Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Engenharia de Software

Bancos de Dados: Análise Comportamental de Desempenho entre Bancos de Dados Relacionais e Não Relacionais para Aplicações Web

Autor: Pedro Igor Oliveira Silva

Orientador: Prof. Dr. Maurício Serrano

Brasília, DF

2023

Pedro Igor Oliveira Silva

Bancos de Dados: Análise Comportamental de Desempenho entre Bancos de Dados Relacionais e Não Relacionais para Aplicações Web

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Prof. Dr. Maurício Serrano

Coorientador: Prof[a]. Dr[a]. Milene Serrano

Brasília, DF

2023

Pedro Igor Oliveira Silva

Bancos de Dados: Análise Comportamental de Desempenho entre Bancos de Dados Relacionais e Não Relacionais para Aplicações Web/ Pedro Igor Oliveira Silva. – Brasília, DF, 2023-

90 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Maurício Serrano

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2023.

1. banco de dados. 2. comparação. I. Prof. Dr. Maurício Serrano. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Bancos de Dados: Análise Comportamental de Desempenho entre Bancos de Dados Relacionais e Não Relacionais para Aplicações Web

CDU 02:141:005.6

Pedro Igor Oliveira Silva

Bancos de Dados: Análise Comportamental de Desempenho entre Bancos de Dados Relacionais e Não Relacionais para Aplicações Web

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 16 de fevereiro de 2023:

Prof. Dr. Maurício Serrano
Orientador

Profª. Drª. Milene Serrano
Coorientador

Prof. Dr. Fernando William Cruz
Examinador 1

M.Sc. Rafael Fazzolino P. Barbosa
Examinador 2

Brasília, DF
2023

Agradecimentos

A Deus, pela graça a mim concedida em estar concluindo um curso de graduação. Pela saúde, perseverança e paz em meio a semestres desafiadores. Pela resiliência, ao me tornar cada vez melhor em cada desafio superado, o que colaborou de forma ímpar para o profissional que tenho me tornado.

Aos meus pais, Pedro e Vilma, e à minha irmã, Ingrid, por terem me apoiado dentro e fora da universidade. Incentivaram a sempre buscar o meu melhor, e ter a certeza de que por meio dos estudos, dedicação e obediência a Deus, alcançaria os meus objetivos.

Aos meus orientadores, Maurício Serrano e Milene Serrano, pela paciência, dedicação e atenção ao me orientarem na construção deste trabalho. Ao corpo docente da Universidade de Brasília, que me motivaram e ensinaram durante todo o período de graduação.

Aos meu amigos, pelos momentos, pela amizade e até mesmo pela motivação em momentos em que me encontrava desanimado.

A todos que participaram de alguma forma desta graduação, meus agradecimentos.

Resumo

A decisão por um banco de dados pode ser uma tarefa árdua, dada a variedade de bancos de dados existentes na atualidade. Os bancos de dados possuem particularidades, as quais podem ser vantajosas para determinadas aplicações ou não. Há aplicações que compartilham de especificidades, que demandam que essa escolha seja mais embasada. Isso ocorre, por exemplo, quando uma aplicação deve se orientar por critérios qualitativos mais específicos, tais como o alto desempenho necessário em sistemas que atendem a muitas requisições ou sistemas de tempo real. Considerando um nível mais generalista de variabilidade de banco de dados, existem os tipos de bancos de dados: relacionais e não relacionais. Escolher por um tipo de banco de dados pode ser considerado um avanço na tomada de decisão sobre qual banco de dados é mais adequado, visto que elimina diversas opções de escolha. Cada um dos tipos de banco de dados armazena e gerencia os dados de forma diferente. Essas diferenças têm como consequência vantagens e desvantagens para a aplicação, dependendo da especificidade da mesma. Diante do exposto, a proposta deste trabalho é implementar funcionalidades comuns a aplicações web, em ambos os tipos de bancos de dados, e comparar, sob a perspectiva de desempenho. O tempo será o principal parâmetro aferido. A intenção é produzir insumos que auxiliem na decisão sobre qual tipo de banco de dados parece mais adequado para uma típica aplicação web, considerando desempenho como um critério qualitativo de relevância. Como possíveis evoluções do trabalho, outros critérios e métricas poderiam ser aferidos, incorporando novos insumos, e contribuindo ainda mais com o intuito principal do trabalho.

Palavras-chave: Banco de Dados, Banco de Dados Relacional, Banco de Dados Não Relacional, Desempenho, Tomada de Decisão.

Abstract

Deciding on a database can be an arduous task, given the variety of databases that exist today. Databases have particularities, which may or may not be advantageous for certain applications. There are applications that share specificities, which demand that this choice be more based. This happens, for example, when an application must be guided by more specific qualitative criteria, such as the high performance required in systems that serve many requests or real-time systems. On a more general level of database variability, there are database types: relational and non-relational. Choosing for a database type can be considered a step forward in making a decision about which database is most suitable, since it eliminates several options to choose from. Each of the database types stores and manages data differently. These differences result in advantages and disadvantages for the application, depending on its specificity. Given the above, the proposal of this work is to implement common functionalities for web applications, in both types of databases, and compare them, under the perspective of performance. Time will be the main parameter measured. The intention is to produce inputs that help in the decision about which type of database seems more adequate for a typical web application, considering performance as a qualitative criterion of relevance. As possible evolutions of the work, other criteria and metrics could be measured, incorporating new inputs, and contributing even more to the main purpose of the work.

Key-words: Database, Relational Database, Non-Relational Database, Performance, Decision Making.

Lista de figuras

Lista de tabelas

Lista de quadros

Lista de abreviaturas e siglas

SGBD	Sistema Gerenciador de Banco de Dados
PK	*Primary Key*
FK	*Foreign Key*
XML	*Extensible Markup Language*
JSON	*JavaScript Object Notation*
BSON	*Binary Javascript Object Notation*
PDF	*Portable Document Format*
SQL	*Structured Query Language*
DDL	*Data Definition Language*
DML	*Data Manipulation Language*
API	*Application Programming Interface*
HTTP	*Hypertext Transfer Protocol*
CSV	*Comma-separated Values*
ORM	*Object-Relational Mapping*

Sumário

1 Introdução

Este capítulo procura acordar uma Contextualização sobre bancos de dados relacionais e bancos de dados não relacionais. A intenção é conferir insumos para a compreensão da proposta desse trabalho, com foco, principalmente, na comparação entre esses tipos de banco de dados. Uma vez contextualizado, tem-se a Justificativa, visando a apresentação das contribuições almejadas com a realização do trabalho, com as quais pretende-se responder a Questão de Pesquisa. Serão apresentados, na sequência, os Objetivos e a Organização do Trabalho.

1.1 Contextualização

Ao iniciar um projeto em *software*, mais especificamente para *backend*, é necessário, como gestor ou líder técnico/tecnológico, tomar algumas decisões quanto a tecnologias, *frameworks* e ferramentas que serão utilizadas para o desenvolvimento do produto de *software*. Dentre essas decisões, existe a decisão de um Sistema Gerenciador de Bancos de Dados, e mais precisamente, a forma como os dados serão armazenados: de forma relacional ou não relacional.

Em um Banco de Dados Relacional, os dados são armazenados em um conjunto de tabelas, onde cada linha corresponde a uma única instância de uma categoria de dados, isto é, uma tupla. Durante a modelagem de um banco de dados relacional, são definidas restrições de integridade para o dado a ser armazenado, como, por exemplo: chave primária, *PRIMARY KEY*(PK), e chave estrangeira, *FOREIGN KEY*(FK). Tais restrições permitem que tabelas diferentes se relacionem. Outra característica que merece ser ressaltada, nessa abordagem relacional, é o processo de normalização. Em linhas gerais, trata-se da aplicação de uma série de passos, com determinadas regras sobre cada tabela do banco de dados, de forma a garantir o projeto adequado dessas tabelas. Dentre os conceitos básicos da normalização, há a separação de dados referentes a elementos distintos em tabelas distintas. Essas tabelas são associadas usando o conceito de chaves. Destaca-se ainda que os bancos de dados relacionais seguem o modelo ACID, sendo esse um conjunto de procedimentos relevante para preservar a integridade de uma transação (DIANA; GEROSA, 2010).

Diante do exposto, no mercado, são encontrados vários banco de dados relacionais,

dentre eles: Oracle[1], MySQL[2], Microsoft SQL Server[3] e PostgreSQL[4].

Já um Banco de Dados Não Relacional armazena a própria estrutura de dados. Como não existem restrições dos dados com relação a outras estruturas, não é possível realizar relacionamentos. Um banco de dados não relacional não garante as propriedades ACID, porém segue as propriedades BASE, sendo esse um conjunto de propriedades de bancos de dados não relacionais. (JATANA et al., 2012). A intenção, segundo Diana e Gerosa(2010), é conferir: escalabilidade, ausência de esquema ou esquema flexível, suporte nativo à replicação, dentre outros intuitos. Portanto, acredita-se na maior flexibilidade na estruturação do banco; melhorias em termos de desempenho, e maior escalabilidade.

Atualmente, existem diversas formas de armazenar os dados em bancos de dados não relacionais: chave valor, orientado a documento, orientado a grafo, orientado a colunas, orientado a objetos, *Grid and Cloud*, orientado a XML, bancos de dados multidimensionais, multivalor, multimodelo. Dentre tantas opções de armazenamento, vale detalhar o banco orientado a documento, visto que será utilizado no desenvolvimento do trabalho. Em um banco de dados orientado a documento, é utilizada uma chave única para referenciamento de um documento. Um documento é armazenado em um formato padrão como JSON, XML, BSON, PDF dentre outros. A grande desvantagem deste método de armazenamento encontra-se na necessidade de algumas ferramentas, em *software*, para que seja possível realizar consultas com base em valores dentro dos documentos (JATANA et al., 2012).

Diante do exposto, no mercado, são encontrados vários banco de dados não relacionais, dentre eles: MongoDB[5], Cassandra[6], e Amazon DynamoDB[7].

Como não é possível comparar desempenho de tipos de bancos de dados, essa comparação ocorre a partir de implementações de cada tipo de banco de dados, relacional e não relacional. O ideal é que várias alternativas, de cada tipo, sejam utilizadas em uma comparação. No entanto, este trabalho se limita a uma alternativa para cada tipo de banco de dados. Foi escolhido um banco de dados para representar cada um dos dois tipos. O PostgreSQL para o banco de dados relacional, e o MongoDB para o banco de dados não relacional. Esta decisão se baseou nos seguintes critérios:

- Custo e disponibilidade;
- Popularidade;

[1] https://www.oracle.com/database/
[2] https://www.mysql.com/
[3] https://www.microsoft.com/pt-br/sql-server
[4] https://www.postgresql.org/
[5] https://www.mongodb.com
[6] https://cassandra.apache.org
[7] https://aws.amazon.com/pt/dynamodb

- Documentação e comunidade, e
- Familiaridade do autor.

1.2 Justificativa

Conhecer os diferentes tipos de bancos de dados é importante para entender qual infraestrutura é a mais indicada para determinado negócio. Segundo Gyorödi, Gyorödi e Sotoc (2015), um banco de dados não relacional possui vantagens em operações de inserção, remoção e atualização, enquanto um Banco de Dados Relacional possui vantagens em operações de consulta. Os autores utilizaram no estudo os bancos MongoDB, como não relacional, e o MySQL[8] como relacional. O estudo realizou a comparação de tempo de resposta, conforme era incrementado o número de registros, para ambos os bancos, na realização de operações básicas: *SELECT*, *INSERT*, *UPDATE* e *DELETE*.

De forma geral, é necessário avaliar como essas vantagens e desvantagens se comportam na prática, visto que é uma escolha que promoverá ganhos e perdas. No entanto, um ganho pode não ser grande o suficiente para compensar uma perda, e vice-versa. Adicionalmente, torna-se pertinente decidir, para cada tipo de projeto, qual abordagem é mais adequada em um produto escalável, visando bom desempenho.

Para uma aplicação de mundo real, esse tipo de comparação, com base na realização de operações básicas sobre uma única tabela, pode não ser suficiente para uma decisão clara e precisa. Aplicações modernas, mais especificamente, aplicações web, nos tempos de hoje, costumam realizar operações sobre várias tabelas ou estruturas de dados. Sendo assim, para uma tomada de decisão mais embasada, torna-se pertinente analisar, complementarmente, operações estratégicas. Tais operações estratégicas são definidas a partir de operações semelhantes às operações realizadas em uma aplicação real, com múltiplos usuários, os quais podem, até mesmo, realizar operações de forma paralela.

1.3 Questão de Pesquisa

Diante do exposto, o presente Trabalho de Conclusão de Curso compreende um estudo exploratório sobre ambos os tipos de banco de dados, banco de dados relacional e não relacional, usando como insumo uma base de dados pública, com grande quantidade de dados. O intuito é realizar operações recorrentes em aplicações web. As operações, em cada tipo de banco de dados, serão comparadas utilizando a métrica tempo, isto é, o tempo entre a solicitação da operação e a disponibilidade da resposta. Com os resultados obtidos, pretende-se responder a seguinte questão de pesquisa: Como se comportam ambas

[8] https://www.mysql.com

as abordagens de banco de dados, relacional ou não relacional, considerando uma aplicação web moderna e o critério desempenho?

1.4 Objetivos

1.4.1 Objetivo Geral

Estudo exploratório em modelagem de banco de dados, relacional e não relacional, com foco em aplicações web modernas, procurando conferir insumos sobre como os tipos de bancos de dados se comportam em termos de desempenho. A métrica de avaliação de desempenho utilizada será o tempo para realização das operações, obtido com base na variação dos registros participantes da operação, tanto para o caso de uma única execução, quanto para o caso de execuções paralelas.

1.4.2 Objetivos específicos

Com o intuito de alcançar o objetivo geral deste trabalho, foram pontuados alguns objetivos específicos:

- Identificar as principais diferenças entre um banco de dados relacional e um não relacional;
- Abordar exemplos de bancos de dados relacionais e não relacionais;
- Descrever a base de dados exemplo escolhida;
- Modelar a base de dados exemplo, considerando um banco de dados relacional;
- Modelar a base de dados exemplo, considerando um banco de dados não relacional;
- Realizar o tratamento necessário sobre a base de dados exemplo para cada uma das abordagens a serem comparadas;
- Construir o protótipo básico para o frontend da aplicação;
- Desenvolver o frontend responsável por solicitar as operações e mostrar as comparações de desempenho;
- Apresentar as operações comuns em aplicações web modernas;
- Desenvolver as operações sobre cada banco de dados;
- Desenvolver os middlewares de monitoramento de tempo de resposta de cada requisição, e

- Analisar os resultados sobre os dados coletados durante o monitoramento da aplicação.

1.5 Organização do Trabalho

- Capítulo 2 - REFERENCIAL TEÓRICO: descreve a base teórica para as comparações realizadas neste trabalho, voltado para a compreensão de algumas definições e conceitos a respeito de bancos de dados;
- Capítulo 3 - REFERENCIAL TECNOLÓGICO: descreve todas as ferramentas utilizadas durante a construção deste trabalho, tanto parte escrita quanto parte prática e aplicada;
- Capítulo 4 - METODOLOGIA: descreve os aspectos metodológicos do levantamento bibliográfico do trabalho, além de apresentar o cronograma de atividades;
- Capítulo 5 - PRODUTO DE SOFTWARE: descreve o contexto de definição da proposta e como esta pretende auxiliar na solução da problemática. Especifica as etapas necessárias para a construção do produto de software, e
- Capítulo 6 - COLETA E ANÁLISE DE RESULTADOS: apresenta os principais resultados alcançados no durante o desenvolvimento e utilização do Produto de Software.

2 Referencial Teórico

Com o objetivo de compreender melhor a utilização de bancos de dados em aplicações web, este capítulo apresenta definições acerca de tópicos de interesse do presente trabalho.

