sistema de arquivos, isso porque a natureza centralizadora do sistema de banco de dados requer um bom sistema de segurança (KAMADA, 2004).

1.3.6 - Manter a Integridade

De acordo com Kamada (2004, p. 13), a dificuldade encontrada em manter a integridade é conseguir que os bancos de dados sejam corretos. O controle centralizado do banco de dados ajuda a evitar problemas de redundância – se for possível evitá-la – pois permite que o DBA defina controles de integridade sempre que for realizada qualquer operação de atualização.

A integridade dos dados é ainda mais importante em sistemas de banco de dados de usuários múltiplos, pois o banco é compartilhado. Sendo assim um usuário poderia atualizar um dado de forma incorreta, e afetando outros usuários que necessitariam deste dado.

1.3.7 - Equilibrar as necessidades conflitantes

O DBA tem a capacidade de solucionar prioridades altas de todos os sistemas, avaliando a real necessidade de cada sistema para priorizar a sua implantação, formando um serviço geral que seja o melhor para a empresa (KAMADA, 2004).

1.3.8 - Independência de dados

Consiste na capacidade de isolar programas de aplicação das mudanças em estruturas de armazenamento, definição dos dados e das estratégias de acesso do banco de dados. Um SGBD que diz oferecer independência de dados garante que programas continuem executando mesmo que os dados armazenados sejam reorganizados para atendem a outra aplicação (BITTENCOURT, 2004).

Segundo Silva (1996, p. 4):

> Representa a forma física de armazenamento dos dados no Banco de Dados e a recuperação das informações pelos programas de aplicação. Esta recuperação deverá ser totalmente independente da maneira com que os dados estão fisicamente armazenados. Quando um programa retira ou inclui dados, o SGBD compacta-os para que haja um menor consumo de espaço no disco. Este conhecimento do formato de armazenamento do campo é totalmente transparente para o usuário.

1.4 - Desvantagens de um SGBD relacional

Apesar da grande revolução que o modelo relacional trouxe para a área de banco de dados, ainda é possível citar algumas limitações presentes nos dias de hoje.

1.4.1 - Custo de escalabilidade

Um dos grandes problemas de um SGBD relacional é a capacidade de lidar com o grande crescimento de dados, que tem se expandido rapidamente. Quando um sistema tem um ganho significativo de usuários o desempenho cai, e para suprir essa perda é necessário fazer um upgrade ou aumentar o número de servidores, mas se o número de usuários continuasse a crescer esse método não se mostraria muito eficiente, pois o problema passa a ser o acesso à base de dados. Para resolver esse problema a solução seria utilizar a escalabilidade vertical (aumento de processamento, memória e armazenamento) ou a escalabilidade horizontal (aumento do número de máquinas no servidor web), mas essas soluções trariam problemas de alto custo e complexidade, já que tais mudanças resultariam em mudanças de configurações das aplicações (KOKAY, 2012).

1.4.2 - Solicitação de mudança

Modelos relacionais apesar de sua maturidade e consistência não seguem as novas dinâmicas de desenvolvimento de software, que sempre estão em constantes mudanças, até mesmo durante o ciclo de desenvolvimento ou a manutenção evolutiva (MIGUEL; CARNEIRO, 2013, p. 1).

1.4.3 - Custo do join e programação orientada a objetos

Desde sua criação o modelo relacional sofreu pequenas mudanças para acomodar exigências vindas dos requisitos de negócio. O banco de dados precisa trabalhar em conjunto com o software, e devido a esse avanço pequeno o modelo de dados não é aderente à estrutura flexível do software. A programação orientada a objetos utiliza conceitos de generalização e especialização, quando aplicada a um modelo relacional pode ser feita de duas formas: criando

uma tabela de dados que contém todos os atributos, sendo algumas informações não preenchidas causando a fragmentação em partes da tabela, ou então, é criada uma tabela adicional para guardar informações específicas, e essa tabela é relacionada pela chave estrangeira, nesse caso evita-se a fragmentação, mas é preciso usar o join, que tem um alto custo de processamento (CARNEIRO; MIGUEL, 2013). A Figura 2 apresenta um exemplo de generalização utilizada em linguagens orientada a objetos.

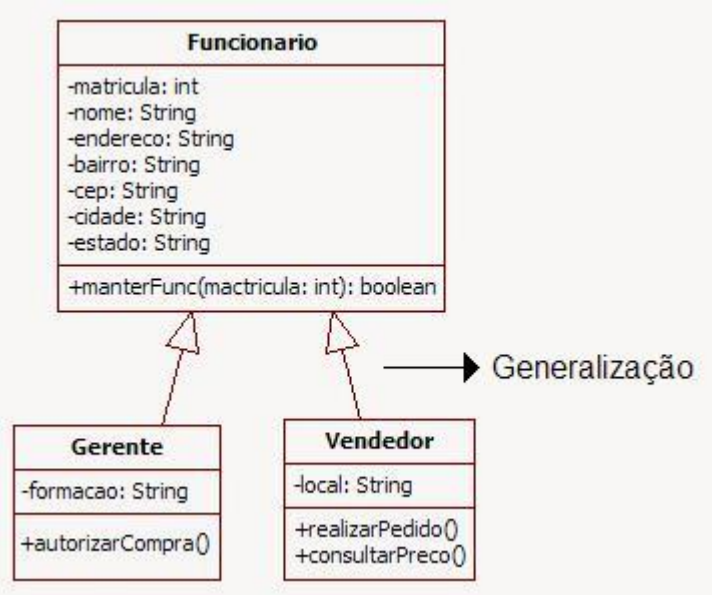


Fig. 2. Exemplo de generalização. (CARNEIRO; MIGUEL, 2013)

1.5 - Desempenho

Ambientes de bancos de dados são dinâmicos, as configurações podem mudar, e ter um ganho significativo de dados e transações. Com isso fica difícil de prever o desempenho de um banco de dados.

De “70% a 80% de todos os problemas de desempenho em aplicações que utilizam banco de dados são causados por consultas SQL mal estruturadas” (MULLINS, 1998 apud ANDRADE, 2005, p. 13).

“Instruções SQL e índices são responsáveis por 60% a 90% dos problemas de desempenho de aplicações” (LECCO, 2003 apud ANDRADE, 2005, p. 13).

“Atividades relacionadas às consultas SQL consomem entre 70% e 90% dos recursos dos bancos de dados” (LECCO, 2003 apud ANDRADE, 2005, p. 14).

O desempenho de banco de dados está ligado desde a sua arquitetura, normalizações, design até as consultas realizadas. E essas consultas são grandes responsáveis por deixar o banco de dados com um bom desempenho.

1.5.1 - Índices

Segundo OLIVEIRA (2009, p. 1), “os índices são estruturas opcionais associadas às tabelas, as quais aumentam o desempenho da recuperação de dados”. Os índices facilitam a localizar informações.

A indexação é a maneira mais fácil e rápida de melhorar a eficiência das consultas SQL. Essa estratégia pode melhorar ou piorar o desempenho de um sistema. Mas depender unicamente do uso de índices não é uma solução confiável. Uma escolha errada de recuperação de índices pode implicar na piora do desempenho de uma consulta SQL. Uma boa prática é utilizar tabelas que são frequentemente utilizadas para a indexação (ANDRADE, 2005).

1.5.2 - Otimização de consultas

Uma consulta SQL primeiramente é lida, analisada e validada. A varredura identifica as palavras chaves SQL, os nomes dos atributos e das relações que aparecem no texto da consulta, o analisador sintático verifica a sintaxe da consulta para determinar se está dentro das regras gramaticais. A consulta é validada, é verificado se os nomes dos atributos e relações são válidos. Então é criada uma estrutura interna, uma árvore de consulta ou grafo de consulta. O SGBD então precisa definir uma estratégia de execução para recuperar os resultados da consulta. O processo de escolha de uma estratégia para o processamento da consulta é chamado de otimização de consulta (NAVATHE; ELMASRI, 2011). A Figura 3 representa o processo de uma consulta no modelo relacional.

Fig. 1. Processo de consulta no modelo relacional. (NAVATHE; ELMASRI, 2011).

Segundo Silberschatz et al. (1999) apud Andrade (2005, p. 14-15):

> O processamento de consultas é uma atividade que permite extrair dados de um banco de dados. Essa atividade inclui a tradução de consultas empresas em linguagens de alto nível do banco de dados em expressões que podem ser implementadas no nível físico do sistema de arquivos, otimizações, traduções e avaliação das consultas.

Segundo Date (2000) apud Andrade (2005, p. 18):

> A otimização representa ao mesmo tempo desafio, visto que, a otimização é uma exigência para os sistemas relacionais quando se espera um desempenho que atinja níveis pré-determinados, e uma oportunidade, já que a otimização é favorecida pelo fato das expressões relacionais estarem em um nível semântico adequado para que seja possível essa otimização.

A otimização de consultas SQL é complexa, pois existe um grande número de formas para isso. Isso porque a sintaxe por si só é complexa, e podem existir vários meios de executa-la.

Segundo Navathe e Elmasri (2011) o termo otimização é empregado de maneira equivocado, pois em alguns casos o plano de execução escolhido não é o melhor, mas sim uma estratégia razoável. Encontrar a estratégia ideal é demorado e pode exigir informações que podem não estar totalmente disponíveis.

No processo de otimização de uma consulta SQL, a query utilizada é traduzida para álgebra relacional estendida, representada como uma árvore de consulta. As consultas SQL são decompostas em blocos. Considere a seguinte consulta:

```
SELECT Unome, Pnome
FROM FUNCIONARIO
WHERE Salario > (SELECT MAX (Salario)
                 FROM FUNCIONARIO
                 WHERE Dnr = 5);
```

A consulta retorna os nomes dos funcionários que ganham um salário maior que o maior salário do departamento cinco. Essa consulta possui uma subconsulta, portando é decomposta em dois blocos (NAVATHE, ELMASRI, 2011).

Uma das técnicas usadas para otimizar consultas SQL é a heurística, que modifica a representação interna da consulta para melhorar o seu desempenho. Aplicar regras de seleção e junção antes de aplicar a junção é uma das principais regras heurísticas, pois o tamanho do arquivo resultante de uma junção, por exemplo, é uma função multiplicativa dos tamanhos dos arquivos de entrada. E as operações de seleção e projeção reduzem o tamanho de um arquivo, por isso devem ser aplicadas antes (NAVATHE; ELMASRI, 2011).

As regras heurísticas não são o único modo de otimizar consultas SQL, usa-se também a técnica de otimização de consulta baseada em custo. Ela usa técnicas tradicionais de otimização que minimiza o custo da busca. As funções de custos são estimativas e não exatas, com isso pode ser que a estratégia adotada não seja a ideal. Nesta técnica são analisados alguns componentes: custo de acesso ao armazenamento secundário (Conhecido como custo de entrada e saída de disco), custo de armazenamento em disco (Custo de armazenar arquivos intermediários que são gerados pela estratégia de execução), custo de computação (Custo de CPU), custo de uso da memória (Número de buffers da memória principal necessários durante a execução da consulta), custo de comunicação (Custo de envio e recebimento da consulta e resultados) (NAVATHE; ELMASRI, 2011).

CAPÍTULO 2 - NOSQL

Pensando em solucionar as limitações dos bancos de dados relacionais, como a dificuldade em tratar dados não estruturados e escalar horizontalmente desenvolvedores criaram o NoSQL, uma alternativa de alto armazenamento com velocidade e grande disponibilidade, se livrando de algumas regras do modelo relacional. Por um lado existe a ruptura das regras do modelo relacional, mas por outro lado houve um ganho significativo de desempenho e flexibilidade (KOKAY, 2012).

2.1 - História do NoSQL

Na literatura o termo NoSQL surgiu em 1998 com Carlo Strozzi. Na época ele utilizou o termo para batizar seu projeto open source que tinha como objetivo de criar um SGBD relacional mais leve e sem interface SQL (PORCELLI, 2010).

No ano de 2006 o Google publicou o artigo “BigTable: A Distributed Storage System for Structured Data”, e trouxe à tona novamente o tema NoSQL (LANNI, 2012). Somente em 2009 que o movimento NoSQL teve seu início, o termo foi usado para nomear um encontro entre Johan Oskarsson e Eric Evans que teve como objetivo discutir o grande número de soluções open source de armazenamento de dados distribuídos não relacionais. E no mesmo ano foi redefinido o uso do tempo NoSQL para descrever soluções de armazenamento de dados não relacionais. Esse novo conceito não veio com a intenção de substituir o modelo relacional, mas sim para auxiliar grandes aplicações que precisam de mais flexibilidade na estruturação do banco, embora o termo NoSQL de a entender “Não ao SQL” significa “Não só SQL” (PORCELLI, 2010).

Com o interesse das grandes empresas da área da tecnologia os SGBD’s não relacionais ganharam força, e hoje são amplamente usados para mitigar os problemas de escalabilidade e custo.

2.2 - Big Data

Um dos motivos para a criação do NoSQL foi o aumento do volume e variedade de dados, o que pode-se chamar de Big Data.

A população passou a ficar cada vez mais conectada, enviando e recebendo dados através da internet. Toda essa informação, que tem como informações dados de redes sociais, envio de vídeos, transações comerciais, cliques em sites, entre outros, fazem parte de um banco

de dados não estruturado. Cerca de 2,5 quintilhões de bytes são gerados a cada dia (CISCO, 2013) apud (AMORIM; MOURA, 2014).

Segundo a INTEL (2012) apud Amorim e Moura (2014):

> A civilização criou até 2003 cinco exabytes de informação e hoje esse volume é criado em apenas dois dias. Em 2012, o universo digital de dados cresceu 2,72 zettabytes (ZB) e irá dobrar a cada dois anos, atingindo 8 ZB em 2015. Isso equivale ao acervo de 18 milhões de Bibliotecas do Congresso dos Estados Unidos. Dois bilhões de dispositivos conectados, que vão desde Computadores e smartphones a dispositivos com sensores tais como, leitores de Rádio Frequency Identification (RFID) e câmeras de tráfego, geram essa enxurrada de dados complexos, estruturados e não estruturados.

O termo Big Data ainda não tem uma definição precisa, porém pode ser definido como o processamento analítico de grande volume de dados complexos produzidos por várias aplicações (CUZZOCREA; DAVIS; SONG, 2011) apud (FIGUEIREIDO et al., 2012).

Os diferentes aspectos do Big Data são caracterizados pelos três Vs, que são definidos pela INTEL (2013) apud Amorim e Moura (2014):

> - Volume. A escala maciça e crescimento de dados não estruturados superar armazenamento tradicional e soluções analíticas.
> - Variedade. Big data é coletado a partir de novas fontes que não foram minadas no passado. Processos tradicionais de gestão de dados não podem lidar com a diversidade e variação dos dados do Big data, que vem em formatos tão diferentes como e-mail, redes sociais, vídeo, imagens, blogs, e sensores de dados.
> - Velocidade. Os dados são gerados em tempo real, com as exigências de informação útil a ser servida.

2.3 - Características

Os SGBD's não relacionais possuem características importantes que fazem esse tipo de sistema ser diferente do modelo relacional, como a facilidade em escalar horizontalmente, por exemplo.

2.3.1 - Escalabilidade horizontal

Modelos relacionais podem escalar horizontalmente, mas essa tarefa não é fácil. A ausência de bloqueios em banco de dados NoSQL é uma característica fundamental para permitir a escalabilidade horizontal, o que o torna adequado a solução de gerenciamento de grande volume de dados que crescem exponencialmente. Uma maneira de alcançar a

escalabilidade horizontal é utilizando “Sharding”, que divide os dados em diversas tabelas a serem armazenadas ao longo de diversos nós de uma rede, essa técnica tem o problema de romper a lógica de relacionamentos, neste caso as aplicações têm que gerenciar a complexidade gerada pela fragmentação de informações (LÓSCIO; OLIVEIRA; PONTES, 2011).

2.3.2 - Ausência de esquema

A ausência de esquema é uma característica forte dos bancos de dados NoSQL, é essa falta de esquema que permite de maneira fácil a escalabilidade e aumento da disponibilidade. Mas devido a isso, bancos de dados NoSQL não garantem a integridade dos dados, diferente dos modelos relacionais que mantem uma estrutura rígida (LÓSCIO; OLIVEIRA; PONTES, 2011).

2.3.3 - Suporte nativo a replicação

A replicação também é uma maneira de escalar, e com a replicação nativa o tempo gasto para a recuperação de informações é mais rápido. Segundo Lóscio, Oliveira e Pontes (2011), as principais maneiras para replicação são a “master-slave” e “multi-master”.

- Master-Slave: Esta abordagem não é muito aconselhada quando se tem um grande volume de dados. A cada escrita no banco, N nós escravos são criados, a escrita é feita no nó mestre, e o mestre refaz a escrita em cada nó escravo. Isso torna a leitura mais rápida.
- Multi-Master: Admite-se vários nós mestres, diminuindo o gargalo gerado pela abordagem master-slave. Mas isso acarreta o problema de conflito de dados.

2.3.4 - API simples para acesso aos dados

A solução NoSQL não está focada em como os dados são armazenados, mas sim em recupera-los de forma eficiente, oferecendo alta disponibilidade e escalabilidade. Para isso APIs simples são desenvolvidas para facilitar o acesso, permitindo que qualquer aplicação possa utiliza-las de forma rápida e eficiente (LÓSCIO; OLIVEIRA; PONTES, 2011).

2.3.5 - Consistência eventual

A consistência nem sempre é mantida entre os diversos pontos de distribuição de dados, esta característica leva em conta o teorema CAP “Consistency, Availability e Partition tolerance”, que foi criado por Eric Brewer. No teorema Eric diz que não é possível obter as três

características em simultâneo, e bancos de dados NoSQL focam em disponibilidade e tolerância a falhas (LÓSCIO; OLIVEIRA; PONTES, 2011).

2.4 - Teorema CAP

Como já mencionado anteriormente, as três letras do CAP referem-se à consistência, disponibilidade e tolerância a partição.

Segundo Diana e Gerosa (2010, p. 4), explicam cada uma das letras do teorema CAP:

> Consistência nesse contexto não tem exatamente o mesmo significado da consistência de transações de banco de dados, mas sim diz respeito á ordem de execuções de requisições, e significa que uma leitura de um item após uma escrita desse item deve retornar o novo valor.
> Disponibilidade é a propriedade de um sistema responder a todas as requisições que chegam a um nó funcionando.
> Tolerância à partição é a propriedade de um sistema continuar funcionando mesmo quando um problema ocorre na rede dividindo o sistema em duas ou mais partições, o que faz com que nós de uma mesma partição não consigam se comunicar com as outras.

Um aplicativo Web deve escolher duas dessas três características para contemplar, não sendo possível utilizar as três com qualquer projeto de banco de dados. Obviamente, qualquer estratégia de escala horizontal é baseada no particionamento de dados, portanto, os designers são forçados a decidir entre consistência e disponibilidade (PRITCHETT, 2008).

2.5 - Técnicas para implementação das funcionalidades do NoSQL

Segundo Lóscio, Oliveira e Pontes (2011, p. 5), existem quatro técnicas importantes para a implementação das funcionalidades do NoSQL:

> O map/reduce dá suporte ao manuseio de grandes volumes de dados distribuídos ao longo dos nós de uma rede. Na fase de map, os problemas são quebrados em subproblemas que são distribuídos em outros nós na rede. E na fase reduce, os subproblemas são resolvidos em cada nó filho e o resultado é repassado ao pai, que, sendo ele também filho, repassaria ao seu pai, e assim por diante até chegar ao nó raiz do problema.
> O consistent hashing suporta mecanismos de armazenamento e recuperação em banco de dados distribuídos, onde a quantidade de sites está em constante modificação. O uso desta técnica é interessante porque evita um grande número de migração de dados entre esses sites, os quais podem ser alocados e desalocados para a distribuição dos dados.
> O multiversion concurrency control (MVCC) é um mecanismo que dá suporte a transações paralelas em um banco de dados. Ao contrário do esquema clássico de gerenciamento de transações, como não faz uso de locks, permite que operações de leitura e escrita sejam feitas simultaneamente.

> Vector clocks são usados para gerar uma ordenação dos eventos acontecidos em um sistema. Pela possibilidade de várias operações estarem acontecendo ao mesmo tempo sobre o mesmo item de dado distribuído, o uso de um log de operações com as suas respectivas datas é importante para determinar qual a versão de um determinado dado é a mais atual.

2.6 - Modelos NoSQL

Este capítulo apresenta os principais modelos de dados NoSQL. Os modelos discutidos são: chave-valor, orientado a colunas, orientado a documentos e orientado a grafos.

2.4.1 - Chave-valor

Modelo bem simples, permite a visualização do banco de dados como uma tabela hash. De maneira simples, o banco é composto por um conjunto de chaves que estão associadas a um único valor, podendo ser string ou binário (LÓSCIO; OLIVEIRA; PONTES, 2011). Segundo Miguel e Carneiro (2013) "informações são armazenadas em objetos indexados por chaves, responsáveis por identificá-los unicamente e realizar sua posterior consulta". A Figura 4 representa um modelo de chave-valor.

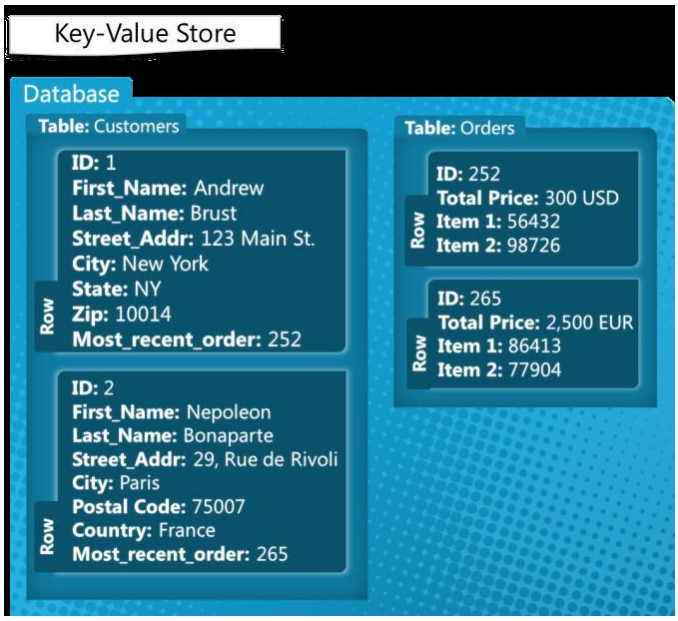


Fig. 4. Modelo NoSQL chave valor. (http://blogs.msdn.com/b/usisvde/archive/2012/04/05/getting-acquainted-with-nosql-on-windows-azure.aspx).

O modelo chave-valor é simples de ser implementado, e por isso permite que os dados sejam acessados rapidamente pela chave. As operações usadas para a manipulação dos dados são o get( ) e set( ), que retornam e capturam os dados, respectivamente. A desvantagem desse

modelo é que não é possível recuperar objetos por meio de consultas complexas (LÓSCIO; OLIVEIRA; PONTES, 2011).

2.4.2 - Orientado a colunas

O modelo orientado a colunas é mais complexo que o modelo chave-valor, e também muda o paradigma de orientação a registros para orientação a colunas, foi criado para armazenar e processar grande quantidade de dados distribuídos em diversas máquinas (KOKAY, 2012).

Os dados deste modelo são indexados por linha, coluna e timestamp, onde linhas e colunas são identificadas por chaves e o timestamp permite que um dado possa ter diversas versões. Este modelo é associado ao conceito de família de colunas, que agrupa colunas que armazenam o mesmo tipo de dados. A Figura 5 representa um exemplo baseado em colunas, onde "primeiroNome" e "sobrenome" fazem parte da família nome, e os campos "endereço" e "cidade" à família local. Pode-se observar que os campos "endereço" e "cidade" possuem mais de uma informação, isso devido ao timestamp permitir mais versões do mesmo dado. Um fato importante é que as ações de escrita e consulta são atômicas, ou seja, quando um valor é consultado todos os dados são levados em conta e retornados para o usuário (LÓSCIO; OLIVEIRA; PONTES, 2011).

