BENCHMARKING DE BANCOS DE DADOS PARA APLICAÇÕES COMPLEXAS: UMA COMPARAÇÃO ENTRE POSTGRESQL, SQL SERVER E MYSQL

BENCHMARKING OF DATABASES FOR COMPLEX APPLICATIONS: A COMPARISON BETWEEN POSTGRESQL, SQL SERVER, AND MYSQL

Gustavo Oliveira da Cunha[1], [i]
Thoris Angelo Pivetta[2], ii
André Cassulino Araujo Souza[3], iii

RESUMO

Este estudo descreve um experimento inicial de benchmark em diferentes bancos de dados para aplicações complexas, com o objetivo de fornecer uma comparação detalhada entre PostgreSQL, SQL Server e MySQL. A pesquisa foca em aspectos críticos como escalabilidade, tempo de resposta e consumo de recursos, avaliando como essas soluções se comportam em operações de leitura, escrita e consulta em grandes volumes de dados. Para esta finalidade, tecnologias e ferramentas de mercado foram utilizadas, incluindo a linguagem de programação C# e .NET Core para desenvolvimento de Web APIs e Apache JMeter para testes massivos. A relevância deste estudo reside na possibilidade de redução de custos através de soluções de código aberto, especialmente considerando a parcela significativa dos gastos com infraestrutura associados aos bancos de dados. Os resultados preliminares indicam diferenças significativas no desempenho de cada banco de dados, oferecendo insights práticos para a possibilitar a tomada de decisão sobre qual a solução mais adequada para infraestruturas complexas e de alta demanda.

Palavras-chave: Benchmark. Bancos de Dados. Postgre SQL. MySQL. SQL Server.

ABSTRACT

This study describes an initial benchmark experiment on different databases for complex applications, aiming to provide a detailed comparison between PostgreSQL, SQL Server, and MySQL. The research focuses on critical aspects such as scalability, response time, and resource consumption, evaluating how these solutions perform in reading, writing, and querying operations with large volumes of data. For this purpose, market-leading technologies and tools were used, including C# programming language and .NET Core for Web API development, and Apache JMeter for load testing. The

[1] Graduando em Análise e Desenvolvimento de Sistemas na Faculdade SENAI de Tecnologia Gaspar Ricardo Junior. E-mail: gustavooliveiracunha812@gmail.com
[2] Engenheiro de Aplicação e Mestre em Engenharia da Computação da 2RPNET. E-mail: thoris.pivetta@2rpnet.com
[3] Docente e Mestre em Ciência da Computação da Faculdade de Tecnologia SENAI Gaspar Ricardo Junior. E-mail: andre.souza@sp.senai.br

relevance of this study lies in the potential cost reduction through open-source solutions, especially considering the significant portion of infrastructure expenses related to databases. Preliminary results indicate significant differences in the performance of each database, offering practical insights to aid decision-making on the most suitable solution for complex and high-demand infrastructures.

Keywords: Benchmark. Databases. Postgre SQL. MySQL. SQL Server.

1 INTRODUÇÃO

Atualmente, empresas demandam cada vez mais soluções complexas para resolver desafios do dia a dia, porém enfrentam obstáculos significativos devido aos custos elevados relacionados aos equipamentos e processos necessários para implementar tais soluções. Para reduzir custos sem comprometer a qualidade, muitas empresas adotam componentes de software de código aberto. Como Lerner e Tirole (2002) destacam, esses componentes, desenvolvidos pela comunidade global de desenvolvedores, são gratuitos e podem ser integrados às aplicações, economizando tempo e recursos. Além de escolher componentes de software adequados, a seleção de um Sistema de Gerenciamento de Banco de Dados (SGBD) adequado para cada cenário é igualmente importante. Yedilkhan et al. (2023) ressaltam que cada SGBD tem arquiteturas e funcionalidades específicas, projetadas para atender a diferentes requisitos. Portanto, é essencial tomar decisões estratégicas baseadas nas especificidades do ambiente em que essas soluções serão aplicadas, garantindo a eficiência e a eficácia do sistema como um todo.

1.1 Problema de pesquisa

A disparidade entre os custos elevados de infraestrutura resulta em altos custos mensais para as empresas, impactando negativamente sua sustentabilidade e eficiência operacional. Essa situação pode levar a uma pressão financeira excessiva, prejudicando a capacidade das empresas de investir em outras áreas estratégicas e comprometendo sua competitividade no mercado.

