Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Aplicação *web* de *GUI* para SGDBs e análise de desempenho

Gustavo Nogai Saito

Monografia Final

mac 499 — Trabalho de Formatura Supervisionado

Supervisora: Prof.ª Dr.ª Ana Cristina Vieira de Melo

São Paulo
2024

Generating a system architecture is not a deterministic process. It requires careful consideration of business requirements, technology choices, existing infrastructure and systems, and actual physical resources, such as budget and manpower.
— Andrew Holdsworth, Oracle9i Database Performance Planning

Agradecimentos

Aos professores do IME-USP, que criaram em mim a base de conhecimento necessária para a elaboração e execução deste trabalho.

À professora Kelly, que ofereceu a matéria MAC0426 - Sistemas de Bancos de Dados, me auxiliando no conhecimento necessário para este TCC, e à professora Ana, pela orientação.

Aos meus colegas durante a faculdade, ajudando uns aos outros durante trabalhos em grupo e aprendendo juntos.

E, principalmente, à minha família, que investiu em meus estudos, me possibilitando mudar de cidade para cursar a faculdade.

Resumo

Gustavo Nogai Saito. **Aplicação *web* de *GUI* para SGDBs e análise de desempenho**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2024.

Sistemas de gerenciamento de banco de dados (SGBDs) são softwares que permitem a organização, armazenamento e manipulação de dados, os quais são armazenados em bancos de dados. Por meio de uma interface gráfica do utilizador (GUI) de uso simples e intuitivo, é possível facilmente testar e interagir com os dados, gerando e modificando-os, bem como executando diversas operações, como, por exemplo, busca, inserção, exclusão e atualização, não necessitando a criação local de um ambiente de desenvolvimento de banco de dados, que pode ser complexo e custoso. Os SGDBs implementados e analisados neste estudo foram o PostgreSQL e o MySQL, escolhidos devido à sua ampla adoção tanto no uso pessoal quanto profissional. A comparação entre os dois sistemas foi realizada com base no desempenho em termos de tempo de execução. Os resultados indicaram que os SGDBs apresentaram desempenho comparável na inserção e na consulta, enquanto o PostgreSQL apresentou melhor desempenho na remoção e o MySQL se sobressaiu na alteração, conforme evidenciado por testes empíricos realizados neste trabalho.

Palavras-chave: Sistema de Gerenciamento de Banco de Dados (SGBDs). Interface Gráfica do Utilizador (GUI). Análise de Desempenho. Aplicação *Web*.

Abstract

Gustavo Nogai Saito. **Web application for GUI for Database Management Systems and performance analysis**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2024.

Database Management Systems (DBMS) are software applications that enable the organization, storage, and manipulation of data, which is stored in databases. Through a user-friendly and intuitive Graphical User Interface (GUI), it is possible to easily test and interact with the data, generating and modifying it, as well as executing various operations such as searching, inserting, deleting, and updating, without the need to create a local database development environment, which can be complex and costly. The DBMS evaluated and analyzed in this study were PostgreSQL and MySQL, chosen due to their widespread adoption in both personal and professional use. The comparison between the two systems was conducted based on performance in terms of execution time. The results indicated that the DBMS exhibited comparable performance in insertion and querying, while PostgreSQL demonstrated better performance in deletion, and MySQL excelled in modification, as evidenced by empirical tests carried out in this work.

Keywords: Database Management System (DBMS). Graphical User Interface (GUI). Performance Analysis. Web Application.

Lista de figuras

Lista de tabelas

Lista de programas

Sumário

Introdução

Os Sistemas de Gerenciamento de Banco de Dados (SGBDs) desempenham um papel fundamental na organização, armazenamento e manipulação de dados, sendo essenciais para o funcionamento eficiente de diversas aplicações modernas. Com a crescente quantidade de informações geradas e a necessidade de acessá-las rapidamente, a escolha do SGBD adequado se torna uma decisão estratégica para desenvolvedores e empresas.

Este trabalho se propõe a criar um aplicativo fullstack com uma interface gráfica do utilizador (GUI) a fim de possibilitar o uso simples e intuitivo das ferramentas de gerenciamento de banco de dados PostgreSQL[1] e MySQL,[2] sendo ambas escolhidas devido a sua popularidade e adoção em ambientes pessoais e profissionais devido às suas características robustas e escalabilidade. Com esta aplicação é possível facilmente testar e interagir com os dados, gerando e modificando-os, bem como executando diversas operações, como busca, inserção, exclusão e atualização utilizando a linguagem SQL, eliminando a necessidade de criar localmente um ambiente de desenvolvimento de banco de dados, que pode ser complexo e custoso.

Além disso, foi feita uma análise direcionada ao ponto de vista de um usuário casual interagindo com os dados, ou seja, execuções de consultas individuais com tempo de espera para usuários humanos, buscando entender as diferenças de desempenho entre os dois sistemas, especialmente em relação ao tempo de execução das operações básicas: inserção, consulta, remoção e alteração. A escolha do PostgreSQL e do MySQL não foi aleatória, pois ambos oferecem soluções distintas que podem atender a diferentes necessidades, desde aplicações simples até sistemas complexos que requerem alta integridade dos dados.

Os objetivos deste estudo incluem:

- Analisar o desempenho dos SGBDs PostgreSQL e MySQL em operações comuns.
- Identificar as vantagens e desvantagens de cada sistema em diferentes cenários de uso.
- Fornecer recomendações para desenvolvedores sobre qual SGBD utilizar com base em suas necessidades específicas.

A abordagem adotada consiste na implementação de um aplicativo completo, desenvolvido do zero, que abrange tanto o frontend (utilizando HTML, JavaScript e CSS) quanto o backend (com Node.js e Express.js). Testes empíricos foram realizados para comparar o

[1] Site oficial do PostgreSQL: https://www.postgresql.org/

[2] Site oficial do MySQL: https://www.mysql.com/

desempenho dos dois SGBDs em termos de tempo de execução durante operações CRUD (Criar, Ler, Atualizar e Deletar).

Os principais resultados indicaram que, embora ambos os SGBDs apresentem desempenho comparável nas operações de inserção e consulta, o PostgreSQL se destacou na remoção, enquanto o MySQL teve melhor desempenho nas alterações. Essas descobertas são relevantes para desenvolvedores que buscam otimizar suas aplicações com base nas características específicas de cada sistema.

A organização deste trabalho está estruturada da seguinte forma: no Capítulo 1, será apresentado um panorama teórico sobre Sistemas de Gerenciamento de Banco de Dados. Já no Capítulo 2, será descrito o desenvolvimento da aplicação completa. Então, no Capítulo 3 serão detalhados os testes de desempenho realizados. Por fim, no Capítulo 4 será feita uma análise dos resultados obtidos. Então, no Capítulo 5, será discutido trabalhos futuros que podem ser implementados considerando o sistema já desenvolvido, com a conclusão no Capítulo 6. Cada seção visa fornecer uma compreensão abrangente do tema abordado, concluindo com uma síntese sobre a aplicação e os resultados obtidos no capítulo final.

Capítulo 1

Sistemas de Gerenciamento de Banco de Dados (SGBDs)

1.1 Definição

Os Sistemas de Gerenciamento de Banco de Dados (SGBDs) são *softwares* projetados para gerenciar, organizar e manipular dados armazenados em bancos de dados. Eles atuam como intermediários entre os usuários e os dados, permitindo a criação, leitura, atualização e exclusão de informações de maneira eficiente e segura. O SGBD é responsável por assegurar a integridade, segurança e consistência dos dados, além de facilitar o acesso a eles através de interfaces que geralmente utilizam a linguagem SQL (*Structured Query Language*) para operações em bancos de dados relacionais.

Os SGBDs podem ser classificados em diferentes tipos, incluindo:

- **Relacionais**: Neste estudo o foco será somente nos SGBDs relacionais, que são os mais utilizados em sistemas reais. Neste tipo, o armazenamento de dados é feito em tabelas que podem se relacionar entre si. Por exemplo o MySQL e o PostgreSQL.
- **Não-relacionais (NoSQL)**: Projetados para lidar com dados não estruturados ou semi-estruturados, como documentos JSON. Por exemplo o MongoDB.
- **Hierárquicos**: Organizam dados em uma estrutura de árvore.
- **De rede**: Permitem múltiplas relações entre dados, formando uma estrutura mais complexa.
- **Orientados a objetos**: Integram conceitos da programação orientada a objetos aos bancos de dados

A importância dos SGBDs no contexto atual é inegável, especialmente considerando o volume crescente de dados que as organizações precisam gerenciar. Algumas das principais razões para a utilização de SGBDs incluem:

- **Centralização dos Dados**: Os SGBDs permitem que as informações sejam armazenadas em um único local centralizado, facilitando o acesso e a gestão dos dados por diferentes usuários e aplicações.
- **Segurança**: Eles implementam medidas robustas para proteger os dados contra acessos não autorizados e garantir que apenas usuários autorizados possam realizar operações sensíveis.
- **Integridade dos Dados**: Os SGBDs garantem que os dados sejam precisos e corretos através da definição de regras e restrições, evitando problemas como redundâncias e inconsistências.
- **Eficiência na Manipulação**: Através do uso de índices e otimizações, os SGBDs permitem um acesso rápido e eficiente aos dados, mesmo em grandes volumes.
- **Suporte à Concorrência**: Eles suportam múltiplos usuários acessando e manipulando os dados simultaneamente sem conflitos, o que é crucial para ambientes de grande escala.
- **Facilidade na Análise de Dados**: Com funcionalidades avançadas para consultas complexas, os SGBDs facilitam a análise de grandes volumes de dados, permitindo ser tomadas decisões baseadas em informações precisas.