2.1 Sistema Gerenciador de Bancos de Dados

É o conjunto de dados interrelacionados unido a um conjunto de programas úteis para acesso a esses dados. A coleção de dados, por si só, pode ser chamada de banco de dados. O Sistema Gerenciador de Bancos de Dados (SGBD) tem o objetivo de acessar os dados de forma eficiente ao realizar operações de:

- Definição: tarefa de especificar os tipos, estruturas e restrições dos dados. Essas informações descritivas dos dados também são armazenadas pelo SGBD e são chamadas de metadados, isto é, dados sobre os dados;
- Construção: processo de armazenar os dados para que seja controlável pelo SGBD;
- Manipulação: disponibilização de funções como consulta ao banco com o objetivo de recuperar dados específicos, e
- Compartilhamento: permite que vários usuários e programas acessem o banco de dados simultaneamente.

Um programa de aplicação acessa o banco de dados por meio de envio de consultas ou soliticações de transações ao SGBD. Uma consulta consiste no processo de recuperação de alguns dados enquanto uma transação consiste em realizar leituras e escritas no banco de dados (ELMASRI;NAVATHE, 2011).

2.2 SQL: Operações Básicas

A linguagem SQL é uma das principais vantagens de um banco de dados relacional. Como ela se tornou um padrão entre bancos de dados desse tipo, existe menor preocupação ao realizar migrações entre diferentes bancos de dados relacionais (ELMASRI;NAVATHE, 2011). A prova disso é a migração realizada de um banco de dados MySQL para um PostgreSQL, conforme exposto na subseção 5.5 Tratamento Sobre a Base de Dados Exemplo.

De acordo com Silberschatz (2020), a linguagem SQL é subdividida a partir dos tipos de comandos existentes:

- DDL: corresponde a comandos para definição e edição de esquemas de tabelas, além de exclusão de tabelas por meio dos comandos *CREATE*, *ALTER* e *DROP*, respectivamente, e
- DML: corresponde a comandos de inserção, atualização e remoção de dados por meio dos comandos *INSERT*, *UPDATE* e *DELETE*;

Existem outros tipos de comandos, além de outros comandos dentro dos tipos mencionados, no entanto, não participam do escopo do trabalho.

2.3 Esquema

O esquema de uma tabela corresponde a uma lista de atributos, seus domínios e tipos correspondentes. Um atributo corresponde a uma informação sobre determinada entidade, a qual deseja-se armazenar informações. Esta entidade é comumente representada por uma tabela ou estrutura.

O domínio de um atributo corresponde aos possíveis valores que um atributo pode assumir. Além disso, um atributo possui um tipo, geralmente vinculado a linguagens de programação. Por mais que o valor de um atributo possa ser alterado, ou assumir diferentes valores ao longo do tempo, um esquema geralmente não muda (SILBERSCHATZ, 2020).

Em resumo, esquema é a definição de tipo e domínio que cada atributo pode assumir dentro de uma tabela ou estrutura. Um banco de dados comumente possui várias tabelas e estruturas, sendo necessária a definição de vários esquemas, um para cada tabela ou estrutura.

Um exemplo de esquema pode ser visualizado no Apêndice B. Cada uma das chaves no primeiro nível corresponde a uma estrutura, enquanto cada chave dentro de cada estrutura corresponde a um atributo, que por sua vez possui como valor seu tipo correspondente.

2.4 ACID

Conjunto de procedimentos relevante para preservar a integridade de uma transação em um banco de dados relacional. Divide-se em quatro propriedades: Atomicidade, Consistência, Isolamento, e Durabilidade. As iniciais de cada propriedade define a sigla. Nesse contexto, as propriedades acordam, respectivamente, que:

1. as ações de uma transação devem ser concluídas com sucesso. Garante que todas as ações de uma transação sejam executadas ou nenhuma seja. Em caso de falha, há necessidade de *rollback*, que retorna para um estado pré-execução das operações;

2. regras e restrições estabelecidas devem ser cumpridas, garantindo a passagem de um estado consistente para outro consistente;
3. uma dada transação não deve ser interferida por outra transação concorrente, e
4. os resultados de uma transação são permanentes. Sendo assim, se salvos, não serão mais perdidos. (DIANA; GEROSA, 2010)

2.5 BASE

São propriedades comuns de bancos de dados não relacionais. As iniciais, em inglês definem a sigla: *Basically Available*(Basicamente disponível), *Soft State*(Estado suave) e *Eventually consistent*(Eventualmente consistente). As propriedades acordam, respectivamente, que:

1. nem sempre um dado estará disponível, isto é, estará disponível na maior parte das vezes no entanto pode-se encontrar indisponível por inúmeras razões;
2. o estado do dado pode ser alterado sem interações da aplicação, e
3. o estado será eventualmente consistente até que o dado seja replicado a todos os servidores, quando alcançará um estado consistente (JATANA et al., 2012).

Ao contrário das propriedades 2.4 ACID que ajudam a manter a integridade do banco de dados quando respeitadas, as propriedades BASE são apenas propriedades comuns entre bancos de dados não relacionais. Podem ser consideradas premissas, e até mesmo desvantagens na escolha de um banco de dados não relacional que devem ser consideradas na escolha de utilização.

2.6 Teorema CAP

O teorema CAP foi proposto, inicialmente, por Eric Brewer[1], em 2000, e foi provado, por Seth Gilbert[2] e Nancy Lynch[3], dois anos após. Este teorema prova que é impossível, para um sistema distribuído, prover consistência, disponibilidade e tolerância em partições ao mesmo tempo. É possível coincidir apenas duas, destas três propriedades como mostrado na Figura 1. É possível prever os três cenários existentes nesse teorema:

- se um sistema distribuído possui requisitos com alta necessidade de consistência e tolerância em partições, então, o usuário deve aceitar falhas em operações de escrita e leitura, devido a indisponibilidade;

[1] https://en.wikipedia.org/wiki/Eric_Brewer_(scientist). Último acesso em 16/09/2022
[2] https://www.comp.nus.edu.sg/~gilbert/. Último acesso em 16/09/2022
[3] https://en.wikipedia.org/wiki/Nancy_Lynch. Último acesso em 16/09/2022

- se o usuário opta por alta disponibilidade e tolerância em partições, a consistência será prejudicada e uma leitura incorreta poderá ocorrer, e
- se o usuário opta por consistência e disponibilidade, não existe garantia de que o sistema continue operante mesmo após uma falha.

Os Bancos de Dados Relacionais costumam optar por alta disponibilidade e consistência, resultando em uma fraca escalabilidade horizontal e pouco espaço para melhorias. Já os Bancos de Dados Não Relacionais costumam prover soluções que equilibram as três possibilidades para cumprirem diversos requisitos.(HE, 2015)

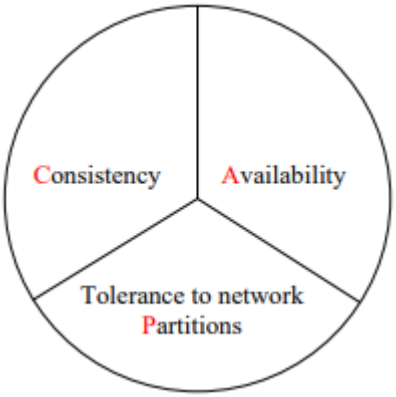

Figura 1 – Teorema CAP

Fonte: HE, 2015.

2.7 Banco de Dados Relacional

Os Bancos de Dados Relacionais foram inventados em 1970 por Edgar Frank Codd. Um banco de dados relacional é uma coleção de itens de dados organizados em tabelas formalmente descritas que podem ser acessados ou modificados de diferentes formas. Uma base de dados relacional é um conjunto de tabelas, também chamadas de relações, e, similarmente a planilhas, possui categoria de dados descritas por meio de colunas. Cada linha contém uma única instância de dados para a categoria de dados correspondente. Durante a criação de uma base de dados relacional, é definido o domínio de possíveis valores por meio de restrições que serão aplicadas aos dados. A relação entre tabelas diferentes permite que este conjunto de tabelas seja chamado de relações.

A maioria dos bancos de dados relacionais utiliza linguagem de consulta estruturada, também conhecida pela sigla SQL, para acessar e modificar os dados armazenados. Originalmente, o SQL foi baseado em cálculos relacionais e álgebra relacional, sendo subdividido em elementos como cláusulas, predicados, consultas e declarações.

Alguns dos benefícios de uma base de dados modelada de acordo com o modelo relacional são:

- a maioria das informações é armazenada no banco de dados e não na aplicação, logo, o banco de dados é auto documentado;
- a facilidade em adicionar, atualizar ou deletar dados;
- benefícios na sumarização, recuperação e predição de dados;
- a base de dados é estruturada em um formulário tabular com tabelas altamente relacionadas; a natureza da base de dados é predição, e
- qualquer alteração no esquema da base de dados pode ser realizada de forma simples (JATANA et al., 2012).

2.8 Banco de Dados Não Relacional

Os Bancos de Dados Não Relacionais são um tipo de sistemas que gerenciam bancos de dados, porém, é amplamente diferente de sistemas relacionais em diversas formas:

- não utiliza relações, isto é, tabelas como estrutura de armazenamento;
- não utiliza linguagens estruturadas de consulta, como o SQL;
- não garante as propriedades ACID, e
- pode ser escalado horizontalmente.

Existem diversas classificações que categorizam os bancos de dados não relacionais. Uma delas, é baseada no 2.6 Teorema CAP. Além disso, conforme (JATANA et al., 2012), os bancos de dados não relacionais podem se diferenciar na forma de organização dos dados:

- *Key Value Stores*
- *Document Store*
- *Graph Database*
- *Column Oriented Databases*
- *Object Oriented Databases*
- *Grid and Cloud Databases*
- *XML Databases*
- *Multidimensional Databases*
- *Multivalue Databases*
- *Multimodel Databases*

2.9 Vantagens e desvantanges entre bancos de dados relacionais e não relacionais

O autor Jatana et al. (2012) fez uma pesquisa de comparação entre bancos de dados relacionais e não relacionais. A partir dos resultados, foi possível construir o Quadro 1.

Quadro 1 – Comparação entre bancos de dados relacionais e não relacionais

Não Relacional	**Relacional**
Alta taxa de transferência de dados	Baixa taxa de transferência de dados
Altamente escalável	Menos escalável
O dado pode ser inserido a qualquer momento sem necessidade de definição de esquema. O dado pode ser alterado a qualquer momento sem muitos problemas. Logo, é altamente flexível.	O dado precisa corresponder as tabelas ou estruturas previamente definidas.
Performance pode ser melhorada ao utilizar cache de dados na memória do sistema.	Cache pode ser utilizado por meio de infraestruturas especiais.
Múltiplas operações de transação podem ser implementadas pela aplicação.	Transacional, isto é, não permite que mais de uma transação seja executada simultaneamente.
Índice único, chave armazena valor.	Índices disponíveis em múltiplas colunas.
Segue as propriedades BASE.	Segue as propriedades ACID.
Podem comprometer a consistência de dados.	Provê melhor consistência de dados que de bancos não relacionais.
Duplicação de dados é permitida, o que ameaça a integridade dos dados. Geralmente, atualizar um registro significa atualizar todos os registros duplicados, o que significa trabalho extra.	O processo de modelagem de banco de dados relacional visa eliminar duplicação de dados, que evita inconsistência.
Consultas são ineficientes especialmente quando baseadas em múltiplos critérios, já que requerem mais de uma comparação.	Permitem a utilização de linguagens de consulta como SQL para utilização das chaves primárias compartilhadas entre tabelas para rapidamente e eficientemente encontrar e retornar os registros solicitados.

Fonte: Jatana et al. (2012). Traduzido e adaptado pelo autor.

2.10 Middleware

Middleware são funções que têm acesso aos objetos de solicitação e resposta de uma requisição, além da próxima função, isto é, o próximo *middleware* a ser executado. Um middleware consegue realizar as seguintes tarefas:

- Executar qualquer código;
- Fazer mudanças nos objetos de solicitação e resposta;
- Encerrar o ciclo de solicitação-resposta, e
- Chamar o próximo middleware da pilha.

No Apêndice A é possível visualizar os elementos de um *middleware*, sendo:

- **get** corresponde ao método *http* utilizado na requisição;
- '/' corresponde a rota a qual a função será aplicada, e
- ***function*** a declaração da função que será executada, isto é, o próprio *middleware*;

Os parâmetros do *middleware* são:

- **req**, que corresponde ao objeto de solicitação da requisição;
- **res** que corresponde ao objeto de resposta da requisição, e
- ***next*** que corresponde a a próxima função a ser executada, isto é, o próximo *middleware*.

Desta forma, vários *middlewares* podem ser empilhados e executados antes e depois do próprio método responsável por tratar a requisição.

Os *middlewares* serão utilizados para o monitoramento e armazenamento de tempo de resposta de cada requisição da aplicação, conforme discutido na subseção 5.9 Métodos para coleta e análise de resultados.

2.11 Considerações Finais

Este capítulo apresentou as bases teóricas que fundamentam este Trabalho de Conclusão de Curso. Inicialmente, foram apresentados conceitos inerentes ao universo de Bancos de Dados, suas diferenças, além de termos relevantes na compreensão deste tema. Em seguida, foi apresentada a definição de *middleware*, representando a camada do software, responsável por solicitar operações aos Bancos de Dados.

3 Referencial Tecnológico

Este capítulo especifica os programas e ferramentas utilizadas durante o desenvolvimento deste trabalho. A seção 3.1 Desenvolvimento da monografia especificará as ferramentas utilizadas na parte escrita deste trabalho, enquanto a seção 3.2 Desenvolvimento técnico especificará as ferramentas utilizadas na parte prática e aplicada deste trabalho. Ao final, têm-se as considerações finais do capítulo.

3.1 Desenvolvimento da monografia

3.1.1 Overleaf v2

Overleaf[1] é um editor de texto LaTeX que permite a escrita/revisão de forma compartilhada e fornece um compilador em tempo real para a visualização dos documentos. O software foi utilizado por meio de sua versão oficial disponível via website.

3.1.2 Git 2.30.2 e GitHub 3.6.0

O Git[2] é um *software* para gerenciamento de versões de código fonte. Cada edição deste documento foi versionada por meio do *software*, que permite controle sobre cada edição realizada, além de restauração de versões específicas. O Git funciona de forma local, isto é, gerando versões que podem ser restauradas sem a necessidade de conexão com a internet. O GitHub[3] foi utilizado como repositório em nuvem para armazenamento de cada uma das versões escritas e versionadas pelo Git.

3.1.3 JabRef 5.7

O JabRef[4] é uma ferramenta para organizar referências bibliográficas. As informações de uma referência são preenchidas na interface da ferramenta, a qual disponibiliza diversos formatos de referência para serem utilizados em artigos e publicações. No trabalho, esta ferramenta foi utilizada para gerar o código BibTex referente a algumas referências, que não se encontravam disponíveis no *site* de publicação.

[1] https://www.overleaf.com. Último acesso em 30/08/2022
[2] https://git-scm.com. Último acesso em 30/08/2022
[3] https://github.com. Último acesso em 30/08/2022
[4] https://www.jabref.org/. Último acesso em 30/08/2022

3.2 Desenvolvimento técnico

3.2.1 Git 2.30.2 e GitHub 3.6.0

O Git também foi utilizado para versionamento de código fonte da aplicação. Assim como na monografia, foram geradas versões, submetidas em nuvem ao GitHub.