1.2 Objetivo(s)

1.2.1 Objetivo Geral

O objetivo geral deste estudo é realizar o benchmark de bancos de dados de código aberto para avaliar a viabilidade em aplicações complexas. Serão identificados características, desempenho e limitações desses bancos em comparação com soluções proprietárias, buscando compreender sua adequação às demandas específicas das empresas de forma econômica.

1.2.2 Objetivos Específicos

- Revisar literatura sobre benchmarking de bancos de dados;
- Analisar bancos de dados de código aberto ou de custo acessível amplamente empregados;
- Desenvolver critérios de avaliação considerando desempenho, escalabilidade, confiabilidade, segurança e complexidade;
- Realizar testes de benchmarking com conjuntos de dados representativos;
- Comparar resultados, contrastando desempenho e características com soluções proprietárias.

1.3 Contribuição

O benchmarking de bancos de dados é crucial para a competitividade empresarial, pois permite a redução de custos operacionais ao adotar soluções de código aberto, direcionando recursos para inovação, e evita investimentos em colocar tecnologias com muito poder computacional para soluções menores, economizando tempo e recursos.

2 REVISÃO DE LITERATURA

2.1 Escolha Adequada de SGBD para Cenários Variados

Yedilkhan et al. (2023) ressaltam que a seleção do SGBD deve considerar suas arquiteturas e funcionalidades específicas, já que são projetados para atender a requisitos diversos. Por exemplo, o PostgreSQL é ideal para consultas complexas e transações ACID (Atomicidade, Consistência, Isolamento e Durabilidade), enquanto o MongoDB é mais indicado para aplicativos que necessitam de escalabilidade e flexibilidade no esquema de dados. Os testes do experimento foram feitos em cenários de leitura e gravação de dados em larga escala, que possibilitou análise de fatores como tempo de resposta e escalabilidade, que é utilizado como insight para melhores tomadas de decisão.

2.2 Impacto da Integração de Algoritmos no Desempenho do PostgreSQL

Ao explorarem a integração de técnicas de mineração de dados no PostgreSQL, Viloria et al. (2019) destacam a relevância do estudo na melhoria de desempenho do SGBD, pois não apenas melhora os tempos de resposta, mas também aumenta a eficiência na obtenção de resultados. Além de evitar a necessidade de exportação de dados para ferramentas externas.

2.3 Flexibilidade e Escalabilidade com ASP.NET Core

O ASP.NET Core, conforme descrito por Lock (2023), é uma plataforma flexível e escalável para desenvolvimento. Ele oferece opções de API (Interface de Programação de Aplicações) como Minimal APIs, MVC Controllers e Blazor Server para atender às diversas necessidades de desenvolvimento, além

de outras opções como gRPC APIs, Blazor WebAssembly e Razor Pages para ampliar suas capacidades. Essas possibilidades em conjunto com uma arquitetura de software robusta e que faz uso de design patterns, como Dependency Injection (DI), Object Relational Mapper (ORM) e Repository permitem a criação de sistemas complexos e de alto desempenho.

2.4 Avaliação do Tempo de Resposta em Aplicações Web

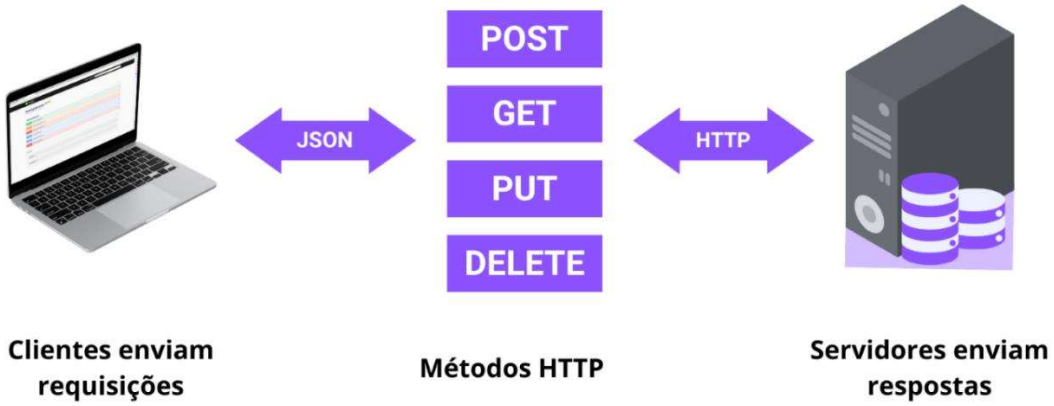
Considerar várias métricas é crucial ao avaliar o tempo de resposta de uma aplicação web, conforme destacado por Matam e Jain (2017). O livro deles serve como um guia abrangente para configurar testes de carga e estresse utilizando o Apache JMeter, abordando elementos essenciais como Thread Groups, Listeners e Assertions. Essas práticas têm implicações diretas nos benchmarks de bancos de dados, pois os testes simulam chamadas na API que realizam operações de leitura e escrita no sistema de gerenciamento de banco de dados (SGBD). À medida que o número de chamadas aumenta, o tempo de resposta é medido, permitindo a identificação de gargalos e pontos de melhoria tanto na API quanto na gestão do SGBD escolhido.

