INSTITUTO FEDERAL DE MINAS GERAIS
CAMPUS SÃO JOÃO EVANGELISTA

ANDRÉ SAYÃO PINTO DE BRITO

ANÁLISE DE DESEMPENHO ENTRE OS PRINCIPAIS SGBDS DO MERCADO

SÃO JOÃO EVANGELISTA
2021

ANDRÉ SAYÃO PINTO DE BRITO

ANÁLISE DE DESEMPENHO ENTRE OS PRINCIPAIS SGBDS DO MERCADO

Trabalho de conclusão de curso apresentado ao Instituto Federal de Minas Gerais - *Campus* São João Evangelista como exigência parcial para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Me. Italo Magno Pereira
Coorientador: Prof. Me. Dayler Vinicius Miranda Alves

SÃO JOÃO EVANGELISTA
2021

B862a Brito, André Sayão Pinto de.
Análise de desempenho entre os principais SGBDS do mercado / André Sayão Pinto de Brito. – 2021.

54 f. : il

Orientador: Italo Magno Pereira
Co-orientador: Dayler Vinicius Miranda Alves

Trabalho de Conclusão de Curso (TCC) – Instituto Federal de Ciência e Tecnologia de Minas Gerais – *Campus* São João Evangelista

1.Banco de dados. 2. Desempenho. 3. MongoDB. 4. SGBDR. 5. JMeter. I.Título.

CDD 005.756

Ficha Catalográfica – Bibliotecária Nirley Dias Leandro CRB 6 2394

ANDRÉ SAYÃO PINTO DE BRITO

ANÁLISE DE DESEMPENHO ENTRE OS PRINCIPAIS SGBDS DO MERCADO

Trabalho de conclusão de curso apresentado ao Instituto Federal de Minas Gerais - *Campus* São João Evangelista como exigência parcial para obtenção do título de Bacharel em Sistemas de Informação.

Aprovado em: 14/07/2021 pela banca examinadora:

Orientador: Prof. Me. Italo Magno Pereira
Instituto Federal de Minas Gerais – *Campus* São João Evangelista

Coorientador Prof. Me. Dayler Vinicius Miranda Alves
Instituto Federal de Minas Gerais – *Campus* São João Evangelista

Convidado: Prof. Me. Rosinei Soares de Figueiredo
Instituto Federal de Minas Gerais – *Campus* São João Evangelista

RESUMO

Este trabalho tem como objetivo realizar uma análise de desempenho comparativa entre cinco SGBDs de grande influência no mercado em suas versões *open source*. Os SGBDs selecionados foram Oracle, MySQL, SQLServer, PostgreSQL e MongoDB e foram submetidos a testes de carga e *stress* em diferentes situações a fim de que se possa apontar qual se saiu melhor no respectivo cenário. A ferramenta utilizada para fornecer os resultados foi o JMeter, ferramenta Java que se comunica com os bancos de dados pelo driver JDBC e é também bastante utilizada em testes de aplicações web. Para povoar os bancos foi criado um conjunto de dados em cada um deles, de forma que foi possível inserir as informações com o mesmo padrão em todos. Os dados fornecidos pelo JMeter foram apresentados em forma de gráficos e tabelas para uma melhor visualização e análise. No fim pode-se perceber destaque para alguns SGBDs em determinadas operações como o desempenho do MongoDB em consultas *full scan* e seu desempenho em um ambiente multi-usuário. O SQLServer apresentou bastante rapidez em operações como *update* e *delete*. O Oracle teve dificuldades em lidar com um grande número de registros e usuários simultâneos.

Palavras-chave: Banco de dados. Desempenho. MongoDB. SGBDR. JMeter.

ABSTRACT

This work aims to obtain a comparative performance analysis between five SGBDs of great influence in the market in their open source version. The DBMS selected were Oracle, MySQL, SQLServer, PostgreSQL and MongoDB and were subjected to load and stress tests in different situations in order to identify which one performed better in a given scenario. The tool used to provide the results of JMeter, a Java tool that communicates with databases through the JDBC driver and is also widely used in testing web applications. To populate the bases, a set of data was created in each one of them, so that it was possible to insert the information with the same pattern in all of them. The data provided by JMeter were presented in the form of graphs and tables for better visualization and analysis. In the end, some DBMSs stand out in functions such as MongoDB's performance in full scan queries and its performance in a multi-user environment. SQLServer was very fast in operations such as update and delete. Oracle struggled to handle a large number of records and concurrent users.

Key-words: Data base. Performance. MongoDB. RDBMS. JMeter.

LISTA DE FIGURAS

Figura 1- Representação do MER 17
Figura 2- Representação lógica de um banco de dados relacional 18
Figura 3- Representação física de dados 19
Figura 4- Representação da coleção criada no MongoDB 33
Figura 5- Plano de teste no JMeter 34
Figura 6- Configuração de conexão JDBC 35
Figura 7- Configuração de conexão do banco de dados 35

LISTA DE QUADROS

Quadro 1- Dados da tabela criada.....33
Quadro 2- *Insert* com 1.000 registros.....36
Quadro 3 - *Update* com 1.000 registros 37
Quadro 4 - *Select* com 1.000 registros.....37
Quadro 5 - *Delete* com 1.000 registros.....37
Quadro 6 - *Insert* com 10.000 registros.....38
Quadro 7 - *Update* com 10.000 registros 39
Quadro 8 - *Select* com 10.000 registros.....39
Quadro 9 - *Delete* com 10.000 registros.....40
Quadro 10 - *Insert* com 100.000 registros.....41
Quadro 11 - *Update* com 100.000 registros 41
Quadro 12 - *Select* com 100.000 registros.....42
Quadro 13 - *Delete* com 100.000 registros.....42
Quadro 14 - Teste de *stress* com 50 usuários.....44
Quadro 15 - Teste de *stress* com 100 usuários.....44
Quadro 16 - Teste de *stress* com 150 usuários.....45
Quadro 17 - Teste de *stress* com 200 usuários.....45

LISTA DE GRÁFICOS

Gráfico 1 - Tempos das operações com 1.000 registros 38
Gráfico 2 - Tempos das operações com 10.000 registros 40
Gráfico 3 - Tempos das operações com 100.000 registros 43
Gráfico 4 - Tempo médio de usuário 46
Gráfico 5 - % de erro 46
Gráfico 6 - Utilização de CPU 47

LISTA DE ABREVIATURAS E SIGLAS

ACID - *Atomicity Consistency Isolation Durability*

AS³AP - ANSI *SQL Standard Scalable and Portable*

CPU - *Central Processing Unit*

DCL - *Data Control Language*

DDL - *Data definition language*

DML - *Data Manipulation Language*

DTL - *Data Transaction Language*

JDBC - *Java Database Connectivity*

MER - Modelo Entidade-Relacionamento

MLD - Modelo Lógico de Dados

NoSQL - *Not Only* SQL

OSDB - *Open Source Database Benchmark*

RAC - *Real Application Cluster*

SGBD - Sistema Gerenciador de Banco de Dados

SGBDR - Sistema Gerenciador de Banco de Dados Relacional

SQL - *Structured Query Language*

TCP-C - *Transaction Process Council*

TI - Tecnologia da Informação

URL - *Uniform Resource Location*

SUMÁRIO

1. INTRODUÇÃO **12**

2. REFERENCIAL TEÓRICO **15**

2.1 Software **15**

2.2 Banco de dados **15**

2.3 Modelagem de dados **16**

2.3.1 Modelo conceitual **17**

2.3.2 Modelo lógico **18**

2.3.3 Modelo físico **19**

2.4 Bancos de dados relacionais **19**

2.4.1 Linguagem SQL **21**

2.5 Bancos de dados não relacionais **22**

2.6.1 Oracle **23**

2.6.2 MySQL **24**

2.6.3 PostgreSQL **24**

2.6.4 SQL Server **25**

2.6.5 MongoDB **25**

2.7 Benchmarking **26**

2.8 JMeter **27**

2.9 Trabalhos relacionados **27**

3. METODOLOGIA **30**

3.1 Natureza da pesquisa **30**

3.2 Materiais **30**

3.3 Métodos **31**

3.3.1 Carga **32**

3.3.2 Stress **32**

3.3.3 Dados Inseridos **32**

3.3.4 Configuração do JMeter **34**

4. RESULTADOS E DISCUSSÕES **36**

4.1 Testes de carga **36**

4.1.1 1.000 registros **36**

4.1.2 10.000 registros **38**

4.1.3 100.000 registros **41**

4.2 Testes de stress **43**
4.1.1 50 usuários **44**
4.2.2 100 usuários **44**
4.2.3 150 usuários **45**
4.2.4 200 usuários **45**
5. CONSIDERAÇÕES FINAIS **48**
REFERÊNCIAS **51**

1. INTRODUÇÃO

A Tecnologia da Informação (TI) oferece recursos computacionais e tecnológicos para o gerenciamento e manutenção de informações dentro de organizações. Com o passar dos anos foram surgindo novas soluções e os sistemas de informação vem se tornando cada vez mais sofisticados, propondo mudanças nos processos, estrutura e estratégia de negócios (BAZZOTTI; GARCIA, 2006).

Somente no ano de 2009, foram investidos nos Estados Unidos quase um trilhão de dólares em *hardware* e *software*, além de outros 275 bilhões em consultoria e serviços de gestão (LAUDON; LAUDON, 2011). Estudos da Gartner apontam crescimentos ainda mais expressivos nos últimos anos. Em 2017 foram projetados investimentos no mercado mundial de serviços de TI no total de 3,5 trilhões de dólares por todo o mundo, um aumento de 1,4% em relação à 2016 (DEANS, 2017). Já em 2019 a Gartner prevê que os gastos mundiais com TI totalizarão 3,79 trilhões de dólares, representando um aumento de 1,1% em relação aos resultados de 2018 (TIINSIDE, 2019).

"A visão da TI como arma estratégica competitiva tem sido discutida e enfatizada, pois não só sustenta as operações de negócio existentes, mas também permite que se viabilizem novas estratégias empresariais" (LAURINDO et al., 2001). A competitividade e demanda do mercado vem crescendo exponencialmente e exigindo das organizações maior aperfeiçoamento quanto à gestão de sua informação. Os bancos de dados tem o fundamental papel no dia a dia das empresas de gerir e acessar registros de fatos ocorridos no exercício de suas atividades operacionais com eficiência e segurança.

Elmasri e Navathe (2011) definem banco de dados como uma coleção de dados relacionados, ou seja, fatos que podem ser registrados e possuem significado implícito. Sendo assim, uma quantidade aleatória de dados sem lógica ou coerência não pode ser classificada como um banco de dados.

De acordo com Silberschatz, Korth e Sundarshan (2016), os bancos de dados como são apresentados hoje surgiram em resposta aos métodos iniciais de gestão dos dados de forma computacional por volta da década de 60. Os dados eram armazenados nos arquivos do sistema operacional formando um sistema de processamento de arquivos. A manutenção das informações nesse tipo de aplicação sofria uma série de inconvenientes, como redundância e inconsistência nos dados, dificuldade no acesso dos dados, isolamento de dados, problemas de integridade, atomicidade, acesso concorrente, segurança, dentre outros.

Nos dias de hoje o acesso e compartilhamento de informações entre as pessoas é

bastante ágil e prático muito devido aos bancos de dados, que realizam um armazenamento de forma otimizada com segurança e organização, além de permitirem também o acesso de forma remota como, por exemplo, o acesso a um banco de dados de uma organização em um servidor em nuvem via internet (ROCHA; DIAS, 2015).

Para gerir grandes massas de informação e definir estruturas para armazená-las, além de garantir segurança para com os dados armazenados, foram projetados os Sistemas Gerenciadores de Banco de Dados (SGBDs) (SILBERSCHATZ; KORTH; SUNDARSHAN, 2016). “O SGBD é um sistema de *software* de uso geral que facilita o processo de definição, construção, manipulação e compartilhamento de banco de dados entre diversos usuários e aplicações.” (ELMASRI; NAVATHE, 2011, p.4).

Ao longo do tempo surgiram diversos SGBDs com diferentes características, funcionalidades e preços que variam de acordo com o seu fabricante. Cabe ao administrador do banco de dados definir qual é o mais indicado entre os disponíveis no mercado para melhor gerenciar as informações de sua empresa de acordo com suas características e que atenda melhor a sua necessidade (MAIELLO, 2016).

Em ambientes computacionais, *benchmarks* são utilizados para mensurar o desempenho de diversos componentes com o objetivo de auxiliar na escolha do sistema ou componente e apontar possíveis melhorias (PIRES; NASCIMENTO; SALGADO, 2009). Por exemplo, na escolha de um *notebook* ou celular, é possível utilizar *benchmarks* para comparar as pontuações de determinado componente afim de tomar melhores decisões na hora da compra.

O objetivo geral deste trabalho foi avaliar o desempenho de SGBDs com base em métricas de eficiência e eficácia. Observou-se o comportamento dos SGBDs quanto a diferentes proporções de entrada, processamento, leitura, pesquisa e exclusão de dados.

Para alcançar o objetivo geral ficaram definidos os seguintes objetivos específicos:

a) estabelecer as métricas a serem utilizadas no trabalho;

b) coletar e avaliar resultados de *benchmark* entre os SGBDs;

c) comparar os SGBDs estudados com base em seus resultados;

d) apurar o sistema com melhores resultados através de análise quantitativa.