3.2.2 Node 16.16.0

O Node[5] é um interpretador de código JavaScript que funciona de forma assíncrona. O interpretador é multiplataforma e escalável. Será utilizado para realizar tanto o tratamento da base de dados exemplo para os bancos MongoDB e PostgreSQL, quanto para criar uma interface de programação de aplicações, também conhecida pela sigla API além de implementar o *dashboard* onde as informações consultadas serão visualizáveis. O usuário realizará uma solicitação via requisição *http*, tendo como retorno os dados solicitados, assim como o tempo necessário para o retorno. O tempo de processamento torna-se importante, visto que será utilizado nas comparações de desempenho entre os bancos.

3.2.3 MongoDB 6.0

MongoDB[6] é um banco de dados não relacional orientado a documentos JSON. O MongoDB, assim como muitos bancos de dados não relacionais, tem foco em escalabilidade e flexibilidade. A escolha deste banco deu-se por ser um dos mais utilizado atualmente, conforme descrito na seção 5.2 Descrição da Base de Dados Exemplo. Neste trabalho, o MongoDB será utilizado para armazenar a base de dados exemplo, a qual permitirá consultas que serão metrificadas em tempo.

3.2.4 PostgreSQL 14.3

PostgreSQL[7] é um Sistema Gerenciador de Bancos de Dados relacional *Open Source*. Seu foco é permitir a implementação de linguagem SQL em estruturas. A escolha deste SGBD deu-se pela popularidade e pela disponibilidade de mais funcionalidades, quando comparado com o MySQL, conforme descrito na 5.2 Descrição da Base de Dados Exemplo. Neste trabalho, o PostgreSQL será utilizado para armazenar a base de dados exemplo, novamente, permitindo consultas que serão metrificadas em tempo.

[5] https://nodejs.org. Último acesso em 30/08/2022
[6] https://www.mongodb.com/. Último acesso em 30/08/2022
[7] https://www.postgresql.org/. Último acesso em 30/08/2022

3.2.5 Docker 4.11.1

Docker[8] é uma plataforma com objetivo de facilitação no desenvolvimento, implantação e execução de aplicações em ambientes isolados. Ao construir uma aplicação, por muitas vezes torna-se necessária a instalação de pacotes, *frameworks* e bibliotecas que podem se comportar de maneiras diferentes a depender do computador e do sistema operacional utilizados. Ao criar um ambiente isolado, é possível construir toda a infraestrutura da aplicação de forma agilizada. Neste trabalho, o Docker será utilizado na construção da infraestrutura da aplicação que envolve um banco de dados PostgreSQL, um banco de dados MongoDB, além de um servidor Node como API. Adicionalmente, foi possível automatizar a etapa de tratamento e recuperação da base de dados exemplo, por meio de *scripts* auto executados no processo de construção da aplicação.

3.2.6 NextJS 13

NextJS [9] é uma estrutura web de desenvolvimento que concilia o framework React JS[10] e a disponibilização de uma API Node. O framework permite a realização de renderização do lado do servidor e geração de sites estáticos para aplicações web. O framework foi escolhido por estar entre os mais utilizados na atualidade além de possuir uma implementação nativa do modelo de *dashboard* definido na seção 5.8.

3.2.7 Python 3.9.12

Python [11] é uma linguagem de programação de alto nível que interpreta scripts. A linguagem foi escolhida para a coleta de resultados por permitir a utilização de programação paralela por meio de *threads*. Cada *thread* terá o papel de simular um usuário utilizando a aplicação.

3.3 Considerações Finais

O Quadro 2 descreve as tecnologias utilizadas tanto na parte escrita, quanto na parte prática deste Trabalho de Conclusão de Curso.

[8] https://www.docker.com/. Último acesso em 30/08/2022
[9] https://nextjs.org/. Último acesso em 15/11/2022
[10] https://reactjs.org/. Último acesso em 15/11/2022
[11] https://www.python.org/. Último acesso em 24/11/2023

Quadro 2 – Resumo das tecnologias utilizadas no trabalho

Nome	Descrição de uso	Versão
Overleaf	Edição do texto LaTeX e compilação do Trabalho de Conclusão de Curso.	V2
Git	Software para gerenciamento de versões tanto da parte escrita quanto da parte aplicada deste trabalho.	2.30.2
Github	Repositório em nuvem para armazenamento das versões do trabalho.	3.6.0
JabRef	Ferramenta para organização das referências bibliográficas do trabalho.	5.7
Node	Interpretador de código JavaScript utilizado na interface de programação da aplicação.	16.6.0
MongoDB	Banco de Dados não relacional orientado a documentos JSON utilizado na aplicação.	6.0
PostgreSQL	Banco de Dados relacional *Open Source* utilizado na aplicação.	14.3
Docker	Ferramenta para isolar e prover as dependências do servidor e do banco de dados em contêineres.	4.11.1
NextJS	Framework para desenvolvimento de aplicações web utilizando ReactJS.	13.0.3
Python	Linguagem de programação, de alto nível, que interpreta scripts.	3.9.12

Fonte: Autor, 2022.

4 Metodologia

No presente capítulo, será especificada a metodologia adotada para o desenvolvimento do trabalho. O desenvolvimento do trabalho envolve parte teórica e prática. Serão apresentadas a Metodologia de Pesquisa Científica, o Fluxo de atividades, o Cronograma e as Considerações Finais.

4.1 Metodologia de Pesquisa Científica

De acordo com Fonseca (2002), metodologia é o estudo do caminho a ser percorrido ao realizar uma pesquisa ou estudo. Já, de acordo com Apolinário (2011), pesquisa é uma investigação sistemática sobre um assunto com o objetivo de obter novas informações ou reorganizar informações existentes sobre determinado problema. GERHARDT T. E.; SILVEIRA (2009) diz que a pesquisa científica pode ser definida como o resultado de um exame cuidadoso que tem como objetivo resolver um dado problema. Neste tipo de pesquisa, são utilizados procedimentos científicos, o que a caracteriza. De acordo com GERHARDT T. E.; SILVEIRA (2009), a pesquisa é a atividade nuclear da ciência. Ela permite uma aproximação e um entendimento da realidade a investigar. É possível diferenciar tipos de pesquisa quanto à abordagem, natureza, objetivos e procedimentos, além de classificar qual a modalidade adequada ao objeto de pesquisa.

Com base em GERHARDT T. E.; SILVEIRA (2009), esta seção tem o objetivo de classificar a presente pesquisa científica quanto à abordagem, natureza, objetivos, procedimento e modalidade adequada.

4.1.1 Abordagem

De acordo com GERHARDT T. E.; SILVEIRA (2009), existem duas abordagens de pesquisa possíveis: quantitativa e qualitativa. A abordagem quantitativa tem o objetivo de ser objetiva em sua busca de analisar dados brutos coletados de forma padronizada. Geralmente, é associada a procedimentos estatísticos. Já a abordagem qualitativa tem o enfoque em aspectos não quantificados, isto é, analisa relações, processos e fenômenos que não podem ser simplificados a utilização de variáveis. A pesquisa científica realizada neste trabalho pode ser definida tanto como qualitativa quanto como quantitiva.

4.1.2 Natureza

Quanto à natureza, a pesquisa deste trabalho classifica-se como aplicada. De acordo com GERHARDT T. E.; SILVEIRA (2009), uma pesquisa científica de natureza aplicada

tem o objetivo de gerar conhecimentos para aplicação prática para a solução de um problema específico.

4.1.3 Objetivos

Para Gil (2002), é possível classificar uma pesquisa quanto a objetivo em três grupos:

- Pesquisa exploratória: Objetivo de trazer familiaridade com o problema;
- Pesquisa descritiva: Objetivo de descrever uma série de informações sobre o que deseja pesquisar, e
- Pesquisa explicativa: Objetivo de identificar os fatores determinantes ou contribuintes para a ocorrência dos fenômenos.

A pesquisa científica do presente trabalho classifica-se como pesquisa exploratória, visto que visa trazer conhecimento sobre a existência do problema.

4.1.4 Procedimentos

De acordo com Fonseca (2002), uma pesquisa processa-se atráves de aproximações repetitivas da realidade, sempre fornecendo insumos para uma intervenção real. Existem diversas classificações para uma pesquisa científica quanto a procedimento. O presente trabalho pode ser classificado em duas:

- Pesquisa bibliográfica: De acordo com Fonseca (2002), é feita a partir de levantamento de referências teóricas previamente analisadas e publicadadas. Qualquer trabalho científico inicia-se com uma pesquisa bibliográfica, e
- Pesquisa-ação: De acordo com Fonseca (2002), é esperada uma participação planejada do pesquisador na situação a ser investigada. Procura-se diagnosticar um problema com a intenção de alcançar um resultado prático.

4.2 Pesquisa Bibliográfica

A pesquisa bibliográfica foi desenvolvida a partir de materiais como livros e artigos científicos previamente publicados. Foram realizadas pesquisas bibliográficas que embasaram a escrita do trabalho, visto que trabalhos científicos iniciam-se com pesquisa bibliográfica (FONSECA, 2002).

4.2.1 *String* de Busca

Uma vez que o tema foi escolhido, foram elaboradas algumas *strings* de busca, úteis para a realização da pesquisa bibliográfica. Para a pesquisa por livros, foi utilizada a base de dados Pearson e Minha Biblioteca, disponíveis no website Biblioteca Central Online[1], da UnB. Como o objetivo, nas pesquisas por livros, era encontrar a definição de conceitos introdutórios gerais inerentes a sistema de bancos de dados, apenas a *string* "Sistema de Banco de Dados" foi utilizada. Foram encontrados os seguintes livros:

- Sistemas de banco de dados, 6ª edição, ELMASRI;NAVATHE, 2011, e
- Sistema de Banco de Dados, 7ª edição, SILBERSCHATZ, 2020.

Para a pesquisa por artigos científicos de temática semelhante, foi utilizada a base de dados Google Scholar[2]. Foram utilizadas as seguintes *strings* de busca:

- "comparativo bancos de dados";
- "relational and non-relational database comparative";
- "relational and non-relational database comparison";
- "mongodb modeling".

A pesquisa resultou em diversos resultados que precisaram ser filtrados. Alguns artigos puderam ser filtrados a partir do título, visto que não se relacionavam com o tema esperado. Outros artigos foram filtrados a partir do *abstract* ou resumo, além das palavras-chaves. Foram utilizados os seguintes artigos:

- Nosql na web 2.0: Um estudo comparativo de bancos não-relacionais para armazenamento de dados na web 2.0, Workshop de Teses e Dissertações de Bancos de Dados do Simpósio Brasileiro de Bancos de Dados, DIANA; GEROSA, 2010;
- Modeling and querying data in mongodb, International of Scientific and Engineering Research, ARORA; AGGARWAL, 2013;
- A comparative study of relational and non-relational database models in a web-based application, International Journal of Advanced Computer Science and Applications, GYORÖDI; GYORÖDI; SOTOC, 2015;
- A survey and comparison of relational and non-relational database, International Journal of Engineering Research & Technology, JATANA et al., 2012;

[1] https://minhabcedigital.bce.unb.br/
[2] https://scholar.google.com.br/

- Pro Hibernate and MongoDB, LEONARD, 2013, e
- Crud operations in mongodb, TRUICA; BOICEA; TRIFAN, 2013.

Além disso, foi utilizado o artigo "Survey on nosql database technology, Journal of Applied Science and Engineering Innovation Vol, HE, 2015" encontrado entre as referências do artigo "A survey and comparison of relational and non-relational database, International Journal of Engineering Research & Technology, JATANA et al., 2012".

4.3 Fluxo de atividades

Para a elaboração deste trabalho, foi seguido um fluxo de atividades. A ordem e paralelização deste fluxo é visualizável na Figura 2. A seguir, é possível visualizar a descrição de cada uma das atividades do fluxo:

- **Definir Tema**: com o auxílio dos orientadores, escolher um tema de interesse para desenvolvimento do trabalho, dentro da área de Engenharia de Software. Decidiu-se trabalhar com uma comparação de desempenho e modelagem entre bancos de dados relacionais e não relacionais, com o objetivo de produzir insumos que auxiliem na decisão de tipo de banco de dados;
- **Realizar levantamento bibliográfico**: reunir materiais que descrevam a utilização de bancos de dados relacionais e não relacionais e seus impactos. Reunir materiais relacionados, que realizam algum tipo de comparação performática em ambos os tipos de bancos de dados;
- **Formular introdução**: desenvolver a definição da problemática além de objetivos gerais e específicos do trabalho;
- **Formular metodologia de pesquisa**: especificar as metodologias de pesquisa e desenvolvimento utilizadas na elaboração do trabalho;
- **Formular referencial teórico**: delinear o referencial teórico em relação a bancos de dados relacionais e não relacionais com o objetivo de inteirar o leitor sobre alguns termos importantes utilizados no desenvolvimento do trabalho;
- **Definir proposta de trabalho**: detalhar o contexto da problemática além da proposta de solução. Definir estratégia utilizada para auxílio na solução da problemática. Descrever os passos necessários para a implementação e desenvolvimento da proposta;
- **Definir métodos de coleta e análise dos resultados**: definir estratégia para coletar os resultados e métodos para análise dos resultados obtidos;

- **Desenvolver prova de conceito**: definir arquitetura de software e desenvolver uma primeira versão do sistema;
- **Estabelecer suporte tecnológico**: definir as tecnologias que possibilitam o desenvolvimento da proposta;
- **Refinar monografia**: realizar ajustes finais em relação à escrita que será entregue na primeira etapa, e
- **Apresentar à banca**: entregar texto e apresentar o trabalho para a banca.

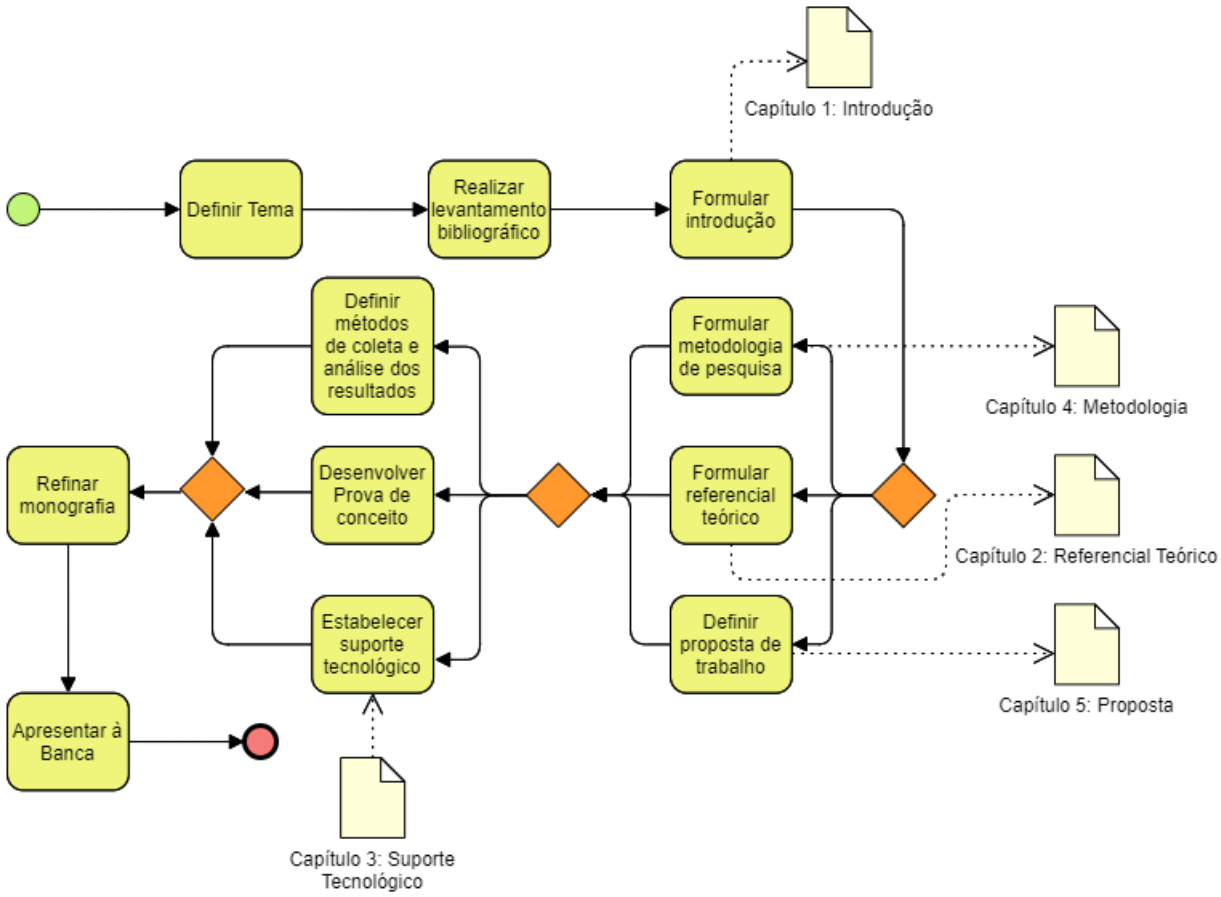

Figura 2 – Atividades relacionadas à elaboração do TCC1

Fonte: Autor, 2022.