Em resumo, os Sistemas de Gerenciamento de Banco de Dados são ferramentas essenciais para qualquer organização moderna. Eles não apenas simplificam o gerenciamento de grandes volumes de dados, mas também garantem segurança, integridade e eficiência. A escolha do tipo adequado de SGBD pode impactar significativamente a capacidade da empresa em lidar com seus dados e extrair significado deles.

1.2 SQL

A Linguagem de Consulta Estruturada (SQL) é uma linguagem de programação fundamental para a interação com bancos de dados relacionais, permitindo a criação, manipulação e consulta de dados armazenados em tabelas. Desenvolvida inicialmente nos anos 70, o SQL se tornou um padrão amplamente adotado, sendo utilizado por diversos sistemas de gerenciamento de banco de dados. A simplicidade e a clareza da sintaxe do SQL, que inclui comandos como SELECT, INSERT, UPDATE e DELETE, tornam a linguagem acessível mesmo para quem não possue um profundo conhecimento de programação em um mundo cada vez mais orientado por dados.

A importância do SQL se reflete na sua capacidade de realizar operações complexas e consultas sofisticadas, permitindo que os usuários extraiam informações valiosas a partir de grandes volumes de dados. Além disso, a linguagem é flexível e padronizada, o que significa que comandos desenvolvidos para um SGBD podem ser facilmente adaptados para outros sistemas, que acabam possuindo pequenas diferenças entre si. Isso proporciona portabilidade e eficiência na gestão de dados.

1.3 MySQL

Figura 1.1: *Logo oficial do MySQL*

O MySQL (figura 1.1) é um dos sistemas de gerenciamento de banco de dados relacionais mais populares e amplamente utilizados no mundo, conhecido por sua robustez, escalabilidade e facilidade de uso. Desenvolvido inicialmente em 1994 por Michael Widenius e David Axmark, o MySQL foi projetado para oferecer um desempenho rápido e confiável, sendo utilizado em uma variedade de aplicações, desde pequenos sites até grandes plataformas como o Facebook[1] e o X (antigo Twitter).[2] A sua natureza de código aberto permite que desenvolvedores e empresas personalizem o software conforme suas necessidades, além de promover uma comunidade ativa que contribui continuamente com melhorias e suporte técnico.

Uma das características mais notáveis do MySQL é seu modelo cliente-servidor, que permite que múltiplos clientes se conectem ao servidor para acessar e manipular dados. O sistema suporta transações ACID (Atomicidade, Consistência, Isolamento e Durabilidade), garantindo a integridade dos dados em operações complexas. Além disso, o MySQL oferece flexibilidade na escolha de mecanismos de armazenamento, como MyISAM e InnoDB, cada um com suas próprias características que atendem a diferentes requisitos de desempenho e volume de dados. Essa versatilidade torna o MySQL uma escolha atraente para desenvolvedores que precisam adaptar suas soluções a diferentes cenários.

A eficiência do MySQL é evidenciada por sua capacidade de lidar com grandes volumes de dados e realizar bilhões de consultas diariamente. O sistema é otimizado para operações rápidas (Győrödi *et al.*, 2021), utilizando técnicas avançadas como indexação e caching para melhorar o desempenho das consultas. A facilidade de integração com diversas linguagens de programação, como PHP, Python e Java, também contribui para sua popularidade entre desenvolvedores. Com uma instalação simples e uma interface intuitiva para consultas SQL, o MySQL se apresenta como uma solução acessível tanto para iniciantes quanto para profissionais experientes no gerenciamento de dados.

[1] https://engineering.fb.com/2021/07/22/data-infrastructure/mysql/

[2] https://blog.x.com/engineering/en_us/a/2012/mysql-at-twitter

1.4 PostgreSQL

Figura 1.2: *Logo oficial do PostgreSQL*

O PostgreSQL (figura 1.2) é um sistema de gerenciamento de banco de dados objeto-relacional de código aberto, amplamente reconhecido por sua robustez e flexibilidade. Desenvolvido inicialmente na Universidade da Califórnia, em Berkeley, o PostgreSQL se destaca por sua conformidade com os padrões SQL e por oferecer uma ampla gama de recursos que suportam tanto dados relacionais quanto não relacionais. Essa versatilidade permite que os desenvolvedores criem aplicações complexas, utilizando tipos de dados variados, como JSON, XML e dados geoespaciais, além de possibilitar a criação de funções personalizadas e extensões. O PostgreSQL é frequentemente escolhido para aplicações que exigem alta confiabilidade e integridade dos dados, sendo utilizado em setores que vão desde startups até grandes corporações, como o Instagram[3] e a Twitch.[4]

Uma das características mais notáveis do PostgreSQL é seu suporte a transações ACID (Atomicidade, Consistência, Isolamento e Durabilidade), o que garante que as operações realizadas no banco sejam seguras e confiáveis. O sistema implementa um controle de concorrência multiversão (MVCC), permitindo que múltiplos usuários acessem os dados simultaneamente sem a necessidade de bloqueios, o que melhora significativamente o desempenho em ambientes com alta carga de trabalho (Neumann *et al.*, 2015). Além disso, o PostgreSQL oferece uma variedade de técnicas de indexação e otimizações para consultas complexas, tornando-o uma escolha ideal para aplicações que requerem análises profundas e rápidas respostas a consultas.

A escalabilidade do PostgreSQL é outro aspecto importante que contribui para sua popularidade. Ele pode gerenciar grandes volumes de dados e suportar um número elevado de usuários simultâneos sem comprometer a performance. A arquitetura do PostgreSQL permite a implementação de replicação síncrona e assíncrona, garantindo alta disponibilidade e resiliência em caso de falhas. Com uma comunidade ativa de desenvolvedores e uma vasta documentação disponível, o PostgreSQL continua a evoluir, incorporando novas funcionalidades e melhorias que atendem às necessidades crescentes das organizações modernas. Essa combinação de confiabilidade, flexibilidade e eficiência torna o PostgreSQL uma escolha preferencial para muitas aplicações críticas em diversos setores.

[3] https://instagram-engineering.com/under-the-hood-instagram-in-2015-8e8aff5ab7c2

[4] https://blog.twitch.tv/en/2016/10/11/how-twitch-uses-postgresql-c34aa9e56f58/

1.5 Comparação entre MySQL e PostgreSQL

Como serão os dois SGDBs utilizados neste projeto, é preciso entender as diferenças fundamentais entre esses dois sistemas. Esta seção explora algumas dimensões de comparação, incluindo arquitetura, tipos de dados, conformidade com ACID e segurança (AWS, 2024):

- **Arquitetura**: MySQL é um banco de dados puramente relacional que se destaca pela simplicidade e facilidade de uso. Ele utiliza um modelo de armazenamento baseado em tabelas que facilita a implementação e a manutenção. Por outro lado, PostgreSQL é um sistema objeto-relacional, o que significa que ele combina características dos bancos de dados relacionais com funcionalidades adicionais que permitem o armazenamento de dados complexos e a implementação de lógica avançada por meio de funções definidas pelo usuário e tipos de dados personalizados.
- **Tipos de Dados**: Uma das principais diferenças entre MySQL e PostgreSQL reside no suporte a tipos de dados. O PostgreSQL oferece uma ampla gama de tipos, incluindo arrays, JSONB, XML, e tipos geométricos, permitindo modelar dados complexos com mais facilidade. Em contraste, o MySQL suporta tipos básicos como números, strings e datas, mas não possui suporte nativo para arrays ou tipos compostos. Essa diferença pode influenciar a escolha dependendo das necessidades específicas do projeto.
- **Conformidade com ACID**: A conformidade com as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) é crucial para garantir a integridade dos dados. O PostgreSQL é totalmente compatível com ACID em todas as suas configurações. Por outro lado, o MySQL só garante essa conformidade quando utilizado com os mecanismos de armazenamento InnoDB ou NDB Cluster2, que são componentes de *software* subjacentes do MySQL utilizados para lidarem com as operações CRUD, entretanto, não são presentes na versão pura. Essa diferença pode ser um fator decisivo em aplicações onde a integridade dos dados é crítica.
- **Segurança**: Em termos de segurança, o PostgreSQL oferece recursos robustos como criptografia SSL, autenticação avançada (incluindo LDAP e Kerberos) e controle detalhado sobre permissões em nível de linha. O MySQL também oferece algumas medidas de segurança, mas sua abordagem é geralmente considerada menos abrangente do que a do PostgreSQL. Para aplicações que lidam com informações sensíveis ou regulamentadas, a escolha do PostgreSQL pode ser mais apropriada.

Enquanto o MySQL pode ser mais adequado para aplicações simples e rápidas, o PostgreSQL se destaca em cenários que requerem complexidade adicional e robustez na manipulação de dados. Logo, essa diferença pode influenciar a escolha dependendo das necessidades específicas do projeto.

Capítulo 2

Desenvolvimento da aplicação

2.1 Ideias iniciais e objetivos

A construção da aplicação *web* foi feita utilizando o modelo de *Single Page Application* (SPA) e o conceito de *client/server model*. As SPAs são projetadas para oferecer uma experiência de usuário fluida, carregando uma única página HTML que é dinamicamente atualizada conforme as interações do usuário, sem a necessidade de recarregar a página inteira. Essa característica é importante para que as aplicações possuam melhor performance e responsividade, sendo ideal para este projeto, que se espera uma execução e resposta rápidas (Kothapall, 2022).

O modelo *client/server* é fundamental para a arquitetura da aplicação, onde o cliente (usuário) é responsável por renderizar a interface do usuário e gerenciar as interações, enquanto o servidor lida com a lógica de negócios e o armazenamento de dados. Com essa separação clara entre as responsabilidades do cliente e do servidor, é possível otimizar o desempenho da aplicação. Essa estrutura permite que os dados sejam recuperados ou enviados ao servidor de forma assíncrona, utilizando chamadas *Fetch API* (será explicado na seção 2.3.2), como feito neste projeto, resultando em uma experiência de navegação mais rápida e eficiente.