Visto que os bancos de dados são de grande importância para manter as informações de uma organização, se torna desejável e vantajoso implantar um SGBD que apresente um melhor desempenho ao manusear esses dados, a fim de que os processos dentro da empresa se tornem mais eficazes e os problemas consequentemente diminuam. Um

benchmark entre esses sistemas apresentará resultados que auxiliarão no processo de determinação e implantação do SGBD (FERREIRA; JÚNIOR, 2016).

Como resultados deste trabalho foi possível apontar com base nos testes que foram submetidos os SGBDs, o sistema mais eficiente em certa operação de acordo com as métricas estabelecidas. Essas informações fornecem resultados baseados nos testes realizados a fim de auxiliar possíveis usuários na seleção de um SGBD que utilizará melhor os recursos do sistema.

Este trabalho está organizado em cinco capítulos. Na introdução é apresentado o problema, os objetivos do trabalho juntamente com sua justificativa. O segundo contempla um referencial teórico, no qual são abordados conceitos ligados ao tema, e trabalhos correlatos. O terceiro descreve a metodologia, onde são apresentadas as diretrizes do estudo, sua natureza além dos materiais e métodos utilizados. No quarto são explicitados os resultados dos testes do estudo. No quinto, por fim, as considerações finais.

2. REFERENCIAL TEÓRICO

Neste capítulo são apresentados conceitos e informações sobre o estudo, tendo como embasamento obras de autores relevantes que se posicionam acerca do tema em questão, além de descrições dos *softwares* e das demais tecnologias utilizadas. O presente capítulo se subdivide em tópicos que abordarão conceitos sobre *software*, banco de dados, modelagem de dados, *benchmark* e, por fim, os trabalhos correlatos.

2.1 Software

Segundo Pressman e Maxim (2016), um *software* é um conjunto de instruções e quando executadas, produzem a função e o desempenho desejado. Sommerville (2011) define software como programas de computador e documentação associada. Produtos de software podem ser desenvolvidos para um cliente em específico ou para o mercado em geral.

A esse respeito é importante apontar que:

> O software distribui o produto mais importante da nossa era - a informação. Ele transforma dados pessoais (por exemplo transações financeiras de um indivíduo) de modo que possam ser mais úteis em determinado contexto; gerencia informações comerciais para aumentar a competitividade; fornece um portal para redes mundiais de informação (Internet) e os meios para obter informações sob todas as suas formas (PRESSMAN; MAXIM, 2016, p.3).

Neste trabalho são utilizados *softwares* de banco de dados conhecidos como servidores, juntamente às suas ferramentas administrativas, os clientes, que auxiliam na utilização dos mesmos, além do *software* para a realização do *benchmark* próprio para análise de sistemas computacionais.

2.2 Banco de dados

Conforme Date (2004), um banco de dados pode ser comparado com um armário de livros, sendo um recipiente para uma coleção de arquivos de dados computadorizados. Neles é possível realizar diversas operações, tais como: inserir, alterar, buscar, mover ou deletar informações. Ou seja, é basicamente um repositório ou recipiente que armazena informações que estejam relacionadas entre si no contexto em que foram ali introduzidas. Um dicionário ou uma agenda telefônica podem ser tidos como exemplo de banco de dados nesse contexto.

Elmasri e Navathe (2011) apontam que um banco de dados representa algum aspecto do mundo real sendo, às vezes, chamado de minimundo ou universo de discurso. As mudanças ocorridas no minimundo são refletidas no banco de dados. O banco de dados é projetado e construído para atender uma necessidade específica e possui um número específico de usuários e algumas aplicações previamente concebidas de interesse desses usuários. “Para manter grandes repositórios compartilhados de dados, ou seja, para manter banco de dados, são usados sistemas de gerência de banco de dados.” (HEUSER, 2009, p. 23).

Um SGBD é um *software* que incorpora as funções de definição, recuperação e alteração de dados em um banco de dados (HEUSER, 2009). Como abordado anteriormente, o uso dos SGBDs proporciona muitas vantagens na manipulação dos dados de determinada organização e no mercado, atualmente, existem diversas opções. Para essa análise serão levados em consideração cinco entre os principais, mais utilizados e conhecidos segundo *ranking* periódico da DB-Engines (2021).

O principal objetivo de um SGBD é proporcionar uma forma de armazenar e recuperar informações de um banco de dados de maneira conveniente e eficiente (SILBERSCHATZ; KORTH; SUNDARSHAN, 2016). De modo geral, o servidor de banco de dados gerencia de forma confiável uma grande quantidade de dados em um ambiente multiusuário para que os usuários possam acessar simultaneamente os mesmos dados. Um servidor de banco de dados também previne acesso não autorizado e fornece soluções eficientes para recuperação de falhas (ORACLE, 2019).

2.3 Modelagem de dados

Em um projeto de banco de dados, assim como em qualquer sistema, é importante criar uma representação do mesmo, com a finalidade de facilitar seu compreendimento e uso. Os modelos de dados auxiliam nesse processo de representação, evitando assim erros de programação e conflitos de entendimento.

Heuser (2009) afirma que um modelo de dados é uma descrição dos tipos de informações que estão armazenadas em um banco de dados. Elmasri e Navathe (2011) reforçam que os modelos de dados são utilizados para alcançar um nível de abstração de dados, ou seja, destacar os recursos essenciais para o melhor entendimento do banco e especificar o comportamento de uma aplicação de banco de dados. Isso permite que o projetista do banco detalhe um conjunto de operações válidas, definidas pelo usuário sobre os

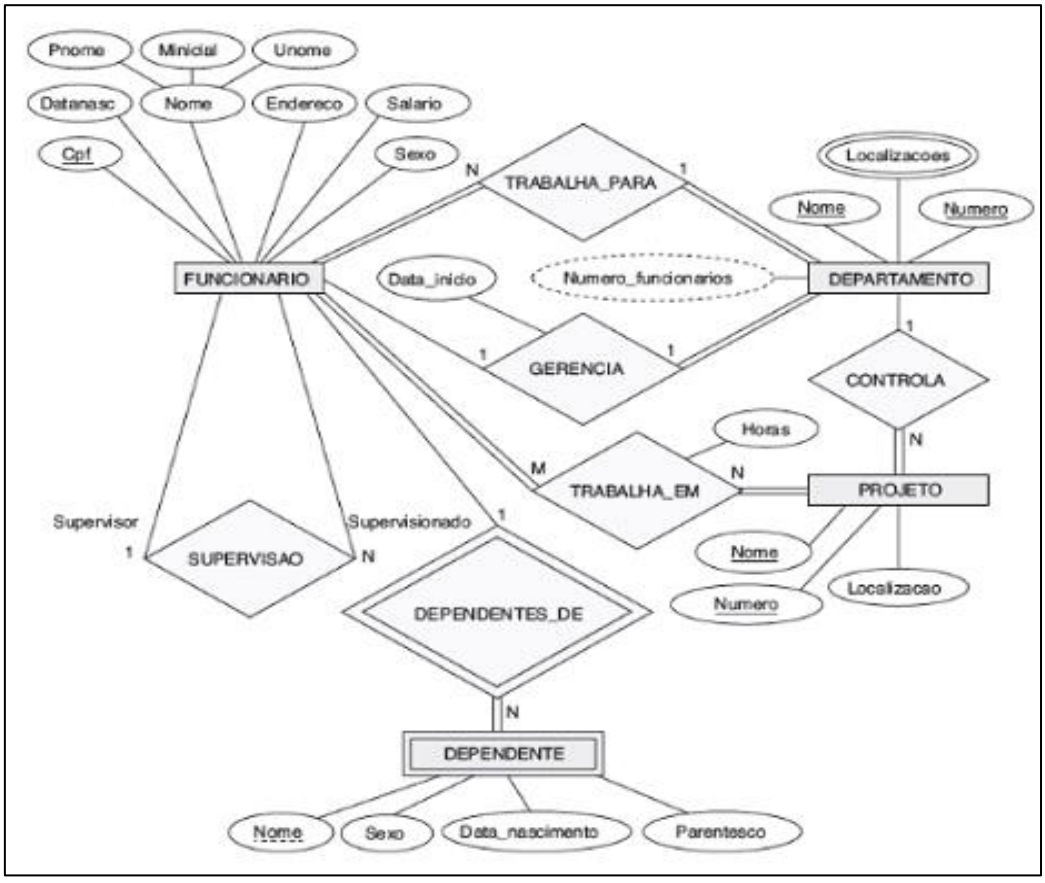
objetos do banco de dados. Um banco de dados pode ser modelado em diversos tipos de abstração de acordo com o modelador e com o usuário a quem será apresentado o modelo (HEUSER, 2009).

2.3.1 Modelo conceitual

Os modelos conceituais ou de alto nível, segundo Elmasri e Navathe (2011), fornecem conceitos mais próximos ao entendimento do usuário, ou seja, são usados para melhor entendimento do cliente. Para isso são utilizados diagramas e fluxogramas que representam as estruturas onde são armazenados os dados.

Na Figura 1 é mostrada a representação conceitual de um banco de dados relacional, o Modelo Entidade-Relacionamento (MER).

Figura 1- Representação do MER



Fonte: ELMASRI; NAVATHE, 2011.

2.3.2 Modelo lógico

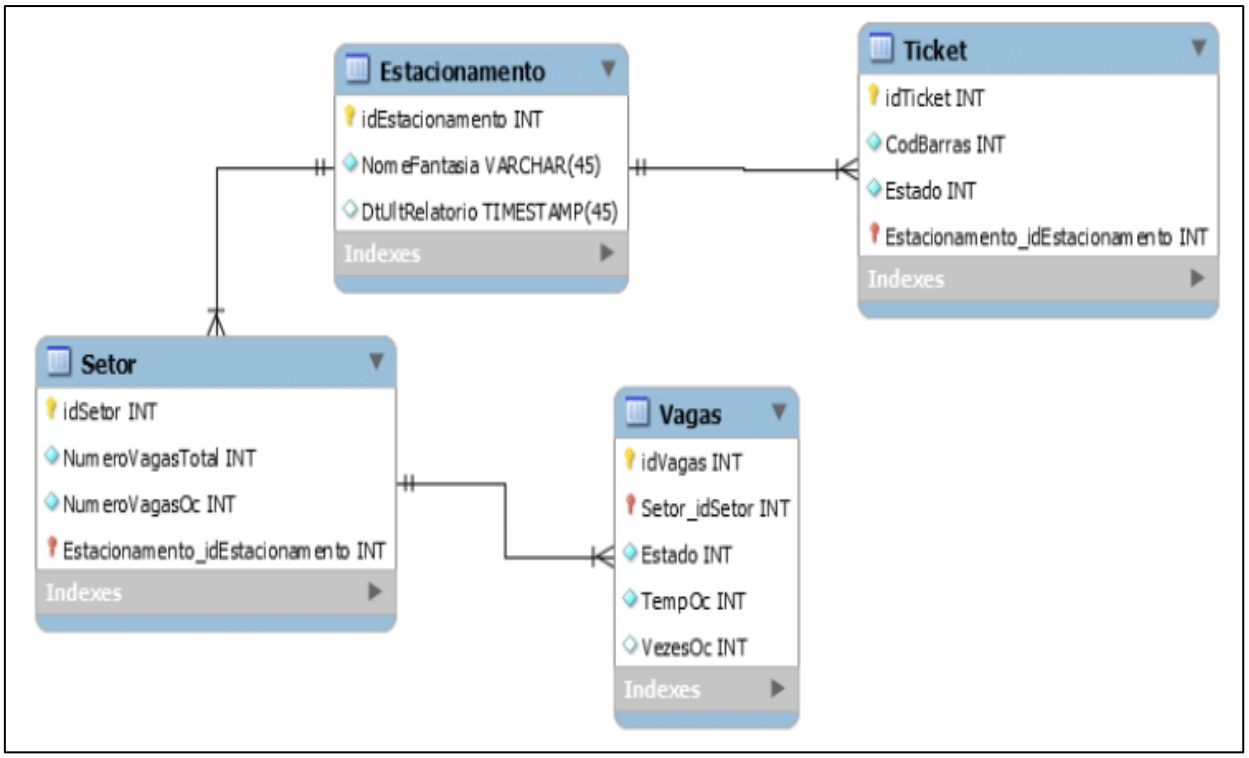
Cougo (2013) define o Modelo Lógico de Dados (MLD) como um modelo de dados em que os objetos, suas características e relacionamentos são representados de acordo com as regras de implementação impostos por algum tipo de tecnologia, independentemente dos dispositivos ou meios de armazenamento físico das estruturas de dados definidas por essa tecnologia.

O MLD deve ser elaborado respeitando e implementando conceitos tais como chaves de acesso, controle de chaves duplicadas, itens de repetição, normalização, ponteiros, *headers*, integridade referencial, entre outros. O modelo lógico é bastante voltado, à implementação sob o ponto de vista de sistemas de informação (COUGO, 2013).

Silberschatz, Sundarshan e Korth (2016) frisam que no nível lógico, cada registro é representado por uma definição de tipo e os programadores que utilizam das linguagens de programação, assim como também os administradores de bancos de dados, trabalham nesse nível de abstração.

A Figura 2 apresenta uma representação lógica de um banco de dados relacional.

Figura 2- Representação lógica de um banco de dados relacional



Fonte: BRITO; COUTINHO; CUNHA, 2016.