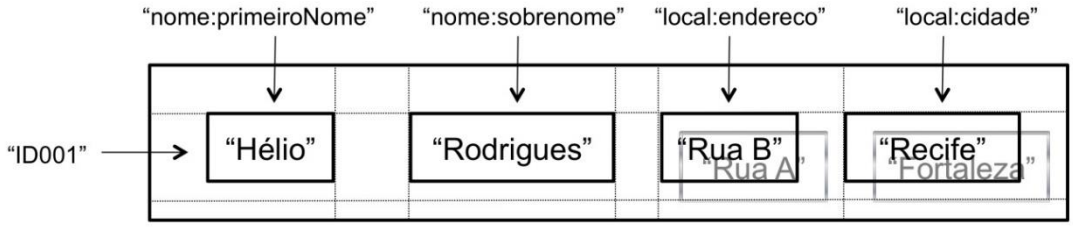


Fig. 5. Exemplo de modelo baseado em colunas. (LÓSCIO; OLIVEIRA; PONTES, 2011).

2.4.3 - Orientado a documentos

Documentos são objetos com um identificador único e um conjunto de campos, que podem ser strings, listas ou documentos aninhados. Estes campos são semelhantes ao chave-valor, a diferença é que o modelo chave-valor cria apenas uma tabela hash para todo o banco, e no modelo orientado a documentos existe um conjunto de documentos; em cada documento um conjunto de campos e os valores dos campos. O modelo orientado a documentos não exige uma estrutura fixa como nos bancos relacionais, isso permite que a estrutura seja atualizada, adicionando novos campos sem trazer problemas (LÓSCIO; OLIVEIRA; PONTES, 2011).

ID: P001
Assunto: “Eu gosto de laranjas”
Autor: “Hélio”
Data: “27/01/2011”
Tags: [“laranjas”, “suco”, “plantas”]
Mensagem: “Hoje estou com vontade de tomar suco de laranja!”

Fig. 6. Exemplo de modelo orientado a documentos. (LÓSCIO; OLIVEIRA; PONTES, 2011).

Como já foi dito, o modelo orientado a documentos pode ter sua estrutura atualizada, então no exemplo da Figura 6, seria possível criar um novo campo. Essa característica é uma das grandes vantagens do modelo.

2.4.4 - Orientado a grafos

O modelo orientado a grafos é composto por três componentes básicos: nós, que são os vértices do grafo, os relacionamentos que são representados pelas arestas e as propriedades dos nós e relacionamentos. Com isso o banco é visto como multigrafo e direcionado, podendo cada par de nós ser conectado por mais de uma aresta (LÓSCIO; OLIVEIRA; PONTES).

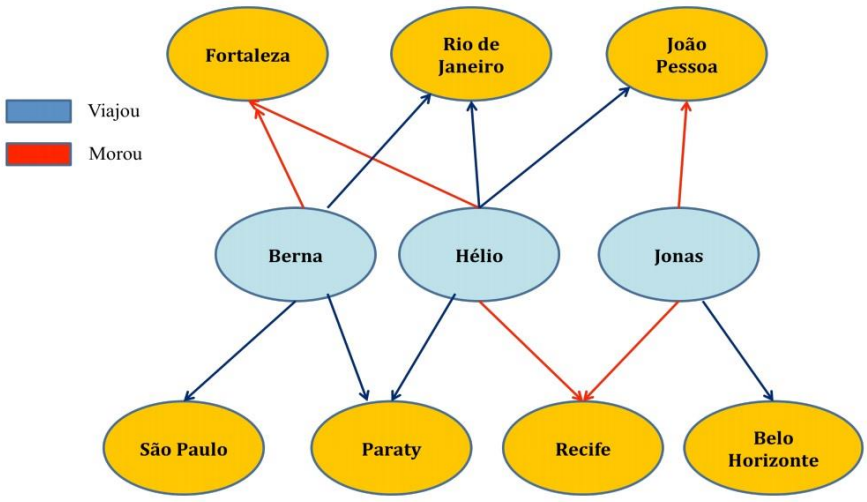


Fig. 7. Exemplo de modelo orientado a grafos. (LÓSCIO; OLIVEIRA; PONTES, 2011).

Na figura 7 pode-se ver um exemplo de informações relativas a locais de viagens, e onde Berna, Hélio e Jonas residem. Pensando em um ambiente como no da figura 7, um banco relacional realizaria consulta complexas, devido ao número de junções necessárias, mas no

modelo orientado a grafos os relacionamentos inerentes aos grafos tornaria a consulta mais simples e vantajosas, pois ganharia em performance.

CAPÍTULO 3 - TESTE DE PERFORMANCE EM BANCO DE DADOS

Com o crescente aumento do volume de dados, os bancos de dados enfrentam mais complexidade em gerenciar os dados oriundos da rede, e cada vez mais são necessários SGBD's que comportem essa demanda. Uma forma de verificar se um SGBD gerenciaria de forma eficiente e eficaz a um volume grande de informações é realizando um teste de performance também conhecido como "benchmark".

Basicamente, um benchmark consiste em um padrão de testes para medir e avaliar um software, utilizado também para analisar o desempenho de um sistema (LORENA, 2007 apud FERREIRA; JÚNIOR, 2012).

O teste de desempenho é um tipo de teste de software, o qual será tratado mais adiante.

3.1 - Teste de Software

Teste de software é o processo de execução de um produto para determinar se ele atingiu suas especificações e objetivos no ambiente em que foi projetado. Seu objetivo é identificar as falhas em um produto, fazendo com que sejam corrigidas antes da entrega final (NETO, 2008). Na fase de desenvolvimento de software a parte de realização dos testes pode ser identificada como verificação e validação.

Segundo Myers (2004) apud Chagas (2009):

> A atividade de teste é o processo de executar um programa com a intenção de encontrar um erro. Um bom caso de teste é aquele que tem uma elevada probabilidade de revelar um erro ainda não descoberto. Um teste bem sucedido é aquele que revela um erro ainda não descoberto.

Quanto mais tarde um defeito é encontrado em uma aplicação mais caro se torna para corrigi-lo, o custo em correção de defeitos cresce cerca de 10 vezes a cada novo estágio em que um software avança. Isso foi estabelecido por Glenford Myers em seu livro *The art of software testing* lançado em 1979. Myers ainda afirmava que os testes unitários poderiam remover de 30% a 50% dos defeitos de um programa, que testes de sistemas poderiam remover de 30% a 50% dos defeitos restantes. Ainda assim os sistemas podem ir para produção com um índice de 49% de defeitos, e que revisões de códigos podem reduzir entre 20% e 30% destes defeitos (BASTOS ET AL., 2012). A Figura 8 representa a regra 10 de Myers.

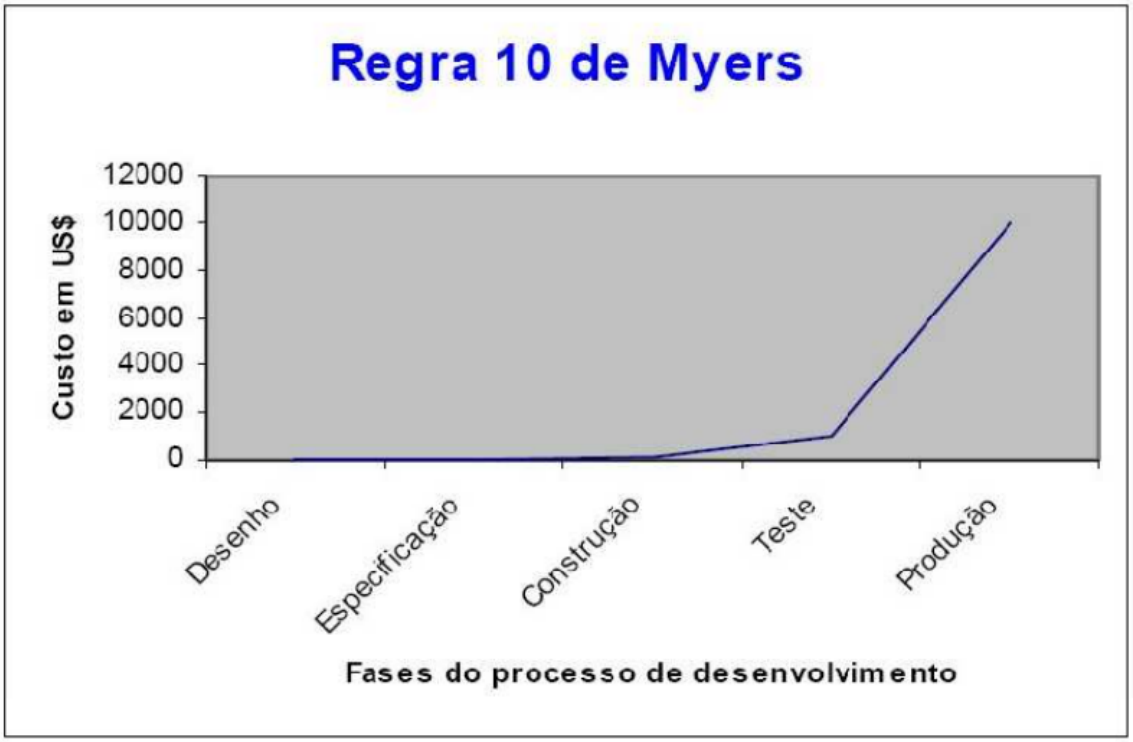


Fig. 8. Regra 10 de Myers. (BASTOS ET AL., 2012).

Antigamente os testes eram realizados pelos próprios desenvolvedores, denominados testes unitários e de integração. Esses testes eram realizados em funcionalidades realizadas apenas para um fim, isso faz com que o nível de cobertura de um teste fique baixo não permitindo que muitos defeitos fossem detectados. Os usuários muitas vezes eram envolvidos para a aprovação dos resultados dos testes, isso gerava um número elevado de defeitos em um produto que já estava em produção, o custo da correção de um defeito que está em produção se torna mais caro (BASTOS ET AL., 2012).

Com o surgimento de sistemas para internet as empresas ficaram mais expostas ao público. Além disso as complexidades das aplicações aumentavam com o surgimento de novas tecnologias. Isso fez com que as empresas procurassem meios para melhorar a qualidade dos softwares desenvolvidos. O processo de desenvolvimento de software foi melhorado, e as atividades de teste de software foram contempladas; o teste de software passou a ser realizado de forma paralela contando com pessoas especializadas. Esse processo fez com que a qualidade do software aumentasse, muitas empresas adotaram o novo modelo conhecido como Capability Maturity Model Integration (CMMI) (BASTOS ET AL., 2012). A Figura 9 representa as áreas do processo do modelo CMMI.

Níveis de maturidade	Áreas de processo
2 - Gerenciado	Gerenciamento de requisitos Planejamento de projeto Controle e monitoração de projeto Medições e análise Garantia de qualidade de produto e processo Gerenciamento de configuração
3 - Definido	Desenvolvimento de requisitos Solução técnica Integração de produto Verificação Validação Foco no processo organizacional Definição do processo organizacional Treinamento organizacional Gerenciamento integrado de projeto Gerenciamento de risco Integração de equipe Gerenciamento integrado de fornecedores Resolução e análise de decisão Ambiente organizacional para integração
4 - Quantitativamente gerenciado	Desempenho do processo organizacional Gerenciamento quantitativo de projeto
5 - Otimizado	Inovação e implantação na organização Análise e resolução de causas

Fig. 9. Áreas do processo do modelo CMMI. (BASTOS ET AL., 2012).

Após a otimização do processo de desenvolvimento de software área de teste começou a ser tratada como como um projeto; isso faz com que seja necessário adotar padrões relacionados a projetos como o Project Management Institute (PMI) por exemplo, uns dos mais utilizados (BASTOS ET AL., 2012). A Figura 10 representa os planos necessários em um gerenciamento de projetos.

- Plano Global do Projeto
- Plano de Gerenciamento de Escopo
- Plano de Gerenciamento de Tempo
- Plano de Gerenciamento de Custos
- Plano de Gerenciamento de Qualidade
- Plano de Gerenciamento de RH
- Plano de Gerenciamento de Comunicação
- Plano de Gerenciamento de Riscos
- Plano de Gerenciamento de Contratos

Fig. 10. Planos necessários para o gerenciamento de projetos. (BASTOS ET AL., 2012).

Desta maneira fica evidente que executar o processo de teste não é uma tarefa trivial, sua execução requer técnicas não conhecidas por desenvolvedores e usuários.

3.2 - Estratégias de Teste

A estratégia de teste descreve os passos a serem executados como parte do teste. Define em que momento eles serão planejados e executados, define também o trabalho, tempo e recursos necessários. Qualquer estratégia de teste deve vincular planejamento aos testes (PRESSMAN, 2011).

Pressman (2011, p. 402) diz que:

> Uma estratégia de teste deverá fornecer diretrizes para o profissional e uma série de metas para o gerente. Devido ao fato de os passos da estratégia de teste ocorrerem no instante em que as pressões pelo prazo começam a aumentar, deve ser possível medir o progresso no desenvolvimento e os problemas devem ser revelados o mais cedo possível.

O processo de software pode ser visto como uma espiral assim como a estratégia de teste. O teste de unidade começa no centro da espiral e se concentra em cada bloco do software. O teste continua em movimento, seguindo para fora da espiral, passando pelo teste de integração, que foca no projeto e na arquitetura do software. Seguindo, passa pelo teste de validação onde os requisitos de modelagem são validados. E finalmente o teste de sistema, onde os elementos são testados como um todo (PRESSMAN, 2011). Este processo é representado na Figura 11.

Fig. 11. Estratégia de teste. (PRESSMAN, 2011).

Considerando o processo de um modo procedimental, o teste é uma série de quatro etapas que acontecem de maneira sequencial. Inicialmente os testes focam nos componentes de forma individual para garantir seu funcionamento como uma unidade. Em seguida o componente deve ser integrado para formar um pacote completo de software, cuidando de problemas associados com aspectos duais de verificação e construção. Depois de integrado, é feito o teste de validação que proporciona a garantia de que o software está de acordo com todos os requisitos estabelecidos (PRESSMAN, 2011). A Figura 12 apresenta as etapas do teste de software em um projeto de desenvolvimento.

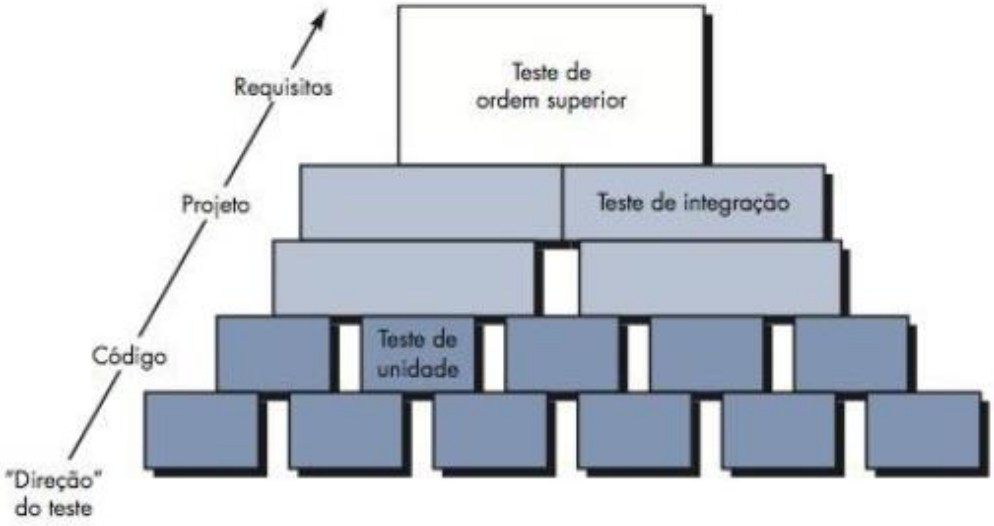


Fig. 12. Etapas do teste de software. (PRESSMAN, 2011).

3.3 - Técnicas de Teste

Atualmente existem diversas maneiras para se testar um software, existem técnicas que sempre foram muito utilizadas em softwares. Apesar de existirem diferentes técnicas o seu objetivo é o mesmo, encontrar defeitos em softwares.

Segundo Howden (1987) apud Modesto (2006):

> É importante ressaltar que as técnicas de teste devem ser vistas como complementares e a questão está em como utilizá-las de forma que as vantagens de cada uma sejam melhor exploradas, possibilitando uma estratégia que leve a uma atividade de teste de boa qualidade, ou seja, eficaz e de baixo custo.

3.3.1 - Teste Caixa Preta

Teste de caixa preta ou teste funcional são testes onde são dadas entradas e analisadas as saídas para verificar se estão de acordo com o especificado. Nestes tipos de testes não são considerados os softwares utilizados para a construção da aplicação assim como a estrutura do código.

Em princípio o teste funcional pode encontrar todos os defeitos, já que o software pode ser submetido a todas as entradas possíveis. No entanto, as entradas podem ser ilimitadas ou muito grandes fazendo com que o tempo de teste fique inviável, por isso o teste é dividido em diversas técnicas, e cada uma com seus critérios fazendo com que a atividade possa ser conduzida de maneira mais sistemática, podendo até mesmo ter uma avaliação quantitativa da atividade de teste (DELAMARO; MALDONADO; JINO, 2007).

"Os testes funcionais garantem o atendimento aos requisitos, ou seja, que os requisitos estejam corretamente codificados." (BASTOS ET AL., 2012).

3.3.2 - Teste Caixa Branca

Teste de caixa branca ou teste estrutural são os testes que verificam a estrutura de um software, suas tecnologias e codificação.

No teste de caixa branca os requisitos de testes são estabelecidos de acordo com uma determinada implementação isso requer a execução de partes ou de componentes elementares do programa. São testados os caminhos lógicos do programa, pondo a prova conjunto específicos de condições, laços e usos de variáveis (DELAMARO; MALDONADO; JINO, 2007).

"Os testes estruturais garantem que os softwares e os programas sejam estruturalmente sólidos e que funcionem no contexto técnico onde serão instalados." (BASTOS ET AL., 2012).

3.3.3 - Teste Caixa Cinza

O teste de caixa cinza é uma combinação do teste de caixa preta com o teste de caixa branca, no teste de caixa cinza é feita a análise da parte lógica e das funcionalidades especificadas para verificar se está tudo sendo feito como previsto. Nesta técnica o testador precisa se comunicar com os desenvolvedores para entender melhor o sistema para melhorar a cobertura dos testes (LEWIS, 2000).

3.4 - Níveis de Teste

Segundo Neto (2008) o planejamento dos testes ocorre em diversos níveis e em paralelo com o desenvolvimento do software. Os principais níveis de teste de softwares são (MALDONADO; ROCHA; WEBER, 2001):

- Teste de unidade: o objetivo deste nível é testar a menor unidade do projeto, procurando por falhas de lógica e implementação em cada módulo, de forma isolada. O alvo são trechos de códigos e os métodos dos objetos.
- Teste de integração: o objetivo deste nível é encontrar defeitos relacionados às interfaces entre os módulos quando são integrados para estruturar o software.
- Teste de sistema: são feitos testes para verificar se o software atende aos requisitos, os testes são feitos simulando o usuário final utilizando as mesmas entradas usadas no dia a dia.
- Teste de aceitação: são realizados por um pequeno grupo de usuários finais. São simuladas as operações de rotina do sistema para se verificar se seu comportamento está de acordo com o solicitado.

Os níveis de teste podem ser vistos no Modelo V, representado na Figura 13, que demonstra como as atividades de testes se relacionam com o desenvolvimento do software.

"O Modelo V foi definido por Paul Rook em 1980, foi apresentado como modelo alternativo ao modelo Waterfall, enfatizava a importante nos testes em todo o processo de desenvolvimento e não somente ao termino do processo." (SILVA, 2011).

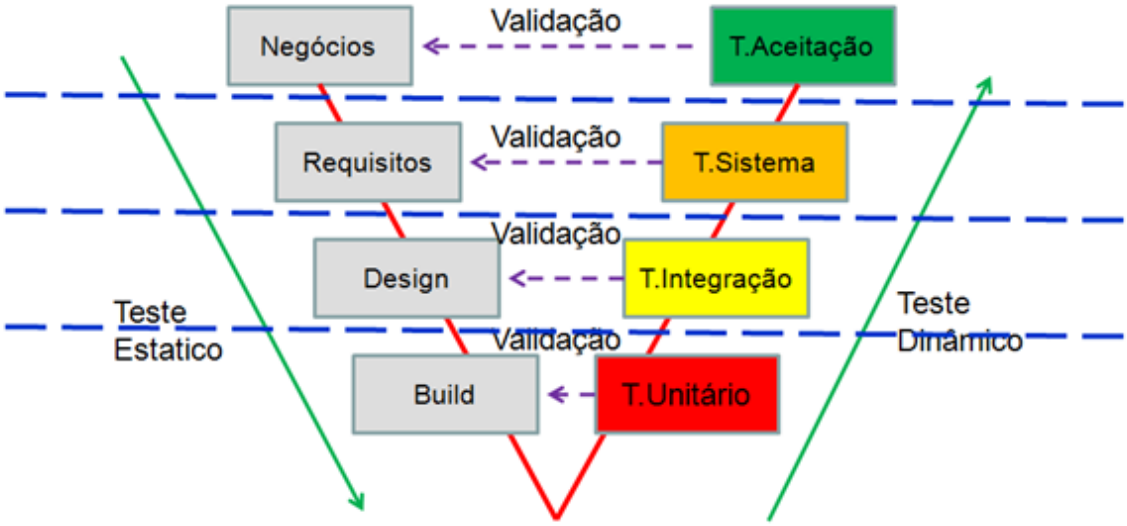


Fig. 13. Modelo V. (http://www.devmedia.com.br/testes-de-software-niveis-de-testes/22282).

3.5 - Teste de Desempenho

O teste de desempenho é um tipo de teste que se encontra no nível de teste de sistema. Com o crescimento do setor tecnológico e com o aumento de pessoas conectadas a preocupação com o desempenho tem crescido no mesmo ritmo, cada fez mais clientes estão preocupados com o desempenho de suas aplicações.

Segundo Survey (2006) apud Costa (2012) foi constatado que metade das empresas de softwares encontraram problemas de desempenho em 20% de suas aplicações. Problemas esses que em sua maioria foram evidenciados em aplicações web, as quais são responsáveis por grandes números de requisições dos usuários.

O teste de desempenho consiste em testar um sistema visando os requisitos de desempenho. Pode-se citar alguns desses requisitos (DENARO; EMMERICH, 2004) apud (NETO; SANTOS, 2012):

- Latência, que é o tempo entre uma requisição e a resposta da operação requisitada;
- Vazão, o número de operações que a aplicação completa em um dado período;
- Escalabilidade, a quantidade de usuários simultâneos que a aplicação pode gerenciar;
- Uso de recursos de máquina, como memória e processamento.

Existem diversos tipos de testes de desempenho, mas os mais comuns são os teste de stress e de carga. Costa (2012) os define como:

- Teste de stress: tem como objetivo determinar o comportamento de uma aplicação quando é submetido além de seus limites normais de carga. Busca determinar os pontos de defeito do sistema;
- Teste de carga: tem como objetivo validar o comportamento de uma aplicação sob condições normais de carga. Verifica se a aplicação cumpre os requisitos de desempenho.