4.4 Cronograma

É possível visualizar o cronograma de atividades planejadas para a escrita do TCC1, assim como o mês de finalização de cada atividade, na Tabela 1. Similarmente ao TCC1, é possível visualizar o cronograma de atividades planejadas para a escrita do TCC2, assim como o mês de finalização esperado de cada atividade, na Tabela 2.

Tabela 1 – Cronograma de Atividades do TCC1

Atividade	Junho/2022	Julho/2022	Agosto/2022	Setembro/2022
Definir Tema	X			
Realizar Levantamento Bibliográfico	X			
Formular Proposta		X		
Contextualizar Problema		X		
Definir Referencial Tecnológico			X	
Definir Metodologia			X	
Refinar Proposta			X	
Desenvolver Prova de Conceito				
Revisar TCC1				X
Apresentar TCC1				X

Fonte: Autor, 2022.

Tabela 2 – Cronograma de Atividades do TCC2

Atividade	Outubro/2022	Novembro/2022	Dezembro/2022	Janeiro/2023	Fevereiro/2023
Aplicar correções sugeridas	X				
Desenvolver o Produto de Sofware		X	X		
Analisar os Resultados			X	X	
Descrever Resultados			X	X	
Realizar Correções Necessárias					X
Apresentar					X

Fonte: Autor, 2022.

4.5 Considerações Finais

Este capítulo apresentou as metodologias utilizadas na escrita deste trabalho. Foi possível classificar a metodologia em diferentes aspectos: quanto à abordagem, trata-se de uma abordagem quantitativa e qualitativa; quanto à natureza, trata-se de um trabalho de pesquisa aplicada; quanto aos objetivos, foi possível classificá-lo como pesquisa exploratória, descritiva e explicativa; quanto aos procedimentos, foram utilizados os procedimentos de pesquisa bibliográfica e pesquisa-ação. Foram apresentadas as estratégias de pesquisa bibliográfica utilizadas no trabalho, assim como as *strings* de busca utilizadas e materiais encontrados. Por fim, foram cobertos o fluxo de atividades realizadas e os cronogramas de entrega de atividades com relação a ambas as etapas de entrega deste Trabalho de Conclusão de Curso, sendo TCC1 e TCC2.

5 Produto de Software

Este capítulo apresentará, em detalhes, o produto de software resultante deste trabalho. Inicialmente, serão tratados o Contexto Temático que embasa a escolha deste tema, assim como a Descrição da Base de Dados Exemplo. Em seguida, serão trabalhados a Modelagem da Base de Dados Exemplo Considerando um Banco de Dados Não Relacional e a Modelagem da Base de Dados Exemplo Considerando um Banco de Dados Relacional. Dando continuidade, serão apresentadas técnicas para o Tratamento Sobre a Base de Dados Exemplo, Operações Comuns em Aplicações Web Modernas e, então, Métodos para coleta e análise de resultados. Por fim, têm-se as Considerações Finais do Capítulo.

5.1 Contexto Temático

Os sistemas de bancos de dados existem desde o início do século XX, quando Herman Hollerith[1] inventou os cartões perfurados. Ainda que este método fosse bastante rudimentar, revela necessidade de armazenamento e recuperação de informações antes mesmo de existirem computadores. Os primeiros computadores modernos foram criados por volta dos anos 1940 por Alan Turing[2]. Com a evolução dos computadores, os dispositivos de armazenamento evoluíram, assim como as técnicas para armazenamento de dados. A história da computação descreve a utilização de fitas magnéticas, uma técnica de armazenamento sequencial, baseada na utilização de cartões perfurados. A década de 70 demarcou a utilização de discos rígidos em computadores. A importância deste método de armazenamento dá-se pela possibilidade de acesso direto aos dados. Ao contrário da fita magnética, que podia ser lida apenas de forma sequencial, a posição de leitura de um disco rígido é indiferente (SILBERSCHATZ, 2020).

A partir da utilização de discos rígidos para armazenamento dos dados, a responsabilidade de evoluções das técnicas de armazenamento foi compartilhada com a evolução de produtos de *software* para gerenciamento de bancos de dados. Em 1970, Edward Codd definiu o modelo relacional e os métodos não procedurais, isto é, métodos declarativos de consulta, dando origem aos bancos de dados relacionais. Já na década de 2000, com a necessidade de desenvolvimento rápido, principalmente para *startups*, houve crescimento na utilização de bancos de dados não relacionais(SILBERSCHATZ, 2020). Segundo Diana e Gerosa (2010), os bancos de dados não relacionais prometem maior flexibilidade, escalabilidade e disponibilidade no uso de uma aplicação, porém, diferentemente dos bancos de dados relacionais, possuem consistência eventual. Consistência é a garantia de que cópias

[1] https://www.britannica.com/biography/Herman-Hollerith. Último acesso em 16/09/2022
[2] https://www.britannica.com/biography/Alan-Turing. Último acesso em 16/09/2022

dos mesmos dados são idênticas.

A falta de suporte à consistência e às consultas declarativas tornaram-se aceitáveis para muitas aplicações. Considerando que ambos os tipos de bancos de dados fornecem vantagens e desvantagens, é necessário tomar esta importante decisão: qual o mais adequado para determinada aplicação?

Tendo em mente uma aplicação que pode ser atendida por ambos os tipos de bancos de dados, os autores Gyorödi, Gyorödi e Sotoc (2015) realizaram comparações de desempenho em ambos os tipos de bancos de dados. Foram realizadas operações básicas de *SELECT*, *INSERT*, *UPDATE* e *DELETE* em um banco de dados MongoDB e um banco de dados MySQL. O tempo de resposta em cada banco foi medido em segundos e realizadas comparações de tempo em cada operação conforme é possível visualizar na Figura 3. É possível verificar na Figura 3, que operações de inserção, atualização e remoção costumam tomar mais tempo no MySQL conforme a quantidade de registros aumenta, enquanto operações de consulta costumam tomar mais tempo no MongoDB.

Essas comparações de tempo ajudam na tomada de decisão ao pensar na experiência do usuário, já que de acordo com Valentim, Silva e Conte (2015), esta é uma métrica importante na avaliação de experiência do usuário. Quanto menos tempo é tomado para que uma tarefa seja realizada, melhor se torna essa experiência. No entanto, a comparação de desempenho em operações básicas de bancos de dados pode não ser suficiente na escolha de um sistema para aplicações modernas. O autor Leonard (2013) cria do zero a modelagem de dados para uma aplicação de *e-Commerce*, e menciona funcionalidades que vão além de operações básicas de bancos de dados. São implementadas funcionalidades como: listagem, listagem com paginação, busca por nome ou palavras dentro do nome e consulta de itens por agregação. Torna-se pertinente a necessidade de comparar desempenho, em tempo de resposta, para cada uma destas funcionalidades visando escolher a melhor alternativa de tipo de bancos de dados para uma aplicação web moderna.

A proposta deste trabalho é fornecer insumos que possam ajudar na escolha do tipo de banco de dados mais adequado para determinada aplicação. Serão implementadas funcionalidades utilizadas em aplicações modernas, realizando consultas, em ambos os tipos de bancos de dados. Será possível comparar tanto a complexidade, na modelagem e implementação de ambas as soluções, quanto o tempo de resposta para cada funcionalidade.

Será construída uma aplicação web que implementa as funcionalidades, realizando consultas, em bancos de dados de ambos os tipos. O usuário será capaz de realizar ações dentro da aplicação e comparar o tempo de resposta para cada ação em cada um dos bancos de dados. Para realização das comparações, foi escolhida uma base de dados com uma grande quantidade de tuplas registradas como base de dados exemplo. A mesma base de dados será utilizada na realização das operações em ambos os tipos de bancos de

dados.

Figura 3 – Comparação de tempo em operações de inserção, atualização, remoção e consulta.

Fonte: Gyorödi, Gyorödi e Sotoc

5.2 Descrição da Base de Dados Exemplo

A base de dados exemplo[3] escolhida é disponibilizada publicamente pelo MySQL e dispõe de 3.919.015 tuplas distribuídas sobre seis tabelas. As seis tabelas são: *employees*, que armazena informações do funcionário; *titles* que armazena os títulos de um funcionário; *salaries* que armazena os salários de um funcionário; *departments* que armazena os departamentos da empresa; *dept_emp* que armazena o relacionamento do cargo funcionário entre departamentos e empregados, sendo um relacionamento de muitos para muitos; *dept_manager* que armazena o relacionamento do cargo gerente entre departamentos e empregados. A base de dados corresponde a um sistema de gerenciamento de funcionários de uma empresa, e foi escolhida pela grande quantidade de tuplas, pela facilidade de acesso a base além do fato de já estar modelado para um banco de dados relacional. Será necessário realizar alguns tratamentos para que esta mesma base de dados esteja

[3] https://dev.mysql.com/doc/employee/en

disponível tanto no banco de dados PostgreSQL[4] quanto no MongoDB. Estes bancos foram escolhidos por estarem entre os mais utilizados atualmente[5], além de, no caso do PostgreSQL, disponibilizar mais funcionalidades e recursos que o MySQL, que podem ser utilizadas futuramente dentro deste trabalho.

5.3 Modelagem da Base de Dados Exemplo Considerando um Banco de Dados Não Relacional

Para a modelagem no MongoDB, foram criadas quatro coleções. A coleção denominada *employees* tem a finalidade de armazenar funcionários, seus salários e seus títulos. A coleção denominada *departments* tem a finalidade de armazenar os departamentos. As coleções *dept_emp* e *dept_manager* têm a finalidade de armazenar os relacionamentos entre a coleção de funcionários e de departamentos e representam o relacionamento de empregado e gerente, respectivamente. A Figura 4 corresponde ao diagrama de esquemas das coleções no banco de dados MongoDB. Foram adicionadas setas simbolizando chaves estrangeiras apenas para representação de relacionamentos. Estes relacionamentos não são implementados por meio de restrições no MongoDB. O esquema utilizado para a criação das coleções no MongoDB pode ser visualizado no Apêndice B.

Foi utilizada a modelagem disponibilizada por Arora e Aggarwal (2013) como base para modelagem no MongoDB. Como o próprio MongoDB disponibiliza índices para cada coleção por meio do atributo '_id', estes podem ser utilizados substituindo os atributos 'emp_no' e 'dept_no' utilizados para identificar tuplas das coleções 'employees' e 'departments', respectivamente. Esta substituição será realizada na etapa de 5.5 Tratamento Sobre a Base de Dados Exemplo.

5.4 Modelagem da Base de Dados Exemplo Considerando um Banco de Dados Relacional

Como a base de dados exemplo originalmente foi criada em MySQL, um sistema de bancos de dados relacional, não foi necessário realizar muitas alterações visto que todas as nomenclaturas relacionadas a restrições são idênticas às disponibilizadas pelo PostgreSQL. Foi necessário apenas criar o diagrama de entidade relacionamento para compreensão e visualização da modelagem de forma simples. A Figura 5 corresponde ao Diagrama Entidade Relacionamento da Base de Dados Exemplo em um Sistema de Banco de Dados Relacional.

[4] https://www.postgresql.org
[5] https://db-engines.com/en/ranking, Último Acesso em 13/08/2022

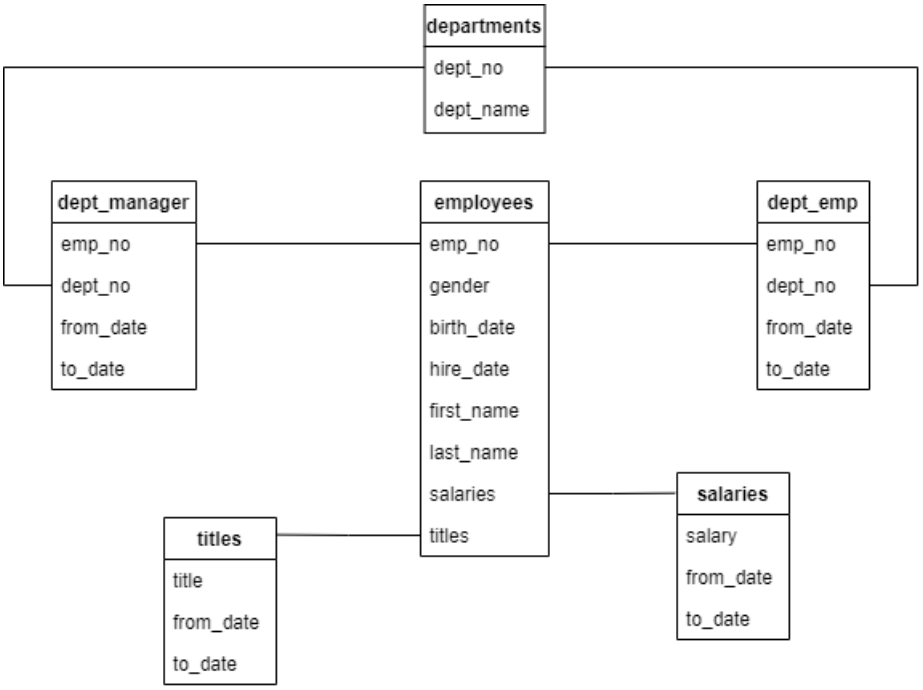

Figura 4 – Diagrama de Esquemas MongoDB.

Fonte: Autor, 2022.

5.5 Tratamento Sobre a Base de Dados Exemplo

5.5.1 PostgreSQL

A sintaxe de instruções DDL e DML são bastante semelhantes entre os bancos MySQL e PostgreSQL. Foi necessário apenas criar um enumerador para gêneros, no atributo *gender*, na tabela *employees*, e os índices para o atributo *dept_no*, tanto na tabela *dept_manager* quanto na tabela *dept_emp*. A criação do enumerador, no MySQL, é permitida na criação da tabela, enquanto no PostgreSQL o enumerador deve existir previamente. O enumerador foi utilizado para restringir o atributo *gender* apenas a valores 'M' e 'F'. A criação dos índices foi necessária já que por padrão, no MySQL, chaves estrangeiras são também índices[6]. O mesmo não acontece no PostgreSQL [7].