3 METODOLOGIA

3.1 Web apis com C# e Visual Studio como IDE

Foram desenvolvidos projetos de Web API utilizando C# no Visual Studio Community 2022 devido à versatilidade e segurança de tipos, conforme destacado por Jesse Liberty (2005). A escolha do Visual Studio Community foi motivada pela sua gratuidade e pelos recursos avançados, como edição de código inteligente e suporte para controle de versão, conforme destacado pela Microsoft (2024). As Web APIs foram selecionadas devido à facilidade de implementação de operações POST e GET, comunicação entre cliente e servidor via HTTP/HTTPS e suporte para troca de dados em formato JSON (Notação de Objeto JavaScript), como mencionado na documentação da Microsoft (2024).

Figura 1 - Esquema simples da comunicação de API com métodos HTTP.



Fonte: Elaborado pelo autor (2024).

3.2 Bancos de Dados selecionados para o experimento

SQL Server, PostgreSQL e MySQL são os três bancos de dados escolhidos para esse levantamento. São amplamente utilizados no mercado atual, cada um com características e usos distintos.

- SQL Server Express, desenvolvido pela Microsoft (2024) e segundo ela mesma, é um sistema gerenciador de banco de dados relacional utilizada para realizar o estudo e desenvolver aplicações web simples e pequenos servidores, além de ser gratuita. Oferece facilidade de integração a outras ferramentas da Microsoft, robustez e segurança avançada, sendo ideal para aplicações corporativas complexas.
- Postgre SQL, de acordo com a documentação da empresa The PostgreSQL Global Development Group (2024), é um sistema gerenciador de banco de dados relacional de código aberto, que é empregado para a gestão e armazenamento de informações em várias esferas empresariais.
- MySQL Community Edition também é um sistema gerenciador de banco de dados relacional de código aberto. De acordo com o MySQL TM (2024), destaca-se pela facilidade de uso, desempenho rápido, escalabilidade e suporte ao NoSQL (Banco de Dados Não Relacional), sendo muito utilizado em diversas aplicações comerciais. Isso torna-o muito popular entre outras soluções do mercado atual.

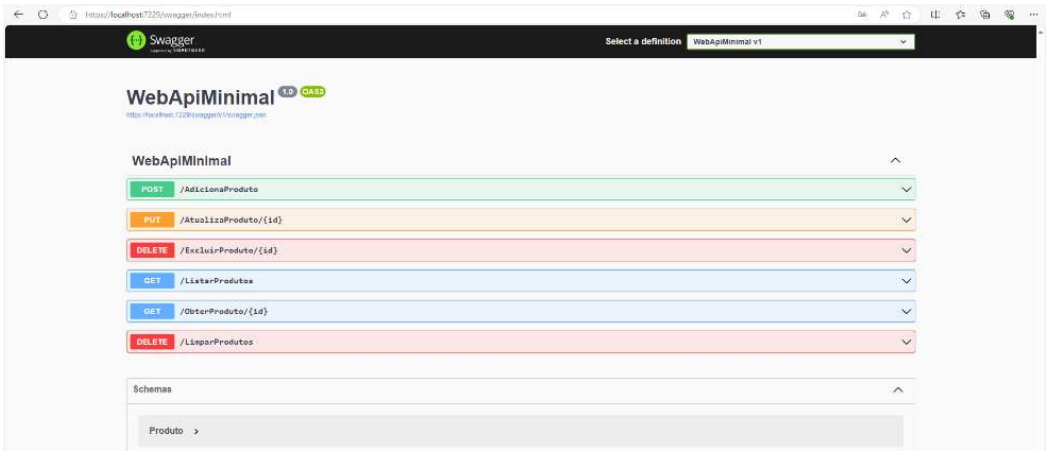
3.3 Conexão com banco de dados e Entity framework como ORM