3.5.1 – Ferramentas para teste de desempenho

O teste de desempenho se mostra importante, por isso existem diversas ferramentas para auxiliar na automação destes testes. Nesta seção serão apresentadas algumas ferramentas para teste de desempenho, suas características e funcionalidades. As ferramentas apresentadas serão: JMeter, LoadRunner, Rational Performance Tester, SilkPerformer e Visual Studio.

Todas as ferramentas utilizam a técnica Record&Playback, que consiste na gravação de todas as interações realizadas pelo usuário. Porém antes de utilizar essa funcionalidade é necessário configurar o parâmetro referente ao endereço IP da aplicação (COSTA, 2012).

Cada ferramenta utiliza uma ou mais linguagens de programação para a criação dos scripts de teste, como pode ser visto a seguir (COSTA, 2012):

- JMeter: grava as informações relativas às interações em um arquivo no formato XML Metadata Interchange (XMI);
- LoadRunner: os scripts podem ser nas linguagens C, Visual Basic e Java;
- Rational Performance Tester: utiliza a linguagem Java para a criação dos scripts;
- SilkPerformer: utiliza uma linguagem específica para a criação de seus scripts, linguagem denominada Benchmark Description Language (BDL);
- Visual Studio: assim como o LoadRunner utiliza mais de uma linguagem de programação para geração de scripts de testes, suporta a linguagem Visual Basic, C# e C++.

Cada ferramenta também possui suas características na configuração do cenário de teste. Nesta etapa os testes são gerados e executados de forma automática. Nesta fase as configurações são basicamente feitas por quatro parâmetros, quantidade de usuários, duração do teste, perfil da carga de trabalho e contadores que avaliam o desempenho do ambiente (COSTA, 2012):

- JMeter: a configuração de quantidade de usuários é feita pelo parâmetro Number of Threads. A configuração da duração do teste é feita pelo parâmetro Duration, que é preenchido por segundos. Não define a carga de trabalho, mas possibilita, pelo parâmetro Ramp-Up determinar a quantidade de usuários virtuais que inicarão o teste em um intervalo de tempo.
- LoadRunner: a configuração de quantidade de usuários é feita pelo parâmetro VUsers. A configuração da duração do teste é feita pelo parâmetro Duration, que é preenchido por loops. A configuração do perfil de carga de trabalho é feita pelo

parâmetro Start VUsers, pode-se definir se todos os VUsers vão iniciar o teste ao mesmo tempo ou de forma gradativa.

- Rational Performance Tester: a configuração de quantidade de usuários é feita pelo parâmetro Quantidade de Usuários. A configuração da duração do teste é feita pelo parâmetro Stop running the schedule after na elapsed time. Trabalha como o JMeter para carga de trabalho, possui o parâmetro Delay between starting each user.
- SilkPerformer: a configuração de quantidade de usuários é feita pelo parâmetro Vusers. A configuração da duração do teste é feita pelo parâmetro Simulation time. A configuração do perfil de carga de trabalho é feita pelo parâmetro Type of workload to run, assim como no LoadRunner é possível definir se os testes irão iniciar ao mesmo tempo ou de forma gradativa.
- Visual Studio: a configuração de quantidade de usuários é feita pelo parâmetro User Count. A configuração da duração do teste é feita pelo parâmetro Run Duration. A configuração do perfil de carga de trabalho é feita pelo parâmetro Load Pattern, também é possível definir se os testes irão executar ao mesmo tempo ou de forma gradativa.

As ferramentas são bem parecidas para a configuração, mudando apenas os nomes de alguns parâmetros. É importante que a escolha de uma ferramenta para o teste de desempenho leve em consideração a experiência da equipe com as linguagens de programação utilizadas por cada ferramenta. Na Figura 14 são representadas as características e funcionalidades das principais ferramentas de teste de desempenho.

Características	Ferramentas				
	JMeter	LoadRunner	RPT	SilkPerformer	Visual Studio
Open Source	X				
Linguagem de *Script*	*XMI*	C Java Visual Basic	Java	BDL	C C++ Visual Basic
Plataforma Windows	X	X	X	X	X
Plataforma Linux	X	X			
Record & Playback	X	X	X	X	X
Contadores	X	X	X	X	X

Fig. 14. Características e funcionalidade das ferramentas de teste de desempenho. (COSTA, 2012).

CAPÍTULO 4 - METODOLOGIA

Este capítulo apresenta as metodologias utilizadas para a realização dos testes e obtenção de maior confiabilidade.

4.1 - Trabalhos Correlatos

Com o surgimento e o crescimento dos bancos de dados NoSQL o interesse em comparar o desempenho desse tipo de SGBD com o modelo relacional aumentou. Com isso as pesquisas de desempenho estão sendo cada vez mais abordadas.

Uma pesquisa de comparação de desempenho de bancos de dados SQL e NoSQL foi realizada por Yashan e Manoharan. Nesta pesquisa eles concluem que nem sempre um banco de dados NoSQL tem um maior desempenho se comparado a um banco de dados SQL. Isso na verdade depende muito do tipo de operação que é usada (YISHAN; MANOHARAN, 2013). Nesta pesquisa eles utilizaram somente um SGBD relacional, sendo ele o Microsoft SQL SERVER Express, no modelo NoSQL foram utilizados o MongoDB, Cassandra, RavenDB, CouchDB e Hypertable.

O fator tamanho é fundamental para a migração de um SGBD SQL para o NoSQL, Brito realizou uma pesquisa em que mostra que a migração de um modelo relacional para o NoSQL só é vantajosa quando o banco de dados é de grande porte, já que somente neste caso é necessária a escalabilidade em alto grau, na qual a disponibilidade é imprescindível. Além disso, não existe, em qualquer abordagem NoSQL, nada que se aproxime da simplicidade e expressividade do SQL, assim como a perda de algumas funcionalidades, como funções e rotinas (BRITO , 2010).

Em sua pesquisa Pichiliani mostra que o MongoDB obteve resultados abaixo do MySQL e PostgreSQL, além disso alerta que os resultados podem depender muito do ambiente em que os testes são realizados (PICHILIANI, 2013). Isso reforça a ideia de que o uso de SGBD’s NoSQL dependem do porte do banco de dados e em qual ambiente em que será utilizado.

4.2 - Ambiente de Testes

Com o intuito de manter os resultados dos testes o mais preciso possível foi tomada a decisão de virtualizar quatro ambientes, isso devido à necessidade de deixar o ambiente de cada SGBD o mais homogêneo possível, cada ambiente terá o mesmo sistema operacional assim como suas configurações.

O objetivo deste trabalho é a análise de desempenho de dois dos bancos de dados mais populares atualmente no SQL e o mais conhecido banco de dados NoSQL, utilizando a ferramenta de virtualização VMWare.

O trabalho irá analisar o desempenho dos bancos de dados de acordo com o tempo de execução das operações transacionais e de recuperação da informação.

4.2.1 - Arquitetura do Ambiente de Testes

A máquina utilizada para os teste foi um notebook DELL Inspiron 15R Special Edition, com processador de 3ª geração Intel(R) Core(TM) i7-3612QM de quatro núcleos e oito threads e clock de 2.10GHz, 8 GB de memória RAM DDR3 1600 MHz, placa de vídeo AMD Radeon HD 7730M com 2 GB de memória RAM dedicada, 1 TB de armazenamento de disco rígido mais 32 GB de SSD. A Figura 15 apresenta a arquitetura do ambiente de testes.

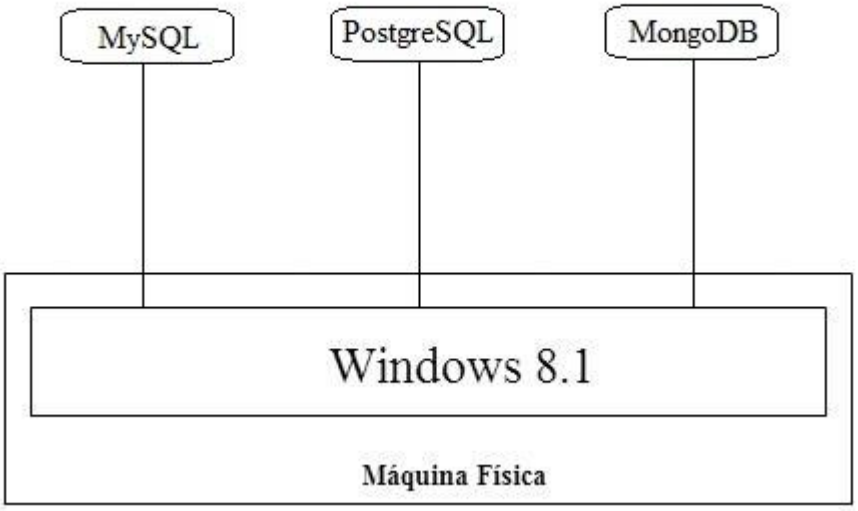


Fig. 15. Arquitetura do ambiente de testes. (Próprio autor).

4.2.2 - Sistemas operacionais utilizados

O sistema operacional nativo do notebook é o Windows 8.1 64 bits, o sistema operacional utilizado no ambiente virtualizado é o CentOS 6.5. A escolha do CentOS não teve um motivo específico a não ser pela sua distribuição gratuita.

4.3 - Máquina Virtual

O software de virtualização utilizado foi o VMWare Player 5.0.4. Em todos os ambientes o sistema operacional assim como as configurações utilizadas foram as mesmas.

4.3.1 - Configurações

Foi disponibilizado até 2GB de memória RAM para os ambientes. Na Figura 16 é representada a configuração de memória RAM da máquina virtual.

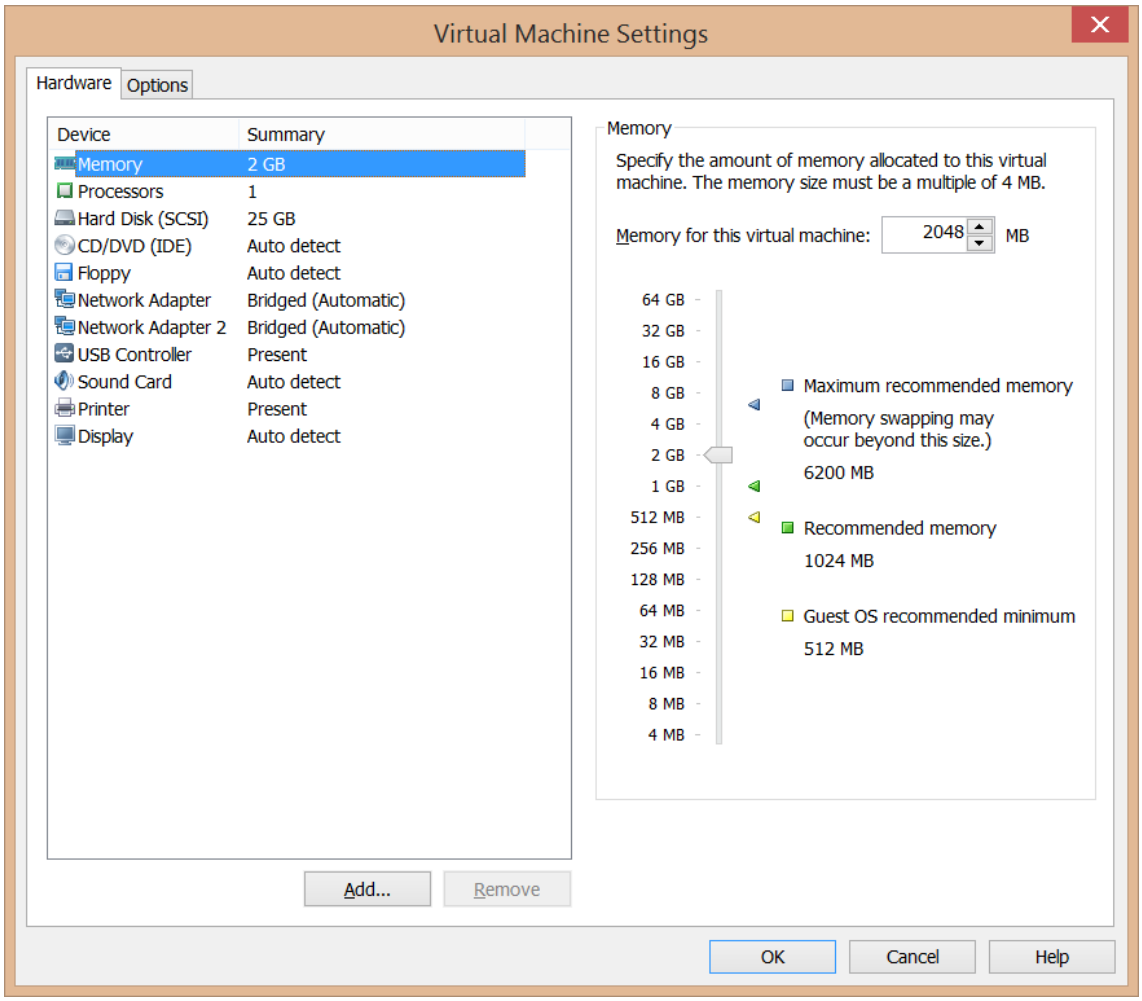


Fig. 16. Configuração de memória RAM. (Próprio autor).

Por padrão a configuração do processador é de 1 core, essa opção foi aumentada para 2 cores, as demais opções ficaram como padrão de instalação. A configuração do processador é representada na Figura 17.

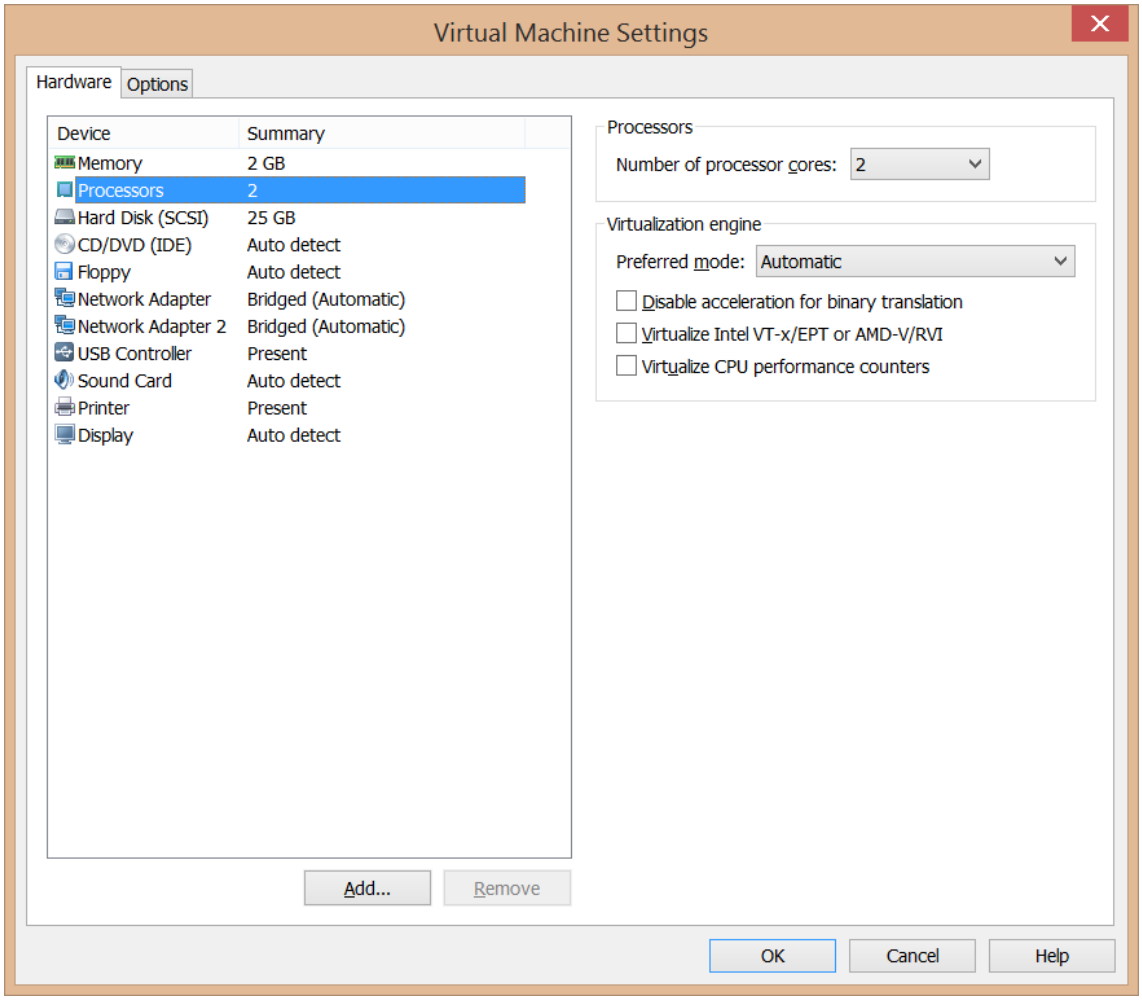


Fig. 17. Configuração do processador. (Próprio autor)

O HD é dinamicamente alocado, isso significa que ele expande conforme a necessidade, neste caso ele se expandiria até 25 GB, que foi o tamanho máximo definido para o HD. A Figura 18 representa a configuração de HD.

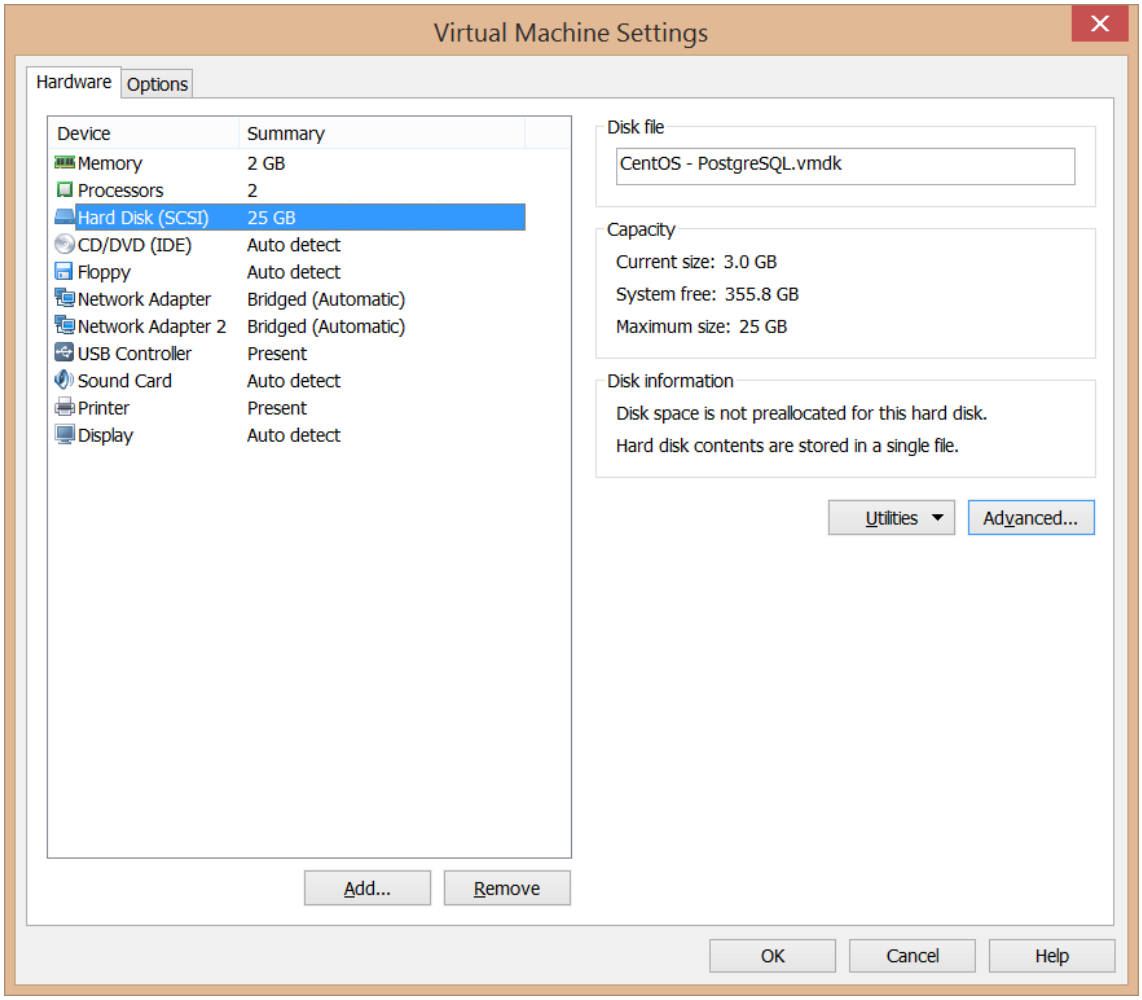


Fig. 18. Configuração da capacidade de disco. (Próprio autor).

4.4 - Bancos de Dados Escolhidos

Os bancos de dados foram escolhidos de acordo com o ranking do site http://db-engines.com/, representado na Figura 19.

Rank	Last Month	DBMS	Database Model	Score	Changes
1.	1.	**Oracle**	**Relational DBMS**	**1502.74**	**-11.34**
2.	2.	**MySQL**	**Relational DBMS**	**1309.10**	**+16.43**
3.	3.	**Microsoft SQL Server**	**Relational DBMS**	**1207.80**	**-2.63**
4.	4.	**PostgreSQL**	**Relational DBMS**	**240.64**	**+10.41**
5.	5.	**MongoDB**	**Document store**	**224.62**	**+10.28**
6.	6.	**DB2**	**Relational DBMS**	**186.47**	**+1.89**
7.	7.	**Microsoft Access**	**Relational DBMS**	**145.36**	**+2.60**
8.	8.	**SQLite**	**Relational DBMS**	**89.29**	**-0.88**
9.	9.	**Cassandra**	**Wide column store**	**81.73**	**+3.01**
10.	10.	**Sybase ASE**	**Relational DBMS**	**80.00**	**+1.87**
11.	11.	Solr	Search engine	67.16	+4.28
12.	12.	Teradata	Relational DBMS	65.53	+3.80
13.	13.	Redis	Key-value store	62.04	+3.58

Fig. 19. Ranking dos bancos de dados mais populares atualmente. (http://db-engines.com/en/ranking).

Para os bancos de dados relacionais foram escolhidos o MySQL, segundo colocado e o PosgreSQL, que aparece em quarto lugar. Para bancos de dados NoSQL foram escolhidos o MongoDB que se encontra em quinto lugar, e o Cassandra em nono lugar.

O método para classificar os bancos de dados é dado pelo número de vezes que o banco de dados é citado via web. Uma pesquisa é feita nos principais buscadores da internet, onde é verificado qual o número de vezes que cada banco de dados aparece, também é verificado quantas vezes cada SGBD aparece nos principais fóruns da área, além de oportunidades de emprego oferecidas (http://db-engines.com).

4.4.1 - Mysql

Foi utilizada a versão 5.1.73 do MySQL, com suas configurações padrões.