Os objetivos principais ao desenvolver uma aplicação full stack com essas tecnologias incluem: garantir uma interface intuitiva e responsiva para os usuários, otimizar o desempenho através da carga dinâmica de conteúdo e facilitar a manutenção e escalabilidade da aplicação. Além disso, a escolha por um modelo SPA visa reduzir a carga no servidor ao minimizar as requisições necessárias para carregar novos conteúdos, transferindo parte do processamento para o lado do cliente. Com isso, espera-se não apenas melhorar a eficiência da aplicação, mas também proporcionar uma base sólida para futuras expansões e integrações com outras ferramentas ou serviços (Haroon-Sulyman, 2014).

2.2 Análise de requisitos

É essencial enfatizar que a aplicação deve ser simples e intuitiva para garantir uma experiência de usuário positiva. Interfaces que permitem uma navegação clara e direta não apenas facilitam o uso, mas também reduzem a curva de aprendizado para novos usuários. Além disso, a implementação de respostas imediatas às ações do usuário é crucial, pois isso melhora a percepção de desempenho da aplicação e mantém o usuário engajado, evitando frustrações que podem surgir com tempos de espera prolongados.

Outro aspecto importante é a facilidade de execução do servidor, especialmente quando a aplicação é projetada para ser executada localmente. Um servidor que pode ser facilmente configurado e iniciado sem complicações técnicas é um grande benefício, pois permite que desenvolvedores e usuários usem a aplicação sem depender de ambientes complexos, assim tornando a aplicação mais acessível para usuários que possam não ter experiência em hospedagem web.

A segurança é um requisito fundamental na análise, abrangendo tanto a proteção contra vazamentos de informações quanto a garantia da corretude dos resultados apresentados pela aplicação. Além disso, fornecer mensagens de erro que sejam fáceis de entender e que expliquem claramente o que ocorreu em caso de falhas contribui para uma experiência mais confiável. A transparência nas mensagens de erro não apenas ajuda os usuários a resolver problemas rapidamente, mas também aumenta sua confiança na aplicação ao mostrar que ela lida adequadamente com situações inesperadas.

2.3 Implementação da aplicação

Para a implementação da aplicação *fullstack* (figura 2.1), no frontend foram utilizados HTML, JavaScript e CSS para criar uma interface de usuário interativa e responsiva, permitindo que os usuários utilizem facilmente a aplicação. No *backend*, a aplicação foi construída com Node.js e Express.js, que oferecem um ambiente leve e eficiente para desenvolver as Interfaces de Programação de Aplicações (APIs) robustas e gerenciar as requisições dos clientes. Para a manipulação de dados foram utilizados MySQL e PostgreSQL, como explicitado anteriormente. Essa arquitetura não apenas permite uma comunicação fluida entre o frontend e o backend, mas também assegura que a aplicação seja escalável e mantenha um alto desempenho em diversas condições de uso. O código completo do frontend e backend pode ser encontrado no repositório da aplicação[1]

[1] Repositório no GitHub: https://github.com/GustavoNogai/MAC499-dbms-app-and-tests/

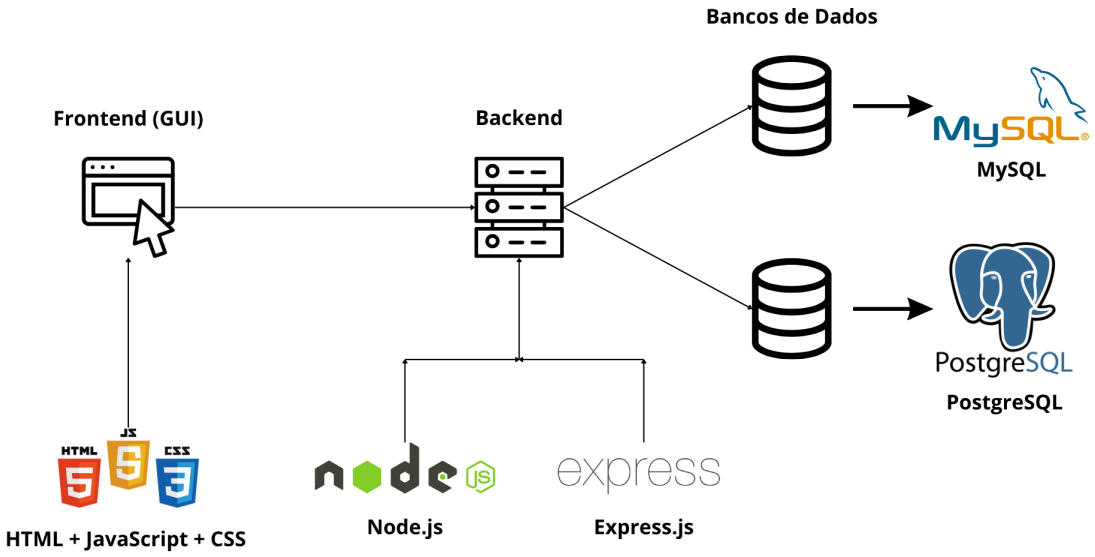

Figura 2.1: *Arquitetura da aplicação fullstack*

2.3.1 Servidor (backend)

No desenvolvimento do servidor (*backend*) da aplicação, foi utilizado Node.js em conjunto com o *framework* Express.js, que se destaca por sua simplicidade e eficiência na criação de servidores *web* e API. O Node.js é um ambiente de execução JavaScript que permite a execução de código no lado do servidor, possibilitando aos desenvolvedores utilizar a mesma linguagem tanto no *frontend* quanto no *backend*. Essa unificação simplifica o processo de desenvolvimento, uma vez que os conhecimentos e bibliotecas entre as duas camadas são as mesmas. O Express.js, por sua vez, oferece uma estrutura robusta que facilita a definição de rotas e o gerenciamento de requisições HTTP, permitindo a criação de APIs de forma rápida e eficiente.

Os métodos HTTP, como GET, POST, PUT e DELETE, são amplamente utilizados para realizar operações básicas de CRUD (*Create, Read, Update, Delete*) na aplicação. O uso desses métodos permite que o servidor responda a diferentes tipos de solicitações de forma adequada. Por exemplo, uma requisição GET pode ser utilizada para recuperar dados do servidor , como visto no código 2.1, enquanto uma requisição POST pode ser utilizada para enviar novos dados, como visto no código 2.2. A flexibilidade do Express.js também se reflete na possibilidade de integrar *middlewares* que podem processar as requisições antes que elas cheguem às rotas definidas, permitindo a implementação de funcionalidades como autenticação, manipulação de erros e validação de dados.

Programa 2.1 Exemplo de GET usado para recuperar dados do PostgreSQL.

```
1   app.get("/pggetdb", async (req, res) => {
2       try {
3          let rows = await pgGetUser();
4          res.status(200).json({
5             status: "recieved",
6             data: rows
7          });
8          console.log("Server:\n Postgres: Sucesso no getdb!" + "\n");
9       } catch (err) {
10         console.log("Server:\n Postgres: Erro no getdb!\n Erro: " + err.
               message + "\n");
11         res.status(400).json({ error: err.message });
12      }
13  });
```

Programa 2.2 Exemplo de POST usado para enviar dados ao MySQL.

```
1   app.post("/mypostdb", async function (req, res) {
2       const { content } = req.body;
3       console.log("Server:\n MySQL: Email: " + content.email + "\n");
4       try {
5           let id = await myInsertUser(content);
6           console.log("Server:\n MySQL: Sucesso no postdb!\n Id da linha
                inserida: " + id + "\n");
7           res.status(200).json({ status: "posted" });
8       } catch (err) {
9           if (err.name === "InputError") {
10              console.warn("Server:\n MySQL: Aviso:\n " + err.message + "\n");
11              res.header("warning", err.message);
12              res.status(200).json({ "warning": err.message });
13          }
14          else {
15              console.error("Server:\n MySQL: Erro no postdb!\n Erro: " + err.
                    message + "\n");
16              res.status(400).json({ "error": err.message });
17          }
18      }
19  });
```

A configuração do servidor com Node.js e Express.js é relativamente simples e direta. Após a instalação das dependências necessárias, como o Express, um servidor básico pode ser criado em poucos passos, como visto no código 2.3. O desenvolvedor define as rotas desejadas e inicia o servidor em uma porta específica, tornando-o acessível para clientes que realizam requisições. Essa abordagem não apenas acelera o desenvolvimento da aplicação, mas também garante um ambiente leve e altamente escalável, capaz de lidar com múltiplas conexões simultaneamente. Em suma, a combinação de Node.js e Express.js proporciona uma base sólida para o backend da aplicação, permitindo um desenvolvimento ágil e eficiente.