O código fonte criado foi reutilizado em diferentes bancos de dados por meio da connection string. O acesso aos dados foi simplificado ao utilizar o Entity Framework (EF) como ORM. Segundo informações da página oficial do EF (2024), esta ferramenta tem objetivo de mapear classes para tabelas do banco, oferecendo uma abstração de alto nível para lidar com dados de forma orientada a objetos. Essa integração do Entity Framework na web API facilita o desenvolvimento e manutenção do sistema, oferecendo um desempenho eficiente e confiável na interação com o banco de dados.

3.4 Swagger como framework para documentação de API

O Swagger foi implantado para melhorar a documentação e a facilidade de uso da web API. Conforme Christoph Nienaber (2024) descreveu na própria página de documentação dessa ferramenta, ela permite aos desenvolvedores visualizar, testar e compreender facilmente os endpoints e parâmetros da API de forma interativa, reduzindo o tempo necessário para entender sua funcionalidade. É possível ver na imagem abaixo um exemplo dessa interface já implementada.

Figura 2 - Interface criada pelo Swagger para consumo da Web-API.

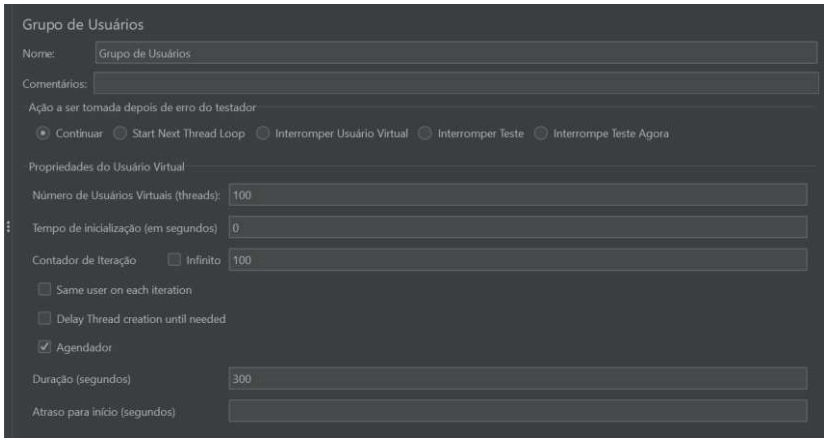


Fonte: Elaborado pelo autor (2024).

3.5 Configuração do Jmeter para testes em massa

O Apache JMeter, de acordo com a página da Apache Software Foundation (2024), é uma ferramenta de teste de desempenho de código aberto amplamente utilizada e ainda é uma das melhores opções do mercado. Ele foi selecionado devido à sua compatibilidade com tecnologias modernas, suporte da comunidade e adequação aos requisitos do projeto, pois permite testes com requisições em massa utilizando HTTP (Protocolo de Transferência de Hipertexto) para as requisições de GET e POST, sendo essencial para os benchmarks dos bancos de dados neste experimento.

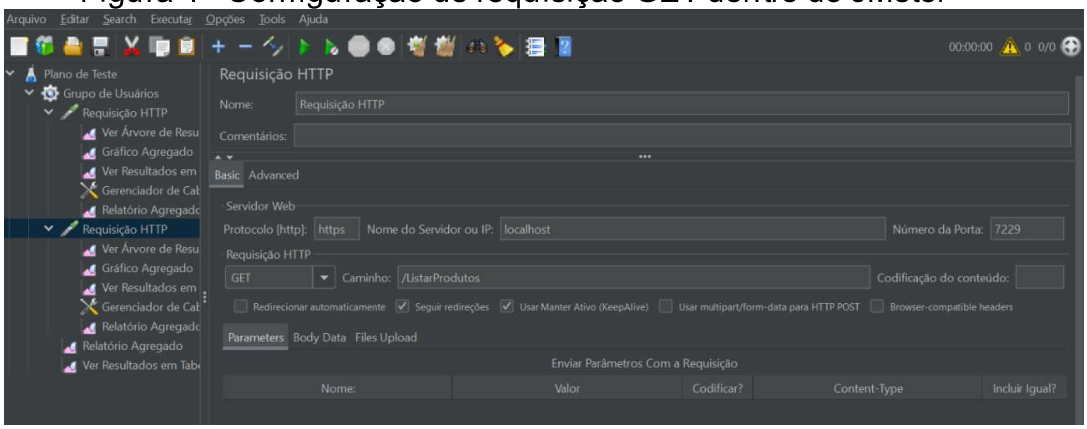
Figura 3 - Configuração do “Grupo de Usuários” dentro do JMeter.