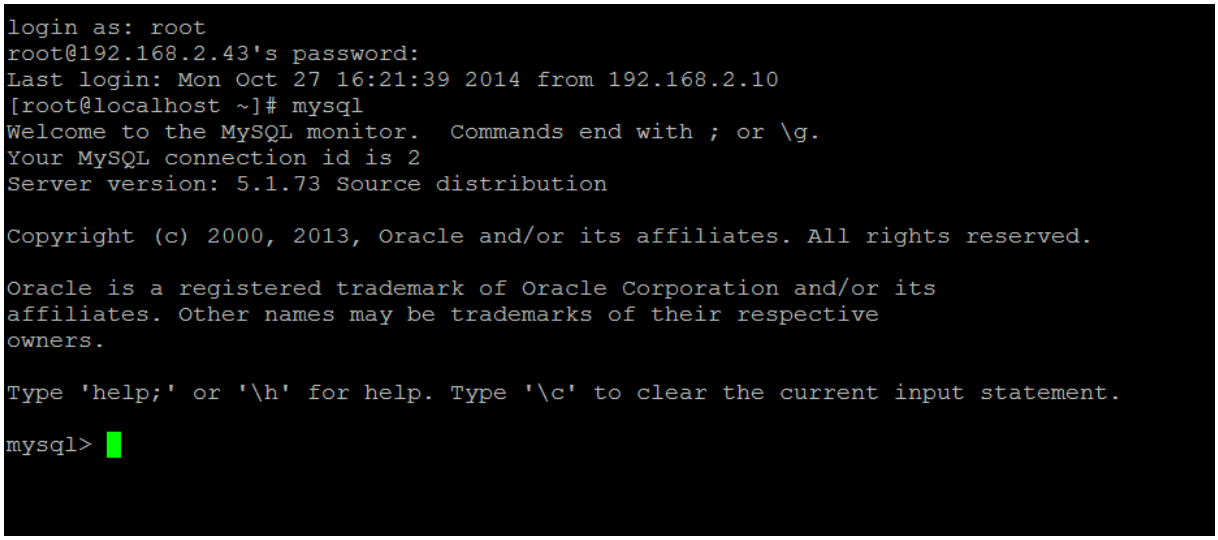


Fig. 20. MySQL versão 5.1.73 instalado no CentOS 6.5. (Próprio autor)

Na Figura 20 é apresentado o MySQL instalado no CentOS 6.5, no ambiente de testes.

O MySQL é um banco de dados relacional que surgiu da necessidade de melhorar o até então utilizado mSQL, em testes realizados pela equipe de desenvolvedores a conclusão foi que o mSQL não era rápido nem flexível o bastante para conectar tabelas com rápidas rotinas de baixo nível. Isso resultou em uma nova interface SQL, mas com praticamente a mesma interface API do mSQL, isso para que a portabilidade de códigos de terceiros escrito para mSQL fosse facilmente adaptado ao MySQL (MYSQL AB, 2003).

O MySQL foi criado por David Axmark, Allan Larsson e Michael Widenius na Suécia. O projeto do MySQL começou a ser desenvolvido em 1994, mas foi em 1995 que saiu oficialmente a primeira versão utilizada internamente pela equipe de desenvolvedores. A empresa desenvolvedora do MySQL, a MySQL AB foi vendida a Sun Microsystems em 2008, e essa em seguida foi vendida para a Oracle.

4.4.1.1 - Características do MySQL

O MySQL é um banco de dados de código aberto (Open Source) liberado sob a licença GNU General Public License (GPL), sendo assim o MySQL pode ser usado sem custos sob a GPL, em outros termos deve-se obter a licença comercial deste SGBD (MYSQL AB, 2003).

A lista a seguir descreve algumas das funções mais importantes do MySQL:

> Multiplataforma, suporta diversos sistemas operacionais como Windows, Linux, FreeBSD por exemplo.
> Escrito em C e C++.

Suporte total a multi-threads, pode facilmente utilizar múltiplas CPUs, se disponível.
Disponíveis APIs para C, C++, Java, Perl, PHP, Python, Ruby.
Tem estrutura para armazenamento transacional e não transacional.
Tabelas em disco baseadas em árvores-B extremamente rápidas com compressão de índices.
Fácil adicionar uma interface SQL a um banco de dados caseiro.
Sistema de alocação de memória rápido e baseado em processo.
Tabelas hash em memória que são usadas como tabelas temporárias.
Aceita diversos tipos de campos.
Suporte a operadores e funções nas partes SELECT e WHERE.
Sistema de privilégios e senhas muito flexível.
Trabalha com banco de dados enormes, existem casos de 60.000 tabelas e aproximadamente 5.000.000.000 linhas.
São permitidos até 32 índices por tabela.
É possível se conectar a servidores MySQL usando sockets TCP/IP.
Atualmente o MySQL não tem restrições quanto ao tamanho das tabelas, essa restrição está diretamente ligada ao sistema operacional. O tamanho máximo para o banco de dados MySQL normalmente é limitado pelas restrições do sistema operacional quanto ao tamanho dos arquivos (MYSQL AB, 2003).

4.4.2 - PostgreSQL

Foi utilizada a versão 9.3.5 do PostgreSQL com suas configurações padrões. Na Figura 21 é apresentado o PostgreSQL instalado no CentOS 6.5.

Fig. 21. PostgreSQL versão 9.3.5 instalado no CentOS 6.5. (Próprio autor)

O PostgreSQL é derivado do pacote POSTGRES que foi desenvolvido na Universidade da Califórnia e liderado pelo professor Michael Stonebraker. O projeto POSTGRES teve sua implementação iniciada em 1986, sendo sua primeira versão de demonstração operacional em 1987, e exibida em 1988 na conferencia AMC-SIGMOD (Documentação PostgreSQL).

Em 1994 surgiu o Postgres95, isso ocorreu logo após Andrew Yu e Jolly Chen adicionarem um interpretador SQL ao POSTGRES, em seguida o Postgres95 foi liberado na

web. Era totalmente escrito em ANSI C, tinha um tamanho reduzido, tinha melhor desempenho, facilidade na manutenção e mais rápido em relação ao POSTGRES (Documentação PostgreSQL).

Ficou evidente que o nome Postgres95 não resistiria ao tempo, então em 1996 nascia o PostgreSQL, este nome foi escolhido para refletir o relacionamento entre o POSTGRES e as versões mais recentes com a capacidade SQL. A partir disso a ênfase no desenvolvimento do PostgreSQL era para o aumento de funcionalidades e recursos (Documentação PostgreSQL).

4.4.2.1 - Características do PostgreSQL

O PostgreSQL é um gerenciador de banco de dados objeto-relacional de código aberto. Além de ser um banco de dados objeto-relacional, Worskey et al (2003) apud Odwazny (2003) cita algumas das características principais do PostgreSQL:

> Altamente extensível - PostgreSQL suporta operadores, funções, métodos de acesso e tipos de dados;
> Suporte a SQL compreensível – PostgreSQL suporta a especificação SQL99 e inclui características avançadas desde SQL92;
> Integridade referencial - é usado para validar os dados de uma base de dados;
> API flexível - a flexibilidade da API do PostgreSQL determina fácil suporte ao desenvolvimento do PostgreSQL ORDBMS. Tais interfaces incluem: Object Pascal (Delphi), Python, Pearl, PHP, ODBC, Java/JDBC, Ruby, TCL, C/C++ e Pike;
> Procedure - suporta linguagem procedural interna, incluindo uma linguagem nativa chamada PL/pgSQL. Outra vantagem do PostgreSQL é a capacidade de usar Pearl, Python ou TCL como linguagem de procedimentos;
> MVCC - Multi-Version Concurrency Control, é a tecnologia que o PostgreSQL usa para não precisar de trava (locking). Outros SGBDs como MySQL, as vezes bloqueiam os dados fazendo o usuário esperar uma resposta;
> Cliente/Servidor - PostgreSQL utiliza um processo por usuário, a arquitetura cliente/servidor tem um processo principal que dispara outros processos adicionais para cada conexão;
> Write ahead log (WAL) - se houver um erro na base de dados, este será um registro de uma transação que será restaurada, isto traz bons benefícios para o evento do erro. Se qualquer mudança for feita na base de dados, pode ser recuperado usando os arquivos de log. Uma vez restaurado, o usuário continua trabalhando do ponto antes do erro ter ocorrido.

4.4.3 - MongoDB

Foi utilizada a versão 2.6.4 do MongoDB com suas configurações padrões. Na Figura 22 é apresentado o MongoDB instalado no CentOS 6.5.

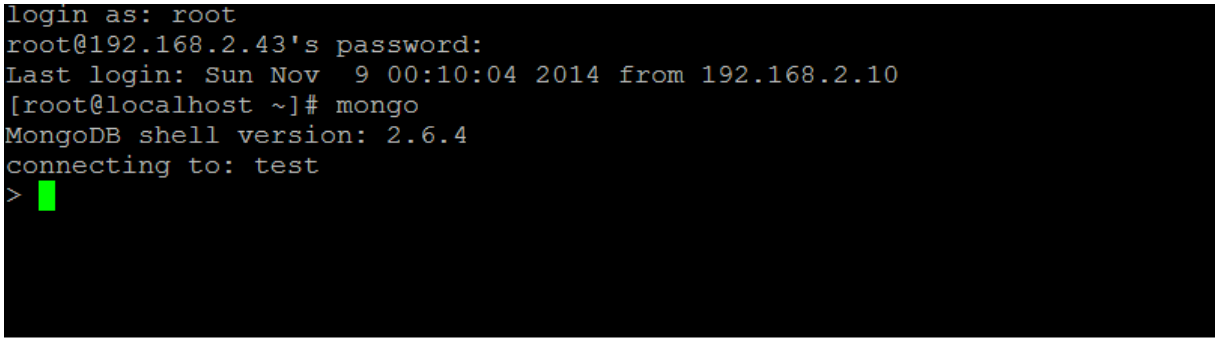


Fig. 22. MongoDB versão 2.6.4 instalado no CentOS. (Próprio autor).

O MongoDB é um banco de dados orientado a documentos, escrito em C++. É um projeto de código aberto e impulsionado pela empresa 10gen Inc, que também oferece serviços profissionais ao MongoDB. De acordo com seus desenvolvedores o principal objetivo do MongoDB é fechar a lacuna entre o rápido e o altamente escalável (STRAUCH, 2010, p. 76).

O projeto do MongoDB teve inicio em meados de 2007 com a startup 10gen, no inicio eles trabalhavam com uma plataforma composta por um servidor de aplicativos e um banco de dados que seria escalado quando necessário. A plataforma da 10gen foi projetada para lidar com dimensionamento e gestão de infraestrutura de hardware e software automaticamente, isso fez com que os desenvolvedores focassem somente no código do aplicativo. Em uma analise a 10gen descobriu que os usuários estavam interessados no banco de dados, isso fez com que os esforços fossem exclusivamente para o banco de dados, que se tornou o MongoDB (BANKER, 1012, p. 5).

4.4.3.1 - Características do MongoDB

O MongoDB têm algumas características importantes que devem ser citadas, Lennon (2011) cita algumas delas:

- Binários oficiais disponíveis para Windows, Mac OS X, Linux e Solaris, distribuição de código de origem disponível para autoconstrução;
- Drivers oficiais disponíveis para C, C#, C++, Haskell, Java, JavaScript, Perl, PHP, Python, Ruby e Scala, com uma ampla variedade de drivers suportados pela comunidade disponíveis para outras linguagens;
- Consultas JavaScript ad-hoc que permitem localizar dados usando qualquer critério de qualquer atributo de documento. Essas consultas são um espelho da funcionalidade de consultas SQL, o que torna muito fácil para desenvolvedores SQL escreverem consultas em MongoDB;
- Suporte a expressões regulares em consultas;
- Os resultados da consulta em MongoDB são armazenados em cursores que fornecem muitas funções para filtragem, agregação e classificação, incluindo limit(), skip(), sort(), count(), distinct() e group();

- Map/reduce – implementação para agregação avançada;
- Armazenamento de arquivos grandes usando GridFS;
- Suporte a indexação de atributo semelhante a RDBMS, onde é possível criar índices diretamente em atributos selecionados de um documento;
- Recursos de otimização de consulta usando dicas, planos de explicação e perfis;
- Replicação mestre/escravo similar ao MySQL;
- Escalada horizontal com auto-sharding
- Atualizações no local para simultaneidade de alto desempenho e livre de disputas;
- Shell on-line que permite testar o MongoDB sem instalá-lo.

4.5 - Metodologia dos testes

Para que os resultados dos testes referentes ao desempenho dos bancos de dados tivessem uma maior confiabilidade e igualdade, além do ambiente homogêneo para cada um deles foram aplicadas algumas metodologias.

4.5.1 - Número de repetições dos testes

Para evitar grandes variações nos resultados recorrentes de falhas, erros e outras interferências tanto de software como de hardware, os testes foram executados dez vezes. O resultado final é a média aritmética dos resultados obtidos.

4.5.2 - Software utilizado para realização dos testes

Para a realização das operações de inserção, alteração, busca e exclusão de cada modelo de banco de dados não foi utilizada nenhuma ferramenta de modelagem existente, já que elas poderiam afetar diretamente os resultados dos testes; mediante a isso, todas as instruções foram executadas diretamente pelo prompt de comando do CentOS com a ajuda do software de emulação de terminal PuTTY.

4.5.3 - Realização dos testes

Neste trabalho foram utilizadas quatro tabelas, uma de um milhão de registros, uma de dez milhões de registros, uma de cinquenta milhões de registros e uma de cem milhões de registros. As instruções realizadas no modelo SQL foram as mesmas para o MySQL e para o PostgreSQL, respeitando as particularidades de cada banco de dados. No modelo NoSQL foram encontradas maneiras de simular da melhor maneira possível instruções que o modelo não possui, como por exemplo instruções de relação entre tabelas.

4.5.4 - Comportamento do sistema após os testes

Como foram feitas dez repetições em cada instrução, executar um teste logo após o outro poderia afetar os resultados de maneira significativa, isso porque as instruções poderiam ficar armazenadas no buffer de memória, e com isso o tempo de execução poderia cair drasticamente em relação aos anteriores. Para que o tempo de execução das instruções não sofressem grandes mudanças, a cada teste executado o sistema era reinicializado.

4.5.5 - Recursos dedicados aos testes

No ambiente virtual não foi instalado nenhum programa a não ser o próprio banco de dados a ser executado, evitando assim excesso de processos consumindo memória e recursos que poderiam interferir nos resultados dos testes. No ambiente físico nenhum programa estava em execução, a não ser a própria máquina virtual, até mesmo o antivírus estava desabilitado. Somente os processos nativos estavam em execução em ambos os ambientes.

4.5.6 - Extração do tempo das instruções

Para cada banco de dados foi utilizada uma maneira diferente de resgatar o tempo de execução de cada teste realizado.

Para o MySQL foi utilizado o próprio prompt de comando, que retorna o tempo em segundos de cada operação realizada.

No PostgreSQL foi utilizada a função timing, que precisa ser ativada sempre que o banco é inicializado.

Para o MongoDB foram utilizadas mais de uma maneira de verificar o tempo das operações. Em operações de busca foi utilizada a função explain(), este método é muito utilizado para otimização de consultas. O explain() executa a busca e mostra o tempo que foi levado para executa-la. Para as demais operações foi usada uma função simples em javascript que pega o tempo inicial e final da execução e mostra o tempo, em milissegundos, que a operação foi executada. Além desses métodos ainda foi utilizado os logs que o MongoDB gera a cada operação realizada.

4.5.7 - Estrutura das tabelas

Foram utilizadas duas tabelas relacionadas para que fosse possível realizar testes de desempenho em buscas complexas. A estrutura das tabelas é apresentada na Figura 23.

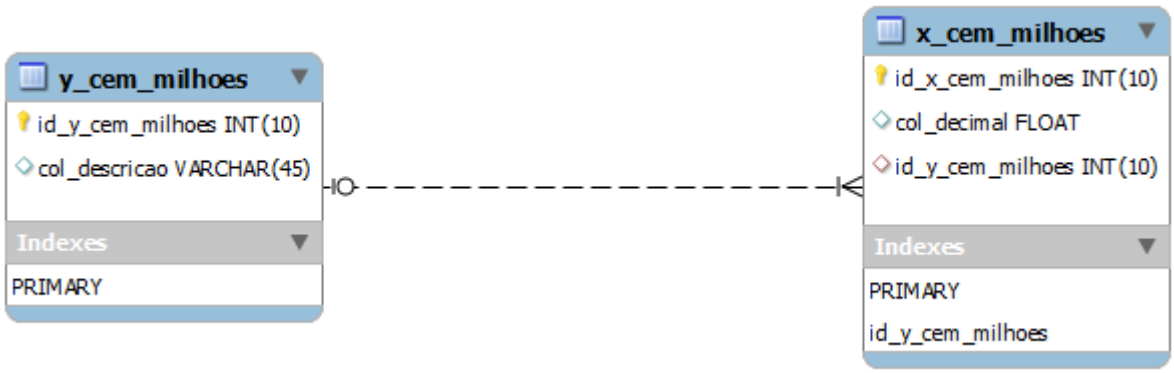


Fig. 23. Modelagem da estrutura das tabelas. (Próprio autor).

As tabelas foram definidas como Y e X. As tabelas Y possuem dois campos, um campo de tipo INT, tamanho 10 como chave primária, e um campo VARCHAR, tamanho 45 contendo a descrição. As tabelas X são as que possuem chave estrangeira das tabelas Y, possuem um campo INT, tamanho 10 como chave primária, um campo FLOAT definido com o valor fixo 13,38 e o campo de chave estrangeira das tabelas Y. As chaves primárias são incrementadas a cada inserção de um registro. A Figura 24 apresenta a estrutura dos registros no MySQL.

```
Database changed
mysql> select * from y_um_milhao limit 3;
+----------------+---------------+
| id_y_um_milhao | col_descricao |
+----------------+---------------+
|              1 | Teste TCC1    |
|              2 | Teste TCC1    |
|              3 | Teste TCC1    |
+----------------+---------------+
3 rows in set (0.00 sec)

mysql> select * from x_um_milhao limit 3;
+----------------+-------------+----------------+
| id_x_um_milhao | col_decimal | id_y_um_milhao |
+----------------+-------------+----------------+
|              1 |       13.38 |              1 |
|              2 |       13.38 |              2 |
|              3 |       13.38 |              3 |
+----------------+-------------+----------------+
3 rows in set (0.02 sec)
```

Fig. 24. Estrutura dos registros das tabelas X e Y. (Próprio autor).

CAPÍTULO 5 - TESTES E RESULTADOS

Neste capítulo serão apresentados os testes que foram realizados e os seus resultados obtidos.

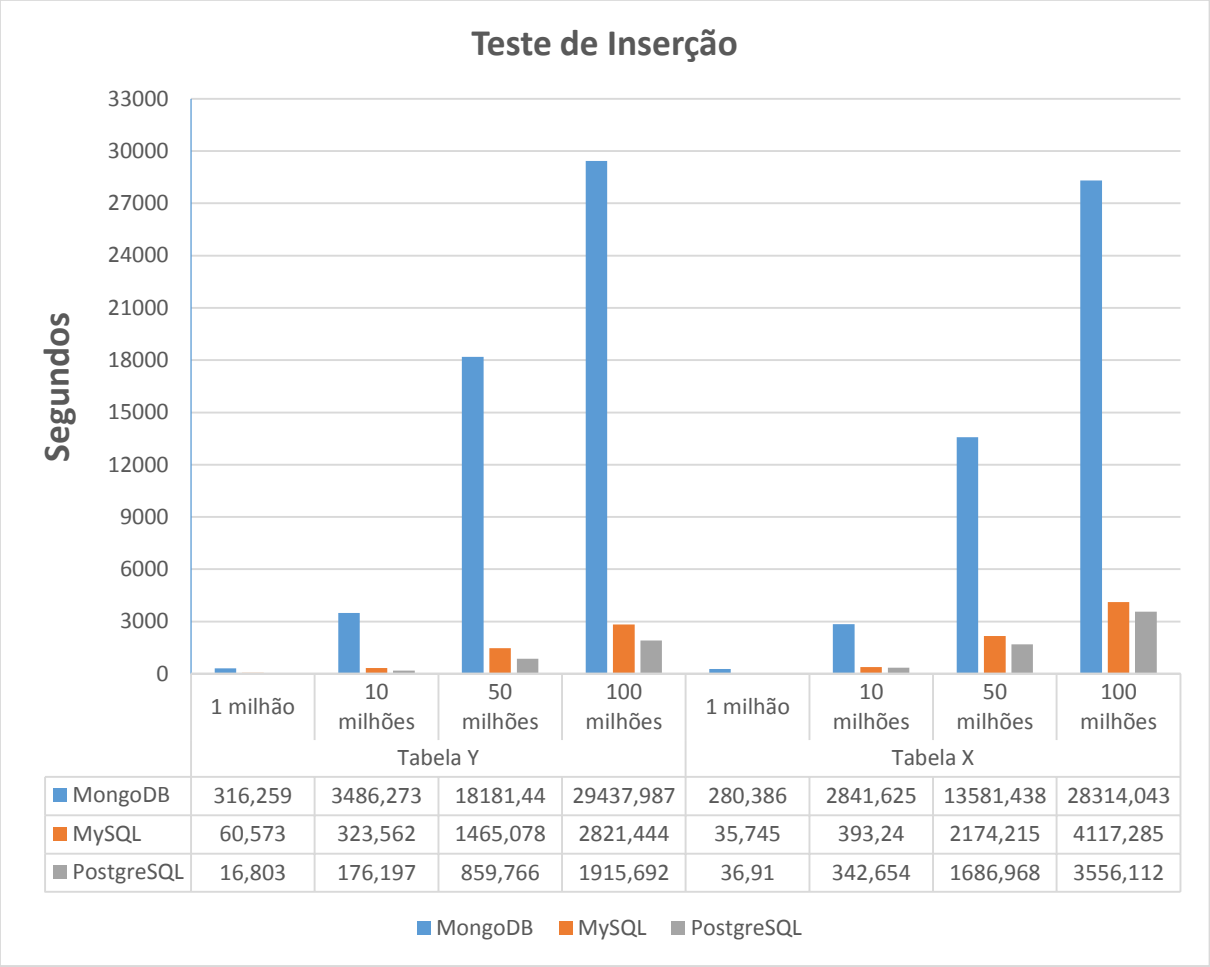
5.1 - Teste de inserção

Alguns termos podem mudar do modelo relacional para o NoSQL, como por exemplo as tabelas, que no MongoDB são chamadas de collections.

Os primeiros testes realizados foram o de inserção, as inserções se deram por meio de procedures no modelo relacional, e para o MongoDB foram utilizadas funções em javascript para resgatar o tempo de execução. Logo após cada execução de uma procedure de inserção era utilizado o comando TRUNCATE no modelo relacional para que nenhum registro ainda se mantivesse na tabela, no MongoDB a solução encontrada foi a de deletar a collection com o comando drop().

O resultado do teste é apresentado em segundos e está representado no Gráfico 1. Quanto menor o tempo melhor o desempenho do banco de dados.

Gráfico 1. Resultados dos testes de inserção das tabelas Y e X.



	1 milhão	10 milhões	50 milhões	100 milhões	1 milhão	10 milhões	50 milhões	100 milhões
	Tabela Y				Tabela X			
MongoDB	316,259	3486,273	18181,44	29437,987	280,386	2841,625	13581,438	28314,043
MySQL	60,573	323,562	1465,078	2821,444	35,745	393,24	2174,215	4117,285
PostgreSQL	16,803	176,197	859,766	1915,692	36,91	342,654	1686,968	3556,112

Ao se analisar o gráfico, pode-se notar que o PostgreSQL é mais rápido na escrita do que os demais bancos, o MongoDB foi o banco de dados que mais demorou para inserir os

dados, a partir da tabela dos dez milhões de registros o seu desempenho teve um aumento drástico em relação ao demais.

Os resultados de todas as repetições dos testes se encontram nas tabelas de 1 a 6.

Tabela 1. Resultados obtidos da inserção de registros da collection Y no MongoDB.

MongoDB				
Teste de Inserção na tabela Y				
	1 milhão	10 milhões	50 milhões	100 milhões
1	316,78	3463,15	18174,62	29841,93
2	317,94	3529,35	18233,31	29145,3
3	315,88	3447,32	17983,8	29285,53
4	314,89	3543,92	18101,35	28982,27
5	315,72	3535,94	18221,95	29590,15
6	316,55	3479,38	18211,3	29976,42
7	315,44	3499,18	18052,67	29368,73
8	316,19	3521,47	18287,63	29428,6
9	316,9	3425,53	18223,43	29274,74
10	316,3	3417,49	18321,38	29486,2
	316,259	3486,273	18181,144	29437,987

Tabela 2. Resultados obtidos da inserção de registros da collection X no MongoDB.