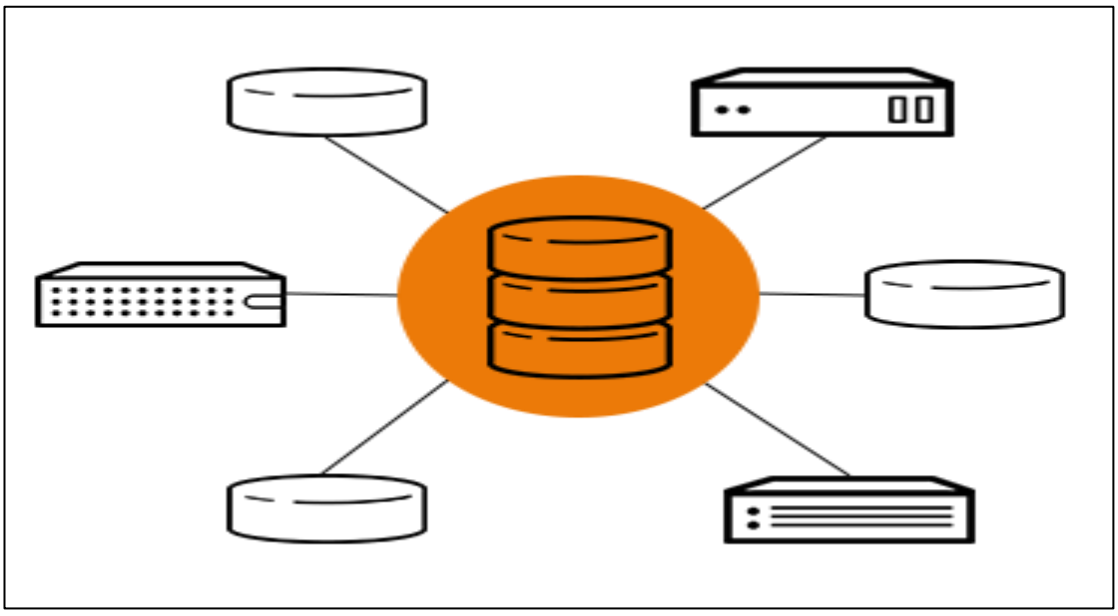
2.3.3 Modelo físico

Os modelos de dados de baixo nível, ou físicos, oferecem conceitos que descrevem detalhes de como os dados são armazenados no computador, em geral em discos magnéticos. Esses conceitos costumam ser voltados para especialistas de computadores, não para usuários finais. (ELMASRI; NAVATHE, 2011)

No nível físico, um registro, por exemplo, de entidades como instrutor, departamento ou aluno pode ser descrito como um bloco de localizações de armazenamento consecutivas. O compilador oculta esse nível de detalhe dos programadores, assim como o SGBD oculta muitos detalhes de armazenamento de nível mais baixo dos programadores de banco de dados (SILBERSCHATZ; KORTH; SUNDARSHAN, 2016).

A Figura 3 é uma representação física dos dados computadorizados, que armazenados em diferentes meios tem acesso ao SGBD e este pode manipulá-los.

Figura 3- Representação física de dados



Fonte: RED HAT, 2021.

2.4 Bancos de dados relacionais

Um banco de dados relacional se trata de uma coleção de uma ou mais tabelas, com nome único e atributos. Existe uma íntima relação entre o conceito tabela e o conceito matemático relação. Cada relação é uma tabela de valores e cada uma de suas linhas é uma coleção de valores de dados relacionados (SILBERSCHATZ, 2016).

“Os bancos de dados relacionais foram propostos originalmente para separar o armazenamento físico dos dados de sua representação conceitual e para fornecer uma base matemática para a representação e consulta dos dados” (ELMASRI; NAVATHE, 2011, p.15).

Algumas vantagens podem ser evidenciadas: tecnologia madura; mecanismo de persistência dominante no mercado; padrões SQL e *Java Database Connectivity* (JDBC) consolidados; extensa base de experiência de desenvolvedores. É importante também citar desvantagens como, por exemplo, o mapeamento de objetos para banco de dados que pode ser uma habilidade difícil, visto que as relações devem estar sempre muito bem definidas (DEVMEDIA, 2011).

Os Sistemas Gerenciadores de Bancos de Dados Relacionais (SGBDR) utilizam conceitos como entidades, atributos e relacionamentos. Uma entidade representa um objeto, como por exemplo, estudante. Um atributo representa uma propriedade dessa entidade, como por exemplo matrícula ou curso. Por fim um relacionamento representa uma associação entre duas ou mais entidades representadas, por exemplo, um relacionamento chamado cursa que relaciona as entidades estudante e disciplina.

O MER é o modelo conceitual mais popular e tradicional utilizado no projeto de banco de dados relacional. Como já abordado, é composto de três conceitos: entidades, relacionamentos e atributos.

As entidades, no MER, são representadas por um retângulo e representam algo no mundo real com existência independente podendo ser um objeto com existência física (estudante, professor, prédio), ou conceitual (curso, disciplina), por exemplo.

Quanto às entidades, Heuser (2009) estabelece que possuem atributos, ou seja, propriedades que os descrevem. São representadas no diagrama como uma elipse ligado à sua respectiva entidade. Os atributos podem ser classificados como simples, compostos, únicos ou multivalorados, e armazenados ou derivados. Atributos compostos podem ser desmembrados, por exemplo, endereço, onde há mais de um campo de informação, ao contrário de atributos simples que são indivisíveis. Um atributo multivalorado é um atributo que pode possuir mais de um valor, por exemplo, o atributo telefone de uma dada entidade pessoa, por exemplo e são representados no diagrama tendo um duplo contorno na sua forma. Um atributo derivado é um atributo que está relacionado com outro armazenado, por exemplo, os atributos idade e data de nascimento, em que se é possível obter a idade por meio da data de nascimento. Uma restrição das entidades é a chave, ou restrição de exclusividade, uma entidade pode possuir um ou mais dessas restrições e servem para representar cada instancia de uma entidade de maneira única, são representados no diagrama tendo seu nome sublinhado.

Ainda de acordo com Elmasri e Navathe (2011), os relacionamentos são exibidos no diagrama do MER como losangos que são conectados às entidades para representar um relacionamento entre duas ou mais entidades. O relacionamento tem grau de acordo com o número de entidades, por exemplo, se o relacionamento possui duas entidades participando, é denominado binário, no caso de três, ternário, e assim por diante.

A razão de cardinalidade para relacionamentos especifica o número máximo de instâncias de relacionamento que uma entidade pode participar. Por exemplo, um relacionamento chamado CURSA, que liga as entidades DISCIPLINA e ESTUDANTE tem razão de cardinalidade 1:N, ou seja, cada disciplina pode estar relacionada a qualquer número de estudantes, enquanto um estudante, neste caso, pode estar relacionado à apenas uma disciplina. As razões de cardinalidade possíveis para relacionamentos binários são 1:1, 1:N, N:1, M:N e esta cardinalidade é aplicada de acordo como funciona o universo de discurso, no exemplo da escola descrita.

Para manipular os comandos em um banco relacional utiliza-se de uma linguagem chamada SQL.

2.4.1 Linguagem SQL

Os SGBDs relacionais, trabalham com linguagens de manipulação de dados para que seja possível realizar as mais diversas operações no banco. A *Structured Query Language* (SQL) ou linguagem de consulta estruturada, na tradução, é a linguagem padrão para se lidar com banco de dados relacionais e é aceita pela maioria dos produtos de banco de dados disponíveis no mercado. Foi desenvolvida originalmente na IBM Research e implementada pela primeira vez em um protótipo da IBM, sendo depois reimplementada em diversos outros produtos da IBM (DATE, 2004).

A SQL inclui operações de definições de dados, ou seja, criar tabelas (relações), domínios, índices, tipos, entre outros. Após o banco ser definido pode-se começar a manipulá-lo por meio das operações da SQL: *select, insert, update e delete*. “Em particular podemos executar operações relacionais de restrição, projeção e junção sobre os dados, utilizando em cada caso a instrução de manipulação de dados *select* da SQL.” (DATE, 2004, p.73). A linguagem SQL divide em subgrupos de acordo com as operações e esses serão explicados nos próximos parágrafos.

A Linguagem de Definição de Dados ou *Data Definition Language* (DDL) se trata da definição da estrutura e organização dos dados, os comandos permitem a criação e

modificação dos esquemas das tabelas (SILBERSCHATZ, 2016). São esses comandos: c*reate*, *drop* e *alter*. Já o subgrupo Linguagem de Manipulação de Dados ou *Data Manipulation Language* (DML), é referente à inclusão, remoção, seleção ou atualização dos dados, tendo os comandos: *select*, *insert*, *update* e *delete* (LIMA, 2011).

A Linguagem de Controle de Dados ou *Data Control Language* (DCL), segundo Silberschatz (2016), garante o controle de acesso ou autorização para manipular os dados. Os comandos são: *grant*, *revoke*, *alter password* e *create synonym.* Linguagem de Transação de Dados ou *Data Transaction Language* (DTL), delimita transações que podem ou não ser concluídas. Pertencem os seguintes comandos: *begin; commit*, *rollback* (LIMA, 2011).

Quanto à Linguagem de Consulta de Dados ou *Data Query Language* (DQL), o seu comando *select* permite realizar consultas simples e complexas através das seguintes cláusulas: *from*, *where*, *group by*, *having*, *order by* e *distinct* (LIMA, 2011).

2.5 Bancos de dados não relacionais

Os bancos de dados não relacionais, chamados NoSQL (*Not only* SQL), não usam o padrão de tabelas com linhas e colunas presentes na maioria dos sistemas de banco de dados. Há várias abordagens para a classificação do banco de dados NoSQL, podem ser orientados a coluna, documentos (o qual foi utilizado no presente trabalho), chave-valor, grafo e multi-modelo (LÓSCIO; OLIVEIRA; PONTES, 2011). Todas essas categorias tem características próprias e armazenam os dados de uma forma diferente, porém a abordagem orientada a documentos foi escolhida pela sua maior popularidade, simplicidade e flexibilidade.

A modelagem dos bancos de dados não relacionais é feita de forma diferente dos relacionais e vai de acordo com sua abordagem, cada uma tem seu próprio tipo de representação. Para isso são utilizadas técnicas e *softwares* específicos, obtendo-se assim diagramas com as particularidades de cada banco.

Indubitavelmente o crescimento da quantidade de dados e informação na web vem crescendo exponencialmente. As aplicações geram uma grande quantidade de dados e isso pode se tornar um problema devido à complexidade da configuração de uma aplicação que usa o banco de dados relacional, podendo assim a experiência do usuário ser afetada negativamente (TOTH, 2011). Os bancos de dados NoSQL surgiram para resolver essa deficiência, mostrando uma abordagem diferente de persistência de dados, baseada em disponibilidade, desempenho e escalabilidade.

Segundo Lóscio, Oliveira e Pontes (2011), algumas de suas características fundamentais que os diferenciam dos sistemas tradicionais tornando-os adequados para o armazenamento de grandes volumes de dados são:

a) escalabilidade: aumento no número de máquinas disponíveis para o armazenamento e processamento de dados para melhorar o desempenho à medida que em que o volume de dados cresce;
b) ausência de esquemas ou esquemas flexível: ausência no esquema que define sua estrutura facilita a escalabilidade, porém não há garantia da integridade dos dados;
c) suporte nativo à aplicação: se trata de prover a escalabilidade através da replicação;
d) API simples para acesso aos dados: o objetivo do NoSQL é prover uma forma eficiente de acesso, oferecendo alta disponibilidade e escalabilidade. É necessário que as API's sejam desenvolvidas para facilitar o acesso à informação;
e) consistência eventual: essa característica dos bancos NoSQL é devida ao fato que nem sempre a consistência é mantida entre os diversos pontos de distribuição de dados.

2.6 SGBDs utilizados

Os SGBDs que serão levados em consideração nessa análise foram escolhidos por apresentar os maiores índices de utilização e aceitação segundo o ranking da DB-Engines (2021), e são tidos como os principais sistemas nesse mercado. Estes são divididos entre banco de dados relacionais: Oracle, MySQL, PostgreSQL e SQLServer e não relacionais: MongoDB. Para todos os bancos utilizados nesta análise foram utilizadas suas versões *open source* por questões de acessibilidade.

2.6.1 Oracle

O Oracle é um SGBD que surgiu no fim dos anos 70 após Larry Ellison, um programador em parceria com Robert Miner, um antigo supervisor da empresa onde trabalhava perceberem a oportunidade de criar uma base de dados compatível com centrais de computadores e diversos terminais em simultâneo, tendo como clientes uma base da força

aérea dos Estados Unidos e a CIA e se tornou a maior empresa de *software* empresarial do mundo (GREENWALD; STACKOWIAK; STERN, 2007).

O banco de dados da Oracle tem como princípio a linguagem PL/SQL, uma extensão da linguagem SQL e é comumente usada para implementar *procedures* e *triggers* entre outras funções. Trata-se de uma linguagem básica para criar programas de grande complexidade, não só no banco de dados, mas como em diversas ferramentas Oracle também.

Ao longo de mais de 40 anos, além de diferentes versões a Oracle subdividiu seu produto em diversas edições para ter um maior alcance em diferentes níveis de mercado. Algumas de suas versões são: Enterprise Edition, Standard Edition, Standard Edition One, Express Edition, Oracle Personal Edition, Oracle Database Lite e Oracle RAC (Real Application Clusters). Essas versões contam com diferentes requisitos de *hardware* para atender diferentes tipos de clientes (ORACLE, 2019). Essa variedade de versões também se dá por razões de marketing e controle de licenças.