Fonte: Elaborado pelo autor (2024).

Nesse exemplo podemos ver a configuração de uma requisição HTTP do método GET, onde foi configurado o tipo de protocolo, nome do servidor ou IP, caminho e o tipo de requisição.

Figura 4 - Configuração de requisição GET dentro do JMeter



Fonte: Elaborado pelo autor (2024).

3.6 Escopo seguido para os testes

Para que houvesse a possibilidade de análise comparativa entre os resultados obtidos, foi necessário que todos os testes realizados tivessem um escopo definido. Com isso, esses parâmetros foram configurados no item" Grupo de Usuários" do JMeter.

Tabela 1 – Parâmetros configurados para o teste.

PARÂMETROS CONFIGURADOS	INFORMAÇÕES ESPECÍFICAS
Nº de Threads simultâneos	100 Threads
Nº de requisições por Thread	10 Requisições
Métodos HTTP	GET e POST
Status do Banco de Dados	Tabela com 10.000 linhas
Métrica avaliada	Tempo de resposta (ms)
Tempo limite	10 minutos
Mesmo thread fazendo requisição	Não
Tempo de inicialização entre um thread e outro	0 segundos

Fonte: Elaborado pelo autor (2024).

3.7 Hardware e Software da máquina utilizada

Antes de proceder com qualquer teste, é importante destacar que todo o processo foi conduzido em um ambiente local, utilizando um desktop pessoal. As tabelas a seguir contém informações detalhadas sobre os componentes de hardware e software utilizados, com o objetivo de permitir a replicação dos experimentos e facilitar comparações de desempenho.

Tabela 2 – Informações sobre o hardware utilizado.

HARDWARE	CARACTERÍSTICAS ESPECÍFICAS
1x CPU - AMD Ryzen 5500	Núcleos de CPU: 6 Threads: 12 Clock base: 3.6 GHz Clock de Max Boost: Até 4.2 GHz Cache L2 total: 3 MB Cache L3 total: 16MB
2x Memória RAM - Husky 8GB (Total 16GB)	Velocidade: 2666MHz Formato: DIMM (Non-ECC) Tecnologia:DDR4 Pinos: 260
1x SSD M.2 Wester Digital Black 500GB	Interface: PCIe Gen3 8 Gb / s Até 3.400 MB / s Leitura Até 2.500 MB / s Gravação Capacidade: 500 GB

Fonte: Elaborado pelo autor (2024).

Tabela 3 – Informações sobre o software utilizado.

SOFTWARE	VERSÃO
Windows 10	64 bits (Atualizado)
Visual Studio Community 2022	Versão 17.8.5
Apache Jmeter	Versão 5.6.2
Excel	Versão online Microsoft365

Fonte: Elaborado pelo autor (2024).

3.8 Tempo de resposta como métrica de avaliação

O Tempo de Resposta, que equivale ao intervalo entre uma solicitação e sua resposta, foi escolhido como métrica inicial devido à sua importância na análise dos dados. Para uma visão abrangente, foram utilizados percentis e quartis, que dividem os dados. Outras métricas, como taxa de transferência e utilização de recursos, também podem ser consideradas em estudos com o mesmo objetivo.