5.5.2 MongoDB

Para o tratamento das tuplas a serem inseridas no esquema do MongoDB, foi necessário realizar a substituição de alguns caracteres com o objetivo de transformar instruções DML em arquivos no formato JSON. Os arquivos em formato JSON podem ser restaurados em um banco de dados MongoDB desde que o tipo de dado de cada campo

[6] Chaves estrangeiras são indexadas no MySQL: https://dev.mysql.com/doc/refman/8.0/en/constraint-foreign-key.html Último acesso em 14/08/2022

[7] Chaves estrangeiras não são indexadas no PostgreSQL: https://www.postgresql.org/docs/current/indexes-unique.html Último acesso em 14/08/2022

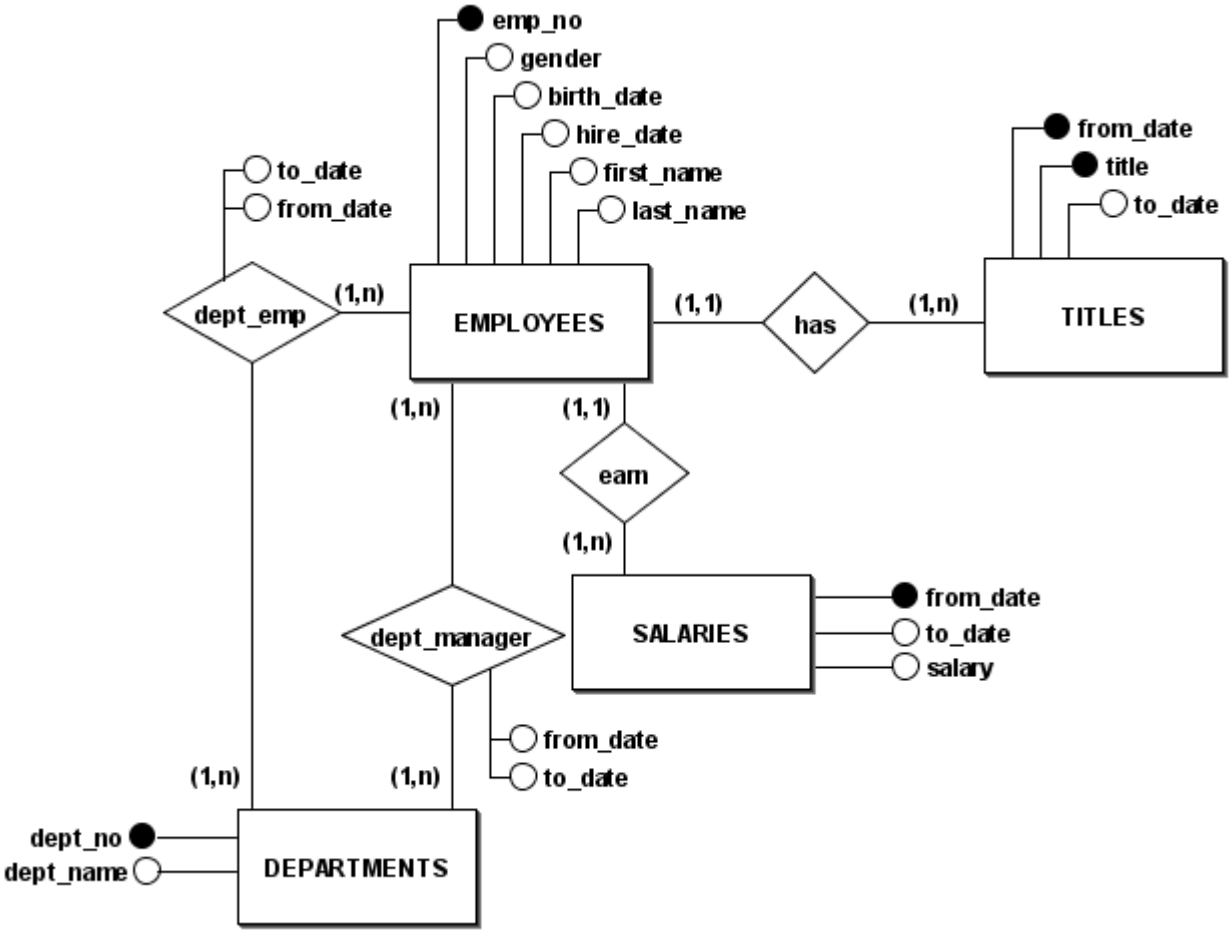

Figura 5 – Diagrama Entidade Relacionamento

Fonte: Autor, 2022.

no JSON corresponda ao esquema no MongoDB.

A substituição de caracteres foi feita em JavaScript com o código disponível no Apêndice C. Para tratamento de tuplas da tabela *departments*, por exemplo, foram passados como parâmetros o SQL, disponível no Apêndice D, e o atributo *fields* igual a ['dept_no', 'dept_name'] para a função *parseTuples*, definida no Apêndice C. O resultado obtido, neste exemplo, pode ser visualizado no Apêndice E.

Outro tratamento necessário foi a adição das tuplas das tabelas *titles* e *salaries* nos campos respectivos no esquema MongoDB. Esta ação foi necessária já que relacionamentos do tipo um para muitos são representados por meio de atributos do tipo lista. Após a transformação dos arquivos SQL em JSON, foi necessário encontrar as tuplas das tabelas *titles* e *salaries* por meio do atributo *emp_no*. As listas de tuplas resultantes foram adicionadas à coleção *employees*.

Como esta estapa de transformação de instruções DML em documentos JSON tomam algumas horas, os arquivos resultantes foram salvos em um arquivo .zip que é extraído na etapa de Carregamento das bases de dados.

5.6 Carregamento das bases de dados

A primeira tarefa a ser realizada no processo de carregamento das bases de dados, é o processo de descompactação dos *backups* das bases de dados, que é disponibilizado no arquivo dump.zip, no repositório [8]. Após a realização da descompactação, é necessário verificar se os *backups* já estão carregados, e evitar assim, que ocorra um erro ao tentar recuperá-los novamente. Para isso, são realizadas consultas aos metadados, tanto do MongoDB quanto do PostgreSQL com o objetivo de verificar se as bases de dados já existem, e, no caso do MongoDB, se esta possui tamanho suficiente.

No MongoDB, o processo de recuperação da base de dados é realizado por meio do comando "mongoimport", disponibilizado no pacote "mongodb-org". O comando completo pode ser visualizado a seguir: "mongoimport –db employees –collection *collection* – –*filedump*/collection.json –jsonArray –username mongo –password mongo –uri mongodb://mongo:mongo@mongo:27017 –authenticationDatabase admin", onde collection corresponde ao nome de cada um dos arquivos JSON disponibilizados no arquivo dump.zip, extraído previamente.

Já no PostgreSQL, é utilizada a própria *ORM*, o *Sequelize*, para a execução de comandos DML em SQL que restaurem os arquivos de backup. Sendo assim, é necessário apenas realizar a configuração inicial da *ORM* e o comando "query"estará disponível.

Todo o processo é automatizado por meio do Docker, que ao iniciar, verifica se as bases já estão carregadas. Se as bases estiverem carregadas, o servidor é iniciado, se não, é realizado o processo acima descrito, e então o servidor é iniciado.

5.7 Operações Comuns em Aplicações Web Modernas

Para a realização das operações em ambas as bases de dados, serão utilizadas as técnicas disponibilizadas por Truica, Boicea e Trifan (2013), que trabalha sobre os bancos MongoDB e MySQL. No entanto, as operações que serão trabalhadas neste trabalho têm o objetivo de serem ainda mais específicas para sistemas web modernos. O autor Leonard (2013), conforme já mencionado no 5.1 Contexto Temático, sugere funcionalidades relevantes em uma aplicação de *e-Commerce*, considerada moderna. O autor Banker (2011) especifica funcionalidades de um banco de dados para uma aplicação web moderna, além de disponibilizar blocos de código para implementação das consultas em MongoDB. São especificadas funcionalidades como: listagem de itens de forma paginada, busca por itens de forma agregada. Com base nas sugestões apontadas pelos autores, e a base de dados exemplo, foram selecionadas as seguintes funcionalidades a serem implementadas:

[8] https://github.com/pedroeagle/database-comparative

- Seleção de funcionários por departamento: Consiste em listar a quantidade de funcionários que pertençam a cada departamento;
- Seleção de funcionários ordenados por data de contratação: Consiste em listar os últimos funcionários contratados;
- Seleção paginada de funcionários: Consiste em listar uma quantidade fixa de funcionários e selecionar os próximos itens conforme solicitado;
- Consulta de de títulos e salários de cada funcionário: Consiste em listar os títulos e os salários de cada funcionário, e
- Consulta de funcionários contratados dentro de um intervalo de tempo: Consiste em listar todos os funcionários que possuem data de contratação dentro de um período de tempo.

O objetivo das funcionalidades é realizar as seguinte operações, respectivamente:

- Seleção por categoria;
- Seleção ordenada por atributo;
- Seleção paginada;
- Seleção por atributo que aplique a determinada regra;
- Seleção de itens pertencentes a um item pai, e
- Seleção de itens, aplicáveis a uma regra, pertencentes a um item pai.

Além disso, foram implementadas funcionalidades sugeridas pelos examinadores, com o objetivo de enriquecer as comparações:

- Contagem de itens;
- Inserção de um novo item, e
- Atualização de um item existente.

Essas funcionalidades serão implementadas e apresentadas em um *dashboard* por meio de gráficos e tabelas.

5.8 Descrição do *dashboard*

Para a implementação de um dashboard que corresponda às funcionalidades mais comuns em aplicações web modernas, foi realizada a escolha de um modelo pré-definido. Para a escolha deste modelo, foi consultado o *ranking* de repositórios do GitHub disponibilizado pelo site Gitstar Ranking[9]. Foi escolhida a biblioteca Material UI[10] por ser a mais bem posicionada no âmbito de bibliotecas de componentes. Para a escolha do modelo, foi realizada uma busca dos modelos mais populares e escolhido o modelo gratuito mais bem posicionado, o modelo Devias Kit - React Admin Dashboard[11], que pode ser visualizado nas figuras Figura 6 e Figura 7. A partir da escolha do modelo de *dashboard*, foi possível implementar as consultas mais comuns, das aplicações web, às bases de dados.

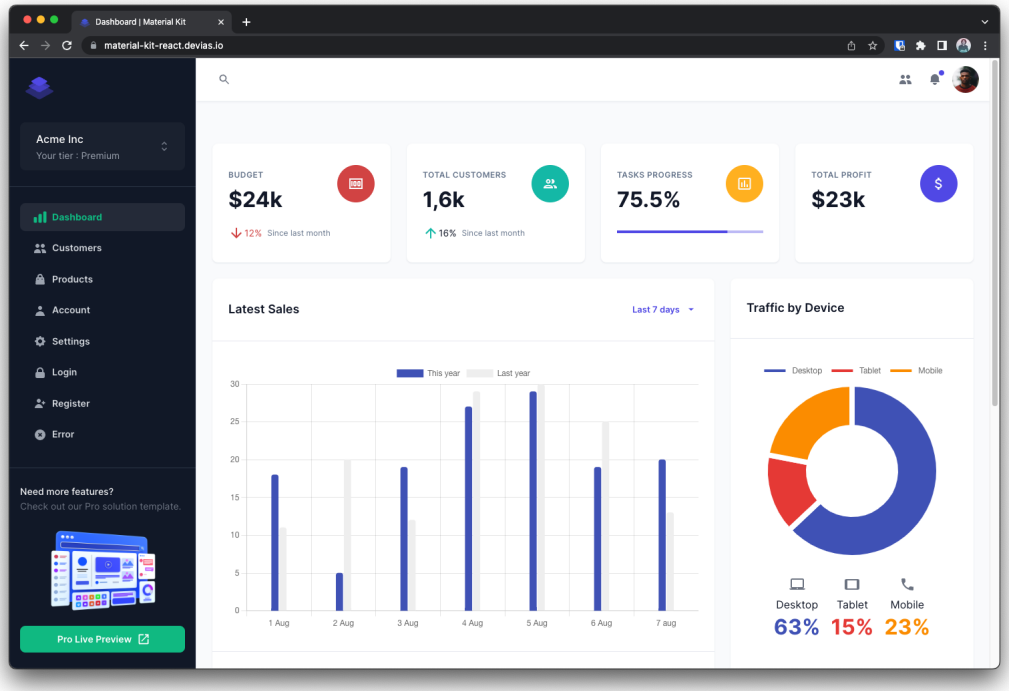

Figura 6 – Template Devias Kit - React Admin Dashboard.

Fonte: Autor, 2022.

5.9 Métodos para coleta e análise de resultados

A métrica utilizada para análise de desempenho das soluções, com banco de dados relacional e não relacional, é o tempo de resposta. O tempo de resposta corresponde ao intervalo de tempo entre o início da solicitação de um resultado até o retorno deste resultado.

[9] https://gitstar-ranking.com/repositories. Último acesso em 15/11/2022
[10] https://mui.com/pt/. Último acesso em 15/11/2022
[11] https://mui.com/store/items/devias-kit/. Último acesso em 15/11/2022

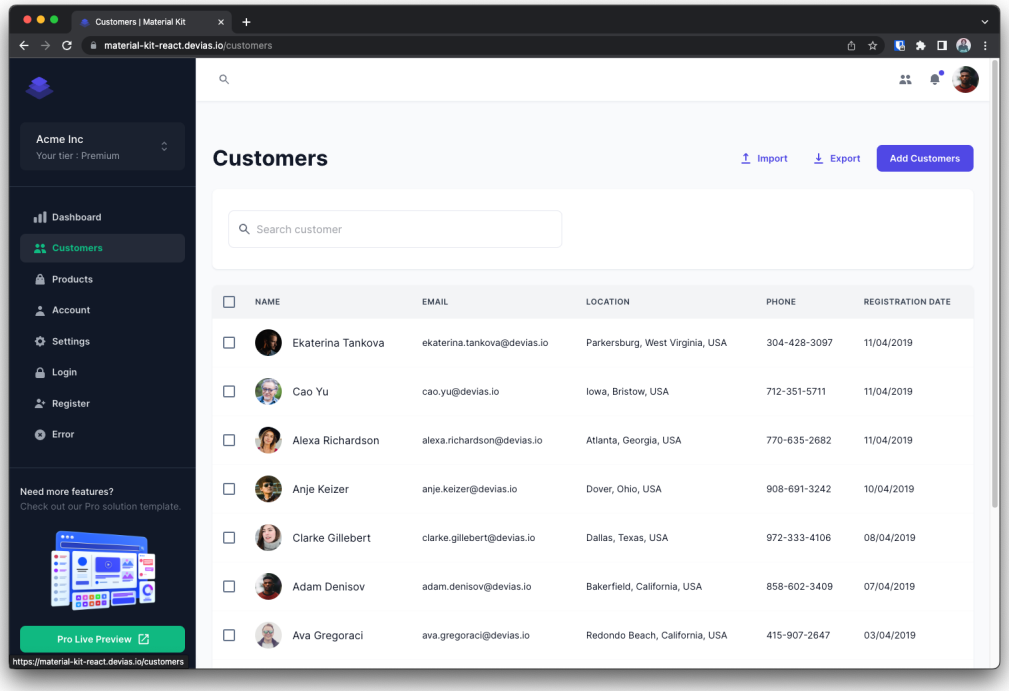

Figura 7 – Template Devias Kit - React Admin Dashboard, Customers page.

Fonte: Autor, 2022.

Para realizar a coleta dos tempos inicial e final, serão utilizados *middlewares* que têm o objetivo de coletar os momentos de recepção de uma requisição *http* e da finalização desta requisição. Além disso, os *middlewares* terão o objetivo de identificar qual banco será utilizado na requisição e qual consulta será realizada. Essas informações serão armazenadas em um arquivo *CSV* e serão utilizadas na comparação de tempo para cada solução.

Para análise dos dados, serão utilizados gráficos que comparam o tempo de cada consulta em cada tipo de banco de dados conforme exemplificado por Gyorödi, Gyorödi e Sotoc (2015) na Figura 3.