MongoDB				
Teste de Inserção na tabela X				
	1 milhão	10 milhões	50 milhões	100 milhões
1	283,4	2861,21	13536,47	28135,59
2	281,29	2830,22	13589,37	28234,8
3	278,12	2843,83	13782,2	28321,52
4	280,48	2833,39	13562,69	28142,38
5	278,39	2843,17	13612,31	28198,48
6	282,38	2823,72	13453,61	28219,18
7	282,26	2854,26	13562,48	28621,35
8	281,91	2831,21	13585,59	28413,62
9	283,94	2840,12	13589,71	28521,31
10	271,69	2855,12	13539,95	28332,2
	280,386	2841,625	13581,438	28314,043

Tabela 3. Resultados obtidos da inserção de registros da tabela Y no MySQL.

MySQL				
Teste de Inserção na tabela Y				
	1 milhão	10 milhões	50 milhões	100 milhões
1	61,63	458,17	1671,67	2882,59

2	52,29	298,09	1404,8	2868,94
3	62,25	301,99	1423,12	2790,17
4	58,28	413,76	1402,36	2836,19
5	64,38	358,01	1466,98	2736,33
6	55,98	275,47	1380,68	2755,43
7	65,36	284,09	1450,44	2849,32
8	64,58	282,14	1408,59	2790,15
9	60,47	278,21	1404,19	2798,89
10	60,51	285,69	1637,95	2906,43
	60,573	323,562	1465,078	2821,444

Tabela 4. Resultados obtidos da inserção de registros da tabela X no MySQL.

MySQL				
Teste de Inserção na tabela X				
	1 milhão	10 milhões	50 milhões	100 milhões
1	37,79	396,54	2073,28	4285,04
2	35,69	393,62	2200,02	4015,98
3	35,48	393,24	2060,87	4018,25
4	35,38	391,74	2447,35	4050,03
5	35,27	393,45	2069,34	4201,01
6	35,27	394,01	2133,09	4278,98
7	35,54	388,34	2039,5	4163,05
8	35,27	393,2	2200,05	4025,69
9	35,73	389,77	2058,34	4088,99
10	36,03	393,24	2460,31	4045,83
	35,745	392,715	2174,215	4117,285

Tabela 5. Resultados obtidos da inserção de registros da tabela Y no PostgreSQL.

PostgreSQL				
Teste de Inserção na tabela Y				
	1 milhão	10 milhões	50 milhões	100 milhões
1	20,27	239,65	933,03	2556,84
2	16,38	174,52	821,52	1743,13
3	16,2	166,33	859,12	1760,02
4	16,53	162,11	865,1	1803,98
5	16,22	168,34	843,92	1921,42
6	16,55	166,69	842,99	1845,52
7	17,01	169,23	882,22	1743,34
8	16,33	173,98	885,21	1949,01
9	16,1	171,1	831,05	1901,23
10	16,44	170,02	833,5	1932,43
	16,803	176,197	859,766	1915,692

Tabela 6. Resultados obtidos da inserção de registros da tabela X no PostgreSQL.

PostgreSQL				
Teste de Inserção na tabela X				
	1 milhão	10 milhões	50 milhões	100 milhões
1	48,39	334,69	1780,39	3505,32
2	32,29	347,73	1673,62	3644,42
3	34,5	333,68	1660,62	3498,59
4	32,98	332,23	1745,23	3501,01
5	32,33	345,2	1688,3	3550,3
6	38,34	348,91	1690,09	3549,89
7	36,15	331,2	1691,2	3499,9
8	32,4	382,9	1630,11	3605,1
9	33,21	335,01	1632,22	3633,49
10	35,51	334,99	1677,9	3573,1
	35,61	342,654	1686,968	3556,112

As tabelas 1 e 2 apresentam os resultados da bateria de testes de inserção nas collections X e Y do MongoDB, as tabelas 3 e 4 apresentam os resultados do MySQL e as tabelas 5 e 6 apresentam os resultados do PostgreSQL. O tempo de execução estão em segundos e o último campo da tabela é a média utilizada no gráfico de resultados.

5.2 - Teste de busca

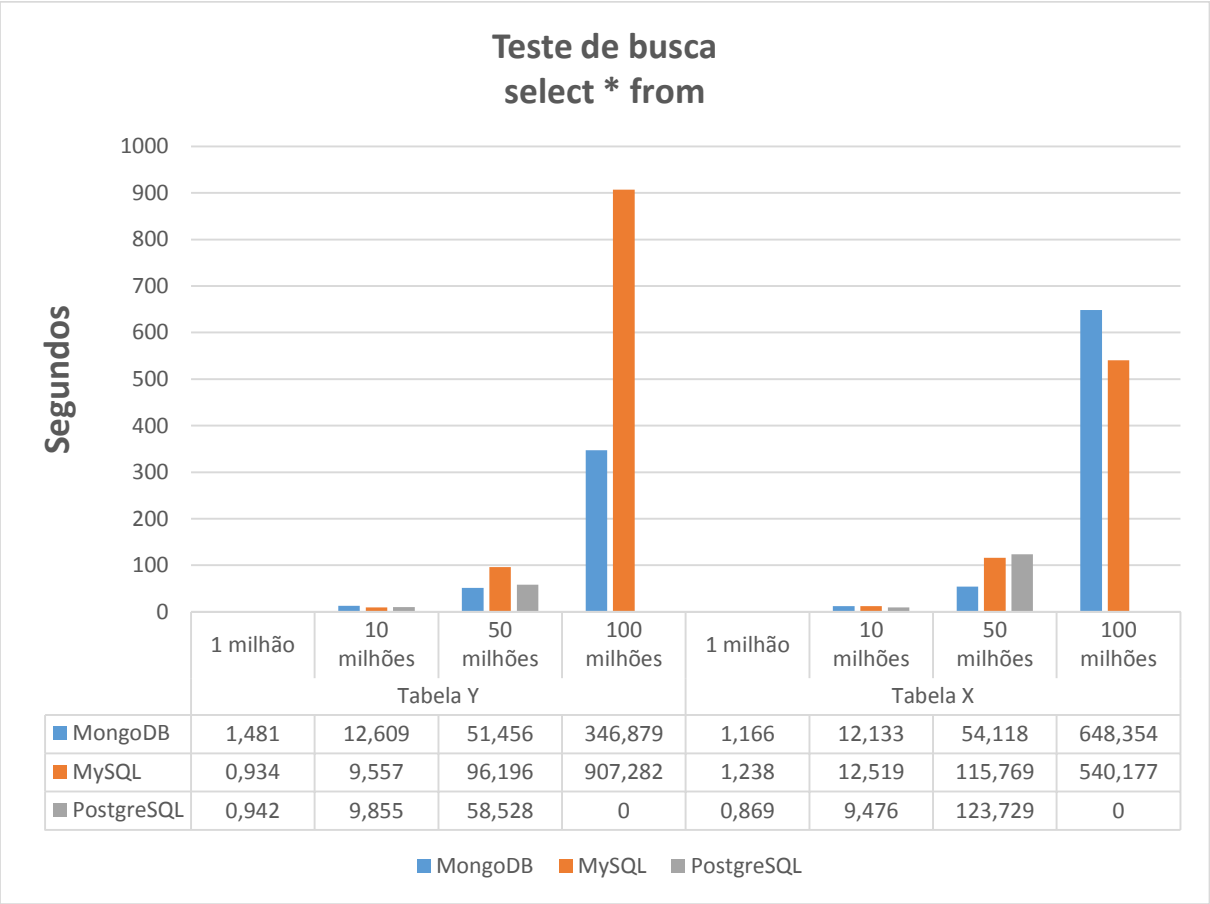
O segundo teste realizado foram os de busca. Como já foi descrito assim que um teste era executado o sistema era reiniciado para que os resultados não fossem afetados pelo buffer de memória.

Os testes de busca foram divididos em diversas partes já que pode-se utilizar diversas formas de executar uma consulta, as consultas utilizadas neste trabalho foram as consultas de todos os registros das tabelas Y e X. E com INNER JOIN foram feitas consultas de todos os registros, consulta de um registro por chave primária e consulta de um registro pela coluna "col_descricao".

5.2.1 - Busca de todos os registros

Nestes testes era realizada uma busca de todos os registros das tabelas X e Y utilizando o "select * from". No MongoDB foi utilizada a função find() e o explain(). O resultado é apresentado no Gráfico 2, quanto menor o tempo melhor o desempenho do banco de dados.

Gráfico 2. Resultados dos testes de busca de todos os registros das tabelas Y e X.



Nota-se que o MongoDB leva uma leve vantagem na leitura de dados a partir dos cinquenta milhões de registros. A partir também dos cinquenta milhões de registros o PostgreSQL precisou ter aumento de memória para que pudesse suportar a busca. O MySQL e o PostgreSQL seguem bem parecidos, é possível notar que as colunas de cem milhões de registros do PostgreSQL não possuem resultados, isso porque a memória não foi suficiente para realizar a busca, mesmo aumentando a memória da máquina virtual não foi possível realizar a operação. A Figura 25 apresenta o erro da memória no PostgreSQL.

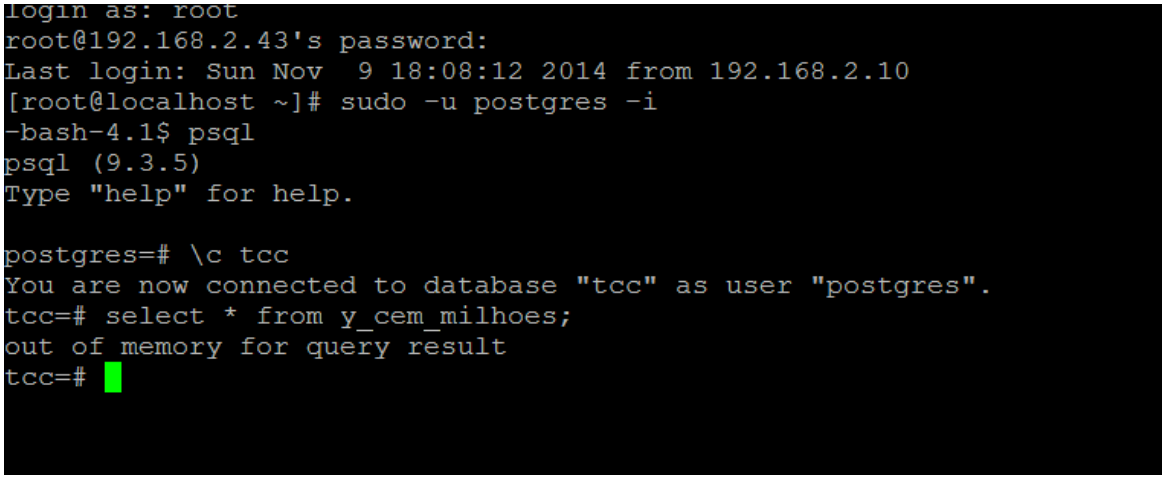


Fig. 25. Problema de memória na consulta do PostgreSQL. (Próprio autor).

Os resultados de todas as repetições dos testes utilizando o "select * from" se encontram nas tabelas de 7 a 12.

Tabela 7. Resultados obtidos da busca de todos os registros da collection Y no MongoDB.

MongoDB				
Teste de Busca na tabela Y				
	1 milhão	10 milhões	50 milhões	100 milhões
1	1,64	12,37	54,72	366,44
2	1,58	13,29	55,11	332,49
3	1,63	12,05	53,55	333,42
4	2,01	12,46	50,68	345,23
5	1,18	12,34	50,65	356,93
6	1,89	12,38	49,51	366,36
7	1,26	12,27	50,12	333,63
8	1,1	13,27	50,27	334,74
9	1,25	13,28	50,17	365,85
10	1,27	12,38	49,78	333,7
	1,481	12,609	51,456	346,879

Tabela 8. Resultados obtidos da busca de todos os registros da collection X no MongoDB.

MongoDB				
Teste de Busca na tabela X				
	1 milhão	10 milhões	50 milhões	100 milhões
1	1,05	11,82	57,12	636,48
2	1,29	12,09	56,12	636,19
3	1,04	12,34	55,28	673,71
4	1,32	11,9	57,02	638,41
5	1,21	11,98	54,1	659,94

6	1,09	12,23	53,21	649,39
7	1,1	12,32	52,19	636,3
8	1,02	12,52	52,18	659,55
9	1,31	12,33	53,84	647,27
10	1,23	11,8	50,12	646,3
	1,166	12,133	54,118	648,354

Tabela 9. Resultados obtidos da busca de todos os registros da tabela Y no MySQL.

MySQL				
Teste de Busca na tabela Y				
	1 milhão	10 milhões	50 milhões	100 milhões
1	0,95	9,61	106,31	827,4
2	0,93	9,36	93,72	991,06
3	0,93	9,82	102,32	805,96
4	0,93	9,38	99,33	990,35
5	0,94	9,53	96,5	855,09
6	0,93	9,63	88,61	945,02
7	0,93	9,32	91,35	902,48
8	0,94	9,79	92,49	889,15
9	0,93	9,77	98,32	874,2
10	0,93	9,36	93,01	992,11
	0,934	9,557	96,196	907,282

Tabela 10. Resultados obtidos da busca de todos os registros da tabela X no MySQL.

MySQL				
Teste de Busca na tabela X				
	1 milhão	10 milhões	50 milhões	100 milhões
1	1,29	12,6	86,31	471,1
2	1,23	12,5	120,32	512,92
3	1,22	12,57	115,18	565,53
4	1,27	12,4	138,61	515,54
5	1,22	12,5	123,69	601,04
6	1,22	12,61	120,12	560,99
7	1,23	12,48	99,89	496,9
8	1,25	12,5	113,9	590,23
9	1,22	12,51	119	565,32
10	1,23	12,52	120,67	522,2
	1,238	12,519	115,769	540,177

Tabela 11. Resultados obtidos da busca de todos os registros da tabela Y no PostgreSQL.

PostgreSQL				
Teste de Busca na tabela Y				
	1 milhão	10 milhões	50 milhões	100 milhões
1	0,99	10,98	61,28	
2	0,94	9,45	58,93	
3	0,96	9,75	57,25	
4	0,92	10,33	58,28	
5	0,94	9,34	56,3	
6	0,95	9,78	57,22	
7	0,93	9,9	56,55	
8	0,94	9,67	59,04	
9	0,92	10,01	58,33	
10	0,93	9,34	62,1	
	0,942	9,855	58,528	

Tabela 12. Resultados obtidos da busca de todos os registros da tabela X no PostgreSQL.

PostgreSQL				
Teste de Busca na tabela X				
	1 milhão	10 milhões	50 milhões	100 milhões
1	0,83	9,12	113,84	
2	0,8	16,38	125,15	
3	1,01	8,54	127,21	
4	0,81	8,37	129,26	
5	0,82	8,55	129,59	
6	0,81	9,01	119,2	
7	1	8,67	121,9	
8	0,88	8,34	122,43	
9	0,91	9,44	123,41	
10	0,82	8,34	125,3	
	0,869	9,476	123,729	

As tabelas 7 e 8 apresentam os resultados da bateria de testes de busca completa nas collections X e Y do MongoDB, as tabelas 9 e 10 apresentam os resultados do MySQL e as tabelas 11 e 12 apresentam os resultados do PostgreSQL. O tempo de execução estão em segundos e o último campo da tabela é a média utilizada no gráfico de resultados. Os campos que estão vazios são os testes que não foram possíveis executar.

5.2.2 – Busca de todos os registros com INNER JOIN

O segundo teste de busca realizado, foram os testes de busca com INNER JOIN. O intuito deste teste é verificar como é o desempenho dos bancos de dados realizando buscas complexas. A consulta é feita buscando todos os campos menos a chave estrangeira da tabela X.

Como o modelo NoSQL não suporta relações, foi necessário utilizar uma função de agregação avançada do MongoDB, o Map/Reduce. A função faz a junção das duas tabelas e gera uma nova tabela contendo os dados dessa junção, como pode ser visto na figura x. A Figura 25 apresenta o exemplo de agregação avançada no MongoDB.

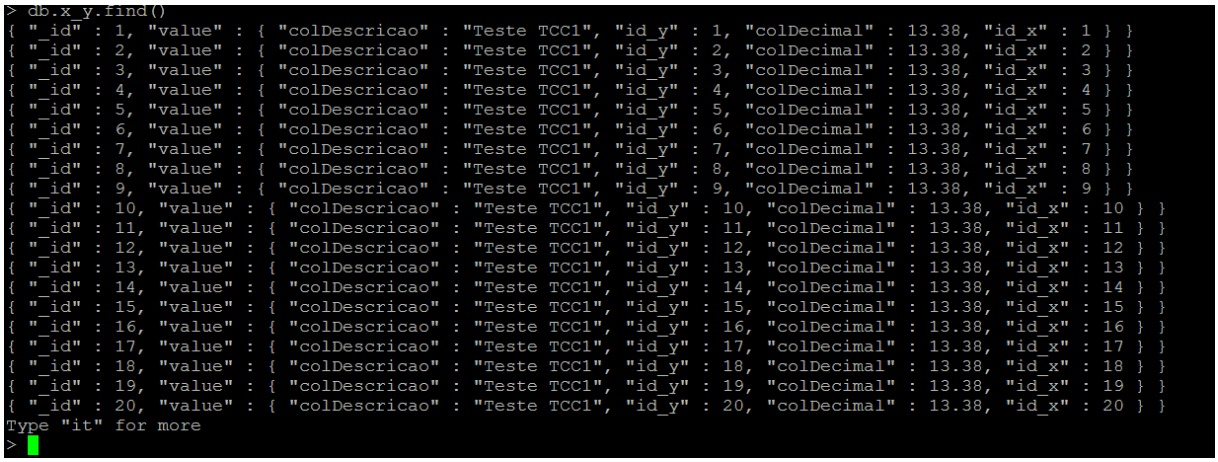
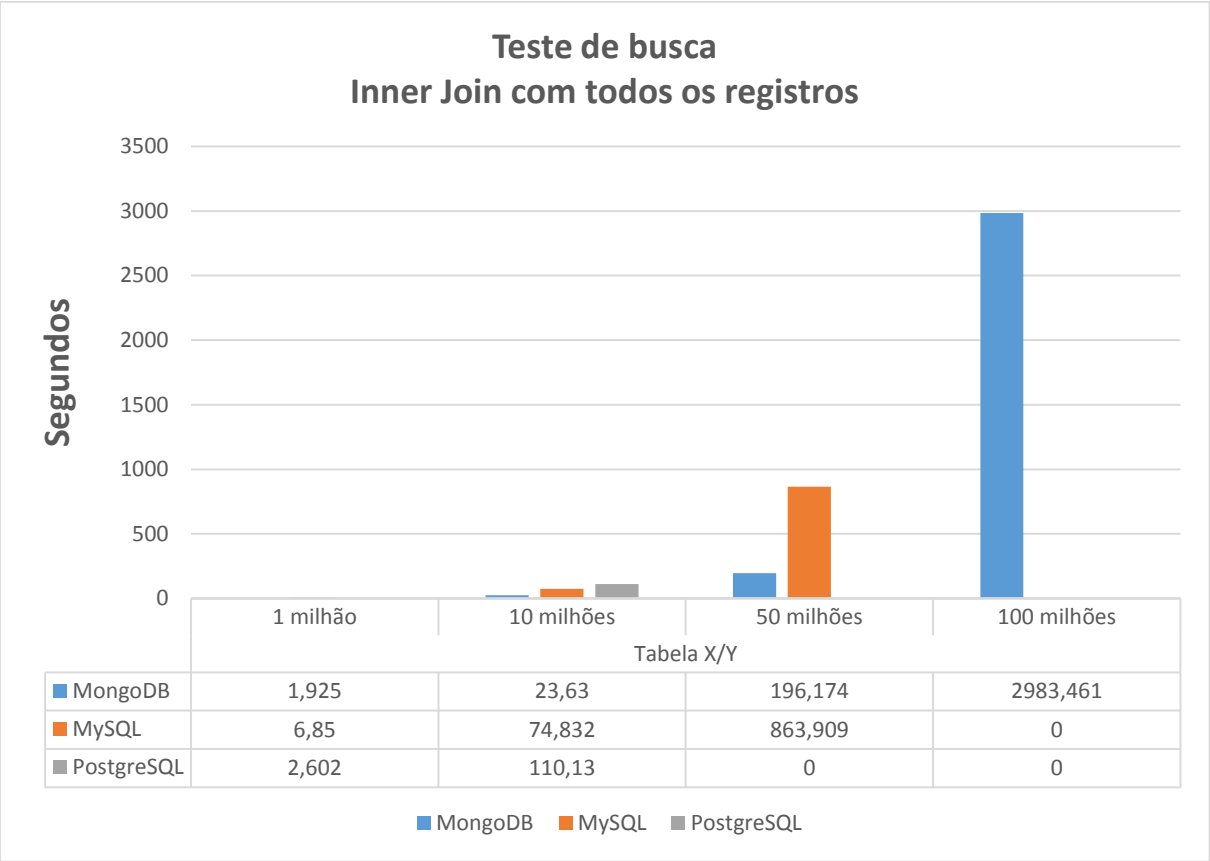


Fig. 26. Exemplo de agregação avançada no MongoDB. (Próprio autor).

O resultado dos testes pode ser visto no Gráfico 3. Quanto menor o tempo melhor o desempenho do banco de dados.

Gráfico 3. Resultado dos testes de INNER JOIN com todos os registros.



Assim como nos primeiros testes de busca o problema de memória também ocorreu com os testes com INNER JOIN, mas neste teste o MySQL também apresentou problemas. A partir de cinquenta milhões de registros foi preciso aumentar a memória da máquina virtual para que o MySQL pudesse executar a operação e a partir dos cem milhões de registros já não era mais possível realizar a consulta. Neste teste o PostgreSQL não conseguiu realizar a operação a partir dos cinquenta milhões de registros. A Figura 27 representa o erro na busca com INNER JOIN no MySQL.

```
mysql> use tcc
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select y.id_y_cem_milhoes, col_descricao, id_x_cem_milhoes, col_decimal f
rom y_cem_milhoes y inner join x_cem_milhoes x on y.id_y_cem_milhoes = x.id_y_ce
m_milhoes;
Killed
[root@localhost ~]#
```

Fig. 27. Por falta de memória a consulta falha no MySQL. (Próprio autor).

Embora os bancos de dados relacionais não tenham conseguido realizar as buscas com INNER JOIN, o MongoDB se mostrou muito superior aos demais, desde a primeira tabela já mostrou grande rapidez, acompanhado de perto do PostgreSQL. Mas a partir dos dez milhões de registros o MongoDB apresentou uma grande vantagem com relação aos outros bancos de dados, o PostgreSQL que tinha aproximadamente o mesmo desempenho do MongoDB na tabela de um milhão de registros aumentou o tempo de execução, ficando com o pior desempenho na tabela de dez milhões de registros.

Todos os resultados obtidos na bateria de testes são apresentados da tabela 13 a 16.

Tabela 13. Resultados obtidos da busca de todos os registros com agregação avançada das collections X e Y no MongoDB.

MongoDB				
Teste de Busca com tabela agregada				
	1 milhão	10 milhões	50 milhões	100 milhões
1	2,01	23,17	194,57	3507,71
2	1,96	24,23	193,33	3025,89
3	1,76	23,23	205,79	3240,8
4	1,97	23,85	194,28	2256,65
5	1,89	23,54	194,38	3255,32
6	1,94	23,43	195,31	3028,62
7	1,88	23,15	194,28	2789,99
8	1,99	24,03	193,8	3088,38
9	1,92	24,22	197,53	3102,33
10	1,93	23,53	198,47	2538,92
	1,925	23,638	196,174	2983,461

Tabela 14. Resultados obtidos da busca de todos os registros com INNER JOIN das tabelas X e Y no MySQL.