3.9 Processo de Teste e Excel para extrair resultados

Com tudo configurado, podemos executar a API e iniciar as requisições massivas no JMeter. A aplicação produzirá uma tabela com milhares de dados de teste. Como citado anteriormente, a métrica selecionada para análise é o tempo de resposta, que retorna valores de tempo em milissegundos (ms). Foi escolhido Excel para construção de gráficos, pois tem uma interface intuitiva e uma variedade de recursos poderosos que o tornam uma ferramenta versátil para analisar de maneira detalhada os resultados produzidos no Jmeter. Segundo a página oficial do Excel no site da Microsoft (2024), ele é uma planilha eletrônica amplamente utilizada para análise de dados, criação de gráficos, manipulação de informações numéricas e gerenciamento de projetos.

4 RESULTADOS E DISCUSSÕES

4.1 Tabelas com dados dos testes

Os dados foram coletados e processados através da extração de uma coluna específica de cada teste, que representa o tempo de resposta. Com base nessas colunas, foi criada as tabelas abaixo, que inclui esses dados e representa os bancos de dados analisados.

Tabela 4 – Fragmento da tabela com tempo de resposta com método POST.

Tempo de resposta com método POST (ms)		
SQL Server	Postgre SQL	My SQL
176	320	111
204	317	121
199	312	122
194	315	131
193	309	127
200	346	129
210	336	132
214	361	131

Fonte: Elaborado pelo autor (2024).

Tabela 5 – Fragmento da tabela com tempo de resposta com método GET.

Tempo de resposta com método GET (ms)		
SQL Server	Postgre SQL	My SQL
586	2189	96

598	2194	171
602	1960	260
611	2001	262
608	2001	279
621	2020	283
631	1969	296
622	2001	319

Fonte: Elaborado pelo autor (2024).

4.2 Tabelas de elementos para construção dos gráficos *boxplot*

Foram calculados quartis, valor máximo, mediana e valor mínimo utilizando fórmulas do Excel com base nos dados anteriores. Observou-se a presença de valores discrepantes, especialmente nos quartis, entre os diferentes bancos de dados. Essa análise comparativa é crucial para avaliar o benchmark e identificar diferenças significativas, auxiliando na escolha do banco de dados mais adequado e compreendendo melhor suas limitações.

Tabela 6 – Elementos presentes no gráfico boxplot do teste de GET.

ELEMENTOS DO BOXPLOT DE GET			
Dados	**SQL Server**	**Postgre SQL**	**My SQL**
Mínimo	48	73	20
Q3	1443	1787	1237
Mediana	1191	1545	912
Q1	923	1311	608
Máximo	2674	3659	2993

Fonte: Elaborado pelo autor (2024).

Tabela 7 – Elementos presentes no gráfico *boxplot* do teste de *POST*.

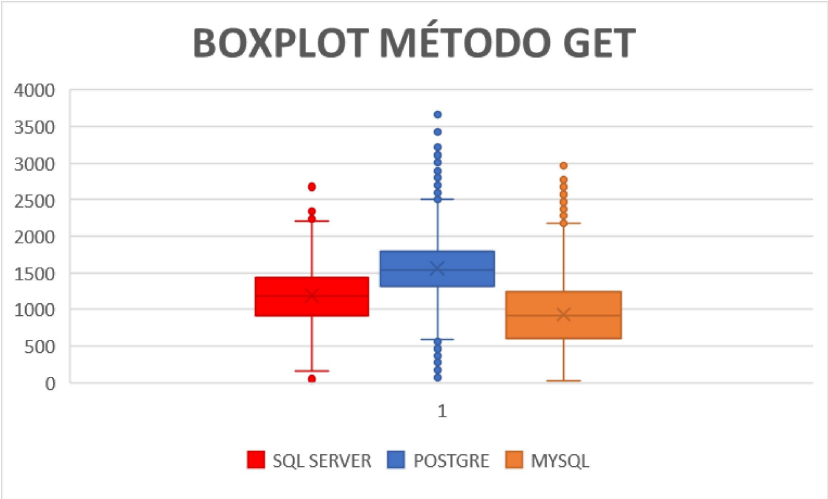
ELEMENTOS DO BOXPLOT DE POST			
Dados	**SQL Server**	**Postgre SQL**	**My SQL**
Mínimo	9	42	15
Q3	155	896	240
Mediana	129	824	188
Q1	108	763	144
Máximo	326	1546	700

Fonte: Elaborado pelo autor (2024).