2.6.2 MySQL

"O *software* MySQL fornece um servidor de banco de dados SQL muito rápido, multiusuário e robusto" (MYSQL, 2019). MySQL é uma marca registrada da Oracle Corporation. O software é *dual licensed*, ou seja, os usuários podem escolher em usar como produto *open source* (sob os devidos termos disponíveis no portal do fabricante) ou podem comprar uma licença comercial padrão. O projeto teve início em 1979, com UNIREG *database tool*, criado por Michael Widenius na companhia sueca TcX. O MySql foi lançado oficialmente em 1995 por David Axmark (DUBOIS, 2008).

Segundo DuBois (2008), o MySQL possui características atrativas para o desenvolvedor dentre elas: velocidade, facilidade de uso, suporte para linguagem SQL, alto desempenho, conectividade e segurança, portabilidade, compactabilidade, baixo custo, código aberto, vasta documentação, suporte de qualidade.

2.6.3 PostgreSQL

O PostgreSQL é um SGBD relacional utilizado para armazenar informações em todas as áreas de negócio, assim como gerenciar o acesso a essas. O *software* teve origem em um projeto chamado POSTGRES na Universidade de Berkeley, na Califórnia em 1986. Uma equipe foi designada para criar o novo sistema de banco de dados com o apoio da Army

Research Office e o National Science Foundation e entregaram a primeira versão em 1987 (MILANI, 2008).

Referente à compatibilidade existem várias bibliotecas e *drivers* de conexão para as principais linguagens e plataformas, por exemplo: C/C++, Java/JSP, PHP, ASP, .NET, Perl, Python, Ruby, Tcl e driver ODBC entre outros. Sobre sistemas operacionais, o PostgreSQL é uma ferramenta muito portátil, estando disponível nos ambientes: Linux, Unix, Mac e Windows (MILANI, 2008).

2.6.4 SQL Server

Microsoft SQLServer é um sistema gerenciador de banco de dados relacional desenvolvido pela Microsoft. O SQLServer suporta aplicações cliente/servidor, que um servidor suporta vários bancos de dados, usado por vários usuários simultaneamente (RAMALHO, 2005).

Algumas versões mais populares são: SQL Server Standart Edition, SQL Server Enterprise, SQL Server Developer, SQL Server Express, Management Studio (MICROSOFT, 2019). Sendo um dos principais bancos de dados do mercado, o SQLServer é um *software* legado consagrado por várias décadas, é capaz de atender as necessidades dos mais simples projetos aos mais complexos (RAMALHO, 2005).

2.6.5 MongoDB

MongoDB é um dos mais populares bancos de dados, bastante à frente de seus concorrentes NoSQL no mercado. É um projeto open-source e escrito em C++ e foi desenvolvido pela empresa 10gen inicialmente, que é hoje conhecida como MongoDB Inc.

MongoDB é um banco de dados orientado a documentos que armazena dados em documentos JSON com esquema dinâmico. Isso significa que é possível armazenar os registros sem se preocupar com a estrutura de dados, como o número de campos ou tipos de campos para armazenar valores. Os documentos do MongoDB são semelhantes aos objetos JSON. Ao contrário de banco de dados relacionais que guardam dados em tabelas formatadas e utilizam SQL para consulta de banco de dados, o MongoDB guarda dados em documentos.

É possível alterar a estrutura de registros (ou documentos, no Mongo) simplesmente adicionando novos campos ou excluindo os existentes. Esta capacidade do MongoDB é útil para representar relações hierárquicas, para armazenar matrizes, e outras

estruturas mais complexas de uma maneira mais simples. MongoDB oferece alto desempenho, alta disponibilidade, fácil escalabilidade com seus serviços de replicaset e sharding (SOARES, 2016).

A sintaxe do MongoDB retém os dados usando pares de chave/valor. É possível instalar o MongoDB tanto no Windows, como no Linux. É possível conectar o MongoDB à várias linguagens de programação (como C, C#, C++, Java, PHP, Python, Ruby, entre outras), para realizar as operações nos sistemas que desenvolver, cada conexão é realizada de forma diferente para cada linguagem. Entretanto é possível realizar as operações no banco através do cliente de shell do MongoDB como em um prompt de comando (SAKIS, 2017).

2.7 Benchmarking

A técnica de *benchmarking* consiste na execução de um conjunto fixo de testes sobre um sistema para avaliar o seu desempenho. Para a análise de desempenho de sistemas computacionais, esta técnica consiste na execução de um conjunto fixo de programas sobre um determinado sistema computacional. A técnica de benchmark pode ser aplicada na comparação de diferentes sistemas, uma vez que é padronizada, sendo os testes invariáveis e bem definidos (CIFERRI, 1995).

"O *benchmarking* tornou-se um dos métodos mais importantes para a avaliação quantitativa do desempenho de processadores e de projetos de sistemas computacionais." (BIENIA, 2011).

Conforme Machado (2010), na área de banco de dados, o *benchmark* é muito útil para o entendimento de como o SGBD responde sob a variação de condições. É por meio do *benchmarking* que será possível medir o desempenho dos SGBDs aqui analisados, pois esses *softwares* realizam uma série de testes padrões e são capazes de mensurar o desempenho dos objetos analisados em números.

Atualmente existem vários *softwares* de *benchmark* como AS³AP (ANSI *SQL Standard Scalable and Portable*), TCP-C (*Transaction Process Council*), BenchmarkSQL, OSDB (*Open Source Database Benchmark*), dentre outros, porém se tratam de *softwares* de difícil obtenção. Existem também ferramentas que realizam diferentes tipos de testes em programas computacionais, como o JMeter, na sua grande maioria de código aberto.

No presente trabalho foi utilizado o Apache JMeter, desenvolvido pela Apache Software Foundation por ser facilmente acessível além de sua grande compatibilidade com diversos tipos de aplicações, entre elas, os bancos de dados. A ferramenta *open source*, 100%

em Java realiza, assim como os *benchmarks* de banco de dados, testes como o de carga e de estresse a fim de medir o desempenho dos bancos de dados. O JMeter pode ser usado para testar recursos estáticos e dinâmicos.

2.8 JMeter

Segundo a Apache Software Foundation (2019), o aplicativo Apache JMeter™ é um software de código aberto, um aplicativo Java 100% puro projetado para carregar o comportamento funcional do teste e medir o desempenho. Ele foi originalmente projetado para testar aplicativos da Web, mas desde então expandiu para outras funções de teste. Dentre os recursos do Apache JMeter está presente a capacidade de carregar e testar o desempenho de muitos tipos diferentes de aplicativos/servidor/protocolo entre eles banco de dados via JDBC. Outra característica importante é a estrutura *multi-threading* completa, que permite a amostragem simultânea por muitos threads e a amostragem simultânea de diferentes funções por grupos de *threads* separados.

Serão apresentados dois tipos de testes não funcionais (carga e *stress*) que podem ser realizados em diferentes tipos de sistemas, incluindo SGBDs, para detectar problemas de requisitos não funcionais, e a automação destes testes utilizando a ferramenta de apoio JMeter, de acordo com Carvalho *et al* (2014):

a) teste de carga: O teste de carga é realizado para verificar a real capacidade do uso, de processamento e acessos do software/sistema/servidor em questão. Esse teste é comumente confundido com o teste de stress, mas possui finalidade diferente. Uma situação que este teste é utilizado é na identificação de qual momento o sistema ou a infraestrutura deverá ser melhorada para atender uma nova demanda;
b) teste de *stress*: O teste de stress serve para assegurar que a aplicação desenvolvida consegue atender a quantidade extremas de usuários acessando e realizando requisições simultâneas.

2.9 Trabalhos relacionados

Existem diversos trabalhos na linha de análise de desempenho de SGBDs. Ferreira e Júnior (2016) mediram o desempenho de quatro SGBDs de bastante influência no mercado: Firebird, PostgreSQL, MySQL, SQL Server com os *softwares* de *benchmark* AS³AP e TCP-C

com diferentes tamanhos de registros. Obtiveram como resultado que em todos os testes o banco que teve o melhor tempo para a inserção dos dados foi o PostgreSQL. Assim, pôde-se constatar que é um banco que, quanto maior o volume de dados inserido, melhor será o tempo de resposta em relação aos outros bancos de dados.

Maiello (2016) realizou uma comparação entre os SGBDs Oracle e PostgreSQL com o uso da ferramenta BenchmarkSQL em que apresenta os bancos, descreve as propriedades denominadas ACID (Atomicidade, Consistência, Isolamento e Durabilidade) que levou em consideração na análise e apresentou um estudo de caso exibindo os resultados e conclusões posteriormente. Foi concluído que para o teste em questão, pode-se dizer que em um ambiente a qual o número de transação concorrente seja maior o PostgreSQL é mais aconselhado, uma vez que apresentou melhor desempenho nos testes de *benchmark*. Já em um ambiente onde não exista concorrências nas transações e o processo de escrita e leitura na base de dados não seja alto, o Oracle se mostrou superior, conforme demonstrado no processo de carga inicial dos dados.

Pires, Nascimento e Salgado (2009) executaram um comparativo de desempenho entre bancos de dados de código aberto utilizando para isso os SGBDs MySQL e PostgreSQL, em plataforma GNU/Linux, utilizando o *benchmark* de código aberto OSDB. O estudo consiste em analisar as métricas geradas pelo OSDB e estimular melhorias nas funcionalidades dos SGBD relacionadas com desempenho. Como resultado obteve-se que o MySQL apresentou melhores resultados na maioria dos testes. O desempenho do PostgreSQL foi superior apenas no módulo de carga e estrutura, especialmente na criação de índices. Como trabalhos futuros espera-se realizar testes semelhantes utilizando algumas bases de dados, além de configurar o ajuste de desempenho de forma ainda mais elaborada.

Filho (2015) comparou o SGBD não relacional orientado a documentos MongoDB com o PostgreSQL com o JMeter, análise essa que importou uma base de dados de grande tamanho e realizou testes de consulta *exact match* retornando dados de acordo com o filtro aplicado. Foi aumentado o número de usuários e variou-se o tempo de inicialização entre eles observando o comportamento dos bancos. No final o MongoDB apresentou resultados largamente superiores.

Machado (2010) realizou um estudo sobre a otimização de desempenho em banco de dados PostgreSQL. Neste trabalho foi utilizado o BenchmarkSQL no banco escolhido para realizar testes de performance, resultando em estatísticas de desempenho com a finalidade de buscar propor estratégias para serem ao elaborar o projeto de um banco de dados, visando a otimização na performance da aplicação. Primeiramente foi realizado um levantamento de

regras gerais existentes para a otimização de banco de dados, a fim de identificar as regras mais utilizadas na hora de otimizar o banco de dados. Em seguida foram propostas estratégias para serem usadas na hora de elaborar o projeto de um banco de dados no SGBD PostgreSQL, visando a otimização da aplicação. Por fim foi apresentado um estudo de caso que demonstrou os principais problemas de desempenho identificados e as soluções propostas para o SGBD PostgreSQL utilizando para isso uma base de dados para teste pré-existente.

A exemplo dos trabalhos descritos, este trabalho visa realizar uma comparação entre SGBDs, utilizando assim determinado *software* para isso, neste caso, o JMeter. São tidos como diferenciais desse trabalho a variação na escolha dos SGBDs analisados, suas versões, versões de suas ferramentas administrativas, a ferramenta utilizada para realizar os testes e a seleção de um ambiente de testes com configurações mais robustas, além da presença de um banco de dados não relacional na análise.

3. METODOLOGIA

Este capítulo descreve a natureza da pesquisa, juntamente com a identificação do caráter da pesquisa realizada, os instrumentos, os materiais e procedimentos, métodos trabalhados, além do tratamento de dados por meio dos instrumentos utilizados.

3.1 Natureza da pesquisa

A metodologia aplicada no desenvolvimento deste estudo apresenta caráter descritivo. Neste tipo de pesquisa realiza-se a análise, o estudo, o registro e a interpretação dos resultados obtidos sem a intervenção do pesquisador. A pesquisa descritiva pode ser comparada a um estudo de caso, onde os dados obtidos são analisados e comparados e são determinados os efeitos e resultados (PEROVANO, 2014). Neste trabalho será observado o comportamento dos SGBDs estudados de acordo com os resultados obtidos.

O estudo também apresenta caráter quantitativo, Wainer (2007), estabelece que este é baseado na medida numérica de variáveis, com ênfase na comparação de resultados e muitas vezes com o uso de técnicas estatísticas. Neste trabalho serão coletados dados pela ferramenta dos testes e com base neles será apresentada uma conclusão.

3.2 Materiais

Considerando o objetivo desse estudo, que consiste em realizar uma análise comparativa, ou um *benchmark*, propriamente dito, entre alguns dos SGBDs mais utilizados hoje em dia, foram realizadas pesquisas, por meio de uma fundamentação teórica, em busca de informações acerca do tema abordado, que sintetizassem a importância do mesmo.

Para a execução dessa pesquisa, foram levados em consideração os SGBDs relacionais Oracle Database 18c 18.4 Express Edition, PostgreSQL 11.1, SQLServer 2019 Express Edition e o MySQL 8.0.23 Community. Para auxiliar na utilização de cada banco serão utilizadas as seguintes ferramentas administrativas para seus respectivos bancos caso necessitem: PGAdmin 4.28 (PostgreSQL), SQL Server Management Studio 18.8 (SQL Server), MySQL Workbench Community 8.0.23 e SQLDeveloper 2.0 (Oracle). Como banco NoSQL, foi escolhido o SGBD MongoDB juntamente com sua ferramenta Robo3T 1.4.2.