MySQL				
Teste de Busca com INNER JOIN				
	1 milhão	10 milhões	50 milhões	100 milhões
1	6,66	71,96	1328,02	
2	7,13	74,18	477,76	
3	6,89	77,43	910,39	
4	6,75	73,69	898,39	
5	6,89	74,38	493,3	
6	7,01	76,01	814,42	
7	6,68	77,5	812,55	
8	6,79	72,05	905,1	
9	6,99	75,23	1100,04	

10	6,71	75,89	899,12	
	6,85	74,832	863,909	

Tabela 15. Resultados obtidos da busca de todos os registros com INNER JOIN das tabelas X e Y no PostgreSQL.

PostgreSQL				
Teste de Busca com INNER JOIN				
	1 milhão	10 milhões	50 milhões	100 milhões
1	3,69	83,21		
2	2,59	153,97		
3	2,61	133,23		
4	2,69	102,25		
5	2,8	95,4		
6	2,38	111,49		
7	2,81	98,5		
8	2,34	99,9		
9	2,1	101,05		
10	2,01	122,3		
	2,602	110,13		

A Tabela 13 apresenta os resultados da bateria de testes de busca com agregação avançada no MongoDB, a Tabela 14 apresenta o resultado do MySQL e a Tabela 15 apresenta os resultados do PostgreSQL. O tempo de execução estão em segundos e o último campo da tabela é a média utilizada no gráfico de resultados. Os campos que estão vazios são dos testes que não foram possíveis a execução.

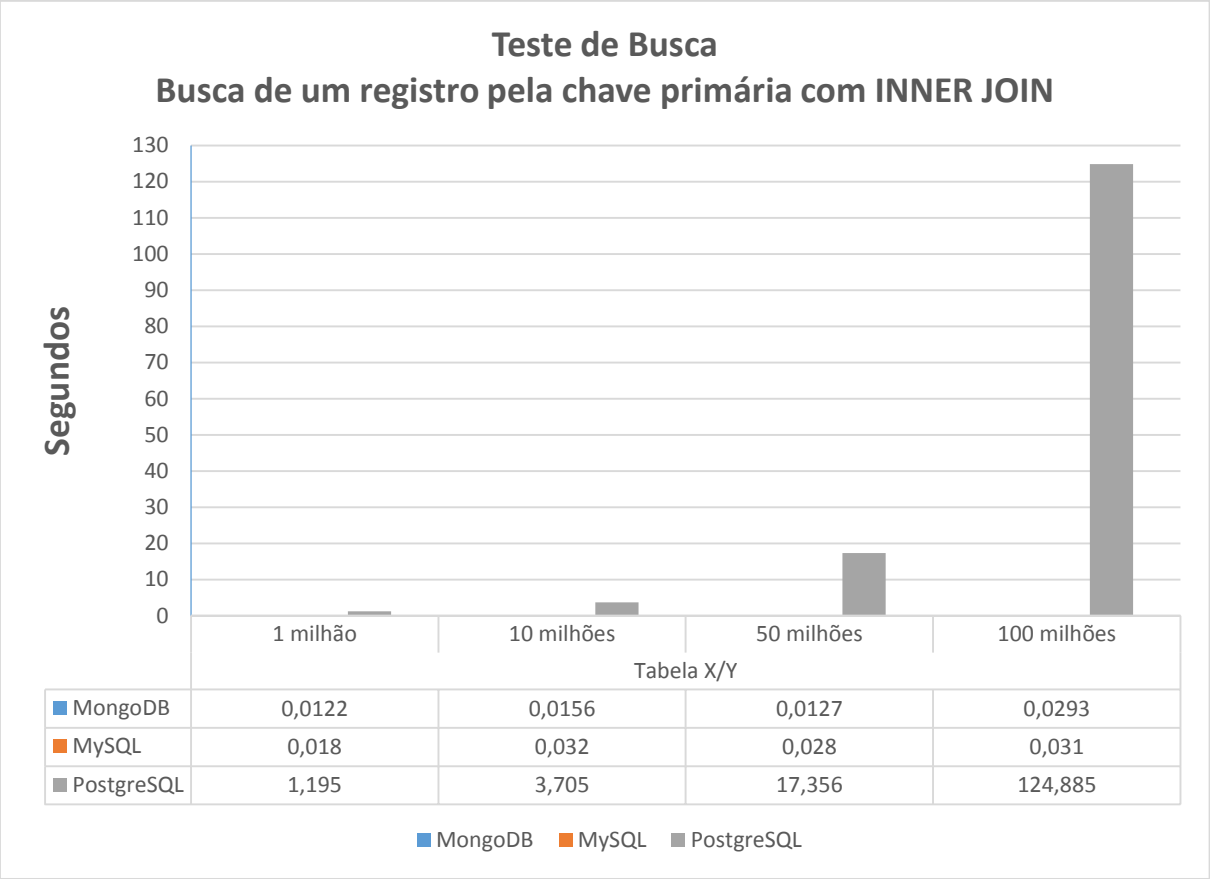
5.2.3 - Busca de um registro por chave primária utilizando INNER JOIN

O terceiro teste de busca realizado são os testes de busca de um registro por chave primária utilizando INNER JOIN. O MongoDB não possui chave primária, ele possui um identificador de documento que equivale à uma chave primária. Esse identificador não precisa ser modelado, isso fica a critério do administrador do banco de dados, caso não seja modelado o identificador fica definido com o valor nativo do MongoDB, o ObjectId.

Neste trabalho foi necessário modelar o identificador de documento do MongoDB, é importante ressaltar que essa modelagem precisa ser feita assim que uma collection é criada, já que um identificador não pode ser modificado depois de sua criação.

Os resultados dos testes podem ser vistos no Gráfico 4. Quanto menor o tempo, melhor o desempenho do banco de dados.

Gráfico 4. Resultado dos testes de busca de um registro pela chave primária e identificador de documentos utilizando INNER JOIN e agregação avançada.



A busca por chave primária é quase que instantânea para o MongoDB e o MySQL, com o MongoDB tendo uma leve vantagem sobre o MySQL, já que o PostgreSQL mostrou um desempenho ruim em relação aos outros, chegando a demorar dois minutos para uma consulta na tabela de cem milhões de registros.

Os resultados obtidos nas baterias de testes são apresentados da tabela 16 a 18.

Tabela 16. Resultados obtidos da busca de um registro pelo identificador de documentos com agregação avançada das collections X e Y no MongoDB.

MongoDB				
Teste de Busca de um registro pela chave primária agregação avançada				
	1 milhão	10 milhões	50 milhões	100 milhões
1	0,012	0,085	0,036	0,038
2	0,009	0,007	0,008	0,032
3	0,007	0,009	0,007	0,031
4	0,007	0,007	0,006	0,03
5	0,01	0,007	0,02	0,022
6	0,009	0,008	0,007	0,025

7	0,009	0,012	0,008	0,033
8	0,009	0,006	0,018	0,032
9	0,02	0,007	0,008	0,028
10	0,03	0,008	0,009	0,022
	0,0122	0,0156	0,0127	0,0293

Tabela 17. Resultados obtidos da busca de um registro pela chave primária com INNER JOIN das tabelas X e Y no MySQL.

MySQL				
Teste de Busca de um registro pela chave primária com INNER JOIN				
	1 milhão	10 milhões	50 milhões	100 milhões
1	0,07	0,14	0,09	0,07
2	0,01	0,01	0,01	0,03
3	0	0,01	0,01	0,01
4	0,01	0,01	0,01	0,01
5	0,01	0,01	0,02	0,01
6	0,02	0,09	0,06	0,07
7	0	0,01	0,01	0,01
8	0,01	0,02	0,05	0,08
9	0,05	0,01	0,01	0,01
10	0	0,01	0,01	0,01
	0,018	0,032	0,028	0,031

Tabela 18. Resultados obtidos da busca de um registro pela chave primária com INNER JOIN das tabelas X e Y no PostgreSQL.

PostgreSQL				
Teste de Busca de um registro pela chave primária com INNER JOIN				
	1 milhão	10 milhões	50 milhões	100 milhões
1	2,74	1,22	17,29	125,36
2	1,04	4,51	17,27	107,8
3	1,03	7,68	17,39	119,22
4	1,02	3,83	17,1	128,49
5	1,01	3,44	17,5	123,42
6	1,04	3,43	17,28	124,82
7	1,07	3,12	17,99	127,29
8	0,99	3,41	17,55	129,97
9	1,02	3,01	16,99	130,1
10	0,99	3,4	17,2	132,38
	1,195	3,705	17,356	124,885

A Tabela 16 apresenta os resultados da bateria de testes de busca de um registro com agregação avançada utilizando identificador de documento no MongoDB, a Tabela 17

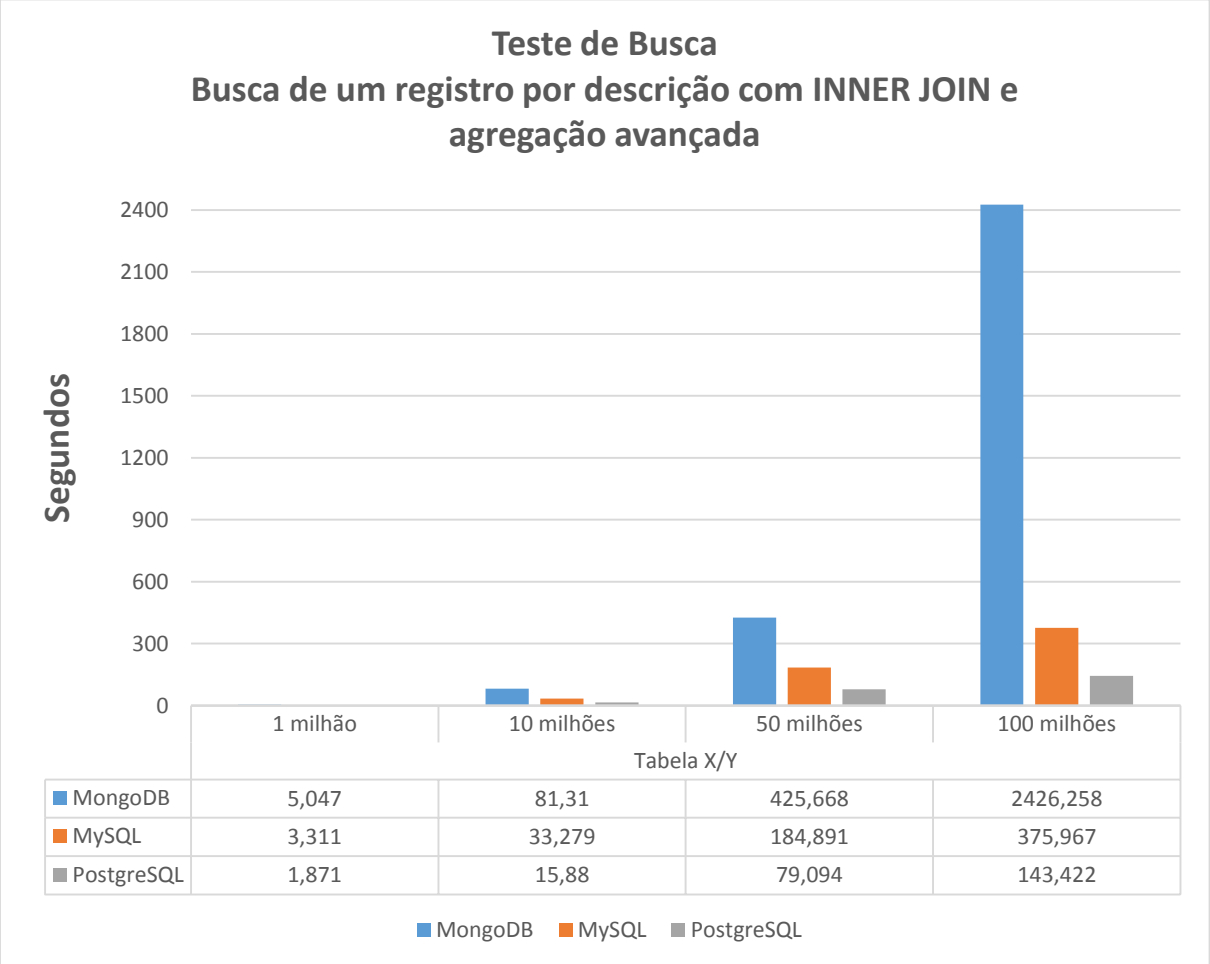
apresenta os resultados do MySQL com INNER JOIN, que utilizou a chave primária da tabela Y e a Tabela 18 apresenta os resultados do PostgreSQL que utiliza a mesma lógica de busca do MySQL. O tempo de execução estão em segundos e o último campo da tabela é a média utilizada no gráfico de resultados.

5.2.4 - Busca de um registro por descrição utilizando INNER JOIN

O quarto teste de busca realizado é parecido com o anterior, mas neste um registro é consultado pela descrição, a coluna “col_descricao”.

Os resultados dos testes são apresentados no Gráfico 5. Quanto menor o tempo de execução melhor o desempenho do banco de dados.

Gráfico 5. Resultado dos testes de busca de um registro pela descrição utilizando INNER JOIN e agregação avançada.



Com relação ao teste anterior houve uma inversão de desempenho. O MongoDB que teve um melhor desempenho ficou com o pior desempenho em busca por descrição, o

PostgreSQL neste teste teve o melhor desempenho e o MySQL continuou sendo o segundo com melhor desempenho. Destaque para o MongoDB que teve cerca de 40 minutos para realizar a busca.

Todos os resultados obtidos na bateria de testes são apresentados da tabela 19 a 21.

Tabela 19. Resultados obtidos da busca de um registro pela descrição com agregação avançada das collections X e Y no MongoDB.

MongoDB				
Teste de Busca de um registro pela descrição com agregação avançada				
	1 milhão	10 milhões	50 milhões	100 milhões
1	6,05	80,49	421,06	2502,24
2	5,72	87,88	410,62	2370,42
3	4,77	79,49	438,31	2430,49
4	4,57	80,35	410,59	2362,28
5	4,68	82,71	427,48	2451,93
6	4,78	80,82	426,41	2512,73
7	4,69	80,17	426,83	2355,22
8	4,96	80,42	422,4	2411,46
9	5,05	79,98	433,88	2423,36
10	5,2	80,79	439,1	2442,45
	5,047	81,31	425,668	2426,258

Tabela 20. Resultados obtidos da busca de um registro pela descrição com INNER JOIN das tabelas X e Y no MySQL.

MySQL				
Teste de Busca de um registro pela descrição com INNER JOIN				
	1 milhão	10 milhões	50 milhões	100 milhões
1	3,75	35,11	188,03	403,69
2	3,17	32,61	184,1	371,28
3	3,18	32,5	184,62	369,53
4	3,21	33,07	181,38	368,17
5	3,34	33,09	183,03	371,09
6	3,22	32,78	181,89	382,69
7	3,15	34,01	187,2	371,11
8	3,19	33,2	186,65	388,59
9	3,69	33,41	184,01	369
10	3,21	33,01	188	364,52
	3,311	33,279	184,891	375,967

Tabela 21. Resultados obtidos da busca de um registro pela descrição com INNER JOIN das tabelas X e Y no PostgreSQL.

PostgreSQL				
Teste de Busca de um registro pela descrição com INNER JOIN				
	1 milhão	10 milhões	50 milhões	100 milhões
1	1,93	21,51	69,96	129,06
2	1,55	17,03	75,4	127,8
3	2,33	20,38	101,93	161,42
4	1,87	18,34	76,38	149,47
5	1,66	12,53	77,02	150,43
6	1,89	13,54	76,82	138,09
7	1,69	12,78	80,5	139,38
8	1,99	14,37	82,43	141,35
9	1,92	12,44	75,3	142,12
10	1,88	15,88	75,2	155,1
	1,871	15,88	79,094	143,422

A Tabela 19 apresenta os resultados da bateria de testes de busca de um registro com agregação avançada utilizando a descrição do campo “col_descricao” no MongoDB, a Tabela 20 apresenta os resultados do MySQL com INNER JOIN, que utilizou o campo “col_descricao” da tabela Y e a Tabela 21 apresenta os resultados do PostgreSQL que utiliza a mesma lógica de busca do MySQL. O tempo de execução estão em segundos e o último campo da tabela é a média utilizada no gráfico de resultados.

5.2.5 - Teste de busca e junção com GROUP BY, INNER JOIN e SUM

O quinto e último teste de busca tem como objetivo analisar o desempenho dos bancos de dados utilizando funções de junção e soma. Assim como no modelo relacional o MongoDB possui suporte as funções GROUP BY e SUM, sendo possível utiliza-las facilmente pelas funções $group e $sum.

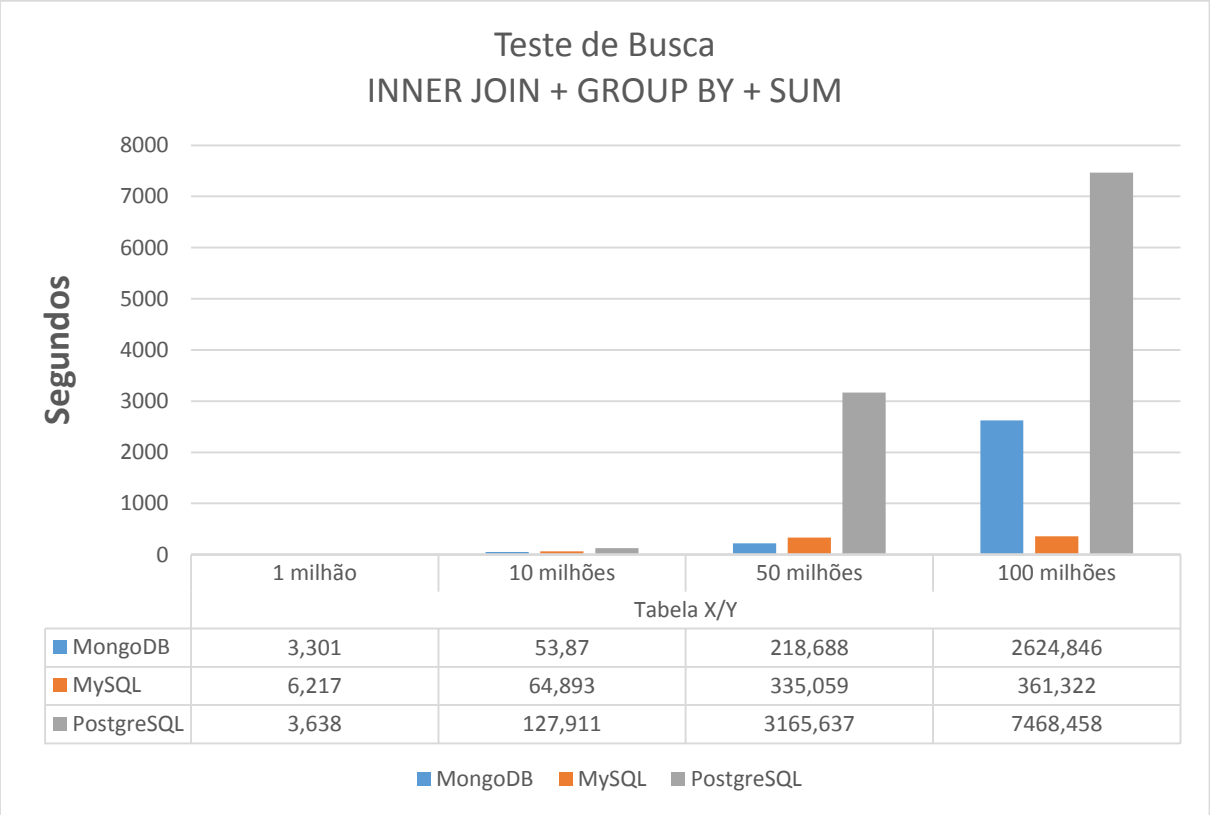
O MongoDB apresentou alguns problemas relacionado a capacidade de agregação da função $group. Por padrão o MongoDB suporta para essa função apenas 16MB para o tamanho do documento, a partir de consultas com cinquenta milhões de documentos essa capacidade é ultrapassada. Para mitigar esse problema é preciso somente habilitar a variável “allowDiskUse” dentro da função de junção, com isso a agregação foi executada sem maiores problemas.

Para este teste todas as tabelas foram divididas em quatro partes iguais, onde cada parte tinha uma descrição. Por exemplo, na tabela de um milhão do registro 1 ao 250000 mil a coluna

“col_descrição” foi definida como “Teste TCC1”, do registro 250001 ao 500000 a “col_descrição” foi definida como “Teste TCC2” e assim sucessivamente para todas as tabelas.

Os resultados dos testes são apresentados no Gráfico 6.

Gráfico 6. Resultado dos testes de busca com GROUP BY, SUM, INNER JOIN e agregação avançada.



Neste teste o MySQL teve melhor estabilidade, até a tabela de cinquenta milhões de dados o MongoDB se mostrou com melhor desempenho, a partir de cem milhões de registros o MongoDB aumentou seu tempo em dez vezes. Com um milhão de registros o MongoDB e o PostgreSQL possuem desempenho parecidos, o Mysql leva o dobro de tempo para realizar as operações.

Analisando o gráfico pode-se dizer que o MySQL teve um melhor desempenho com as operações realizadas na bateria dos testes.

Os resultados obtidos nas baterias de teste são apresentados da tabela 22 a 24.

Tabela 22. Resultados obtidos da busca com $GROUP, $SUM, e agregação avançada das collections X e Y no MongoDB.

MongoDB

Teste de Busca com agregação avançada + $GROUP + $SUM				
	1 milhão	10 milhões	50 milhões	100 milhões
1	1,93	21,51	69,96	129,06
2	1,55	17,03	75,4	127,8
3	2,33	20,38	101,93	161,42
4	1,87	18,34	76,38	149,47
5	1,66	12,53	77,02	150,43
6	1,89	13,54	76,82	138,09
7	1,69	12,78	80,5	139,38
8	1,99	14,37	82,43	141,35
9	1,92	12,44	75,3	142,12
10	1,88	15,88	75,2	155,1
	1,871	15,88	79,094	143,422

Tabela 23. Resultados obtidos da busca com INNER JOIN, GROUP BY e SUM das tabelas Y e X no MySQL.

MySQL				
Teste de Busca com INNER JOIN + GROUP BY + SUM				
	1 milhão	10 milhões	50 milhões	100 milhões
1	1,93	21,51	69,96	129,06
2	1,55	17,03	75,4	127,8
3	2,33	20,38	101,93	161,42
4	1,87	18,34	76,38	149,47
5	1,66	12,53	77,02	150,43
6	1,89	13,54	76,82	138,09
7	1,69	12,78	80,5	139,38
8	1,99	14,37	82,43	141,35
9	1,92	12,44	75,3	142,12
10	1,88	15,88	75,2	155,1
	1,871	15,88	79,094	143,422

Tabela 24. Resultados obtidos da busca com INNER JOIN, GROUP BY e SUM das tabelas Y e X no PostgreSQL.