4.3 Gráficos *boxplot*

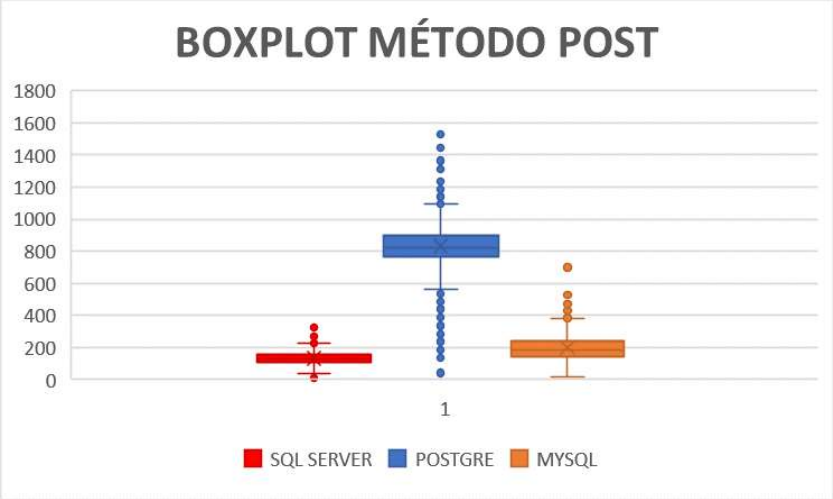
Com os cálculos realizados, tornou-se possível a criação dos gráficos *boxplot*, que facilitou a análise detalhada de seus componentes, incluindo a localização da mediana, a detecção de valores atípicos, as propriedades das caudas e a distribuição dos quartis. Na figura seguinte, esses aspectos podem ser visualizados de forma clara e concisa.

Gráfico 1 – *Boxplot* com resultados dos testes com método GET.



Fonte: Elaborado pelo autor (2024).

Gráfico 2 – *Boxplot* com resultados dos testes com método *GET*.



Fonte: Elaborado pelo autor (2024).

4.4 Resultados Obtidos

Os resultados dos testes de GET e POST indicam comportamentos distintos entre os DBMS, revelando suas forças e fraquezas em diferentes

cenários operacionais:

4.4.1 Teste de GET

O MySQL se destacou em termos de tempo de resposta geral, evidenciando sua eficiência em operações de leitura, sendo uma escolha adequada para aplicações que realizam consultas constantes de grandes volumes de dados. No entanto, sua maior dispersão de dados, em comparação com o PostgreSQL e SQL Server, pode indicar uma variabilidade de performance em cenários de alta demanda.

4.4.2 Teste de POST

No caso de operações de escrita, o SQL Server apresentou o melhor desempenho, com tempos de resposta menores e menor dispersão de valores, inclusive dos outliers. Este resultado reforça sua adequação para cenários que exigem alta consistência e baixa latência, como sistemas financeiros ou de transações críticas. A menor dispersão também indica que o SQL Server oferece uma previsibilidade maior em termos de performance, tornando-o uma solução confiável para operações de escrita.

4.4.3 Análise de Outliers:

Os outliers detectados, principalmente no PostgreSQL, indicam que em determinadas cargas de trabalho ou configurações, há variações de performance que merecem atenção. Esses valores discrepantes podem resultar de queries mal otimizadas, concorrência elevada ou até limitações momentâneas do hardware.

5 CONCLUSÃO

O estudo alcançou o seu objetivo de iniciar uma investigação simplificada nas análises comparativas de banco de dados, com o propósito de identificar alternativas viáveis às soluções comerciais predominantes. Adicionalmente ao seu objetivo declarado, o experimento proporcionou uma ampla gama de informações relacionadas a tecnologias, métodos de análise, design patterns, interfaces de programação de aplicativos em C#, estruturas de documentação, ORM, boas práticas de desenvolvimento de software e conhecimento abrangente de diversos sistemas de bancos de dados amplamente reconhecidos internacionalmente.

5.1 Estudos futuros

- Troca do App Service da Azure por Docker: Avaliar a migração do App Service da Azure para Docker para determinar benefícios em escalabilidade, desempenho, custo e gerenciamento eficiente.
- Análise de Outra Linguagem Equivalente a C#: Comparar TypeScript,

JavaScript com Node.js e outras linguagens alternativas com C#.