A ferramenta selecionada para realizar os testes de carga e estresse nos bancos, como mencionado anteriormente foi o JMeter, uma ferramenta Java bastante utilizada para medir desempenho em diversos tipos de programas computacionais.

O ambiente de testes onde foram instalados todos os *softwares*, configuradas as conexões com os bancos e realizados os testes de desempenho foi uma máquina virtualizada no servidor no servidor do laboratório de redes do IFMG *Campus* São João Evangelista com as seguintes especificações: processador Intel Xeon E5630 dual core de 2.53 GHz (dois processadores), 16GB de memória RAM com o Windows Server 2016 Datacenter Evaluation como sistema operacional.

3.3 Métodos

Com a máquina virtual devidamente configurada, foram instalados e configurados os SGBDs e ferramentas utilizadas no projeto. A conexão com os SGBDs se dá por meio do *driver* Java Database Connectivity (JDBC), um conjunto de classes e interfaces escritas em Java que fazem o envio de instruções SQL para qualquer banco de dados. Desse modo, é possível enviar requisições do JMeter para os SGBDs e receber dados dessa conexão.

Os testes foram divididos em duas categorias: carga e *stress*. No teste de carga foi observado o comportamento de cada SGBD ao aumentar o número de registros no banco em cada uma das operações e coletados os tempos que cada um deles levou para concluir a solicitação, afim de testar sua eficácia naquela operação com aquele número de registros.

Já o teste de *stress* consiste em submeter o banco à uma quantidade de requisições com um alto número de usuários (ou *threads*) simultâneos com a finalidade de observar se o banco suporta e é eficaz nesse processo, além de coletar o uso total de *Central Process Unit* (CPU), ou, Unidade Central de Processamento do sistema no momento da operação para medir a eficiência que o banco lida com os recursos do ambiente.

Essas métricas tem como objetivo medir o desempenho dos SGBDs pelo tempo que este leva para concluir a solicitação, se ele conseguiu executar e também mensurar em qual ponto o banco não conseguiu responder totalmente às solicitações devido ao *stress* aplicado apresentando erros.

Para realizar os testes, foi criada a mesma base de dados em cada um dos SGBDs, ou seja, as mesmas informações foram inseridas de forma que também os retornos das solicitações fossem os mesmos.

3.3.1 Carga

O procedimento consiste em realizar a primeira inserção com 1.000 registros e obter seu tempo de finalização. Logo após já se efetua os comandos de busca, alteração e exclusão, também coletando seus respectivos tempos. Então a máquina é reiniciada para eliminar qualquer influência na memória e é realizado novamente o passo anterior para as iterações de 10.000 e 100.000 sempre excluindo a tabela, ou coleção no caso do MongoDB, e recriando-a. Essa etapa é realizada com um usuário, ou *thread*, apenas.

Esse tempo coletado representa o tempo total da conexão do JMeter com o banco até este concluir a solicitação.

3.3.2 Stress

Nesse teste é observado o comportamento da aplicação ao realizar um *full scan* em cada um dos bancos, ou seja, percorrer a tabela inteira para retornar todos os registros. A quantidade de registros é de 10.000. Observa-se o comportamento do banco ao aumentar a quantidade de usuários simultâneos em 50, 100, 150 e 200. Além disso, é também verificada a utilização máxima de *Central Processing Unit* (CPU), ou unidade central de processamento, que a aplicação consumiu.

3.3.3 Dados Inseridos

Como objeto de comparação dos testes foi criado em cada banco uma estrutura de armazenamento, ou seja, uma tabela nos bancos relacionais, e uma coleção no caso do MongoBD que é não relacional. Nela foram criados os mesmos campos para que seja armazenada a mesma informação em todos os SGBDs. Trata-se de uma estrutura simples, com um número de campos considerável e não muito grande para não causar travamentos no JMeter, sendo definida de forma empírica tendo como referência os trabalhos correlatos e foi utilizada em todos os testes propostos.

Se trata de uma estrutura simples e única por razão da proposta do trabalho que visa comparar os SGBDs em suas quatro operações essenciais, sendo desejável a observação de cada tabela, ou coleção de forma individual. Caso desejável uma base de dados maior com várias estruturas, a operação de *select* retornaria uma consulta exata, retornando resultados específicos relacionados por essas várias estruturas, como observado em alguns trabalhos

relacionados.

Nos bancos relacionais foi criada uma tabela com sete campos conforme o quadro 1.

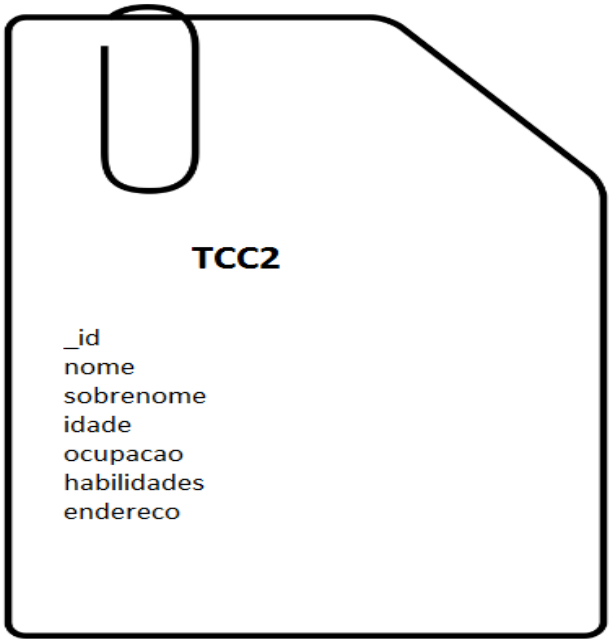
Quadro 1- Dados da tabela criada

TABELA TCC2	TIPO DE DADO
ID	INTEGER
NOME	VARCHAR(50)
SOBRENOME	VARCHAR(50)
IDADE	INTEGER
OCUPACAO	VARCHAR(50)
HABILIDADES	VARCHAR(50)
ENDERECO	VARCHAR(50)

Fonte: Elaborado pelo autor, 2021.

Já no MongoDB é possível representar sua coleção conforme a Figura 4.

Figura 4- Representação da coleção criada no MongoDB



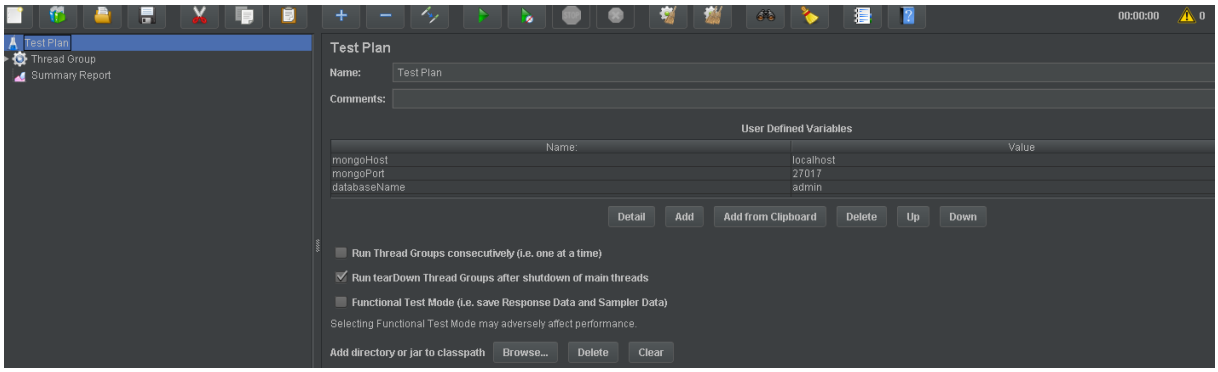
Fonte: Elaborado pelo autor, 2021.

3.3.4 Configuração do JMeter

Inicialmente, é configurado um plano de teste, onde foi adicionada a configuração JDBC e adicionadas as requisições, chamadas *samplers*. Nesse plano de teste é possível declarar variáveis que podem ser necessárias nos scripts dos testes, como é o caso do MongoDB, além de poder adicionar plug-ins e arquivos desejados.

A Figura 5 exibe a tela de configuração do plano de teste do JMeter.

Figura 5- Plano de teste no JMeter

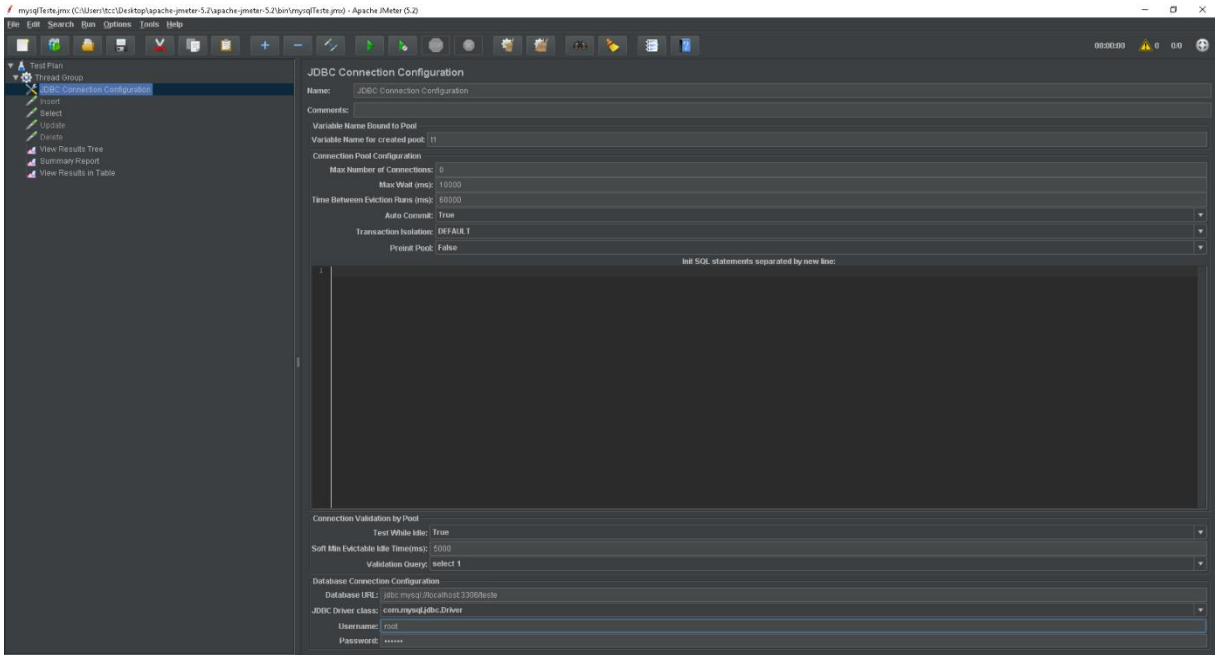


Fonte: Print screen do JMeter, 2021.

Em seguida, é criada uma configuração de conexão JDBC. Para conectar os SGBDs no JMeter é necessário estabelecer uma Uniform Resource Locator (URL) de conexão localizada nesse elemento de configuração JDBC. Cada banco possui uma URL diferente e, ao se conectar, é possível uma transferência de dados entre os programas.

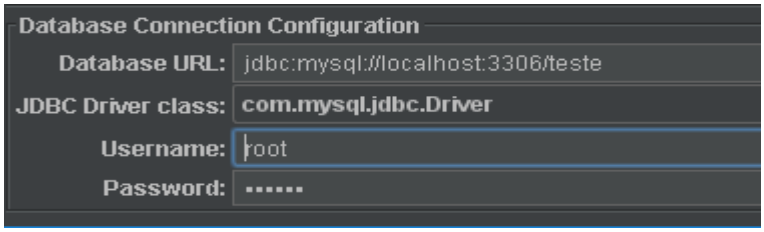
Na Figura 6 é mostrada a tela de configuração JDBC, onde em seu final se encontra a configuração de bancos de dados conforme a Figura 7.

Figura 6- Configuração de conexão JDBC



Fonte: Print screen do JMeter, 2021.

Figura 7- Configuração de conexão do banco de dados



Fonte: Print screen do JMeter, 2021.

Por fim, é possível adicionar os *samplers*, um para cada operação. Os *samplers* podem ser chamados testadores, eles permitem o JMeter enviar requisições para o servidor desejado. Neles são inseridos os scripts que executam as operações e os *samplers* utilizados são o JDBC Request para os bancos relacionais e o JSR223 para o MongoDB, este último sendo uma estrutura para incorporar scripts no código-fonte Java devido o formato de documentos com que o MongoDB trabalha.

4. RESULTADOS E DISCUSSÕES

Neste capítulo, os dados coletados foram registrados e apresentados em forma de quadros e gráficos, de modo que seja possível visualizar uma comparação entre os objetos analisados.

Para se chegar nos valores finais das variáveis utilizadas, como o número de usuários, tempo entre o início de cada usuário e a quantidade de registros para o preenchimento dos bancos, foram realizada uma pré-testes com diferentes configurações do JMeter para o reconhecimento de um intervalo de valores que atenda os testes propostos e não sobrecarregue o sistema, assim como utilizadas referências dos trabalhos correlatos e arbitrando valores para as variáveis empiricamente até chegar nos valores utilizados. Assim alcançou-se resultados mais coerentes nos testes.

4.1 Testes de carga

Como abordado anteriormente, foi coletado o tempo de conclusão de quatro operações em cada banco: *insert*, *update*, *select* e *delete*. Esta estapa foi dividida em três partes, cada uma delas com um tamanho de registro: 1.000, 10.000 e 100.000.