PostgreSQL				
Teste de Busca com INNER JOIN + GROUP BY + SUM				
	1 milhão	10 milhões	50 milhões	100 milhões
1	1,93	21,51	69,96	129,06
2	1,55	17,03	75,4	127,8
3	2,33	20,38	101,93	161,42
4	1,87	18,34	76,38	149,47
5	1,66	12,53	77,02	150,43
6	1,89	13,54	76,82	138,09

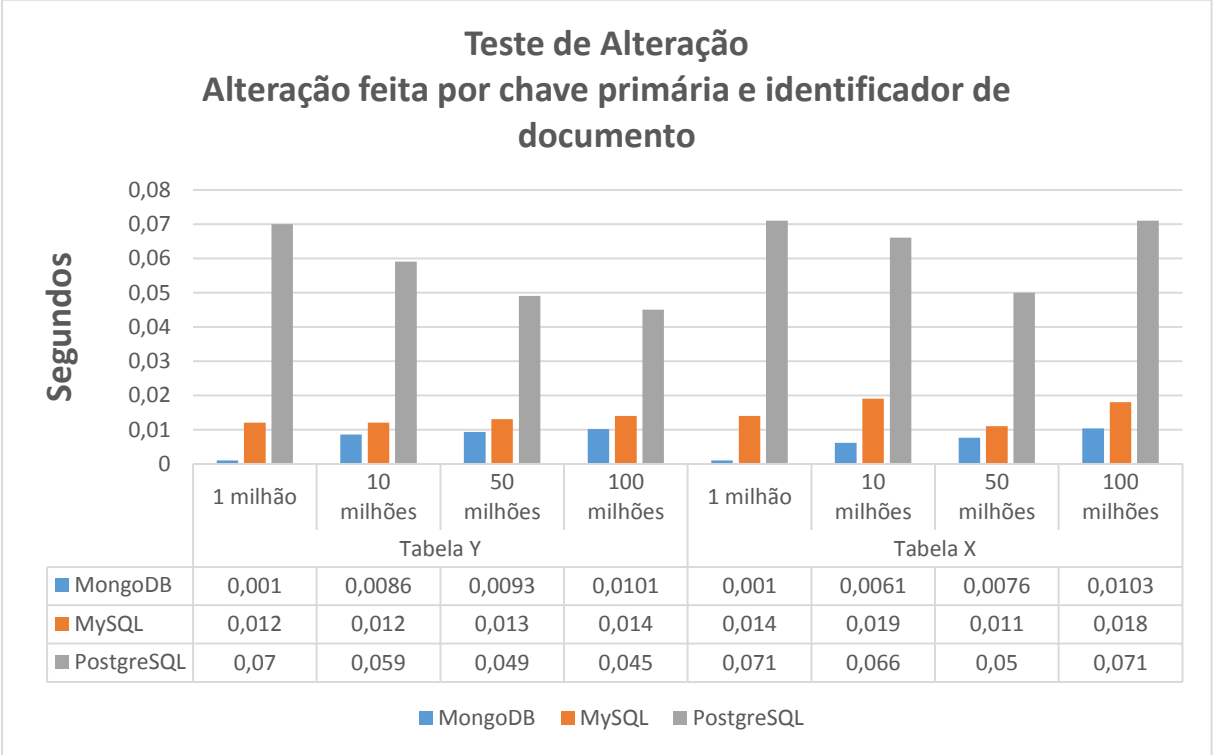
7	1,69	12,78	80,5	139,38
8	1,99	14,37	82,43	141,35
9	1,92	12,44	75,3	142,12
10	1,88	15,88	75,2	155,1
	1,871	15,88	79,094	143,422

A Tabela 22 apresenta os resultados da bateria de testes de busca com agrupamento e a função “$SUM” no MongoDB, a Tabela 23 apresenta os resultados do MySQL e a Tabela 24 apresenta os resultados do PostgreSQL. O tempo de execução estão em segundos e o último campo da tabela é a média utilizada no gráfico de resultados.

5.3 - Teste de alteração

O sexto teste feito foi o de alteração dos registros. Nestes testes foram feitas alterações por chave primária e pela coluna “col_descricao” da tabela Y, e pela coluna “col_decimal” da tabela X. O Gráfico 7 apresenta os resultados dos testes.

Gráfico 7. Resultado dos testes de alteração por chave primária e identificador de documento.

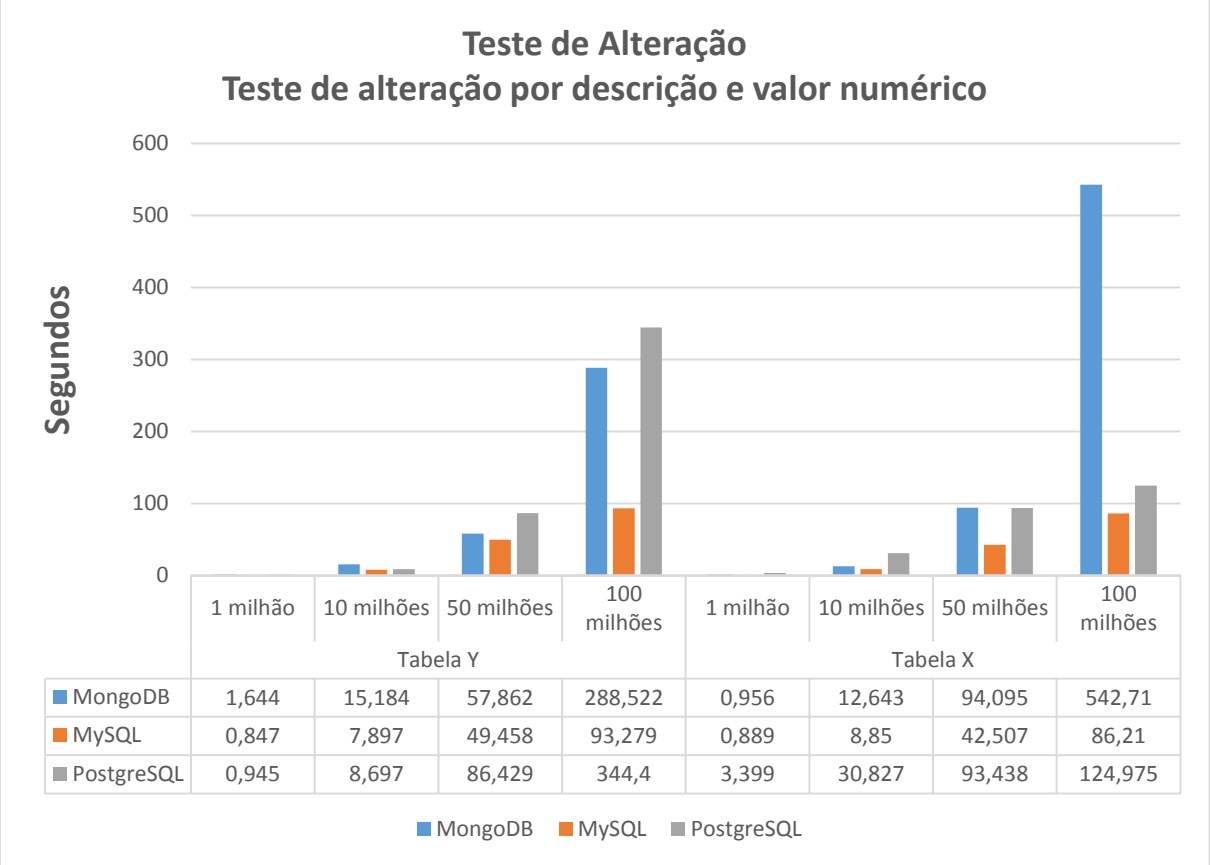


Embora o tempo de execução dos testes seja muito baixo o MongoDB teve o melhor desempenho em alteração de registro por identificador ou chave primária seguido de perto pelo MySQL, já o PostgreSQL teve um desempenho modesto. Mas isto levando em conta o cenário

representado no gráfico, já que mesmo utilizando o PostgreSQL o resultado seria quase que instantâneo.

Os resultados da execução dos testes da alteração por descrição ou valor decimal das colunas são representados no gráfico 8.

Gráfico 8. Resultado dos testes de alteração descrição e valor numérico.



O MySQL teve o melhor desempenho em nas buscas por descrição e valor numérico. O PostgreSQL e o MongoDB variaram seus desempenhos, em busca por descrição até dez milhões de registros o PostgreSQL tem melhor desempenho e partir dos cinquenta milhões o MongoDB leva vantagem. Na busca por valor numérico o MongoDB tem melhor desempenho até dez milhões de registros, e a partir dos cinquenta milhões de registros o PostgreSQL tem melhor desempenho.

Os resultados obtidos nos testes estão representados da tabela 25 a 36.

Tabela 25. Resultados obtidos da alteração por identificador de documento da collection Y no MongoDB.

MongoDB

Teste de alteração por identificador de documento da collection Y				
	1 milhão	10 milhões	50 milhões	100 milhões
1	0,002	0,005	0,003	0,026
2	0,001	0,003	0,005	0,006
3	0,002	0,001	0,015	0,004
4	0,001	0,002	0,009	0,018
5	0	0,002	0,018	0,019
6	0,001	0,001	0,001	0,005
7	0,002	0,058	0,022	0,007
8	0	0,012	0,004	0,001
9	0	0,001	0,007	0,014
10	0,001	0,001	0,009	0,001
	0,001	0,0086	0,0093	0,0101

Tabela 26. Resultados obtidos da alteração por identificador de documento da collection X no MongoDB.

MongoDB				
Teste de alteração por identificador de documento da collection X				
	1 milhão	10 milhões	50 milhões	100 milhões
1	0,001	0,009	0,018	0,004
2	0,002	0,002	0,005	0,017
3	0,002	0,012	0,007	0,015
4	0,002	0,008	0,003	0,005
5	0	0,002	0,001	0,003
6	0	0,001	0,02	0,002
7	0,001	0,002	0,005	0,021
8	0,001	0,002	0,007	0,02
9	0,001	0,022	0,009	0,007
10	0	0,001	0,001	0,009
	0,001	0,0061	0,0076	0,0103

Tabela 27. Resultados obtidos da alteração por descrição da collection Y no MongoDB.

MongoDB				
Teste de alteração por descrição da collection Y				
	1 milhão	10 milhões	50 milhões	100 milhões
1	1,57	15,69	59,1	279,29
2	1,76	14,86	56,97	286,54
3	1,62	14,99	57,39	290,36
4	1,65	15,21	58,47	290,35
5	1,69	15,25	58,92	289,8
6	1,73	14,78	58,24	288,39
7	1,59	15,42	57,36	289,52
8	1,61	15,62	57,84	290,16

9	1,62	15,22	56,99	290,19
10	1,6	14,8	57,34	290,62
	1,644	15,184	57,862	288,522

Tabela 28. Resultados obtidos da alteração por valor numérico da collection X no MongoDB.

MongoDB				
Teste de alteração por valor numérico da tabela X				
	1 milhão	10 milhões	50 milhões	100 milhões
1	0,86	13,11	94,84	527,91
2	0,95	12,63	92,65	531,77
3	1,05	11,99	93,37	637,52
4	0,89	13,03	94,28	531,29
5	0,93	12,75	93,86	532,41
6	1,01	12,82	94,28	537,9
7	0,95	12,57	94,23	532,61
8	0,93	13,03	94,87	533,42
9	0,97	11,98	93,98	528,37
10	1,02	12,52	94,59	533,9
	0,956	12,643	94,095	542,71

Tabela 29. Resultados obtidos da alteração por chave primária da tabela Y no MySQL.

MySQL				
Teste de alteração por chave primária da tabela Y				
	1 milhão	10 milhões	50 milhões	100 milhões
1	0,03	0,03	0,03	0,01
2	0	0,01	0	0
3	0,01	0	0,01	0
4	0,03	0,01	0,01	0,03
5	0,03	0	0	0,02
6	0	0	0	0,03
7	0	0,02	0,01	0,01
8	0,02	0,01	0,01	0,02
9	0	0,02	0,05	0,02
10	0	0,02	0,01	0
	0,012	0,012	0,013	0,014

Tabela 30. Resultados obtidos da alteração por chave primária da tabela X no MySQL.

MySQL				
Teste de alteração por chave primária da tabela X				
	1 milhão	10 milhões	50 milhões	100 milhões
1	0,04	0,02	0,01	0,03

2	0,01	0,04	0,01	0,01
3	0,01	0,02	0	0
4	0,01	0,02	0	0,01
5	0,01	0,02	0,02	0,02
6	0,02	0,01	0,02	0,03
7	0,02	0,02	0,02	0,01
8	0,01	0,01	0,01	0,01
9	0,01	0,01	0	0,03
10	0	0,02	0,02	0,03
	0,014	0,019	0,011	0,018

Tabela 31. Resultados obtidos da alteração por descrição da tabela Y no MySQL.

MySQL				
Teste de alteração por descrição da tabela Y				
	1 milhão	10 milhões	50 milhões	100 milhões
1	0,94	9,07	57,6	93,45
2	0,82	6,04	48,88	95,99
3	0,84	8,28	48,82	90,97
4	0,88	6,09	47,87	93,19
5	0,83	9,01	48,89	91,46
6	0,84	8,8	47,57	93,55
7	0,86	8,78	50,01	92,98
8	0,84	6,9	49,04	93,12
9	0,8	7,99	48,02	93,21
10	0,82	8,01	47,88	94,87
	0,847	7,897	49,458	93,279

Tabela 32. Resultados obtidos da alteração por valor numérico da tabela X no MySQL.

MySQL				
Teste de alteração por valor numérico da tabela X				
	1 milhão	10 milhões	50 milhões	100 milhões
1	1,33	9,54	42,27	85,76
2	0,78	8,95	42,75	87,56
3	0,73	8,23	41,36	85,93
4	0,87	8,66	42,65	85,84
5	0,8	8,65	42,66	86,03
6	0,9	8,77	43,05	85,55
7	0,89	9,05	43,07	87,43
8	0,92	9,01	42,89	86,12
9	0,77	8,65	41,93	85,78
10	0,9	8,99	42,44	86,1
	0,889	8,85	42,507	86,21

Tabela 33. Resultados obtidos da alteração por chave primária da tabela Y no PostgreSQL.

PostgreSQL				
Teste de alteração por chave primária da tabela Y				
	1 milhão	10 milhões	50 milhões	100 milhões
1	0,26	0,23	0,12	0,07
2	0,03	0,05	0,06	0,06
3	0,04	0,06	0,05	0,05
4	0,04	0,01	0,05	0,03
5	0,06	0,09	0,02	0,04
6	0,05	0,02	0,01	0,04
7	0,07	0,01	0,06	0,05
8	0,03	0,1	0,04	0,06
9	0,09	0,01	0,05	0,02
10	0,03	0,01	0,03	0,03
	0,07	0,059	0,049	0,045

Tabela 34. Resultados obtidos da alteração por chave primária da tabela X no PostgreSQL.

PostgreSQL				
Teste de alteração por chave primária da tabela X				
	1 milhão	10 milhões	50 milhões	100 milhões
1	0,16	0,11	0,09	0,29
2	0,06	0,05	0,05	0,05
3	0,05	0,05	0,05	0,1
4	0,05	0,09	0,05	0,03
5	0,03	0,05	0,04	0,04
6	0,04	0,07	0,04	0,05
7	0,06	0,08	0,05	0,12
8	0,07	0,05	0,05	0,01
9	0,09	0,1	0,06	0
10	0,1	0,01	0,02	0,02
	0,071	0,066	0,05	0,071

Tabela 35. Resultados obtidos da alteração por descrição da tabela Y no PostgreSQL.

PostgreSQL				
Teste de alteração por valor descrição da tabela Y				
	1 milhão	10 milhões	50 milhões	100 milhões
1	1,06	10,56	110,1	320,3
2	0,7	9,67	79,08	318,07
3	0,7	8,13	75	371,82
4	2,63	7,96	86,82	353,41
5	0,78	8,32	79,4	371,3
6	0,74	7,8	81,32	342,2
7	0,9	8,3	82,47	353,28

8	0,43	8,02	86,36	323,82
9	1,01	9,01	88,91	345,4
10	0,5	9,2	94,83	344,4
	0,945	8,697	86,429	344,4

Tabela 36. Resultados obtidos da alteração por valor numérico da tabela X no PostgreSQL.

PostgreSQL				
Teste de alteração por valor numérico da tabela X				
	1 milhão	10 milhões	50 milhões	100 milhões
1	3,96	33,01	99,18	124,67
2	3,38	31,07	91,13	126,34
3	3,52	29,6	89,71	124,92
4	3,33	30,5	92,16	125,82
5	3,42	31,41	93,12	124,37
6	3,28	29,89	93,81	123,48
7	3,19	30,28	93,17	124,93
8	3,41	30,49	94,01	125,43
9	3,1	30,12	93,89	124,82
10	3,4	31,9	94,2	124,97
	3,399	30,827	93,438	124,975

As tabelas 25 e 26 apresentam os resultados da bateria de testes de alteração de um registro por identificador de documento, das collections X e Y respectivamente no MongoDB, as tabelas 27 e 28 apresentam os resultados de alteração de um registro pelo campo “col_descricao” da collection Y e “col_decimal” da collection X no MongoDB.

As tabelas 29 e 30 apresentam os resultados da bateria de testes de alteração de um registro por chave primária das tabelas Y e X do MySQL, as tabelas 31 e 32 apresentam os resultados de alteração de um registro pelos campos “col_descricao” e “col_decimal” das tabelas X e Y respectivamente no MySQL.

As tabelas 33 e 34 apresentam os resultados da bateria de testes de alteração de um registro por chave primária do PostgreSQL, as tabelas 35 e 36 apresentam os resultados da bateria de testes de alteração de um registro no PostgreSQL seguindo o mesmo modelo do MySQL.

Todos os resultados são em segundos e o último campo das tabelas são as médias dos testes executados.

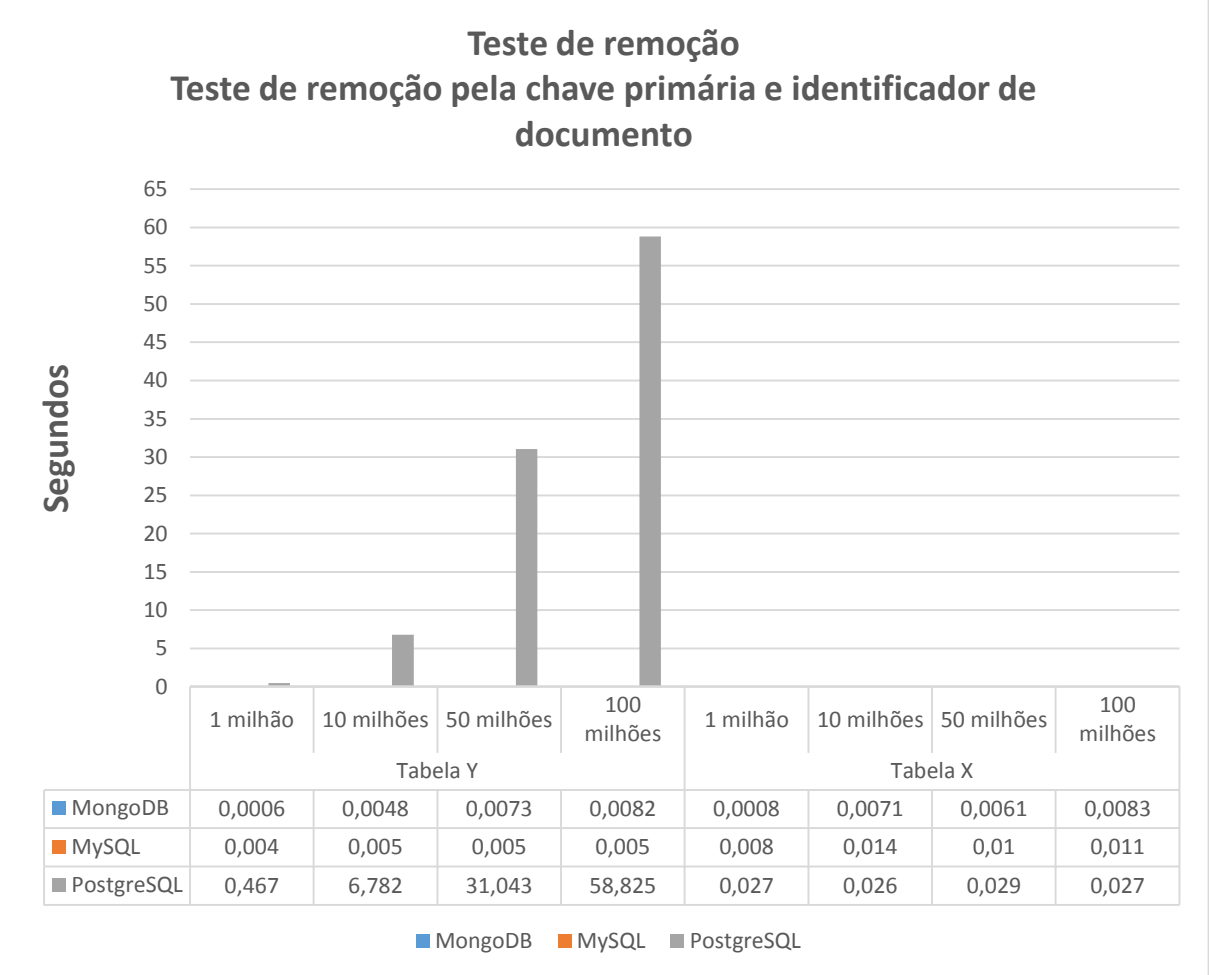
5.4 - Teste de remoção

O último teste realizado foi o de remoção de registros. Assim como nos testes de alteração, os testes de remoção foram feitos pela chave primária e identificador de documentos, por descrição da tabela e collection Y, e valor numérico da tabela e collection X.

Para os testes de remoção as tabelas do MySQL precisaram ter seu tipo modificado, já que por padrão elas tem o tipo MyISAM, que não suportam a função rollback, que será necessária para que nenhum registro seja de fato apagado. Então as tabelas foram modificadas para o tipo InnoDB.

Os resultados dos testes estão representados no Gráfico 9.

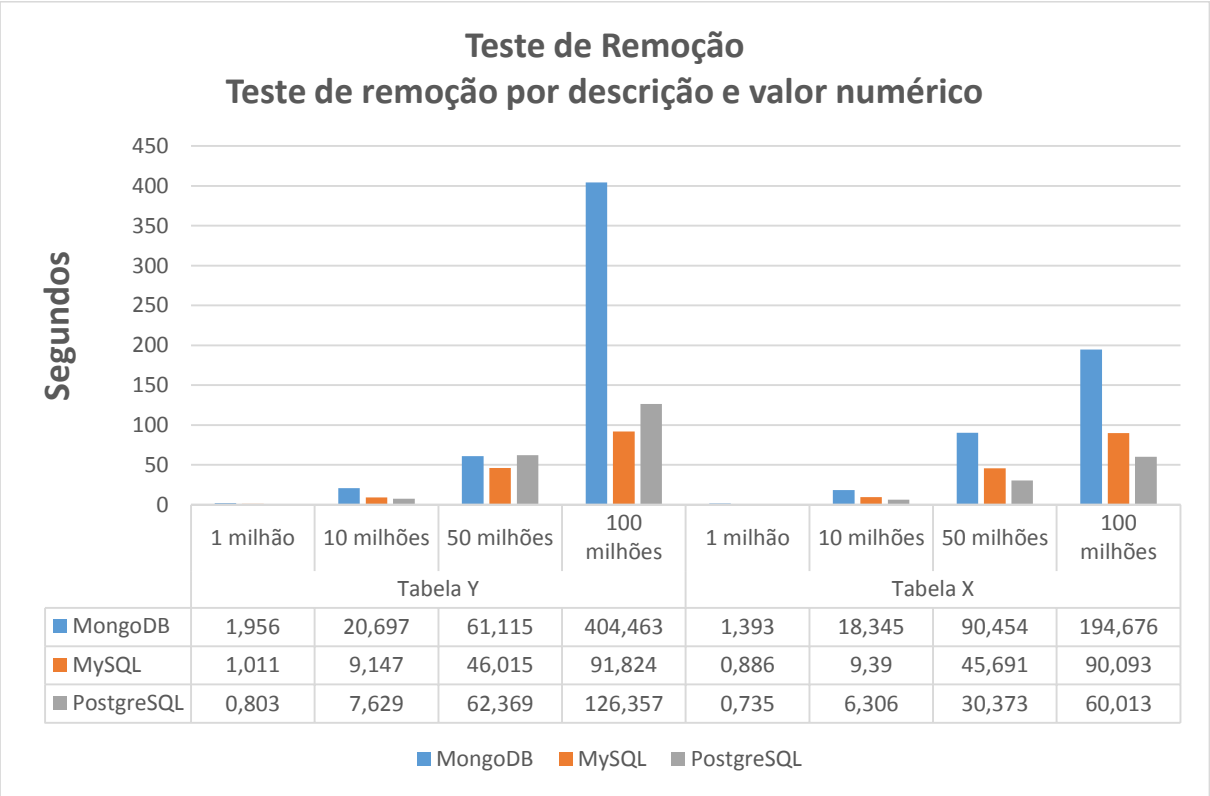
Gráfico 9. Resultado dos testes de remoção pela chave primária e identificador de documento.