5.10 *Database Comparative*

A aplicação utiliza como base as duas primeiras telas do *dashboard* apresentado na subseção Descrição do *dashboard*. Foram removidos alguns componentes que não foram utilizados com o objetivo de manter o *dashboard* organizado para as funcionalidades implementadas. Para cada informação, consultada no banco de dados, e, apresentada na interface, foi apresentado um gráfico que compara o tempo de resposta, em milisegundos, em 3 implementações diferentes:

- Postgres, em sua implmentação padrão;

- MongoDB, em sua implementação padrão, e
- MongoDB, com a utilização de índices, conforme descrito no seção 6.1.1 Utilização de índices para a melhoria de desempenho no MongoDB.

No gráfico, a cor azul representa o tempo de resposta para a implementação utilizando o banco de dados Postgres. A cor amarela representa o tempo de resposta para a implementação utilizando o MongoDB sem índices. E, a cor verde, a implementaçãso utilizando o MongoDB com índices.

5.10.1 Telas da aplicação

Na Figura 8 é possível visualizar as *features* de contagem de elementos; seleção por atributo que aplique a determinada regra, e seleção por categoria.

Na Figura 9 é possível visualizar as *features* de seleção ordenada por atributo; seleção por atributo que aplique a determinada regra; seleção de itens pertencentes a um item pai, e seleção de itens, aplicáveis a uma regra, pertencentes a um item pai.

Na Figura 10 é possível visualizar as *features* de seleção paginada; seleção por atributo que aplique a determinada regra, e inserção de um novo item.

Na Figura 11 é possível visualizar a *feature* de edição de item.

Além das funcionalidades previamente descritas, foi adicionado um botão de *download* a barra superior direita da página. Ao clicar no botão, é realizado o *download* do arquivo de relatório de utilização da sessão. É possível visualizar informações como:

- *timestamp* inicial e final da requisição;
- tempo de resposta;
- rota consultada;
- banco de dados utilizada na consulta, e
- parâmetros passados por meio de *query string*.

É possível visualizar a descrição da funcionalidade implementada em cada rota no Quadro 5 da seção 6.1 Coleta de Resultados.

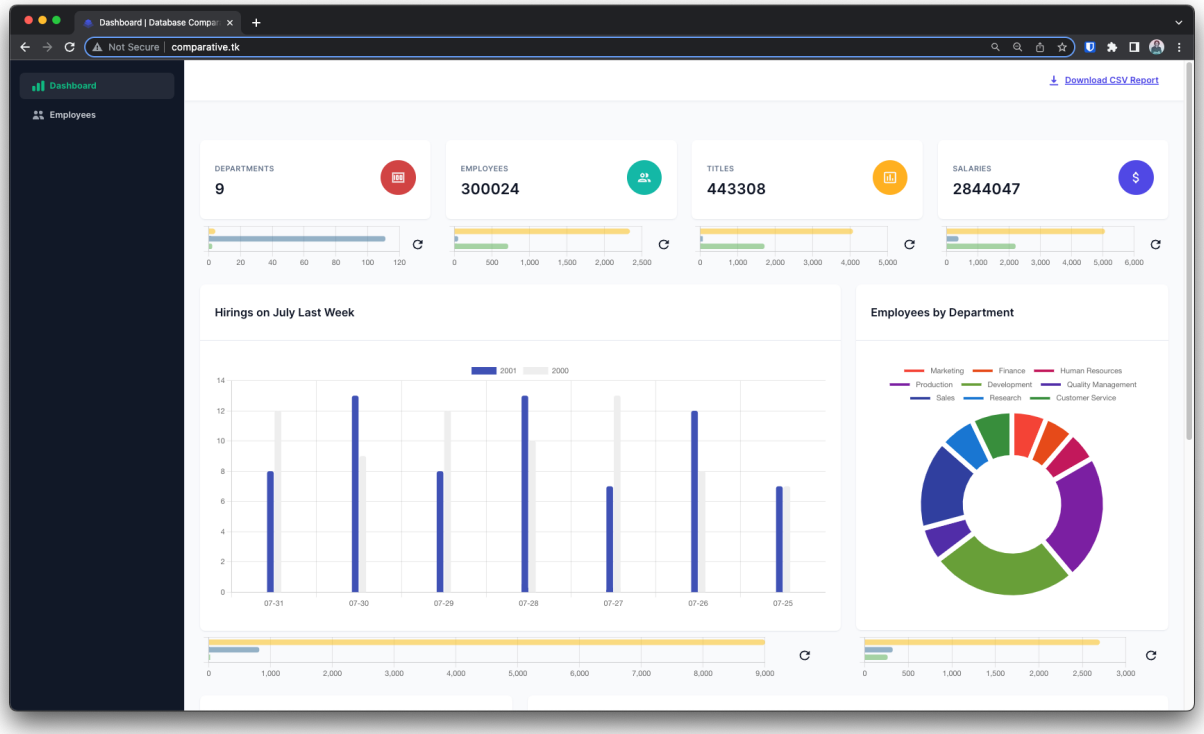

Figura 8 – Primeira parte do Dashboard

Fonte: Autor, 2023.

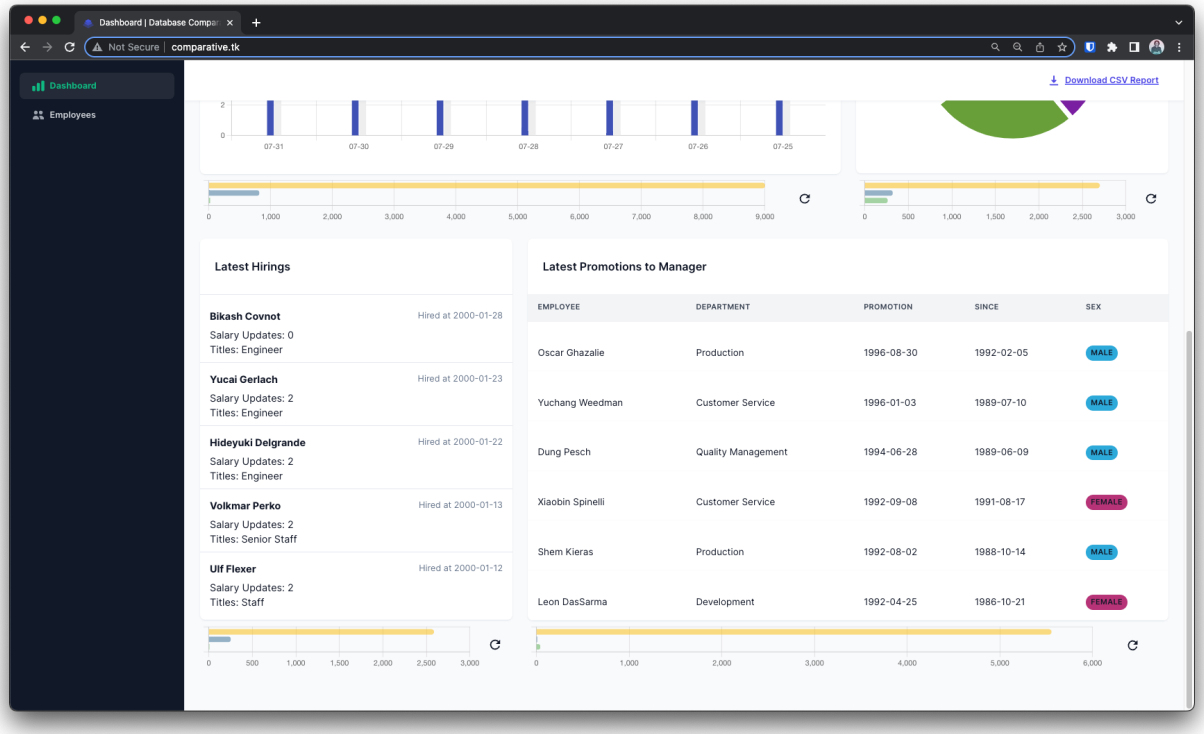

Figura 9 – Segunda parte do Dashboard

Fonte: Autor, 2023.

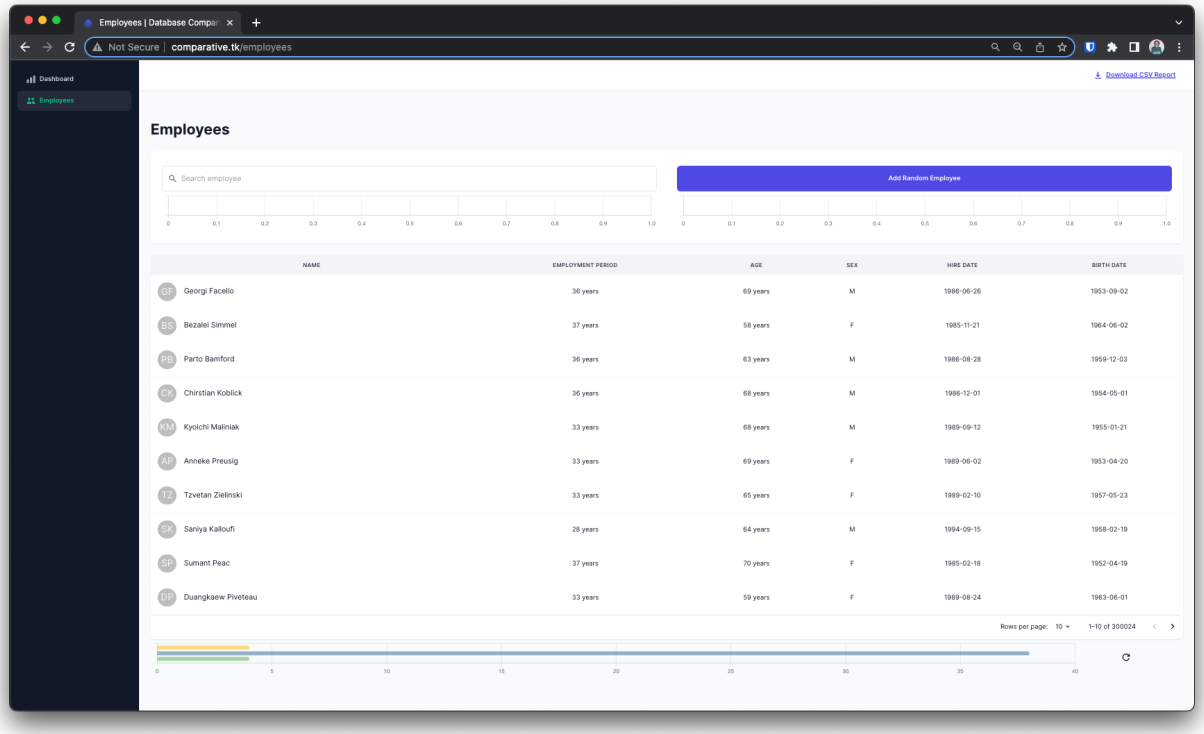

Figura 10 – Tela de listagem de funcionários

Fonte: Autor, 2023.

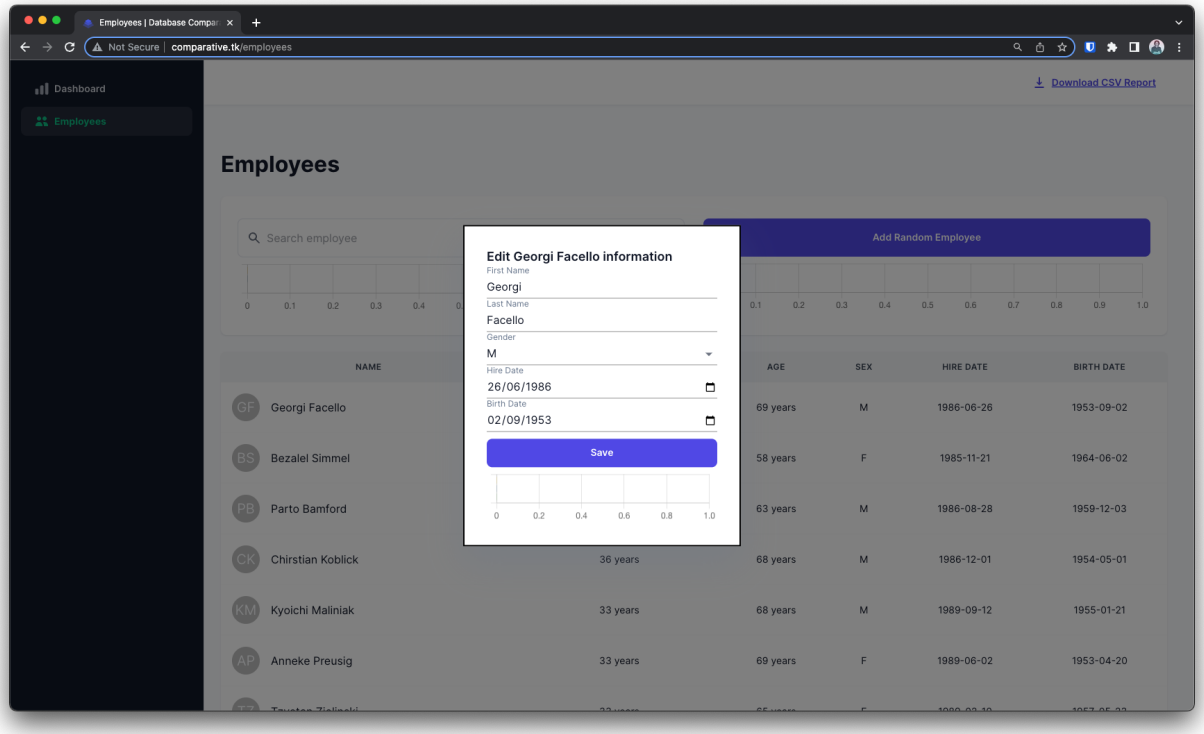

Figura 11 – Tela de edição de funcionário

Fonte: Autor, 2023.

5.11 Considerações Finais

Este capítulo apresentou o contexto temático, dentro da história da computação, até o momento onde é definida a problemática. Foram apresentados trabalhos com problemáticas semelhantes com o objetivo de enriquecer o conhecimento sobre o assunto. A partir dos trabalhos apresentados, foi possível identificar que o MongoDB possui vantagens de desempenho em operações de inserção, remoção, e atualização quando comparado com o MySQL, que possui vantagens em operações de seleção. Por último, foi apresentado produto de software por meio de algumas subseções. O objetivo do produto de software resultante deste trabalho é fornecer insumos que auxiliem na escolha do tipo de banco de dados mais adequado para determinada aplicação.

Para desenvolvimento do produto, foi definido o contexto de uma aplicação que utilize bancos de dados de ambos os tipos: relacional e não relacional. Para uma comparação justa, foi escolhida uma base de dados pública e, então, realizada a modelagem de dados considerando ambos os tipos de bancos de dados. A base de dados exemplo escolhida foi a disponibilizada pelo MySQL[12]. Em seguida, foram especificados os tratamentos realizados sobre a base de dados exemplo, com o intuito de utilizá-la nos dois bancos de dados escolhidos: PostgreSQL e MongoDB. E, então, descrito o processo de carregamento das bases de dados, utilizando o Docker.

Em seguida, foram escolhidas, e expostas, operações comuns em aplicações web modernas a serem implementadas em cada tipo de banco de dados: seleção por categoria, seleção ordendada por atributo; seleção paginada; seleção por atributo que aplique a determinada regra; seleção de itens pertencentes a um item pai, e seleção de itens, aplicáveis a uma regra, pertencentes a um item pai. Foi apresentada a definição de um modelo de *dashboard* que corresponda a uma aplicação web moderna, o modelo Devias Kit - React Admin Dashboard[13]. Foram apresentadas técnicas utilizadas para coleta e análise de resultados da aplicação. O tempo de execução de cada operação será medida a partir de *middlewares*, e armazenados em um arquivo CSV que será analisado por meio de gráficos. Por último, foi apresentada a aplicação final desenvolvida, assim como capturas de tela, úteis para visualização de cada funcionalidade.