4.1.1 1.000 registros

Nessa primeira etapa foi utilizado um volume na ordem de 1.000 registros. Os resultados de cada SGBD estão representados nos seguintes quadros, sendo os testes realizados duas vezes em cada banco para uma maior precisão e tentar cobrir melhor possíveis variações.

Quadro 2- *Insert* com 1.000 registros

SGBD	TEMPO 1	TEMPO 2	TEMPO MÉDIO
ORACLE	4618	4903	4760,5
MYSQL	2415	947	1681
SQLSERVER	300	326	313
POSTGRESQL	1182	1192	1187
MONGODB	1006	801	903,5

Fonte: Elaborado pelo autor, 2021.

Na operação de inserção o SQLServer teve considerável vantagem com a primeira carga. PostgreSQL e Mongo apresentaram resultados relativamente próximos, mas ligeiramente superior ao MySQL. O Oracle não respondeu bem à essa requisição e teve tempo bem acima dos demais.

Quadro 3 - *Update* com 1.000 registros

SGBD	TEMPO 1	TEMPO 2	TEMPO MÉDIO
ORACLE	156	188	172
MYSQL	221	182	201,5
SQLSERVER	25	24	24,5
POSTGRESQL	178	351	264,5
MONGODB	99	212	155,5

Fonte: Elaborado pelo autor, 2021.

Em atualização dos dados, o SQLServer também obteve boa vantagem em relação aos demais, que tiveram seus tempos equiparados tendo o PostgreSQL o pior deles.

Quadro 4 - *Select* com 1.000 registros

SGBD	TEMPO 1	TEMPO 2	TEMPO MÉDIO
ORACLE	189	195	192
MYSQL	243	103	173
SQLSERVER	113	72	92,5
POSTGRESQL	132	221	176,5
MONGODB	50	56	53

Fonte: Elaborado pelo autor, 2021.

A operação de consulta dessa etapa apresentou melhor resultado para o MongoBD, seguido do SQLServer, MySQL, PostgreSQL e Oracle respectivamente.

Quadro 5 - *Delete* com 1.000 registros

SGBD	TEMPO 1	TEMPO 2	TEMPO MÉDIO
ORACLE	13	31	22
MYSQL	129	119	124

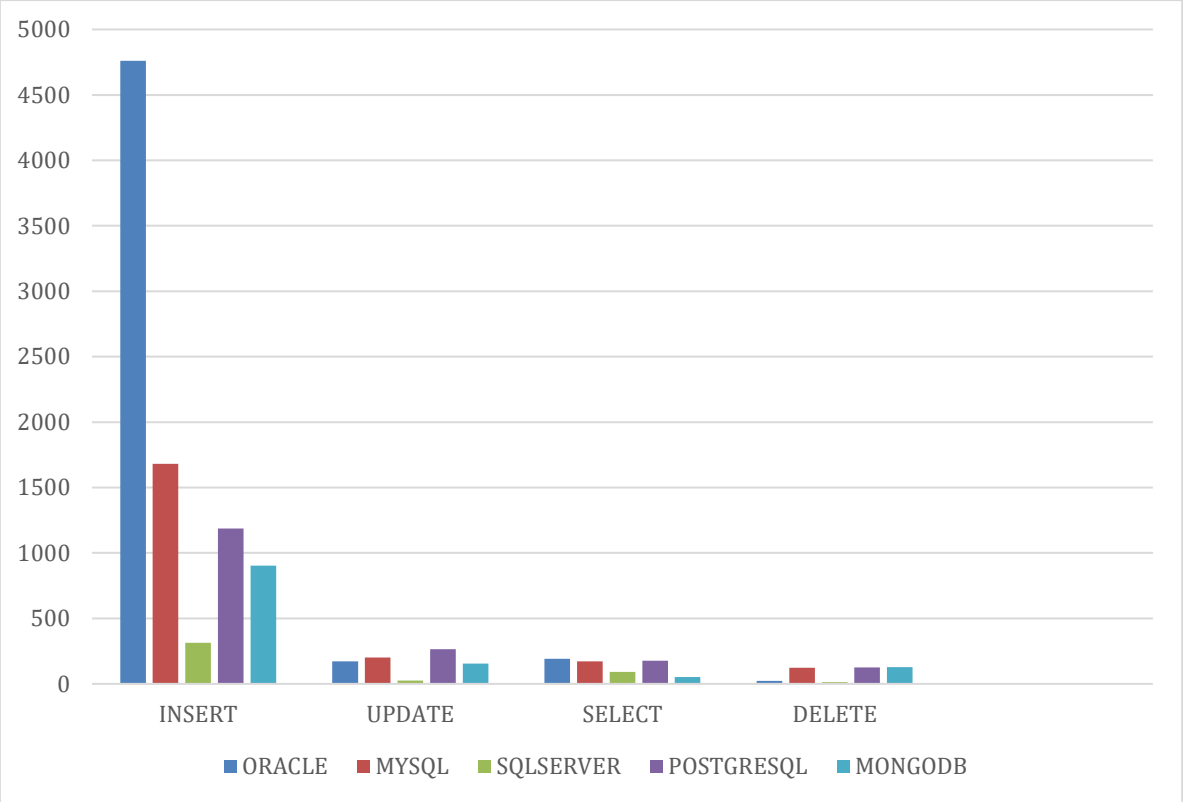
SQLSERVER	14	10	12
POSTGRESQL	142	111	126,5
MONGODB	140	116	128

Fonte: Elaborado pelo autor, 2021.

Na exclusão de dados da ordem de 1.000 destacaram-se SQLServer e Oracle, o primeiro ligeiramente a frente. Os demais tiveram tempos parecidos.

O gráfico abaixo fornece uma visualização dos tempos lado a lado.

Gráfico 1 - Tempos das operações com 1.000 registros



Fonte: Elaborado pelo autor, 2021.

4.1.2 10.000 registros

Em sequência, foi a vez de testar com uma carga maior, na ordem de 10.000 registros.

Quadro 6 - *Insert* com 10.000 registros

SGBD	TEMPO 1	TEMPO 2	TEMPO MÉDIO

ORACLE	825558	848463	837010,5
MYSQL	1144	1251	1197,5
SQLSERVER	3246	3315	3280,5
POSTGRESQL	834	658	746
MONGODB	4271	4789	4530

Fonte: Elaborado pelo autor

Aumentando a quantidade de registros notou-se melhor resultado para o PostgreSQL, seguido pelo MySQL. SQLServer e MongoDB levaram alguns segundos a mais para concluir. Já o Oracle a exemplo da primeira etapa não respondeu bem e teve tempo muito acima dos demais nessa operação.

Quadro 7- *Update* com 10.000 registros

SGBD	TEMPO 1	TEMPO 2	TEMPO MÉDIO
ORACLE	412	465	488,5
MYSQL	560	343	451,5
SQLSERVER	56	47	51,5
POSTGRESQL	212	165	188,5
MONGODB	604	558	581

Fonte: Elaborado pelo autor, 2021.

O SQLServer apresentou o melhor tempo em *update* com carga de 10.000, juntamente com SQLServer ficaram bem distantes dos demais. O MongoDB novamente terminou na última posição.

Quadro 8 - *Select* com 10.000 registros

SGBD	TEMPO 1	TEMPO 2	TEMPO MÉDIO
ORACLE	254	334	294
MYSQL	196	269	232,5
SQLSERVER	230	199	214,5
POSTGRESQL	249	241	245
MONGODB	48	42	45

Fonte: Elaborado pelo autor, 2021.

Na operação de busca do segundo teste de carga, o MongoDB seguiu demonstrando grande superioridade em relação aos outros que tiveram resultados semelhantes.

Quadro 9 - *Delete* com 10.000 registros

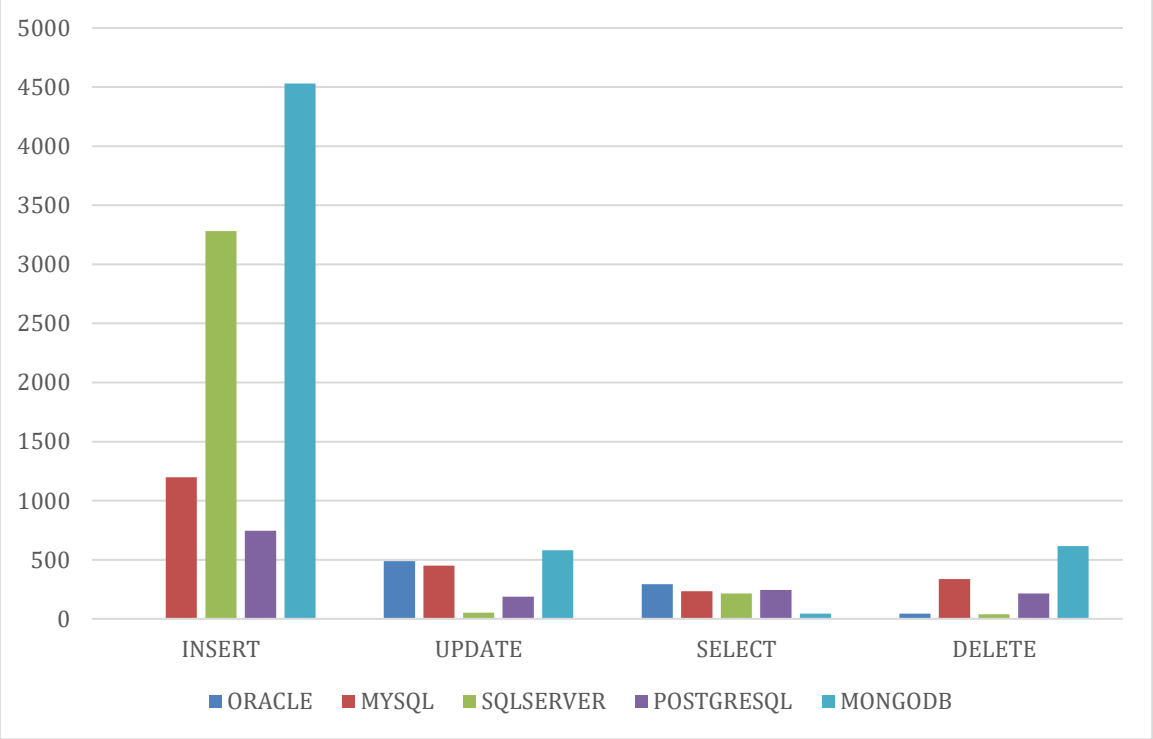
SGBD	TEMPO 1	TEMPO 2	TEMPO MÉDIO
ORACLE	35	54	35
MYSQL	325	350	337,5
SQLSERVER	42	38	40
POSTGRESQL	165	266	215,5
MONGODB	550	679	614,5

Fonte: Elaborado pelo autor, 2021.

Em exclusão foram espelhados os resultados da primeira etapa de teste com Oracle e SQLServer na ponta, seguidos de longe por PostgreSQL e MySQL. O MongoDB novamente demorou mais para finalizar.

Os resultados podem ser comparados no Gráfico 2.

Gráfico 2 - Tempos das operações com 10.000 registros



Fonte: Elaborado pelo autor, 2021.

No gráfico acima foi desconsiderado o tempo de inserção do Oracle devido à grande desequiparação com o resultado dos demais.

4.1.3 100.000 registros

Finalmente foi realizada a última etapa com 100.000 registros, com resultados apresentados no Quadro 10.

Quadro 10 - *Insert* com 100.000 registros

SGBD	TEMPO 1	TEMPO 2	TEMPO MÉDIO
ORACLE	-	-	-
MYSQL	4947	4755	4851
SQLSERVER	29840	29879	29859,5
POSTGRESQL	5660	6100	5880
MONGODB	34519	35183	34851

Fonte: Elaborado pelo autor, 2021.

No terceiro e último teste de carga da operação de inserção o Oracle não conseguiu terminar a requisição. MySQL e PostgreSQL disputaram o primeiro lugar com vantagem para o MySQL. SQLServer e MongoDB se mostraram menos eficientes com esse tamanho de registros.

Quadro 11 - *Update* com 100.000 registros

SGBD	TEMPO 1	TEMPO 2	TEMPO MÉDIO
ORACLE	6547	6735	6641
MYSQL	4080	4295	4187,5
SQLSERVER	352	356	354
POSTGRESQL	4390	3756	4073
MONGODB	6462	5542	6002

Fonte: Elaborado pelo autor, 2021.

Em *update* houve grande disparidade do SQLServer. PostgreSQL e MySQL com tempos próximos vieram em sequência, porém bem atrás. Já O MongoDB e Oracle apresentaram maior demora.

Quadro 12 - *Select* com 100.000 registros

SGBD	TEMPO 1	TEMPO 2	TEMPO MÉDIO
ORACLE	1925	1876	1900,5
MYSQL	1046	946	996
SQLSERVER	704	571	637,5
POSTGRESQL	630	507	568,5
MONGODB	58	63	60,5

Fonte: Elaborado pelo autor, 2021.

Novamente em *select* o MongoDB demonstrou superioridade, agora com 100.000 registros.