Programa 2.3 Criação de rotas com o Express.js.

```
1  import express from "express";
2  import path from "path";
3  import { fileURLToPath } from "url";
4
5  // Configuracoes iniciais do express
6  const app = express();
7  const port = 3000;
8  const __filename = fileURLToPath(import.meta.url);
9  const __dirname = path.dirname(__filename);
10
11 // Indica o caminho absoluto da pasta 'frontend'
12 app.use(express.static(__dirname + '/../frontend'));
13 app.use(express.json());
14 app.set("json spaces", 2);
15
16 // ...
17 // Código dos métodos HTTP para o uso das SGDBs
18 // ...
19
20 // GET para definir a página inicial do aplicativo
21 app.get("/", function (req, res) {
22 console.log(path.join(__dirname, '../frontend/index/index.html' + "\n"));
23 res.sendFile(path.join(__dirname, '../frontend/index/index.html'));
24 });
25
26 // Inicia o servidor no port definido
27 var server = app.listen(port, function () {
28 console.log(`Listening on port ${port}!` + "\n");
29 });
```

2.3.2 Cliente (frontend)

No desenvolvimento do *frontend* da aplicação, a Fetch API desempenha um papel crucial na comunicação entre o cliente e o backend. Essa interface permite que o JavaScript realize requisições HTTP de forma assíncrona, como mostrado no código 2.4 requisitando dados fornecidos pelo *backend* com o código 2.1, facilitando a interação com os recursos disponíveis no servidor, pois ao utilizar a Fetch API, é possível fazer chamadas para *endpoints* RESTful definidos no backend, utilizando os métodos HTTP para gerenciar dados de maneira eficiente. Além disso, a sintaxe intuitiva da Fetch API, que retorna *Promises*, simplifica o tratamento de respostas e erros, permitindo uma manipulação fácil de dados recebidos.

Programa 2.4 Exemplo da Fetch API requisitando (GET) dados do PostgreSQL.

```
1   async function getDB(e){
2       e.preventDefault();
3       const res = await fetch(baseURL + "pggetdb",
4           {
5               method:"GET"
6           }
7       );
8       const data = await res.json();
9       if (res.ok) {
10          // ...
11          // Se o comando foi executado com sucesso exibir os dados na página
                do usuário
12          // ...
13      }
14      else {
15          // ...
16          // Se não exibir o erro na página do usuário
17          // ...
18      }
19  }
```

A aplicação foi projetada como uma RESTful, o que significa que segue os princípios da arquitetura REST (*Representational State Transfer*). Isso implica que a comunicação entre o *frontend* e o *backend* é baseada em recursos identificados por URLs, onde cada recurso pode ser acessado e manipulado através de métodos HTTP apropriados. O usuário interage diretamente com a interface do *frontend*, realizando ações que resultam em chamadas à Fetch API, essas interações são fundamentais para a aplicação funcionar como uma SPA, pois permitem que ele visualize e manipule dados em tempo real sem a necessidade de recarregar a página. Além disso, a arquitetura RESTful não apenas melhora a eficiência das operações, mas também promove uma estrutura clara e organizada para a comunicação entre as diferentes partes da do projeto.

Assim, a Fetch API facilita a implementação de funcionalidades dinâmicas no *frontend*, tornando a interação do usuário com a aplicação mais responsiva e eficiente. Com essa abordagem, é possível criar aplicações *web* rápidas e eficientes, mostrando a importância da arquitetura RESTful para a comunicação entre o *frontend* e o *backend* e da Fetch API no desenvolvimento da aplicação.

A figura 2.2 mostra a página PostgreSQL da aplicação, onde pode ser feito a interação com o banco de dados do PostgreSQL pelo caixa de texto ou pela UI na parte direita.

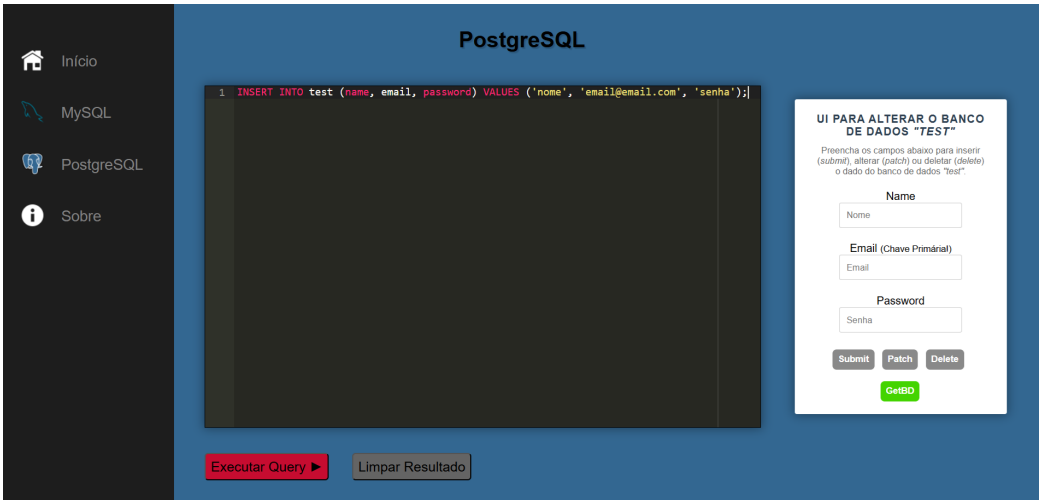

Figura 2.2: *Página PostgreSQL da aplicação*

2.3.3 Comunicação com os bancos de dados

A comunicação entre os SGBDs MySQL e PostgreSQL com o *backend* da aplicação é facilitada pelo uso do Node.js e suas bibliotecas específicas, que permitem a integração com esses bancos. Ao utilizar o npm (*Node Package Manager*), é possível instalar módulos como *mysql2* ou *pg*, que são essenciais para estabelecer conexões e executar operações de banco de dados. O módulo *mysql2*, por exemplo, permite a criação de pools de conexões, otimizando o gerenciamento de múltiplas requisições ao banco, enquanto o módulo *pg*, oferece uma interface robusta para interagir com PostgreSQL, suportando funcionalidades avançadas como transações e consultas assíncronas, como visto no código 2.5.

Esses módulos não apenas simplificam a conexão e a execução de comandos SQL, mas também garantem que as operações sejam realizadas de forma eficiente e segura. A utilização de um pool de conexões é uma prática recomendada, pois permite a reutilização de conexões existentes, reduzindo a sobrecarga associada à abertura e fechamento frequente de conexões com o banco. Além disso, ao utilizar métodos HTTP apropriados nas rotas definidas no *backend* com Express.js, é possível mapear as operações CRUD (*Create, Read, Update, Delete*) diretamente às requisições do cliente, garantindo uma comunicação clara e organizada entre o frontend e o backend.

A estrutura modular do Node.js e a flexibilidade do npm permitem a fácil integração de diferentes SGBDs na aplicação, sendo possível escolher aquele que melhor atender às necessidades específicas para cada projeto. Além disso, futuras expansões e manutenções da aplicação se tornam mais fáceis por conta de tal estrutura modular fornecida pelo npm.

Programa 2.5 Conexões do *backend* com as SGDBs.

```
1   import pg from "pg"; // PostgreSQL
2   import mysql from "mysql2"; // MySQL
3
4   const { Pool } = pg;
5
6   var pgPool = new Pool({
7        database: database,
8        host: "localhost",
9        user: userPostgreSQL,
10       password: passwordPostgreSQL,
11       keepAlive: true,
12       idleTimeoutMillis: 100000
13  });
14
15  var myPool = mysql.createPool({
16       database: database,
17       host: "localhost",
18       user: userMySQL,
19       password: passwordMySQL,
20       enableKeepAlive: true,
21       idleTimeout: 100000
22  });
```

Capítulo 3

Testes de desempenho

3.1 Metodologia para testes de performance

A abordagem utilizada para avaliar a eficiência das SGDBs MySQL e PostgreSQL foi a execução de *scripts* em *bash*, como visto no código 3.1, para realizar 30 repetições de cada consulta, que foram 22 consultas diferentes. Essa metodologia permite coletar dados consistentes sobre o tempo de resposta, proporcionando uma visão clara sobre o desempenho em diferentes cenários pois os scripts foram projetados para automatizar as consultas, garantindo que os testes sejam realizados de forma sistemática, permitindo que as conclusões sobre o desempenho da aplicação sejam fundamentadas em evidências empíricas.

Além disso, foi utilizado o ambiente Jupyter Notebook (ipynb) para processar os dados coletados e gerar gráficos que ilustram os resultados dos testes. A integração do Jupyter Notebook com códigos em Python permite a análise de dados e a identificação de tendências ou padrões nos tempos de resposta das consultas. Gráficos como o *boxplot*, como por exemplo o visto no código 3.2, e o *barplot* foram criados para representar visualmente a performance das SGDBs, facilitando a vizualização dos resultados.

Quanto aos dados utilizados em ambos os SGDBs serão os mesmos, permitindo uma comparação clara e consistente dos tempos de resposta das consultas. Neste caso foi utilizado o banco de dados do Stackoverflow de 2010,[1] possuindo aproximadamente 10gb de dados, sendo um tamanho significativo para o teste proposto e se aproximando da realidade, gerando mais fidelidade aos testes.