Assim como no teste de alteração por chave primária o PostgreSQL apresentou desempenho ruim também na exclusão de registros pela chave primária, mas neste caso somente na tabela Y. Mais uma vez o MongoDB mostrou ótimo desempenho utilizando o identificador de documento, seguido de perto pelo MySQL.

Os resultados dos testes de remoção de registros por descrição e valor numérico estão representados no Gráfico 10.

Gráfico 10. Resultado dos testes de remoção pela descrição e valor numérico.



Tanto no teste de remoção por descrição quanto por valor numérico o MongoDB teve a pior performance, a disputa pelo melhor desempenho ficou entre o MySQL e o PostgreSQL, houve alternância entre os desempenhos dos dois bancos de dados relacionais. Para a remoção por valor numérico o PostgreSQL mostrou que tem o melhor desempenho, já em remoção por descrição o PostgreSQL tem vantagem até dez milhões de registros, o MySQL apresenta melhor desempenho a partir dos cinquenta milhões de registros.

Os resultados obtidos das baterias de testes nos bancos de dados estão representados da tabela 37 a 48.

Tabela 37. Resultados obtidos da remoção de um registro por identificador de documento da collection Y no MongoDB.

MongoDB				
Teste de remoção por identificador de documento da collection Y				
	1 milhão	10 milhões	50 milhões	100 milhões
1	0,001	0,006	0,009	0,013

2	0	0,003	0,005	0,009
3	0	0,004	0,007	0,005
4	0,001	0,012	0,003	0,018
5	0	0,009	0,002	0,001
6	0	0,005	0,013	0,003
7	0,002	0,001	0,008	0,007
8	0,001	0	0,006	0,001
9	0,001	0,007	0,015	0,014
10	0	0,001	0,005	0,011
	0,0006	0,0048	0,0073	0,0082

Tabela 38. Resultados obtidos da remoção de um registro por identificador de documento da collection X no MongoDB.

MongoDB				
Teste de remoção por identificador de documento da collection X				
	1 milhão	10 milhões	50 milhões	100 milhões
1	0,001	0,011	0,016	0,002
2	0,001	0,01	0,006	0,006
3	0	0,008	0,004	0,009
4	0	0,006	0,008	0,017
5	0,002	0,002	0,003	0,007
6	0,001	0,001	0,001	0,009
7	0	0,022	0,004	0,019
8	0,001	0,002	0,007	0,003
9	0,001	0,003	0,011	0,006
10	0,001	0,006	0,001	0,005
	0,0008	0,0071	0,0061	0,0083

Tabela 39. Resultados obtidos da remoção de um registro por descrição da collection Y no MongoDB.

MongoDB				
Teste de remoção por descrição da collection Y				
	1 milhão	10 milhões	50 milhões	100 milhões
1	1,91	21,08	60,22	402,84
2	1,96	20,6	60,19	406,58
3	2,06	20,29	62,63	404,79
4	1,89	20,42	61,63	403,51
5	2,01	21,32	61,22	402,5
6	1,94	20,34	61,5	402,98
7	1,87	21,48	61,49	405,63
8	2,03	20,31	60,2	405,77
9	1,9	20,61	60,73	405,93
10	1,99	20,52	61,34	404,1
	1,956	20,697	61,115	404,463

Tabela 40. Resultados obtidos da remoção de um registro por valor numérico da collection X no MongoDB.

MongoDB				
Teste de remoção por valor numérico da collection X				
	1 milhão	10 milhões	50 milhões	100 milhões
1	1,34	19,03	90,33	195,2
2	1,37	17,63	89,37	194,21
3	1,42	18,59	91,76	194,22
4	1,39	17,88	90,4	193,9
5	1,42	18,41	89,59	195,28
6	1,44	19,01	91,58	194,74
7	1,37	18,6	90,51	194,58
8	1,4	17,99	90,09	195,13
9	1,41	17,79	91,33	194,49
10	1,37	18,52	89,58	195,01
	1,393	18,345	90,454	194,676

Tabela 41. Resultados obtidos da remoção de um registro por chave primária da tabela Y no MySQL.

MySQL				
Teste de remoção por chave primária da tabela Y				
	1 milhão	10 milhões	50 milhões	100 milhões
1	0	0,03	0,01	0
2	0,01	0,01	0	0,01
3	0,01	0	0	0,01
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0,01	0	0,02	0,01
9	0,02	0,01	0,01	0,02
10	0	0	0,01	0
	0,005	0,005	0,005	0,005

Tabela 42. Resultados obtidos da remoção de um registro por chave primária da tabela X no MySQL.

MySQL				
Teste de remoção por chave primária da tabela X				
	1 milhão	10 milhões	50 milhões	100 milhões
1	0,02	0,02	0,03	0,03
2	0,01	0,02	0,01	0,01
3	0	0,02	0	0
4	0,02	0,02	0	0

5	0,01	0,01	0	0
6	0,01	0	0,01	0,03
7	0,01	0	0,02	0,01
8	0	0,03	0,02	0,01
9	0	0,01	0,01	0,01
10	0	0,01	0	0,01
	0,008	0,014	0,01	0,011

Tabela 43. Resultados obtidos da remoção de um registro por descrição da tabela Y no MySQL.

MySQL				
Teste de remoção por descrição da tabela Y				
	1 milhão	10 milhões	50 milhões	100 milhões
1	1,04	9,56	47,88	91,3
2	0,97	8,36	44,94	92,75
3	1,1	9,54	46,54	91,68
4	1,01	8,45	44,88	91,49
5	0,99	9,21	45,6	91,27
6	1,01	9,4	47,45	91,38
7	0,93	8,99	45,19	92,99
8	1,02	9,2	46,01	91,22
9	0,99	9,21	46,66	92,47
10	1,05	9,55	45	91,69
	1,011	9,147	46,015	91,824

Tabela 44. Resultados obtidos da remoção de um registro por valor numérico da tabela X no MySQL.

MySQL				
Teste de remoção por valor numérico da tabela X				
	1 milhão	10 milhões	50 milhões	100 milhões
1	0,78	9,14	42,64	92
2	0,94	14,76	61,08	91,5
3	0,88	8,67	44,13	91,38
4	0,94	8,81	45,15	89,64
5	0,88	8,84	43,9	88,3
6	0,94	8,23	44,7	89,32
7	0,96	9,04	44,77	88,83
8	0,85	9,07	44,3	88,23
9	0,86	8,84	43,44	89,43
10	0,83	8,5	42,8	92,3
	0,886	9,39	45,691	90,093

Tabela 45. Resultados obtidos da remoção de um registro por chave primária da tabela Y no PostgreSQL.

PostgreSQL				
Teste de remoção por chave primária da tabela Y				
	1 milhão	10 milhões	50 milhões	100 milhões
1	0,11	7,4	34,68	55,98
2	0,67	6,5	31,9	60,07
3	0,83	6,34	29,23	56,67
4	0,76	6,38	30,42	58,14
5	0,5	6,99	31,45	59,4
6	0,3	6,74	29,68	60,1
7	0,34	6,31	31,84	59,19
8	0,51	7,37	31,69	58,9
9	0,45	6,59	30,64	60,3
10	0,2	7,2	28,9	59,5
	0,467	6,782	31,043	58,825

Tabela 46. Resultados obtidos da remoção de um registro por chave primária da tabela X no PostgreSQL.

PostgreSQL				
Teste de remoção por chave primária da tabela X				
	1 milhão	10 milhões	50 milhões	100 milhões
1	0,03	0,04	0,04	0,07
2	0,04	0,02	0,02	0,03
3	0,02	0,02	0,02	0,02
4	0,02	0,02	0,03	0,03
5	0,02	0,03	0,04	0,02
6	0,03	0,04	0,02	0,02
7	0,02	0,03	0,02	0,02
8	0,04	0,02	0,03	0,02
9	0,03	0,02	0,04	0,02
10	0,02	0,02	0,03	0,02
	0,027	0,026	0,029	0,027

Tabela 47. Resultados obtidos da remoção de um registro por descrição da tabela Y no PostgreSQL.

PostgreSQL				
Teste de remoção por descrição da tabela Y				
	1 milhão	10 milhões	50 milhões	100 milhões
1	0,79	8,06	62,43	124,94
2	0,88	7,42	61,99	126,13
3	0,84	7,54	62,09	132,19

4	0,77	7,35	61,77	123,89
5	0,78	7,93	62,93	128,42
6	0,8	7,5	62,37	124,7
7	0,82	7,49	62,4	123,93
8	0,75	7,37	62,41	126,32
9	0,8	7,62	62,5	126,12
10	0,8	8,01	62,8	126,93
	0,803	7,629	62,369	126,357

Tabela 48. Resultados obtidos da remoção de um registro por valor numérico da tabela X no PostgreSQL.

PostgreSQL				
Teste de remoção por valor numérico da tabela X				
	1 milhão	10 milhões	50 milhões	100 milhões
1	0,72	6,54	29,61	57,37
2	0,71	6,06	30,05	61,22
3	0,78	6,27	30,61	61,82
4	0,73	6,72	30,21	58,03
5	0,74	6,29	30,38	60,43
6	0,75	6,38	30,41	59,59
7	0,72	6,26	30,5	60,03
8	0,74	6,09	30,6	60,4
9	0,75	6,17	30,73	60,73
10	0,71	6,28	30,63	60,51
	0,735	6,306	30,373	60,013

As tabelas 37 e 38 representam os resultados da bateria de testes de exclusão de um registro por identificador de documento, das collections X e Y respectivamente no MongoDB, as tabelas 39 e 40 apresentam os resultados de exclusão de um registro pelo campo “col_descricao” da collection Y e “col_decimal” da collection X no MongoDB.

As tabelas 41 e 42 representam os resultados da bateria de testes de exclusão de um registro por chave primária das tabelas Y e X do MySQL, as tabelas 43 e 44 apresentam os resultados de exclusão de um registro pelos campos “col_descricao” e “col_decimal” das tabelas X e Y respectivamente no MySQL.

As tabelas 45 e 46 apresentam os resultados da bateria de testes de exclusão de um registro por chave primária do PostgreSQL, as tabelas 46 e 47 apresentam os resultados da bateria de testes de exclusão de um registro no PostgreSQL seguindo o mesmo modelo do MySQL.

Todos os resultados são em segundos e o último campo das tabelas são as médias dos testes executados.

CAPÍTULO 6 - CONCLUSÃO

A utilização dos bancos de dados NoSQL veem se tornando uma saída para aplicações que gerenciam um grande fluxo de dados, seu modelo não estruturado faz com que uma grande massa de dados possa ser armazenada sem muitos problemas. Embora os bancos de dados NoSQL veem crescendo no mercado o modelo relacional ainda se mostra muito eficiente em seu desempenho.

Todos os testes realizados neste trabalho foram feitos no CentOS, um sistema operacional baseado no Linux, leve e de fácil instalação. O software utilizado para virtualizar os ambientes de teste foi o VMware Player, uma distribuição gratuita, de fácil instalação e utilização.

O objetivo deste trabalho é comparar o desempenho do MySQL, PostgreSQL e MongoDB utilizando operações de inserção, busca, alteração e remoção de registros. Com sua conclusão, fica evidente que embora o NoSQL tenha sido criado para mitigar alguns problemas de performance, o modelo relacional ainda sim é um modelo com desempenho satisfatório, e em algumas ocasiões até superior, levando em conta o cenário estabelecido e os bancos de dados escolhidos.

Nos testes de inserção ou escrita, o MongoDB teve um desempenho ruim comparado com o MySQL e o PostgreSQL. O MongoDB demorou cerca de vinte vezes a mais para realizar as operações de inserção do que o PostgreSQL, que teve o melhor desempenho nos testes de inserção. Isso devido o MongoDB criar documentos a cada registro inserido.

Já nos testes de busca o MongoDB superou os demais e teve o melhor desempenho, o MySQL manteve um bom desempenho, mantendo regularidade no desempenho em todos os testes. Os testes de busca, principalmente nos testes de busca complexa utilizando INNER JOIN se tornaram um gargalo para os bancos de dados relacionais, a partir dos cinquenta milhões de registros os bancos de dados precisaram de mais memória para executar as buscas, e em alguns casos não conseguiram. O MongoDB não teve nenhum problema de memória, mostrando grande vantagem aos demais nos testes de busca.

Nos testes de alteração de dados o MySQL teve o melhor desempenho seguido do MongoDB que teve um bom desempenho mas oscilou quando foi necessário realizar alterações por descrição ou valor numérico. Já o PostgreSQL teve um desempenho modesto.

Nos testes de remoção o MongoDB novamente oscilou entre o melhor e o pior desempenho, isso porque o MongoDB trabalha muito bem utilizando seu identificador de documentos, e não muito bem quando é preciso realizar operações por texto ou números, caso

contrário do PostgreSQL que trabalha melhor quando é necessário realizar operações por texto. O MySQL novamente foi o banco de dados que manteve melhor estabilidade.

Pode-se concluir também que o MongoDB trabalha muito bem com seu identificador de documento, que é equivalente a chave primária no modelo relacional, nos testes que foram necessários a manipulação de dados utilizando esse identificador o MongoDB teve o desempenho superior aos bancos de dados relacionais.

Vale ressaltar que nenhum tipo de otimização foi feita para a realização dos testes, o que pode ter influenciado no desempenho dos bancos de dados testados. Desta forma destaca-se que o MongoDB teve o melhor desempenho em operações de busca, o MySQL foi o banco de dados que teve a melhor estabilidade nos testes e o PostgreSQL teve o melhor desempenho nos testes de inserção de dados.

REFERÊNCIAS BIBLIOGRÁFICAS

AMORIM, Dinani Gomes; MOURA, Aristoteles Lamartine Teles. **Big data:** O impacto e sua funcionalidade na sociedade tecnológica. Revista Opara: Ciências Contemporâneas Aplicadas, v. 4, p. 57-70, abr. 2014.

ANDRADE, A. D. **Otimização de consultas de aplicações T-SQL em ambiente SQL Server 2000.** Disponível em <http://disciplinas.dcc.ufba.br/pub/MATA67/TrabalhosSemestre20051/Monografia_Luciano_Andrade.pdf>. Acesso 30/04/2014.

BALDO, S. M.; MACÁRIO, C. G. N. **O modelo relacional.** Disponível em <http://www.ic.unicamp.br/~geovane/mo410-091/Ch03-RM-Resumo.pdf>. Acesso em 20/02/2014.

BANKER, Kyle. **MongoDB in Action**, Manning Publications Co., 2012.

BASTOS, Aderson; et al. **Base de conhecimento em teste de software**. 3ª ed. São Paulo, Martins Fontes, 263p. 2012.

BITTENCOURT, Rogério Gonçalves. **Aspectos básicos de banco de dados**. Disponível em <http://www.marilia.unesp.br/Home/Instituicao/Docentes/EdbertoFerneda/BD%20-%20Aspectos%20Basicos.pdf>. Acesso 01/09/2014.

CARNEIRO, Felipe Cauê Fraga; MIGUEL, Sérgio Barriviera. **Repositório de dados relacional ou NoSQL?** Revista Java Magazine, v. 114, abr. 2013.

CHAGAS, Camila Ayub de Barros. **Teste como parcela no processo de qualidade de software**. Disponível em <http://www.avm.edu.br/docpdf/monografias_publicadas/k210758.pdf>. Acesso 04/10/2014.

COSTA, E. R. **Bancos de dados relacionais.** Disponível em <http://www.fatecsp.br/dti/tcc/tcc0025.pdf>. Acesso 24/04/2014.

COSTA, Leandro Teodoro. **Conjunto de características para teste de desempenho:** Uma visão a partir de ferramentas. Disponível em <http://repositorio.pucrs.br/dspace/bitstream/10923/1615/1/000440236-Texto%2bCompleto-0.pdf>. Acesso 14/10/2014.

DATE, C. J**. Introdução a sistemas de banco de dados, 8ª ed.,** Campus - RJ Inativar, 2004.

DELAMARO, Márcio Eduardo; JINO, Mario; MALDONADO, José Carlos. **Introdução ao teste de software**. Rio de Janeiro, Elsevier, 394p. 2007.

DIANA, Maurício 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: IX Workshop de Teses e Dissertações em Banco de Dados, 8, 2010. **Workshop...** Belo Horizonte: SBC, 2010. p. 68-75.

DIOGO, A. L. G.; SANTOS, D. R.; FERREIRA, F. A. P.; SANTOS, T. F. **Arquitetura/Funcionamento interno de um sistema gerenciador de banco de dados.** Disponível em <http://pt.slideshare.net/diogocbj/arquitetura-e-sgbd-de-um-banco-de-dados>. Acesso em 23/04/2014.

Documentação do PostgreSQL 8.0.0. Rio de Janeiro, 2007. Disponível em <http://ftp.unicamp.br/pub/apoio/postgresql/pgdocptbr800-1.2.pdf>. Acesso 06/10/2014.

ELMASRI, Ramez; NAVATHE, Shamkant. **Sistemas de Banco de Dados, 6ª ed.**, Pearson Addison Wesley, 2011.

FERREIRA, E. R.; JÚNIOR, S. M. T. **Análise de desempenho de bancos de dados.** Disponível em <http://www.unipac.br/site/bb/tcc/tcc-15ee06c022b6b866f2815b76757c667f.pdf>. Acesso 23/07/2014.

FIGUEIREDO, Josiel Maimone de; et al. **Bancos de dados noSQL:** Conceitos, ferramentas, linguagens e estudos de casos no contexto de Big data. Disponível em <http://data.ime.usp.br/sbbd2012/artigos/pdfs/sbbd_min_01.pdf>. Acesso 13/10/2014.

FRIESS, Ivan Isaías. **Análise de bancos de dados NoSQL e desenvolvimento de uma aplicação.** Santa Maria: UFSM, 2013.

KAMADA, C. M. **Bancos de dados relacionais**. Disponível em <http://pt.slideshare.net/carloscastro2108/bancos-de-dados-relacionais>. Acesso 26/04/2014.

KOKAY, Marília Cavalcante. **Banco de dados NoSQL: Um novo paradigma.** Disponível em <http://www.devmedia.com.br/websys.5/webreader.asp?cat=2&artigo=4773&revista=sqlmagazine_102#a-4773>. Acesso 13/05/2014.

LANNI, Vinicius. **Introdução aos bancos de dados NoSQL**. Disponível em <http://www.devmedia.com.br/introducao-aos-bancos-de-dados-nosql/26044>. Acesso 13/05/2014.

LENNON, Joe. **Explore o MongoDB**. Disponível em http://www.ibm.com/developerworks/br/library/os-mongodb4/>. Acesso 08/10/2014.

LEWIS, W. E. **Software testing continuous quality improvement**. 3ª ed. Boca Raton, Auerbach Publication, 688p. 2000.

LÓSCIO, Bernadette Farias; OLIVEIRA, Hélio Rodrigues; PONTES, Jonas César de Sousa. **NoSQL no desenvolvimento de aplicações web colaborativas**. Disponível em <http://www.addlabs.uff.br/sbsc_site/SBSC2011_NoSQL.pdf>. Acesso 14/05/2014.

MALDONADO, José Carlos; ROCHA, Ana Regina Cavalcanti; WEBER, Kival Chaves. **Qualidade de software: teoria e prática**. Prentice Hall, São Paulo, 303p. 2001.

MATTEUSSI, Kassiano José. **Protótipo de interface web com PHP para gerenciamento de banco de dados couchDB**. Disponível em <http://eltonminetto.net/docs/monografia_kassiano.pdf>. Acesso 07/10/2014.

MATTOSO, M. **Introdução a banco de dados.** Disponível em <http://www.cos.ufrj.br/~marta/BdRel.pdf>. Acesso em 23/04/2014.

MODESTO, Lisandro Rogério. **Teste funcional baseado em digramas da UML**. 2006. 114 f. Dissertação (Mestrado em Ciência da Computação) – Fundação de Ensino Euripedes Soares da Rocha, 2006.

NETO, Arilo Cláudio Dias. **Introdução a Teste de Software**. Disponível em <http://www.isacaguiar.com.br/arquivos/academico/engenharia/iii/ART-03.pdf>. Acesso 27/09/2014.

NETO, Pedro de Alcântara dos Santos; SANTOS, Ismayle de Sousa. **Automação de teste de desempenho e estresse com o JMeter**. Disponível em <http://www.ufpi.br/subsiteFiles/pasn/arquivos/files/artigoJMeter(1).pdf>. Acesso 14/10/2014.

ODWAZNY, Paulo Fernando. **Protótipo para administração de SGBD PostgreSQL**. Disponível em <http://dsc.inf.furb.br/arquivos/tccs/monografias/2003-2paulofernandoodwaznyvf.pdf>. Acesso 06/10/2014.

OLIVERA, Kaui Aires. **Os conceitos primordiais em banco de dados**. Disponível em <http://www.devmedia.com.br/os-conceitos-primordiais-em-banco-de-dados/3173>. Acesso 30/04/2014.

PALMEIRA, T. V. V. **Arquitetura de um SGBD.** Disponível em <http://www.devmedia.com.br/arquitetura-de-um-sgbd/25007>. Acesso em 23/04/2014.

PICHILIANI, Mauro. **Comparação de desempenho entre bancos SQL e NoSQL**. Revista Java Magazine, v. 109, mar. 2013.

PORCELLI, Alexandre. **O que é NoSQL?** Revista Java Magazine, v. 87, jan. 2011.

PRESSMAN, Roger S. **Engenharia de software:** Uma abordagem profissional, 7ª ed., AMGH Editora Ltda, 2011.

PRITCHETT, Dan. **BASE:** An acid alternative. Disponível em <http://delivery.acm.org/10.1145/1400000/1394128/p48-pritchett.pdf?ip=200.211.20.27&id=1394128&acc=OPEN&key=4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E6D218144511F3437&CFID=581885570&CFTOKEN=46433493&__acm__=1412874158_9044ff122760a1c65bad0c5f2143ab41>. Acesso 09/10/2014.

REZENDE, R. **Conceitos fundamentais de banco de dados.** Disponível em <http://www.devmedia.com.br/conceitos-fundamentais-de-banco-de-dados/1649>. Acesso em 15/03/2014.

SANCHES, A. R. **Arquiteturas de banco de bados.** Disponível em <http://www.ime.usp.br/~andrers/aulas/bd2005-1/aula4.html>. Acesso 23/04/2014.

SILVA, Claudio Ribeiro da. **Banco de Dados.** 1996.

SILVA, Fernando Rodrigues Da. **Testes de software - Níveis de testes.** Disponível em <http://www.devmedia.com.br/testes-de-software-niveis-de-testes/22282>. Acesso em 13/10/2014.

STRAUCH, Christof. **NoSQL Databases.** Disponível em <http://oak.cs.ucla.edu/cs144/handouts/nosqldbs.pdf >. Acesso 07/10/2014.