Quadro 13 - *Delete* com 100.000 registros

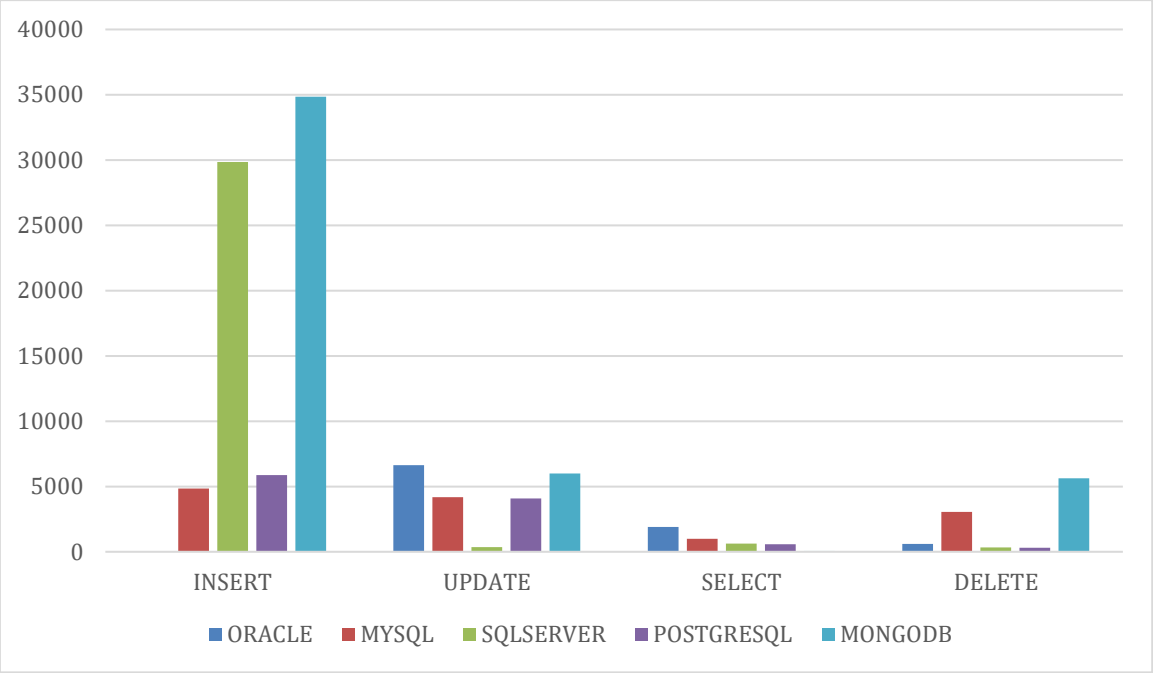
SGBD	TEMPO 1	TEMPO 2	TEMPO MÉDIO
ORACLE	583	625	604
MYSQL	2893	3241	3067
SQLSERVER	419	267	343
POSTGRESQL	225	373	299
MONGODB	5747	5515	5631

Fonte: Elaborado pelo autor, 2021.

Na última operação desse teste, o PostgreSQL se mostrou mais rápido na exclusão dos dados, acompanhado de perto pelo SQLServer. Destaques negativos para o MongoDB e o MySQL que destoaram dos demais.

No Gráfico 3 observa-se os resultados dos testes de forma comparativa.

Gráfico 3 - Tempos das operações com 100.000 registros



Fonte: Elaborado pelo autor, 2021.

Observa-se uma ausência no tempo de inserção do Oracle. Isso se dá pelo fato de o SGBD não suportar tamanha quantidade dessa operação simultaneamente ocasionando na mensagem de erro “SQL Error: ORA-00913: too many values”.

Analisando os três testes, observa-se uma notável dificuldade no processo de inserção de dados no Oracle devido sua limitação a grandes quantidades de dados. O banco apresentou tempo de inserção nas etapas de 1.000 e 10.000 registros consideravelmente abaixo dos demais. Na operação de *full scan* também apresentou o resultado mais baixo. Obteve bom desempenho nas demais operações principalmente em *delete* em que foi líder.

4.2 Testes de stress

Como abordado anteriormente, o teste de *stress* visa checar se o banco consegue atender a grandes quantidades de usuários acessando e realizando requisições simultâneas. Foi definido que essas seriam de 50, 100, 150 e 200 por serem valores que a maioria dos SGBDs suporta. A operação escolhida foi a de *full scan* pela grande exigência dos recursos do sistema para um grande retorno de dados dos bancos.

4.1.1 50 usuários

A primeira etapa é realizada com 50 usuários simultâneos e apresentou os seguintes resultados.

Quadro 14 - Teste de *stress* com 50 usuários

SGBD	TEMPO MÉDIO (ms)	% ERRO	USO DE CPU TOTAL %
ORACLE	4905	0,00%	67,00%
MYSQL	1084	0,00%	98,00%
SQLSERVER	1756	0,00%	81,00%
POSTGRESQL	2264	0,00%	99,00%
MONGODB	878	0,00%	35,00%

Fonte: Elaborado pelo autor, 2021.

É possível observar um alto uso de CPU principalmente no MySQL e no PostgreSQL, além do tempo médio do Oracle que ficou bastante acima dos demais.

4.2.2 100 usuários

Em sequência foi aumentado o número de usuários para 100.

Quadro 15 - Teste de *stress* com 100 usuários

SGBD	TEMPO MÉDIO (ms)	% ERRO	USO DE CPU TOTAL %
ORACLE	6323	46,00%	60,00%
MYSQL	2289	0,00%	98,00%
SQLSERVER	2211	0,00%	97,00%
POSTGRESQL	3497	0,00%	99,00%
MONGODB	499	0,00%	40,00%

Fonte: Elaborado pelo autor, 2021.

Nota-se a primeira ocorrência de erros no Oracle e maior utilização geral de CPU, com exceção do MongoDB.

4.2.3 150 usuários

Os resultados dos testes com 150 usuários são descritos no quadro abaixo.

Quadro 16 - Teste de *stress* com 150 usuários

SGBD	TEMPO MÉDIO (ms)	% ERRO	USO DE CPU TOTAL %
ORACLE	8183	54,00%	70,00%
MYSQL	2392	0,00%	98,00%
SQLSERVER	2818	0,00%	99,00%
POSTGRESQL	5004	33,33%	99,00%
MONGODB	516	0,00%	47,00%

Fonte: Elaborado pelo autor, 2021.

Aqui houve o início de erros no PostgreSQL, que juntamente com o Oracle, apresentaram tempo médio maiores.

4.2.4 200 usuários

Por fim o último teste teve 200 *threads*.

Quadro 17 - Teste de *stress* com 200 usuários

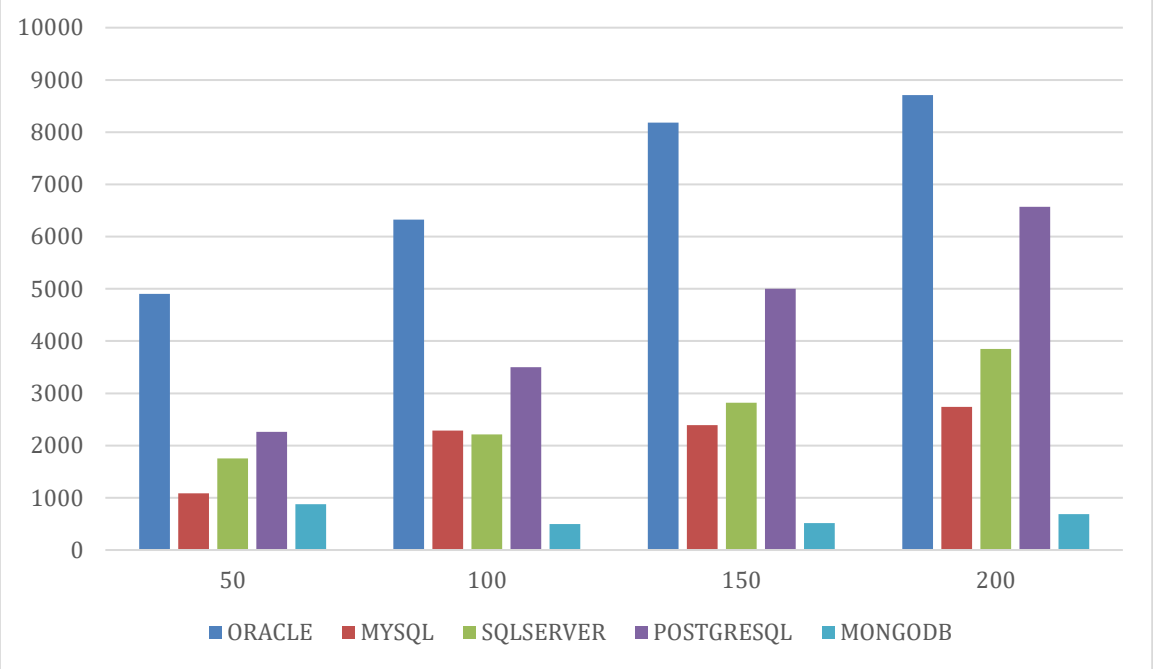
SGBD	TEMPO MÉDIO (ms)	% ERRO	USO DE CPU TOTAL %
ORACLE	8710	71,50%	47,00%
MYSQL	2742	24,00%	99,00%
SQLSERVER	3850	0,00%	99,00%
POSTGRESQL	6572	50,00%	99,00%
MONGODB	690	0,00%	51,00%

Fonte: Elaborado pelo autor, 2021.

Por fim o MySQL não conseguiu concluir a tarefa e também apresentou erros. Destaque para o MongoDB e SQLServer, este último concluindo apesar da grande utilização de CPU.

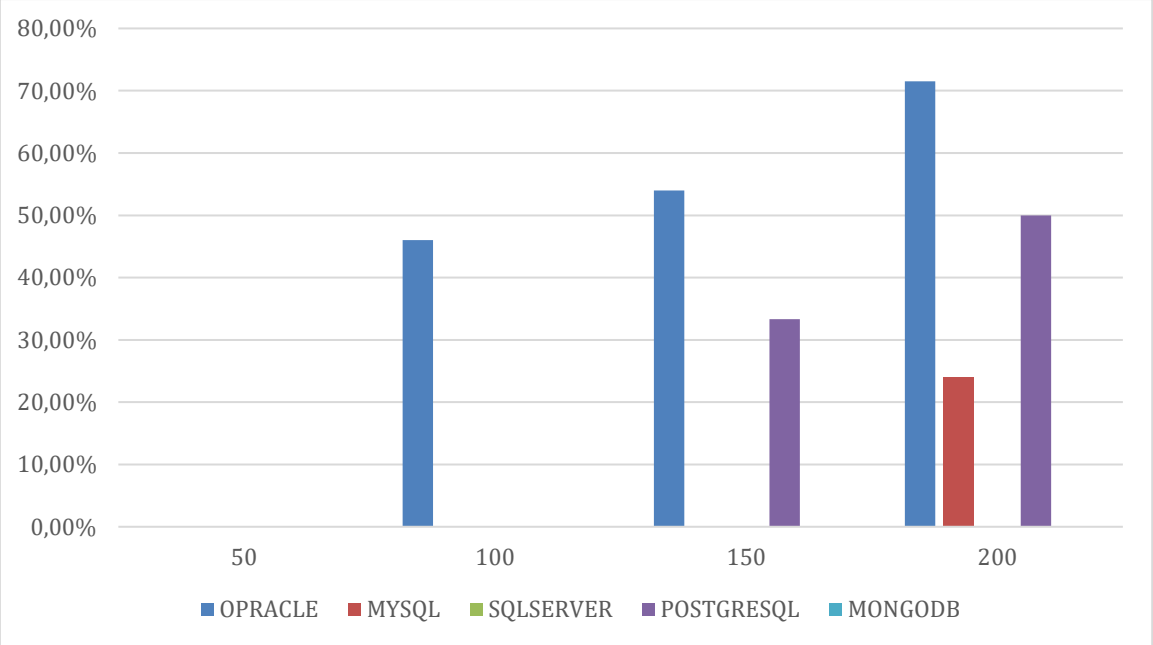
Nos gráficos abaixo é possível uma visualização das quatro etapas, divididas pelas vertentes analisadas para efeitos comparativos.

Gráfico 4 - Tempo médio de usuário



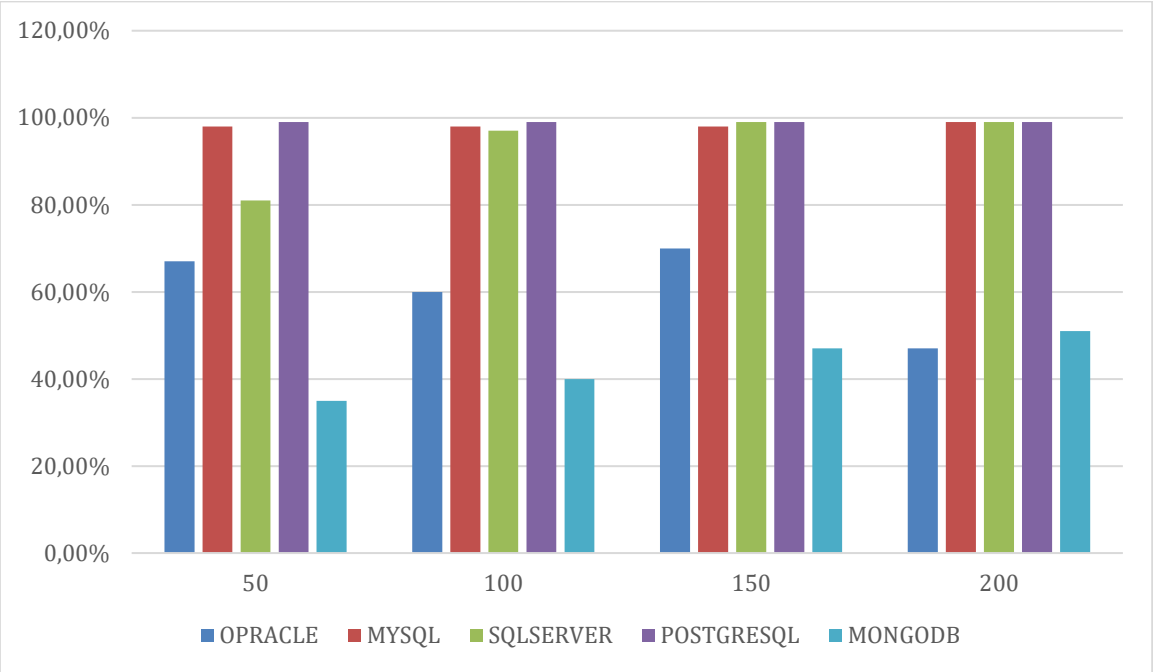
Fonte: Elaborado pelo autor, 2021.