As consultas podem ser vistas no repositório do projeto,[2] possuindo consultas como por exemplo: a consulta de inserção *"SELECT * FROM Users WHERE DisplayName = 'Alan Turing';"* e a consulta de alteração *UPDATE Users SET Reputation = 111 WHERE Id = 1;*

[1] https://www.brentozar.com/archive/2015/10/how-to-download-the-stack-overflow-database-via-bittorrent/

[2] Repositório no GitHub: https://github.com/GustavoNogai/MAC499-dbms-app-and-tests/tree/main/performance-test/queries

Programa 3.1 Script Bash para automatizar os testes do MySQL.

```
#!/bin/bash

# Fazer o separador ser a quebra de linha
IFS=$'\n'
# Faz o comando time formatar em tempo real
TIMEFORMAT=%R

# Número de queries
QUERY_NUM=0
# Número de repetições
NUM_RUNS=30

# Cabeçalho dos resultados
echo "Query Number|Query|Run|Elapsed Time" >> results.txt

# Executar testes
for (( i=1; i<=$NUM_RUNS; i++ ))
do
for query in $(cat mysql_queries.txt)
do
    QUERY_NUM=$((QUERY_NUM + 1))
    echo "query $i: $query"
    ELAPSED_TIME=$({ time mysql --user=$USER --password=$PASSWORD --database=
        StackOverflow -e $query; } 2>&1 > /dev/null )
    echo "${QUERY_NUM}|${query}|${i}|$ELAPSED_TIME" >> results.txt
    echo "Elapsed Time: $ELAPSED_TIME"
done

QUERY_NUM=0
done
```

Programa 3.2 Código em Python que gera um *boxplot* dos tempos do MySQL no ipynb.

```
fig = plt.figure(figsize =(13, 3.9), dpi=150)

ax = fig.add_axes([0, 0, 0.5, 1])
ax.yaxis.grid(True, linestyle="--")
ax.set_title("MySQL")

aux = timesMySQL[:].transpose()

bp = ax.boxplot(aux, sym='.')

plt.xlabel("Consulta")
plt.ylabel("Tempo de Execução (s)")
plt.show()
```

3.1.1 Ambiente computacional utilizado

Os experimentos foram realizados em um computador com as especificações de *hardware* mostradas na tabela 3.1. As informações do processador foram retiradas do site oficial da Intel.[3]

	Especificação
Processador	Intel Core 15-7500 (4 núcleos de 3.4Ghz)
RAM	8GB
Tipo de Armazenamento	Disco Rígido
Sistema Operacional	Windows 10 Home 22H2

Tabela 3.1: *Especificações do computador utilizado para os testes.*

3.2 Resultados

Primeiramente, definiu-se para os experimentos um conjunto de operações sobre o banco de dados que abrange uma grande variedade de consultas e alterações. Para cada tipo de operação, existem ao menos duas variações, atuando sobre diferentes relações e atributos, isso permite avaliar o comportamento do SGDB em diferentes parâmetros mantendo o mesmo tipo de consulta.

A fim de facilitar a identificação dessas operações ao longo do relatório, foi abreviado a operação baseado na tabela 3.2.

[3] https://www.intel.com.br/content/www/br/pt/products/sku/97123/intel-core-i57500-processor-6m-cache-up-to-3-80-ghz/specifications.html

Prefixo	Descrição	Variações
I	Inserção	I1, I2
R	Remoção	R1, R2, R3, R4
A	Alteração	A1, A2, A3
CCP	Consulta por Chave Primária	CCP1, CCP2, CCP3, CCP4, CCP5, CCP6
CNP	Consulta por Chave Não Primária	CNP1, CNP2, CNP3
CS	Consulta por Padrão de String	CS1, CS2, CS3, CS4

Tabela 3.2: *Abreviaturas dos tipos de operações realizadas nos testes*

Além disso, foram testados dois cenários: um sem índices adicionais e um com índice, ambos com o mesmo conjunto de operações, sendo que o cenário com índices adicionais utiliza um *index* com a expectativa de facilitar as consultas.

3.2.1 Sem índices adicionais

O primeiro teste foi realizado sem índices adicionais em um banco de dados recém criado, porém, é preciso se atentar ao fato de que índices primários são criados automaticamente pelo SGDB.

O *boxplot* da figura 3.1 mostra a variação dos tempos de execução das 30 repetições de cada consulta no MySQL, enquanto a figura 3.2 mostra a do PostgreSQL. Já a tabela 3.3 mostra os tempos médios de execução e o desvio padrão das 30 execuções de cada consulta.

Note como as consultas R2 (4), A2 (8), CNP2 (17), CS2 (20) e CS4 (22) possuem resultados lentos quando comparados à outras consultas, isso se deve pelo fato das consultas em si serem complexas, como a CS2 (20): *"SELECT * FROM Posts WHERE Title LIKE '%enigma%';"*, que possui uma expressão *LIKE* complexa, que busca todos os *posts* que contém a palavra *'enigma'*, precisando buscar percorrer todos os *posts* e comparar o *title* de cada um.

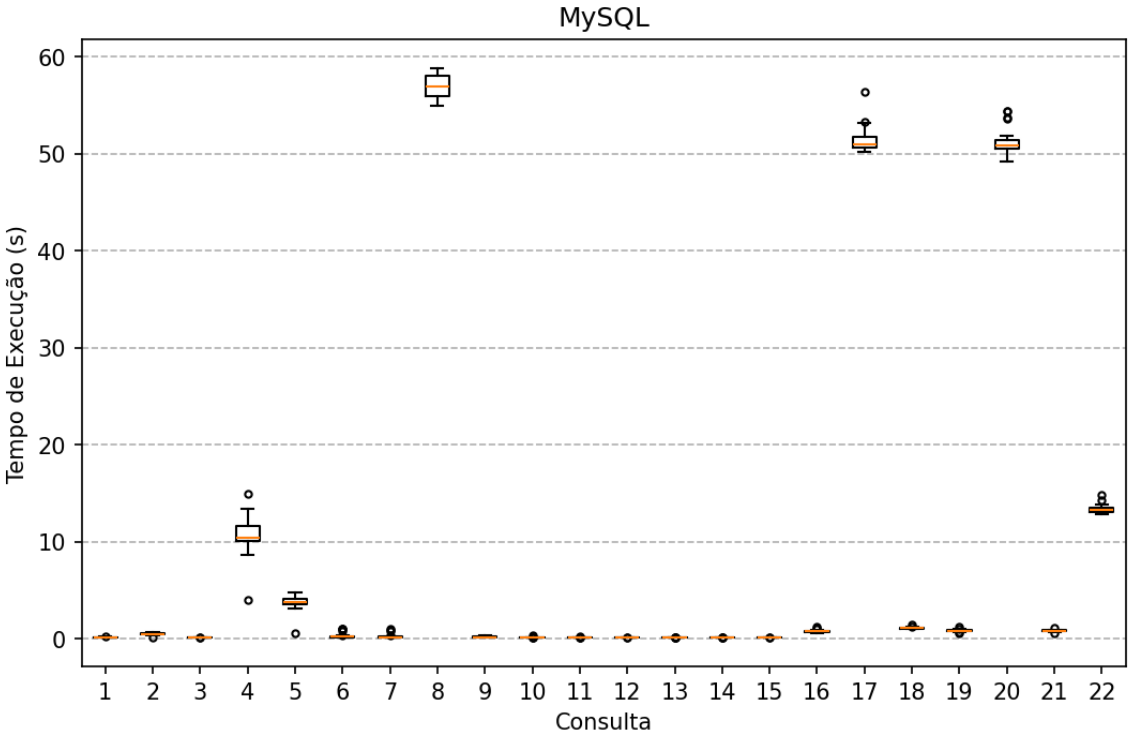

Figura 3.1: *Boxplot do tempo de execução das operações sem índices adicionais no MySQL.*

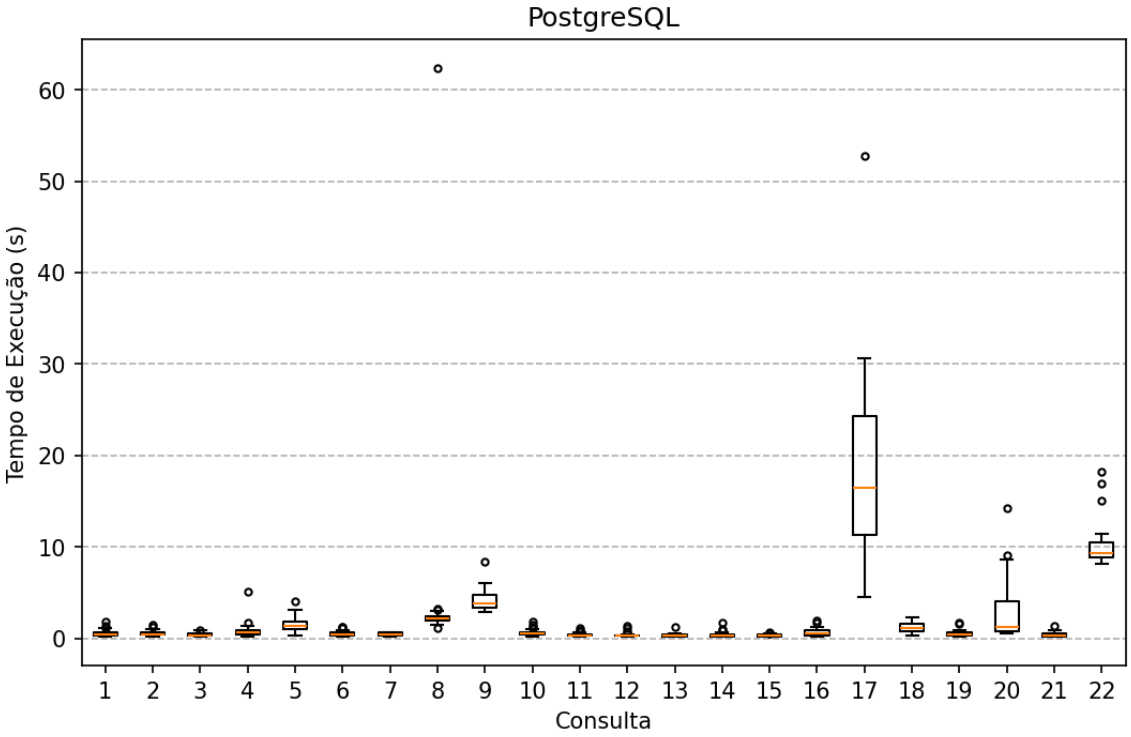

Figura 3.2: *Boxplot do tempo de execução das operações sem índices adicionais no PostgreSQL.*