[12] https://dev.mysql.com/doc/employee/en. Último acesso em 15/11/2022
[13] https://mui.com/store/items/devias-kit/. Último acesso em 15/11/2022

6 Coleta e Análise de Resultados

Este capítulo apresenta a análise dos resultados obtidos ao longo da coleta de dados de utilização do produto de software, resultante deste Trabalho de Conclusão de Curso. Os resultados foram coletados de forma automática, dentro do próprio produto de sofware, por meio de *middlewares*, conforme descrito na seção 5.9 Métodos para coleta e análise de resultados. Será descrito o *script* em Python utilizado para a simulação de utilização do site e obtenção de dados. Em seguida, serão apresentados os resultados, por meio de gráficos, assim como suas respectivas interpretações. Por fim, apresenta-se um breve resumo sobre o capítulo.

6.1 Coleta de Resultados

Para simular a utilização de múltiplos usuários simulâneos, foi escrito um *script* em Python. O *script* realiza a solicitação de todas as rotas da aplicação em ordem aleatória e com intervalo aleatório entre requisições. Para a simulação de vários usuários simultâneos, foi utilizada a programação paralela por meio de *threads*.

Threads são fluxos de controle que permitem que várias ações sejam realizadas simultaneamente, isto é, de forma paralela, pelo processador. Para a coleta dos resultados, foram criadas 100 *threads*. Cada *thread* realiza ao menos uma requisição a cada uma das 12 rotas disponíveis no produto de software. Somente a rota responsável pela funcionalidade de paginação recebe mais de uma requisição por thread, totalizando 6 requisições com tamanhos de páginas diferentes. Para esta rota, são consultadas páginas de 10, 25, 50, 100, 1000 e 10000 itens.

Para simular um usuário real, cada thread realiza as consultas em uma ordem aleatória e aguarda um intervalo aleatório, entre 5 e 10 segundos, para a realização da próxima requisição. Após a finalização da rotina realizada por cada uma das 100 threads, é realizado o *download* do arquivo de relatório gerado pelo produto. O scripts em Python para a coleta de resultados estão disponíveis nos apêndices F e G.

6.1.1 Utilização de índices para a melhoria de desempenho no MongoDB

No MongoDB, assim como no Postgres, existe a funcionalidade de índice. Índice é uma estrutura criada sobre um ou mais atributos de uma coleção com o objetivo de melhorar o desempenho. Um índice pode ser classificado quanto ao tipo:

- ASC: Ascendente;

- DESC: Descendente, e
- 2dsphere: Útil para cálculos geométricos de esferas semelhantes a terra,

e pode ser único ou não. Além disso, um índice formado por mais de um atributo é chamado de índice composto.

Como a diferença da primeira implementação, em MongoDB, se mostrou demasiadamente inferior, em relação a desempenho, quando comparado com o Postgres, foi necessária a utilzação de índices. Com o objetivo de melhorar o resultados, foram criados alguns índices nas coleções do MongoDB. Outros índices foram criados, no entanto não tiveram efeito no desempenho da aplicação. Os índices criados são visualizáveis no Quadro 3

É possível visualizar a comparação de tempo de resposta para o MongoDB, em sua implementação mais simples; em sua implementação com índices, e para o Postgres no Quadro 4. Para melhor entendimento dos dados, pode ser o utilizado o Quadro 5 que corresponde a rota da aplicação com a funcionalidade implementada.

6.2 Análise de Resultados

Para a análise dos resultados foi realizada a plotagem de gráficos de barra que comparam o tempo de resposta, de cada funcionalidade, em cada banco de dados. O tempo é analisado em milisegundos e os resultados são comparados a partir da média de tempo de resposta aferida nas 100 consultas realizadas. Para a análise da funcionalidade de paginação, foram utilizados gráficos de dispersão com o objetivo de relacionar o tempo de resposta com o número da página solicitada.

6.2.1 Contagem de departamentos

É possível visualizar a comparação da média de resultados na Figura 12. Nesta consulta, até mesmo sem a utilização de índices, o MongoDB se mostrou mais rápido que o Postgres. A consulta realiza a contagem de itens de uma tabela/esquema. Isso significa que, para a contagem de poucos itens, o MongoDB pode ser mais indicado. Esta é a consulta com menor tempo médio em ambos os bancos de dados.

6.2.2 Contagem de salários

É possível visualizar a comparação da média de resultados na Figura 13. Nesta consulta, o Postgres se mostrou 10 vezes mais eficiente que a primeira implementação em MongoDB, e cerca de 5 vezes mais eficiente que o MongoDB implementado com índices. Isso se deve ao fato de que, no Postgres, a lista de salários é armazenada em uma tabela

Quadro 3 – Índices criados para a base de dados exemplo no MongoDB

Coleção	Atributos	Tipo	Único
departments	dept_no	ASC	Sim
dept_emp	dept_no	ASC	Não
dept_emp	emp_no	ASC	Não
dept_emp	from_date	ASC	Não
dept_manager	from_date	ASC	Não
employees	emp_no	ASC	Sim
employees	hire_date	ASC	Não

Fonte: Autor, 2023.

Quadro 4 – Comparação de desempenho de cada rota para o MongoDB, MongoDB com Índices e o Postgres

Operação	MongoDB (ms)	Postgres (ms)	MongoDB com Índices (ms)	Ganho (%)
/departments/count	5,65	10,16	2,49	126,90
/employees/all	496,23	129,77	470,40	5,49
/employees/by/department	850,06	110,18	48,32	1659,24
/employees/by/year	5159,84	737,99	33,62	15247,53
/employees/count	373,02	163,57	158,94	134,69
/employees/new	9,37	31,61	5,08	84,19
/employees/search	607,04	110,72	264,57	129,44
/employees/update	239,31	14,91	5,04	4648,21
/last/hirings	473,25	58,67	4,22	11114,45
/last/promotions	2990,10	9,025	54,67	5369,37
/salaries/count	1498,44	145,81	685,19	118,68
/titles/count	1072,69	47,34	447,71	139,59

Fonte: Autor, 2023.

Quadro 5 – Rota de cada funcionalidade da aplicação

Rota	Descrição da Funcionalidade
/departments/count	Contagem de departamentos
/employees/all	Listagem paginada de funcionários
/employees/by/department	Contagem de funcionários por departamento
/employees/by/year	Contagem de contratações em um mesmo período do ano em dois anos seguidos
/employees/count	Contagem de funcionários
/employees/new	Inserção de um novo funcionário
/employees/search	Busca por funcionário pelos campos nome e sobrenome
/employees/update	Atualização de um funcionário
/last/hirings	Consulta das 5 últimas contratações
/last/promotions	Consulta das 6 últimas promoções
/salaries/count	Contagem de salários
/titles/count	Contagem de títulos

Fonte: Autor, 2023.

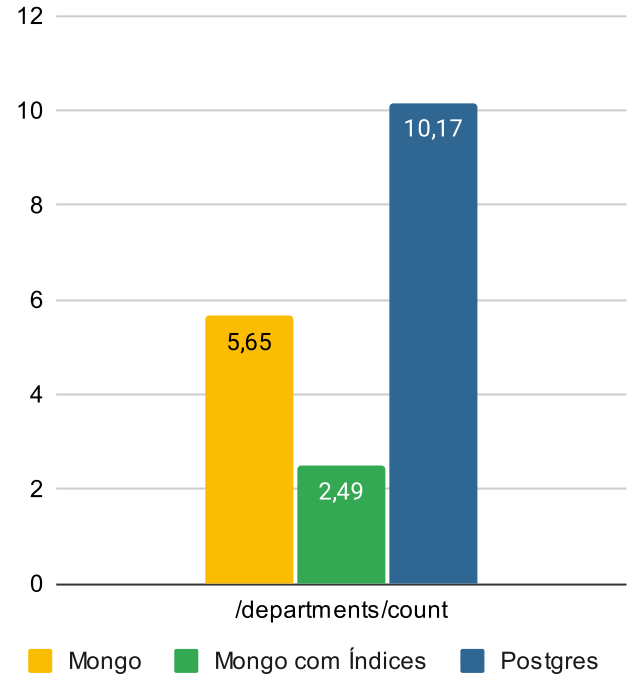

Figura 12 – Contagem de departamentos

Fonte: Autor, 2023.

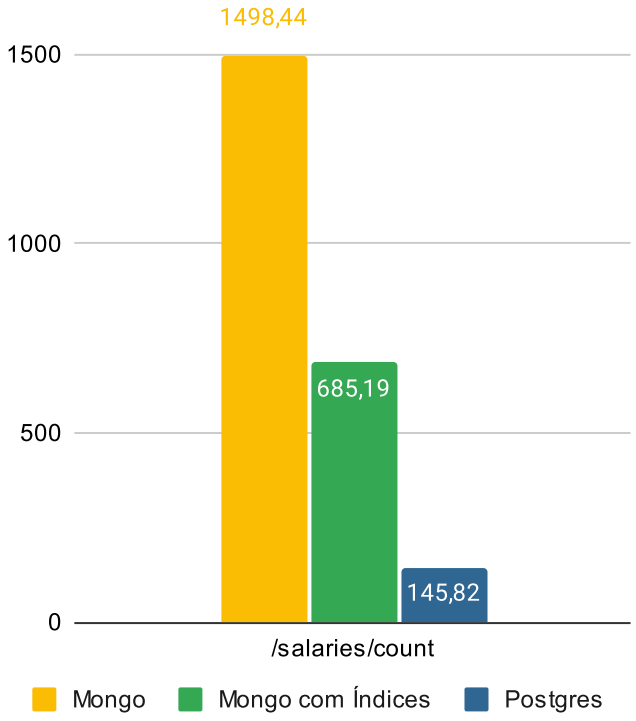

Figura 13 – Contagem de salários

Fonte: Autor, 2023.

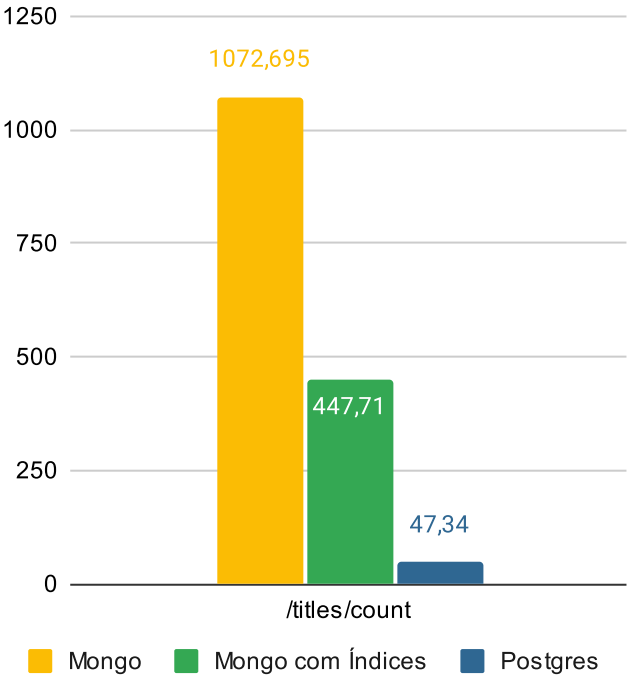

Figura 14 – Contagem de títulos

Fonte: Autor, 2023.

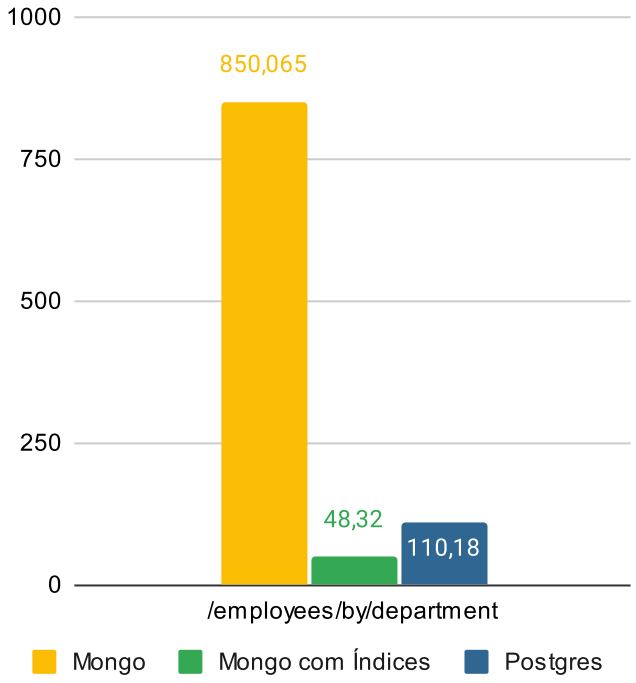

Figura 15 – Contagem de funcionários por departamento

Fonte: Autor, 2023.

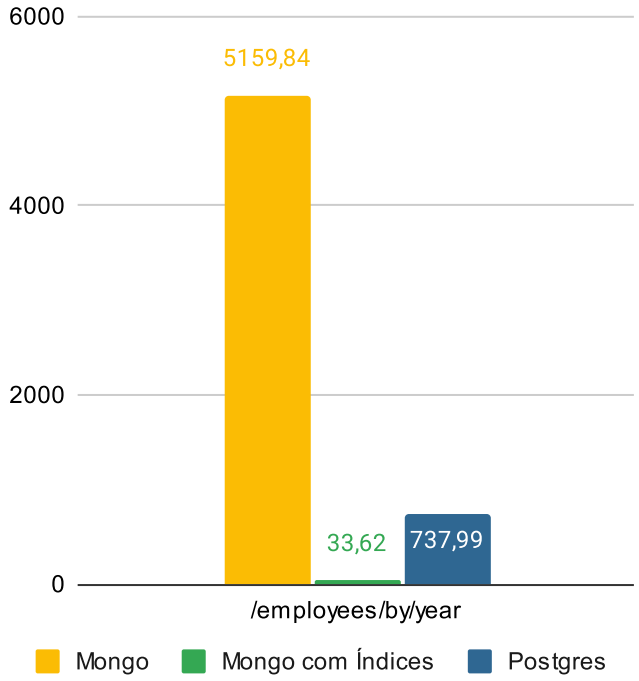

Figura 16 – Contagem de contratações em um mesmo período do ano em dois anos seguidos

Fonte: Autor, 2023.

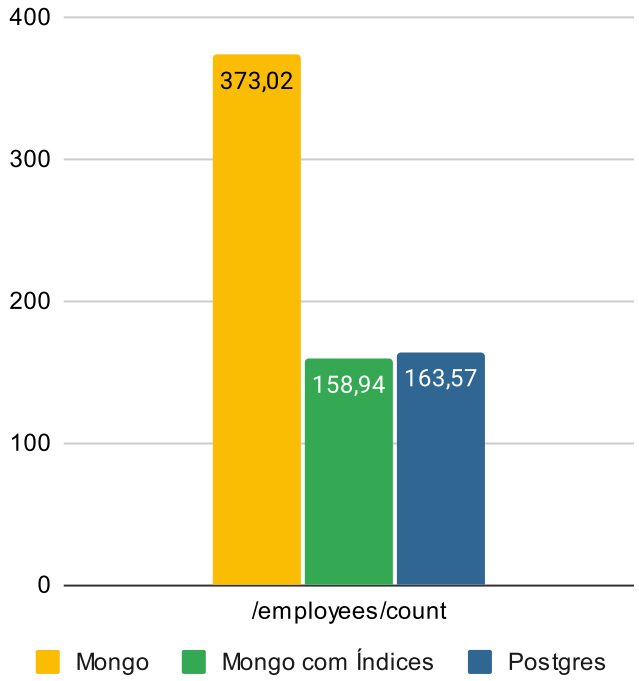

Figura 17 – Contagem de funcionários

Fonte: Autor, 2023.