Gráfico 5 - % de erro



Fonte: Elaborado pelo autor, 2021.

Gráfico 6 - Utilização de CPU



Fonte: Elaborado pelo autor, 2021.

Analisando os gráficos observa-se grande domínio do MongoDB que concluiu todos os testes assim como o SQLServer, este consumindo mais do sistema. À medida que se aumentou o número de usuários simultâneos identificou-se a variação no tempo médio das requisições e os pontos onde não foi mais possível continuar com a tarefa no caso do Oracle, PostgreSQL e MySQL.

5. CONSIDERAÇÕES FINAIS

Baseando-se nos dados provenientes dos resultados das análises dos SGBDs através da ferramenta Apache JMeter, acerca de testes de carga e *stress*, é possível realizar uma análise quantitativa dos resultados e apontar algumas conclusões.

Primeiramente foi notável uma dificuldade no processo de inserção de dados no Oracle devido sua limitação a um grande número de inserção de registros de uma única vez. O banco apresentou tempo de inserção nas etapas de 1.000 e 10.000 registros consideravelmente abaixo dos demais e não concluiu os testes com 100.000. Na operação de *full scan* também apresentou o resultado mais baixo. Obteve bom desempenho nas demais operações principalmente em delete em que foi líder à medida que se aumentou a carga.

No geral, os bancos MySQL e PostgreSQL foram bem equiparados tendo resultados bastante regulares. O primeiro teve o segundo pior tempo em inserção na etapa com 1.000 registros, porém obteve uma melhora nas seguintes deixando o posto para o MongoDB, e, no final apresentou o melhor tempo de inserção, indicando ter maior desempenho à medida que se aumenta o número de registros.

O MongoDB apresentou o segundo pior resultado em *insert* e *delete* com o aumento do tamanho dos registros. Por outro lado, obteve o melhor resultado disparado no *full scan*. O MongoDB e o SQLServer, a exemplo do Oracle, também apresentaram limitações quanto à inserção de registros na última etapa dos testes de carga. O SQLServer passou do melhor colocado no primeiro teste para o terceiro colocado no último, em compensação, teve os melhores tempos gerais em *update* e *delete*. O MongoDB finalizou em penúltimo neste quesito.

No teste de *stress* houve destaque para o MongoDB que suporta muito bem altas quantidades de usuários e tem um rápido retorno dos documentos em suas coleções. O SQLServer também respondeu bem, não apresentou erros pelo excesso de conexões, porém teve alto uso de processamento indicando que com o aumento de *threads* e quantidade de registros seria possível ocorrência de eventuais falhas. Os demais apresentaram erros com o aumento dos usuários e um alto uso de CPU.

Pode-se concluir que o SQLServer leva vantagem para o uso geral, vencendo seus concorrentes em 3 de 4 operações no teste de carga da ordem de 1.000 registros, ou seja, seria o mais indicado para operações dessa ordem, sendo também o mais indicado para se realizar *updates* no geral. Além disso foi o segundo melhor no teste de *stress*, perdendo apenas para o MongoDB, suportando bem vários usuários simultâneos. Apresentou dificuldades na

operação de inserção a partir da ordem de 10.000 registros, então caso o sistema requisite um grande número de inserção em detrimento de outras operações, seria mais indicada a utilização do MySQL ou do PostgreSQL, que tiveram os melhores resultados de *insert* em grandes quantidades de dados.

O PostgreSQL e o MySQL vem logo em seguida nesse teste com 10.000 registros e podem ser apontados como alternativas, uma vez que tiveram os tempos mais próximos do SQLServer em *update* e *delete*, além de terem obtido melhores resultados em *insert* dessa etapa, porém com resultados gerais um pouco melhores para o PostgreSQL nessa fase do teste.

No último dos testes de carga, SQLServer, PostgreSQL e MySQL obtiveram os melhores resultados, os dois últimos bastante equiparados com o primeiro continuando a apresentar tempo de inserção consideravelmente maior. Assim, caso a aplicação demande grande volume de inserção de dados, o PostgreSQL e o MySQL levariam vantagem, com o MySQL apresentando um resultado um pouco melhor em inserção, porém, ainda com destaque para o PostgreSQL por vencer em mais operações.

O MongoDB no geral apresentou grande vantagem nos demais na operação de *select* e no acesso por vários usuários simultâneos, fazendo dele a melhor escolha de um sistema que tenha esses requisitos. Esse resultado reflete o crescimento recente na utilização do MongoDB com várias grandes empresas atualmente o empregando como por exemplo, Globo, LinkedIn, MTV, entre outras, que provavelmente demandam grande fluxo de dados e requisições simultâneas diariamente. Seria também indicado para retornar grandes quantidades de informações de uma vez só por apresentar ótimos resultados em *select*.

Vale ressaltar o uso do Oracle em sua versão *open source*, o que possivelmente influenciou no resultado visto que a empresa é referência no mercado e possui diversos produtos com diferentes especificações, sendo escolhida a presente por ser *open source* como os demais.

É importante ressaltar também que estes testes foram realizados com a configuração padrão de cada SGBD, sendo assim possível utilizar de recursos disponíveis dos próprios para otimizar os resultados. Sendo assim, no momento da escolha do SGBD pode-se levar em consideração que de acordo com a finalidade que se deseja implementar o banco, pode haver ganho de desempenho. Por exemplo, alguma configuração adicional no Oracle que o permita aceitar melhor grandes quantidades de *inserts*, utilização de índices nas tabelas, ou, *scripts* com sintaxes particulares que favoreçam determinado SGBD.

Como trabalhos futuros nesta linha de pesquisa podem-se incluir a utilização de outros bancos de dados NoSQL com diferentes abordagens; configurações adicionais dos SGBDs; análise de importação de dados por meio de processos como, por exemplo, o *bulk insert;* uso de base de dados maiores, com mais estruturas e com mais registros inseridos.

Este estudo teve como finalidade apresentar resultados de testes de desempenho para comparação entre os SGBDs em um determinado ambiente com as configurações em questão e serve como uma referência quanto ao desempenho dos mesmos.

REFERÊNCIAS

APACHE SOFTWARE FOUNDATION. **Sobre**. Disponível em: <https://jmeter.apache.org/index.html>. Acesso em: 13 de maio de 2019.

BAZZOTTI; GARCIA. **A importância do sistema de informação gerencial na gestão empresarial para tomada de decisões.** Cascavel: UNIOESTE, 2006.

BIENIA, Christian; LI, Kai. **Benchmarking modern multiprocessors**. Princeton: Princeton University, 2011.

BRITO; COUTINHO; CUNHA. **Desenvolvimento e Integração de uma Aplicação Móvel a um Sistema Distribuído**: Estudo de Caso de um Sistema de Gerenciamento de Vagas. Fortaleza: UFC, 2016.

CARVALHO et al. **Testes não funcionais com apoio da ferramenta jmeter**. Revista de trabalhos acadêmicos Brasil, n. 9. Universidade Salgado Oliveira (UNIVERSO), 2014.

CIFERRI, Ricardo Rodrigues et al. **Um benchmark voltado a análise de desempenho de sistemas de informações geográficas**. Campinas: UNICAMP, 1995.

COUGO, Paulo. **Modelagem conceitual e projeto de banco de dados**. Rio de Janeiro: Elsevier Brasil, 2013.

DATE, C. J. **Introdução a Sistemas de Bancos de Dados**. 8. ed. Rio de Janeiro: Elsevier, 2004.

DB-ENGINES. **DB-Engines Ranking**. Disponível em: <https://db-engines.com/en/ranking>. Acesso em: 21 de julho de 2021.

DEANS, David H.; **Worldwide IT Revenue will Reach $3.5 Trillion in 2017**. Disponível em: <https://blog.geoactivegroup.com/2017/04/worldwide-it-revenue-will-reach-35.html>. Acesso em: 21 de maio de 2019.

DEVMEDIA. **Bancos de Dados Relacionais**. Disponível em: <https://www.devmedia.com.br/bancos-de-dados-relacionais/20401>. Acesso em: 21 de maio de 2019.

DIEHL, A. A. **Pesquisa em ciências sociais aplicadas: métodos e técnicas.** São Paulo:

DUBOIS, Paul. **MySql Developer´s Library**. 4 ed. 2008.

ELMASRI; NAVATHE. **Sistemas de banco de dados**. 6. ed. São Paulo: Pearson, 2011.

FERREIRA; JÚNIOR. **Analise de desempenho de Bancos de Dados**. Barbacena: Unipac, 2016.

FILHO, Marcos André. **SQL X NoSQL:** Análise de desempenho do uso do MongoDB em relação ao uso do PostgreSQL. Recife: UFPE, 2015.

GREENWALD; STACKOWIAK; STERN. **Oracle Essentials.** Oracle Database 11g**.** 4 ed. Sebastopol, CA, 2008.

HEUSER, Carlos Alberto. **Projeto de banco de dados**. 6. ed. Porto Alegre: Bookman, 2009.

LAUDON, Kenneth; LAUDON, Jane. **Sistemas de informação gerencial**. 9. ed. São Paulo: Pearson, 2011.

LAURINDO et al. **O papel da tecnologia da informação (ti) na estratégia das organizações**. In: Gestão & Produção v.8, n.2, p.160-179. São Paulo: USP, 2001.

LIMA, Evandro Cezar. **Banco de Dados**. 2011.

LÓSCIO; OLIVEIRA; PONTES. **NoSQL no desenvolvimento de aplicações Web colaborativas**. VIII Simpósio Brasileiro de Sistemas Colaborativos, v. 10, n. 1, p. 11, 2011.

MACHADO, Carlos Augusto. **Estudo sobre a otimização de desempenho em banco de dados PostgreSQL**. Lages: UNIPLAC, 2010.

MAIELLO, Pollyanna Espinosa. **Comparação dos sistemas gerenciadores de bancos de dados oracle e postgresql com o uso da ferramenta benchmarksql**. Araraquara: Uniara, 2016.

MICROSOFT. **SQL SERVER**. Disponível em: <https://www.microsoft.com/pt-br/sql-server>. Acesso em: 15 de maio de 2019.

MILANI, André. **PostgreSQL-Guia do Programador**. Novatec Editora, 2008.

ORACLE. **Produtos de Banco de Dados Oracle**.
Disponível em: <https://www.oracle.com/ br/database/products-a-z>. Acesso em: 21 de maio de 2019.

ORACLE. **Database Concepts.** Disponível em: <https://docs.oracle.com/en/database/oracle/oracle-database/19/cncpt/introduction-to-oracle-database.html#GUID-A42A6EF0-20F8-4F4B-AFF7-09C100AE581E>. Acesso em: 10 de maio de 2019.

PEROVANO, D. G. **Manual de metodologia científica para a segurança pública e defesa social**. Curitiba: Juruá, 2014.

PIRES; NASCIMENTO; SALGADO. **Comparativo de desempenho entre bancos de dados de código aberto**. Recife: UFPE, 2009.

PRESSMAN; MAXIM. **Engenharia de software**. 8. ed. São Paulo: AMGH, 2016.

RED HAT. **O que é armazenamento definido por software?** Disponível em: https://www.redhat.com/pt-br/topics/data-storage/software-defined-storage. Acesso em: 25 de junho de 2021.

ROCHA; DIAS. **Importância do banco de dados nas aplicações**. Umuarama: UNIPAR, 2015.

SAKIS, Isabella. **MongoDB: Uma introdução ao NoSQL**. Disponível em: <http://coral.ufsm.br/pet-si/index.php/mongodb-uma-introducao-ao-nosql/>. Acesso em: 13 de maio de 2019.

SILBERSCHATZ; KORTH; SUNDARSHAN. **Sistema de banco de dados.** 6. ed. Rio de Janeiro: Elsevier, 2016.

SOARES, Jhonathan. **O que é MongoDB e porque usá-lo?** Disponível em: <https://codigosimples.net/2016/03/01/o-que-e-mongodb-e-porque-usa-lo/>. Acesso em: 15 de maio de 2019.

SOMMERVILLE. **Engenharia de software**. 9. ed. São Paulo: Pearson, 2011.

TIINSIDE. **Gartner anuncia que os gastos globais de TI crescerão 1,1% em 2019**. Disponível em: <https://tiinside.com.br /tiinside/22/04/2019/ gartner-anuncia-que-os-gastos-globais -de-ti-crescerao-11-em-2019>. Acesso em: 05 de junho de 2019.

TOTH, Renato Molina. **Abordagem NoSQL–uma real alternativa.** Sorocaba, São Paulo, Brasil: Abril, v. 13, 2011.

WAINER et al. **Métodos de pesquisa quantitativa e qualitativa para a Ciência da Computação**. Atualização em informática, v. 1, p. 221-262, 2007.