	PostgreSQL		MySQL	
Operação	**Tempo médio (em segundos)**	**Desvio Padrão**	**Tempo médio (em segundos)**	**Desvio Padrão**
I1	0,368	0,4949	0,253	0,1006
I2	0,303	0.3608	0,455	0,1299
R1	0,232	0,1289	0,107	0,0139
R2	0,370	0,3285	11,393	1,6814
R3	1,260	1,3637	3,380	1,3127
R4	0,351	0,4766	0,289	0,2496
A1	0,246	0,1405	0,258	0,2067
A2	9,615	16,3443	59,721	6,7294
A3	2,115	3,6207	0,218	0,0700
CCP1	0,689	0,3338	0,114	0,0100
CCP2	0,556	0,1969	0,113	0,0133
CCP3	0,574	0,2939	0,133	0,0137
CCP4	0,501	0,2192	0,102	0,0061
CCP5	0,569	0,4825	0,100	0,0031
CCP6	0,503	0,3043	0,101	0,0084
CNP1	0,531	0,3580	0,647	0,0546
CNP2	1,064	1,0122	51,644	2,2180
CNP3	0,589	0,3441	1,131	0,0601
CS1	0,650	0,5636	0,662	0,0373
CS2	0,952	0,3318	51,590	3,5726
CS3	0,436	0,2275	0,673	0,0401
CS4	9,178	1,1094	13,270	0,2043

Tabela 3.3: *Tempo médio e desvio padrão das operações sem índice.*

3.2.2 Com índices adicionais

Neste cenário, foram criados em ambos os SGDBs alguns índices do tipo Árvore B, sobre os atributos que eram comuns às operações, sendo eles: *Posts.Title*, *Users.DisplayName*, *Badges.Name* e *Comments.Score*. E como as comparações entre textos geralmente não utilizam tantos caracteres, foi limitado a 10 caracteres para cada índice.

Com os índices criados, esperamos que esses as execuções sejam afetadas positivamente nas operações CNP1, CNP2, CS1 e CS4, mas afetem negativamente I1, I2, R1, R3, R4 e A2, devido à necessidade de atualizar os índices após cada inserção, remoção ou atualização.

O *boxplot* da figura 3.3 mostra a variação dos tempos de execução das 30 repetições de cada consulta no MySQL, enquanto a figura 3.4 mostra a do PostgreSQL. Já a tabela 3.4 mostra os tempos médios de execução e o desvio padrão das 30 execuções de cada consulta.

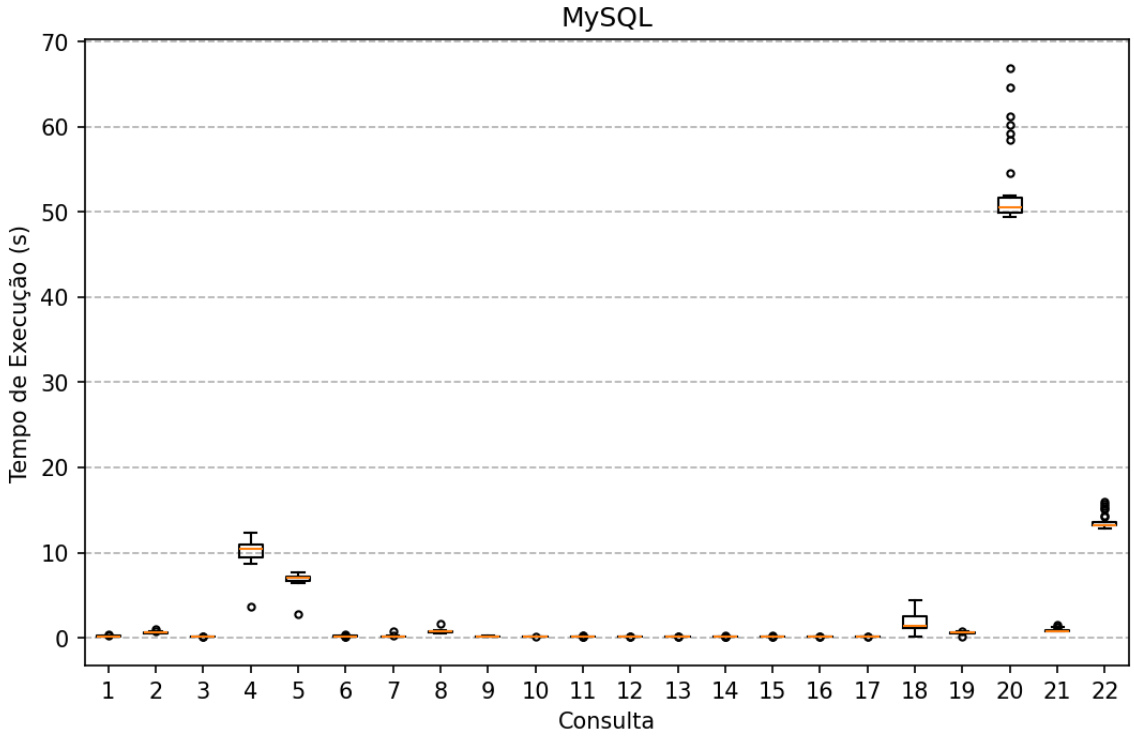

Figura 3.3: *Boxplot do tempo de execução das operações com índices adicionais no MySQL.*

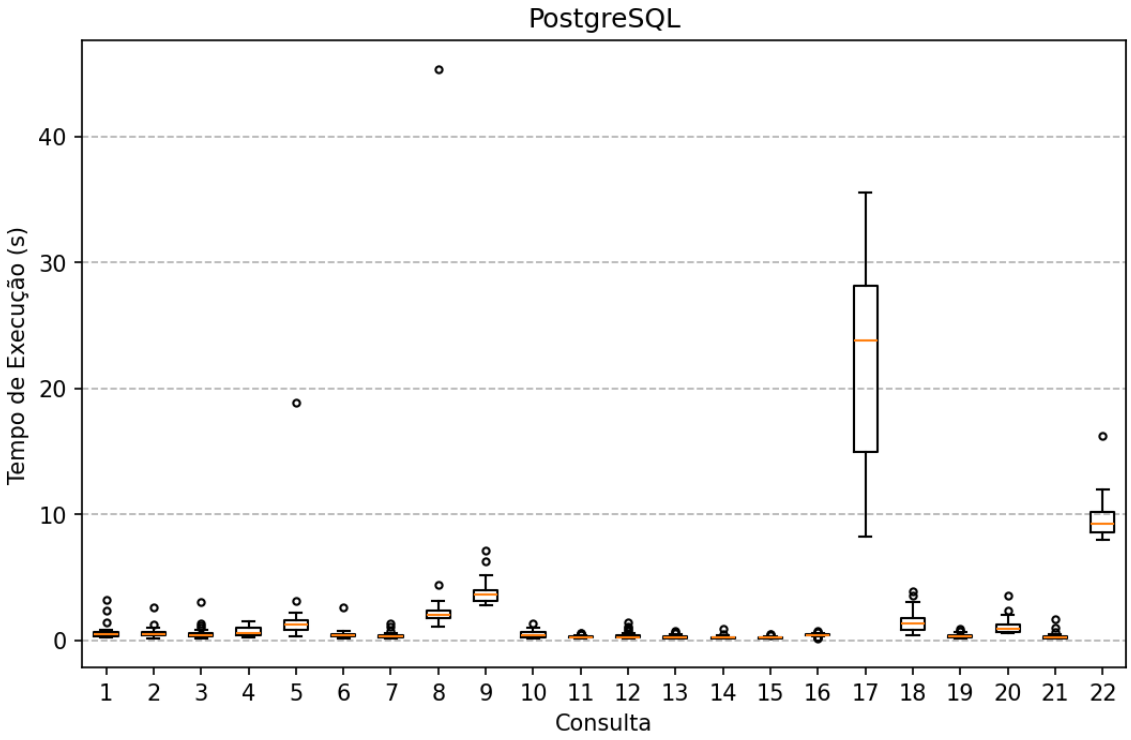

Figura 3.4: *Boxplot do tempo de execução das operações com índices adicionais no PostgreSQL.*

	PostgreSQL		MySQL	
Operação	**Tempo médio (em segundos)**	**Desvio Padrão**	**Tempo médio (em segundos)**	**Desvio Padrão**
I1	0,695	0,6287	0,212	0,0447
I2	0,619	0,4341	0,612	0,1058
R1	0,577	0,5431	0,103	0,0058
R2	0,692	0,3757	10,135	1,5127
R3	1,874	3,2104	6,891	0,8405
R4	0,481	0,4172	0,223	0,0447
A1	0,397	0,2601	0,194	0,1223
A2	3,564	7,7897	0,774	0,1885
A3	3,833	0,9356	0,177	0,0266
CCP1	0,509	0,2976	0,123	0,0185
CCP2	0,303	0,1219	0,129	0,0448
CCP3	0,411	0,3138	0,130	0,0256
CCP4	0,286	0,1481	0,126	0,0238
CCP5	0,250	0,1420	0,114	0,0317
CCP6	0,250	0,0985	0,107	0,0281
CNP1	0,444	0,1413	0,121	0,0159
CNP2	22,109	7,4943	0,115	0,0192
CNP3	1,545	0,8778	1,983	1,0014
CS1	0,368	0,1959	0,611	0,1097
CS2	1,097	0,6296	52,795	4,7757
CS3	0,343	0,3091	0,910	0,2128
CS4	9,701	1,5891	13,678	0,8936

Tabela 3.4: *Tempo médio e desvio padrão das operações com índice.*

Capítulo 4

Análise dos resultados

Para ser feita esta análise, é fundamental considerar a percepção do usuário em relação ao tempo de resposta das interações com o sistema. Segundo o estudo *"A Study on Tolerable Waiting Time: How Long Are Web Users Willing to Wait?"* Nah, 2003, tempos de resposta abaixo de 0,1 segundos são percebidos como imediatos, enquanto intervalos entre 0,1 e 1 segundo são notados como uma leve demora que não interrompe a linha de raciocínio do usuário. No entanto, quando os tempos se aproximam de 2 segundos, a experiência do usuário muda drasticamente, passando de aceitável para inaceitável, o que pode resultar em insatisfação e reclamações.