- Possibilidade de Banco de Dados Não Relacional: Explorar bancos de dados não relacionais, como documentos ou grafos como alternativas.
- Complexidade de análise com outras métricas: Realizar uma comparação de resultados utilizando métricas alternativas para benchmark do desempenho de bancos de dados, tais como latência e escalabilidade.

6 REFERÊNCIAS

CAMARGO ACUÑA, Genesis Yulie et al. Integration of data mining techniques to PostgreSQL database manager system. 2019.

CHRISTOPH, Nienaber. ASP.NET Core web API documentation with Swagger / OpenAPI. Disponível em: https://learn.microsoft.com/en-us/aspnet/core/tutorials/web-api-help-pages-using-swagger?view=aspnetcore-8.0. Acesso em: 14 fev. 2024.

LIBERTY, Jesse. Programming C#: building .NET applications with C#. "O'Reilly Media, Inc.", 2005.

LOCK, Andrew. ASP.NET Core in Action. Simon and Schuster, 2023.

MATAM, Sai; JAIN, Jagdeep. Pro Apache JMeter: web application performance testing. Apress, 2017.

MICROSOFT. Documentação do ASP.NET. Disponível em: https://learn.microsoft.com/pt-br/aspnet/core/?view=aspnetcore-8.0. Acesso em: 20 fev. 2024.

MICROSOFT. Hub de documentação do Entity Framework. Disponível em: https://learn.microsoft.com/pt-br/ef/. Acesso em: 19 fev. 2024.

MICROSOFT. SQL Server technical documentation. Disponível em: https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16. Acesso em: 16 fev. 2024.

MICROSOFT. Visual Studio Community. Disponível em: https://visualstudio.microsoft.com/vs/community/. Acesso em: 16 fev. 2024.

MICROSOFT. Microsoft Excel. Disponível em: https://www.microsoft.com/pt-br/microsoft-365/excel. Acesso em: 21 fev. 2024.

MYSQL TM. MySQL Community Edition. Disponível em: https://www.mysql.com/products/community/. Acesso em: 20 fev. 2024.

SCHWERTMAN, Neil C. et al. A simple more general boxplot method for identifying outliers. Computational Statistics & Data Analysis, v. 47, n. 1, p. 165-174, 2004.

THE APACHE SOFTWARE FOUNDATION. Apache JMeter™. Disponível em: https://jmeter.apache.org/index.html. Acesso em: 10 fev. 2024.

THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. PostgreSQL Documentation. Disponível em: https://www.postgresql.org/docs/current/. Acesso em: 15 fev. 2024.

YEDILKHAN, Didar et al. Performance analysis of scaling NoSQL vs SQL: a comparative study of MongoDB, Cassandra, and PostgreSQL. In: 2023 IEEE International Conference on Smart Information Systems and Technologies (SIST). IEEE, 2023. p. 479-483.

AGRADECIMENTOS

Agradeço ao Senai - Gaspar Ricardo Júnior - Sorocaba/SP, e a Empresa 2RP NET pelo apoio nesta pesquisa.

Sobre os autores:

---

[i] **GUSTAVO OLIVEIRA DA CUNHA (Aluno)**

Possui graduação Técnica em Mecatrônica pelo INSTITUTO FEDERAL DE SÃO PAULO (2021) e atualmente cursa a graduação em Tecnologia de Análise e Desenvolvimento de Sistemas pela FACULDADE DE TECNOLOGIA SENAI GASPAR RICARDO JÚNIOR. Atualmente é estagiário como desenvolvedor de software na empresa 2RP Net LTDA.

[ii] **THORIS ANGELO PIVETTA (Orientador)**

Possui graduação em Análise de Sistemas pela faculdade Universidade Paulista (2002) com especialização em Automação Industrial pela Faculdade Universidade de Taubaté (2006), e Mestrado (2016) em Engenharia da Computação pela Instituto Tecnológico de Aeronáutica. Atualmente é Engenheiro de Aplicação contratado da Empresa 2RPNet.

[iii] **ANDRÉ CASSULINO ARAUJO SOUZA (Coorientador)**

Possui graduação em Análise e Desenvolvimento de Sistemas pelo Instituto Federal de São Paulo (2012) e mestrado em Ciência da Computação pela Universidade Federal de São Carlos (2017). Atualmente é professor de ensino superior - SENAI - Departamento Regional de São Paulo. Tem experiência na área de Ciência da Computação, com ênfase em Desenvolvimento de Sistemas.