Levando esse estudo em conta, será comparado o desempenho entre as duas SGDBs com o intuito de entender sua adequação para o ambiente de desenvolvimento de aplicações na seção 4.2, levando mais em conta os tempos absolutos de execução das operações do que os tempos relativos.

4.1 Interpretação dos desempenhos entre MySQL e PostgreSQL

Como ambas as SGDBs possuem grande popularidade, vários estudos já existem sobre o desempenho dessas ferramentas, logo, é possível construir algumas expectativas sobre as características de cada sistema, como, por exemplo, o estudo mostrado no artigo *"Comparison of the performance of relational databases PostgreSQL and MySQL"* Skublewska-Paszkowska, 2021, que destaca que, embora o MySQL seja amplamente reconhecido por sua eficiência em operações de leitura simples, o PostgreSQL se sobressai em consultas mais complexas e em cenários que exigem integridade referencial. A pesquisa revela que o MySQL apresenta tempos de resposta mais rápidos para consultas básicas, tornando-o ideal para aplicações web que priorizam acessos frequentes a dados. No entanto, quando se trata de operações que envolvem múltiplas tabelas e transações complexas, o PostgreSQL demonstra uma performance superior, beneficiando-se de sua arquitetura robusta e suporte a processamento paralelo.

Complementando essa análise, o estudo "A Performance Analysis of the DBMS - MySQL vs PostgreSQL"Andjelic, 2008 ressalta a importância de considerar o tipo de operação ao avaliar o desempenho de cada sistema. Os resultados mostram que, enquanto o MySQL é mais eficiente em inserções e operações simples, o PostgreSQL se destaca em atualizações e consultas que exigem um maior processamento de dados. Essa diferença se torna evidente em ambientes onde a complexidade das consultas pode impactar significativamente os tempos de execução.

Assim, como neste estudo foram feitas diversas consultas de diversos tipos, será esperado resultados similares, acrescentando o caso com índices adicionais.

4.1.1 Sem índices adicionais

No caso sem índices adicionais, observando os dados coletados, o SGDB PostgreSQL demonstrou uma eficiência superior em comparação ao MySQL, como visto na figura 4.1. Embora existam diversos cenários nos quais o MySQL se mostrou mais rápido que o PostgreSQL, incluindo casos com menor desvio padrão, esses geralmente envolvem operações extremamente rápidas, onde, apesar da diferença relativa ser significativa, a diferença entre o tempo absoluto das execuções é muito pequeno. Em contrapartida, são raros os casos em que o PostgreSQL apresenta um desempenho consideravelmente inferior ao do MySQL, e quando isso ocorre, a desvantagem do MySQL tende a ser expressiva.

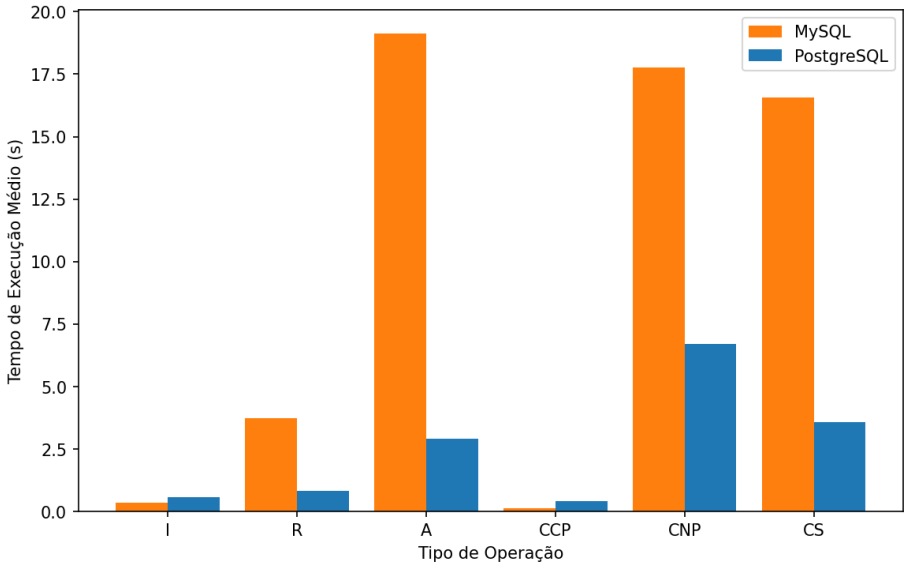

Figura 4.1: *Barplot da comparação dos tempos de execução das operações sem índices adicionais.*

De maneira geral, o PostgreSQL revelou resultados muito mais consistentes do que o MySQL. Enquanto este último apresenta não apenas uma média de tempos de execução mais elevada, mas também um número consideravelmente maior de *outliers*, o primeiro se destacou pela estabilidade em suas respostas. Em certos casos, ambos os SGBDs se mostraram altamente eficientes em determinadas buscas, enquanto em outras suas performances foram bastante insatisfatórias. Essa variação pode ser atribuída à presença de índices que os sistemas gerenciadores de banco de dados possam ter criado.

4.1.2 Com índices adicionais

Quanto aos teste com índices adicionais, curiosamente, apenas a operação CNP1 apresentou uma melhoria consistente em ambos os SGBDs. A operação CNP2, por sua vez, teve uma grande melhora no MySQL, reduzindo o tempo de execução de mais de 51 segundos para menos de um segundo. No entanto, observou-se uma degradação significativa no desempenho do PostgreSQL nessa mesma consulta. Todas as outras consultas que esperávamos que apresentassem um desempenho superior mantiveram-se muito próximas entre os dois SGBDs. Como era esperado, as operações que envolvem a atualização dos índices resultaram em aumentos consideráveis na média do tempo de execução, exceto para a operação A2 no MySQL, que surpreendentemente teve um ganho expressivo, passando de 59 segundos para menos de um segundo, como pode ser visto na figura 4.2.

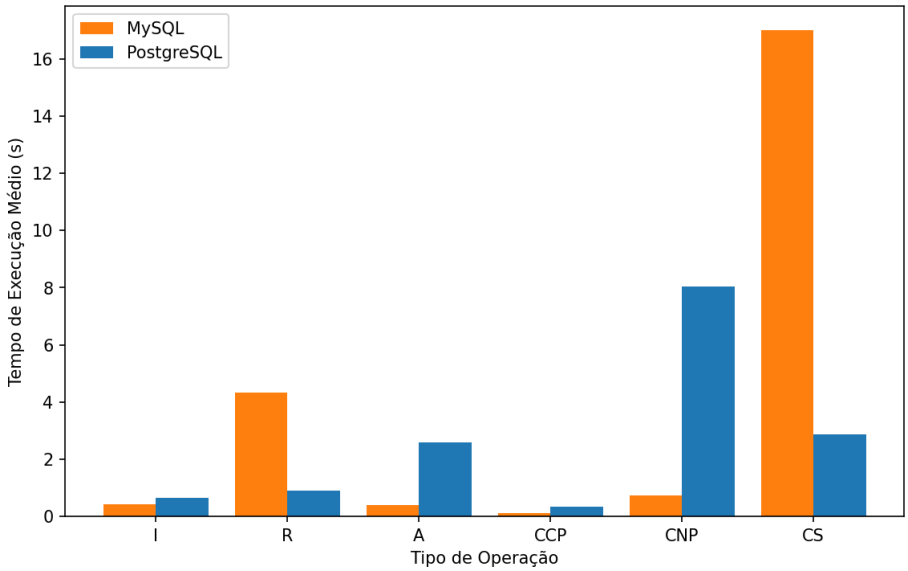

Figura 4.2: *Barplot da comparação dos tempos de execução das operações com índices adicionais.*

Os gráficos apresentados evidenciam essa melhora, especialmente para o MySQL, onde os dados estão "agrupados"em regiões com tempos de execução mais baixos. Em contrapartida, o PostgreSQL mostrou uma quantidade maior de *outliers* que excedem consideravelmente a média. Esses fenômenos vão na contramão das expectativas iniciais. Ao comparar com o gráfico correspondente do cenário sem índices adicionais, confirmamos que o MySQL apresentou uma boa melhora na primeira metade das operações, mas manteve os números para as operações de CS. Já o PostgreSQL teve um desempenho inferior na primeira metade e manteve-se na segunda. O fato de que essas diminuições ainda não conseguem ser consistentes em melhorar os valores observados no cenário sem índices adicionais sugere que não apenas os índices parecem não ser efetivos para os casos de teste, mas também levanta questionamentos sobre a validade das melhorias observadas, que se mostraram muito específicas e sem um padrão aparente.

4.2 Impacto dos resultados no desenvolvimento de aplicações

Ao longo da análise, observou-se que, embora a maioria das consultas apresentasse tempos de execução próximos a 1 segundo, algumas operações se destacaram por levar consideravelmente mais tempo. Essa variação nos tempos de resposta é crucial para entender como os usuários interagem com a aplicação e como a performance do banco de dados pode impactar a experiência geral. A identificação dessas discrepâncias permite que desenvolvedores e administradores de banco de dados tomem decisões informadas sobre otimizações necessárias e ajustes na indexação, visando melhorar a eficiência das operações e garantir que os tempos de resposta permaneçam dentro dos limites aceitáveis para o usuário final.

Entretanto, é comum que a indexação seja percebida como uma solução mágica para problemas de desempenho em bancos de dados, a expectativa é que, ao implementar índices, as consultas se tornem instantâneas mais rápidas e a eficiência geral do sistema aumente de forma significativa. De fato, os índices são projetados para acelerar o acesso a dados, permitindo que o banco de dados localize registros relevantes sem precisar percorrer toda a tabela, no entanto, essa visão simplista ignora a complexidade envolvida na escolha e manutenção de índices adequados. A implementação de índices não é trivial, ao contrário, pode ser um processo desafiador que requer uma análise cuidadosa das operações mais frequentes e dos padrões de acesso aos dados.

A escolha inadequada de índices pode levar a resultados opostos, onde o desempenho das operações é prejudicado em vez de melhorado. Embora os índices possam acelerar as consultas, eles também introduzem custos adicionais durante as operações de escrita, como inserções e atualizações, uma vez que cada modificação nos dados exige também uma atualização nos índices correspondentes, o que pode resultar em tempos de execução mais longos para essas operações, especialmente em tabelas com alta taxa de modificação. Além disso, o uso excessivo ou inadequado de índices pode levar ao aumento do consumo de espaço em disco e à necessidade de manutenção constante. Portanto, é crucial que os desenvolvedores adotem uma abordagem estratégica na criação de índices, garantindo que eles sejam realmente necessários e benéficos para o desempenho do sistema, evitando assim a armadilha da "magia"da indexação que pode resultar em um desempenho subótimo.

Capítulo 5

Trabalhos futuros

Neste capítulo serão apresentadas diversas direções que podem ser exploradas para aprimorar esta aplicação e expandir suas funcionalidades e buscar entender melhor a performance dos SGDBs. A seguir, discutiremos três áreas principais que podem ser abordadas para a aplicação: a expansão da aplicação com novas funcionalidades, a implementação de novos sistemas de gerenciamento de banco de dados e sugestões para a correção de códigos. Além disso, será discutido como a análise das consultas pode ser reanalisada baseada em especificação de *hardware* diferentes. Essas direções não apenas visam melhorar a experiência do usuário, mas também garantir que a aplicação se mantenha relevante e eficiente em um ambiente tecnológico em constante evolução.

5.1 Expansão da aplicação

Uma das principais direções para o futuro da aplicação é a expansão de suas funcionalidades. Isso inclui a capacidade de executar várias linhas de comandos SQL simultaneamente, não necessitando escrever códigos SQL um por vez. Além disso, aprimorar a interface de usuário (UI), permitindo aos usuários criar e gerenciar diferentes bancos de dados sem necessidade de escrever comandos SQL diretamente. Essa abordagem tornaria a aplicação mais acessível para usuários com diferentes níveis de conhecimento técnico, permitindo que eles interajam com os dados de maneira mais direta e fácil.

5.2 Implementação de novos SGDBs

Outra área promissora é a implementação de novos sistemas de gerenciamento de banco de dados. Com a rápida evolução dos SGBDs populares, como MongoDB, SQLite e outros, é fundamental que a aplicação se adapte às novas tecnologias disponíveis, que mudam constantemente cerca de 10 em 10 anos, dependendo de mudanças na tecnologia em si e na própria demanda dos usuários Patel *et al.*, 2023. Graças ao npm (*Node Package Manager*), a integração desses SGBDs pode ser realizada de forma relativamente simples e rápida. Isso não apenas ampliaria as opções disponíveis para os usuários, mas também permitiria que a aplicação aproveitasse as características únicas de cada SGDB e as *"trendings"* da

área de dados.

5.3 Sugestão de correção de códigos

A utilização da inteligência artificial para sugerir correções de códigos é uma tendência crescente que pode ser incorporada à aplicação. Implementar algoritmos que analisem o código escrito pelos usuários e ofereçam sugestões automáticas para melhorias ou correções pode não apenas aumentar a eficiência do desenvolvimento, mas também educar os usuários sobre boas práticas de programação. Além disso, fornecer guias e explicações detalhadas sobre as sugestões apresentadas pode auxiliar no aprendizado contínuo dos usuários, promovendo um ambiente mais acolhedor e intuitivo. Essa abordagem não só melhoraria a qualidade do código gerado, mas também incentivaria um maior engajamento dos usuários com a plataforma.

5.4 Reanálise das performances das consultas

Por fim, uma reanálise das performances das consultas é essencial para compreender melhor a eficiência real e a consistência dos resultados obtidos, pois o *hardware* pode ter sido um fator limitante durante a realização dos testes, havendo a possibilidade de ter ocorrido um gargalo em algumas operações mais complexas com o banco de dados de 10gb utilizado. Assim, a variação nas especificações do *hardware*, como CPU, RAM e tipo de armazenamento, pode influenciar significativamente os tempos de resposta das operações, especialmente em cenários de alta concorrência ou consultas complexas. No entanto, a realização de testes com diferentes configurações de *hardware* para esta pesquisa se mostrou desafiadora devido a limitações logísticas e orçamentárias. Assim, futuros trabalhos poderiam se beneficiar da implementação de um ambiente de testes mais diversificado, permitindo comparações diretas entre diferentes configurações de hardware e suas respectivas influências no desempenho das consultas. Essa abordagem não apenas enriqueceria a análise, mas também proporcionaria *insights* valiosos para otimizar a escolha do *hardware* em implementações reais.

Capítulo 6

Conclusão

A aplicação desenvolvida neste trabalho cumpre com o seu objetivo inicial, de proporcionar uma interação fácil e intuitiva com os sistemas de gerenciamento de banco de dados MySQL e PostgreSQL. A interface foi projetada para ser acessível, permitindo que usuários com diferentes níveis de experiência possam realizar operações básicas de forma eficiente. Essa abordagem visa auxiliar o acesso e aprendizagem dos usuários em relação aos SGDBs, facilitando a vizualização das operações de manipulação e a consulta às informações armazenadas.

Considerando a análise proposta, os resultados indicaram que ambos os SGBDs desempenharam bem nas operações de inserção e consulta. No entanto, observou-se que o PostgreSQL apresentou um desempenho mediano nas operações de atualização, enquanto o MySQL teve um desempenho inferior nas remoções. Essa variação no desempenho sugere que o tempo de resposta esperado da aplicação web criada pode ser insatisfatório em determinadas situações, dependendo da complexidade da consulta realizada, das especificações da máquina do usuário e até mesmo do estado emocional do usuário durante a interação com a aplicação.

Em suma, as conclusões deste trabalho reafirmam a importância de uma escolha criteriosa do SGBD e da implementação de índices adequados para otimizar as operações, assim como bom senso do prórpio usuário na hora de escolher a consulta a ser realizada e suas expectativas sobre o desempenho. Além disso, as limitações identificadas nas operações de atualização e remoção abrem espaço para futuras melhorias na aplicação. Desdobramentos potenciais incluem a expansão das funcionalidades da aplicação, como a implementação de novas opções de SGBDs e o uso de inteligência artificial para sugerir correções de código. Assim, não apenas a experiência do usuário será melhorada, mas também garantiriam que a aplicação se mantenha relevante e não distante do mundo real, evoluindo com o passar do tempo.

Referências

[Andjelic 2008] Svetlana Andjelic. *A performance analysis of the DBMS - MySQL vs PostgreSQL*. Dez. de 2008. url: https://www.researchgate.net/publication/296259511_A_performance_analysis_of_the_DBMS_-_MySQL_vs_PostgreSQL (acesso em 10/12/2024) (citado na pg. 28).

[AWS 2024] Amazon Web Services AWS. *What's the Difference Between MySQL and PostgreSQL?* Nov. de 2024. url: https://aws.amazon.com/compare/the-difference-between-mysql-vs-postgresql/ (acesso em 10/11/2024) (citado na pg. 7).

[Győrödi *et al.* 2021] Cornelia A. Győrödi *et al. Performance Impact of Optimization Methods on MySQL Document-Based and Relational Databases*. Jul. de 2021. url: https://www.mdpi.com/2076-3417/11/15/6794 (acesso em 10/11/2024) (citado na pg. 5).

[Haroon-Sulyman 2014] Shakirat Haroon-Sulyman. *Client-Server Model*. Jan. de 2014. url: https://www.researchgate.net/publication/271295146_Client-Server_Model (acesso em 12/11/2024) (citado na pg. 9).

[Kothapall 2022] Mounika Kothapall. *Performance Analysis of Single Page Applications*. Jan. de 2022. url: https://www.researchgate.net/publication/384459075_Performance_Analysis_of_Single_Page_Applications (acesso em 12/11/2024) (citado na pg. 9).

[Nah 2003] Fiona Fui-Hoon Nah. *A Study on Tolerable Waiting Time: How Long Are Web Users Willing to Wait?* Jan. de 2003. url: https://www.researchgate.net/publication/220893869_A_Study_on_Tolerable_Waiting_Time_How_Long_Are_Web_Users_Willing_to_Wait (acesso em 23/11/2024) (citado na pg. 27).

[Neumann *et al.* 2015] Thomas Neumann, Tobias Mühlbauer e Alfons Kemper. *Fast Serializable Multi-Version Concurrency Control for Main-Memory Database Systems*. Mai. de 2015. url: https://dl.acm.org/doi/10.1145/2723372.2749436 (acesso em 10/11/2024) (citado na pg. 6).

[Patel *et al.* 2023] Sindhubala Patel, Jitendra Choudhary e Govinda Patil. *Revolution of Database Management System: A literature Survey*. Jul. de 2023. url: https://www.researchgate.net/publication/372974295_Revolution_of_Database_Management_System_A_literature_Survey (acesso em 14/12/2024) (citado na pg. 31).

[Skublewska-Paszkowska 2021] Maria Skublewska-Paszkowska. *Comparison of the performance of relational databases PostgreSQL and MySQL for desktop application*. Mar. de 2021. url: https://ph.pollub.pl/index.php/jcsi/article/view/2314 (acesso em 10/12/2024) (citado na pg. 27).