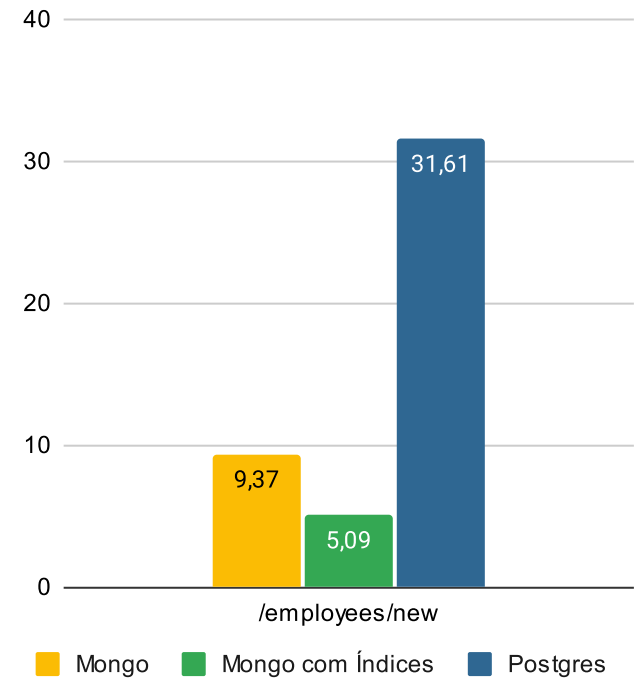

Figura 18 – Inserção de um novo funcionário

Fonte: Autor, 2023.

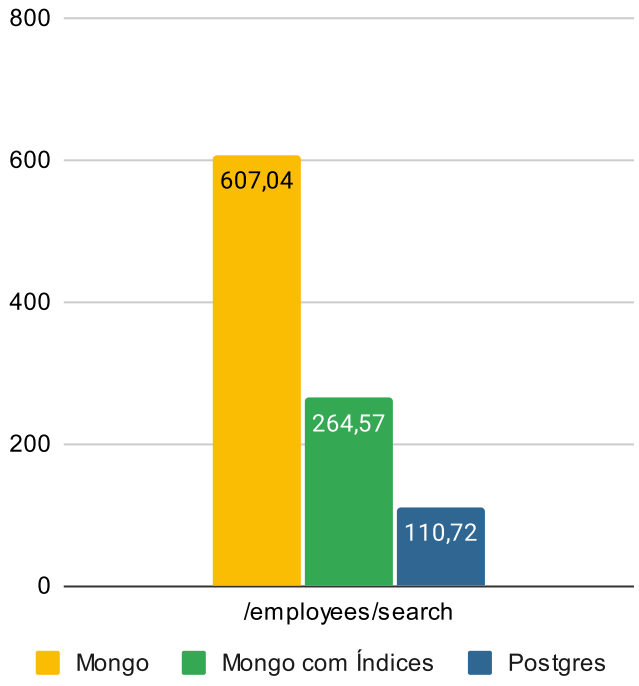

Figura 19 – Busca por funcionário pelos campos nome e sobrenome

Fonte: Autor, 2023.

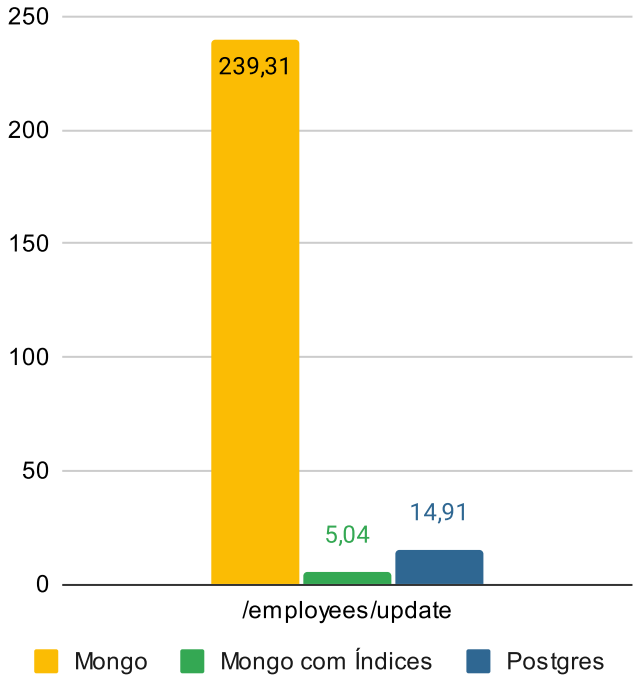

Figura 20 – Atualização de um funcionário

Fonte: Autor, 2023.

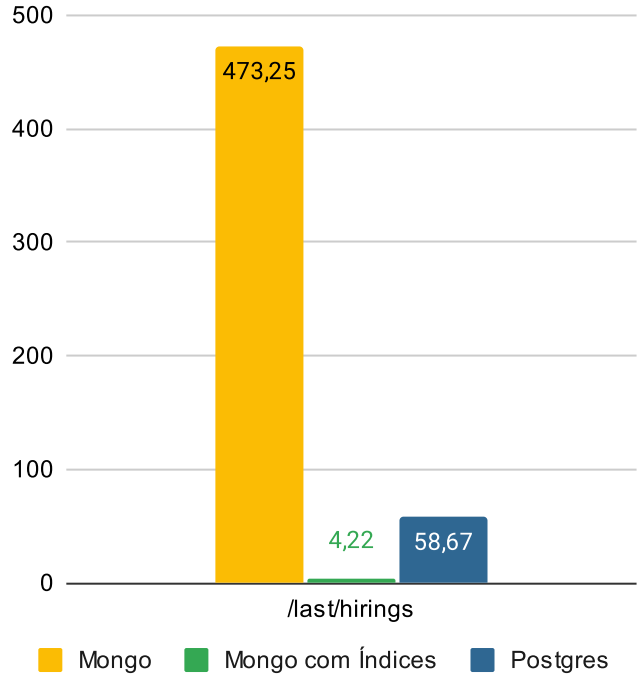

Figura 21 – Consulta das 5 últimas contratações

Fonte: Autor, 2023.

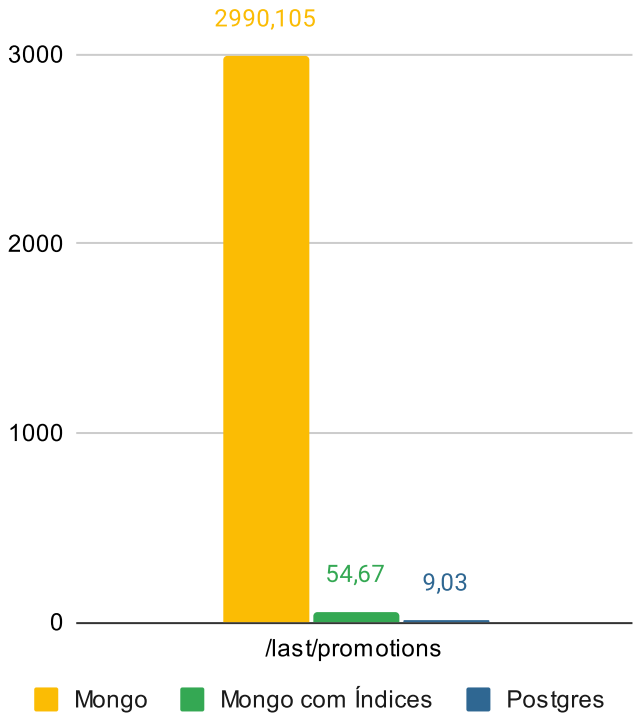

Figura 22 – Consulta das 6 últimas promoções

Fonte: Autor, 2023.

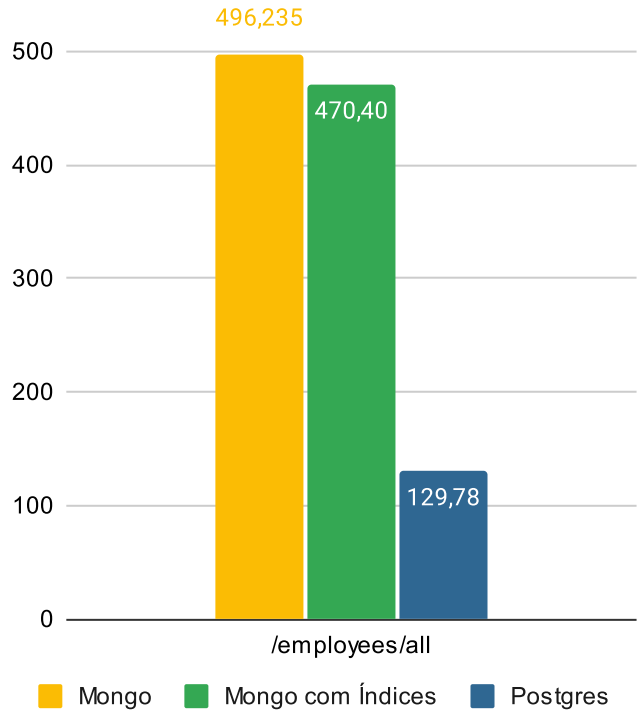

Figura 23 – Listagem paginada de funcionários

Fonte: Autor, 2023.

exclusiva. É armazenado o salário e o identificador do usuário. Enquanto no MongoDB, a lista de salários é armazenada como uma lista dentro do objeto de funcionário. Logo, no Postgres é necessário apenas realizar uma operação de contagem de itens, enquanto no MongoDB é necessário percorrer a lista de funcionários, e então realizar a contagem de itens de salário dentro de cada funcionário. No Mongo, essa operação é realizada por meio de uma agregação. Além disso, é interessante observar que a implementação com índices do MongoDB conseguiu reduzir pela metade o tempo de resposta quando comparada com a implementação sem índices.

6.2.3 Contagem de títulos

É possível visualizar a comparação da média de resultados na Figura 14. O motivo do MongoDB se apresentar tão lento nesta operação é idêntico ao da Contagem de salários. É necessário realizar a contagem da lista de títulos dentro da lista de funcionários. Similarmente a contagem de salários, o tempo de resposta para a implementação com índices reduziu para menos da metade da implementação sem índices.

6.2.4 Contagem de funcionários por departamento

É possível visualizar a comparação da média de resultados na Figura 15. O MongoDB, sem índices, se mostrou quase 8x mais lento que o Postgres para esta consulta. Nesta operação é consultada a lista de departamentos cadastrados e então a quantidade de funcionários pertencentes a cada departamento. Com a utilização de índices, o MongoDB conseguiu reduzir drasticamente o tempo de resposta, ganhando inclusive do tempo de resposta do Postgres.

6.2.5 Contagem de contratações em um mesmo período do ano em dois anos seguidos

É possível visualizar a comparação da média de resultados na Figura 16. A implementação com índices, do MongoDB, novamente surpreendeu com uma média cerca de 22 vezes mais rápida que o Postgres e cerca de 153 vezes mais rápida que a implementação sem índices.

6.2.6 Contagem de funcionários

É possível visualizar a comparação da média de resultados na Figura 17. Esta consulta é idêntica a apresentada na subseção 6.2.1 Contagem de departamentos, porém, para a tabela/esquema de funcionários. Este resultado mostra que, conforme a quantidade de itens em uma mesma coleção aumenta, o desempenho do MongoDB com índices se aproxima do desempenho do Postgres, para a contagem de itens.

6.2.7 Inserção de um novo funcionário

É possível visualizar a comparação da média de resultados na Figura 18. Conforme apresentado na Figura 3, e previsto pelos Autores Gyorödi, Gyorödi e Sotoc (2015), as operações de inserção tomam mais tempo para finalizar em um banco de dados relacional. O tempo de inserção no Postgres se apresentou mais lenta até mesmo da implementação do MongoDB sem índices. Essa diferença se tornou ainda maior, quando comparada com a implementação utilizando índices.

6.2.8 Busca por funcionário pelos campos nome e sobrenome

É possível visualizar a comparação da média de resultados na Figura 19. A pesquisa de funcionário por nome e sobrenome se mostrou mais rápida no Postgres. Foi feita uma tentativa de criação de índice sobre os atributos de nome, da coleção *employees* no MongoDB, no entanto não surtiram efeitos sobre esta consulta.

6.2.9 Atualização de um funcionário

É possível visualizar a comparação da média de resultados na Figura 20. Ao contrário do apresentado na Figura 3, e previsto pelos Autores Gyorödi, Gyorödi e Sotoc (2015), a operação de atualização se apresentou mais de 15 vezes mais lenta no MongoDB sem utilização de índices. Um dos possíveis motivos, pode ser pelo fato de que a operação de busca pelo registro, para então atualizá-lo, toma bastante tempo no MongoDB. A utilização de índices novamente se mostrou bastante eficiente, obtendo um resultado quase 3 vezes mais eficiente que a implementação em Postgres, e cerca de 48 vezes, quando comparada a implementação sem índices.

6.2.10 Consulta das 5 últimas contratações

É possível visualizar a comparação da média de resultados na Figura 21. O MongoDB com índices se mostrou bem mais eficiente que a implementação em Postgres. Novamente, a implementação sem índices se mostrou bastante ineficiente.

6.2.11 Consulta das 6 últimas promoções

É possível visualizar a comparação da média de resultados na Figura 22. A diferença de tempo de resposta entre o Postgres e o Mongo se mostrou bem alta nesta consulta, cerca de 30 vezes mais lenta no MongoDB sem índices, e cerca de 6 vezes mais lenta no MongoDB com índices. Essa diferença pode ter se tornado alta, nesta consulta, visto que são feitas 3 buscas pelo identificador dos registros. Uma para a procura por funcionário, outra para a procura por departamento e então na coleção *'dept_emp'*.

6.2.12 Listagem paginada de funcionários

É possível visualizar a comparação da média de resultados na Figura 23. O valor apresentado corresponde a média de 100 usuários realizando consultas paginadas com páginas de tamanho 10, 25, 50, 100, 1000 e 10000. Logo, a média de resultados para tamanhos tão distintos não é a melhor abordagem para esta análise. A análise desta funcionalidade é melhor aprofundada na subseção Gráficos de dispersão para paginação. No entanto, é possível observar que, em média, paginações são quase 4 vezes mais rápidas no Postgres e a diferença na utilização de índices no MongoDB é quase imperceptível.

6.2.13 Gráficos de dispersão para paginação

Os gráficos de dispersão são úteis para a detecção de relação entre duas variáveis. Para as consultas paginadas, o objetivo é observar se quanto maior o número da página solicitada, maior o tempo de resposta. Além disso é possível comparar os gráficos entre si e verificar se o tempo médio de resposta aumenta proporcionalmente ao tamanho das páginas.

Como é possível observar nas Figuras 24 a 29, enquanto no MongoDB é apresentada uma certa proporcionalidade no tempo de resposta, conforme o número da página solicitada aumenta, no Postgres o tempo se aproxima da constância. A constância no tempo de resposta, para o Postgres, pode ser uma desvantagem quando solicitada uma das primeira páginas, no entanto, para as últimas páginas o tempo de resposta permanece baixo. Além disso, para páginas de tamanho 1000 e 10000, o MongoDB passa a apresentar uma constância maior no tempo resposta, embora na maioria das vezes seja mais lento que o Postgres. Para páginas de tamanho 1000, foi possível observar alguns pontos com tempo bem maior que a maioria, no Postgres. Foi utilizada a implementação do MongoDB com índices para esta análise.

6.3 Considerações Finais

Como foi possível observar, a implementação em MongoDB sem a utilização de índices só obteve melhor desempenho nas funciondalides de contagem de coleção com poucos itens e inserção de um novo item. No entanto, a implementação de índices no MongoDB promoveu ganhos bastante relevantes, obtendo desempenho melhor que o Postgres em alguns casos.

É possível visualizar um breve resumo de qual banco de dados obteve melhor desempenho em cada funcionalidade no Quadro 6. Além disso, é possível visualizar a proporção de desempenho do banco mais adequado sobre a outra alternativa.

Em resumo, o Postgres obteve vantagem desempenho em consultas como:

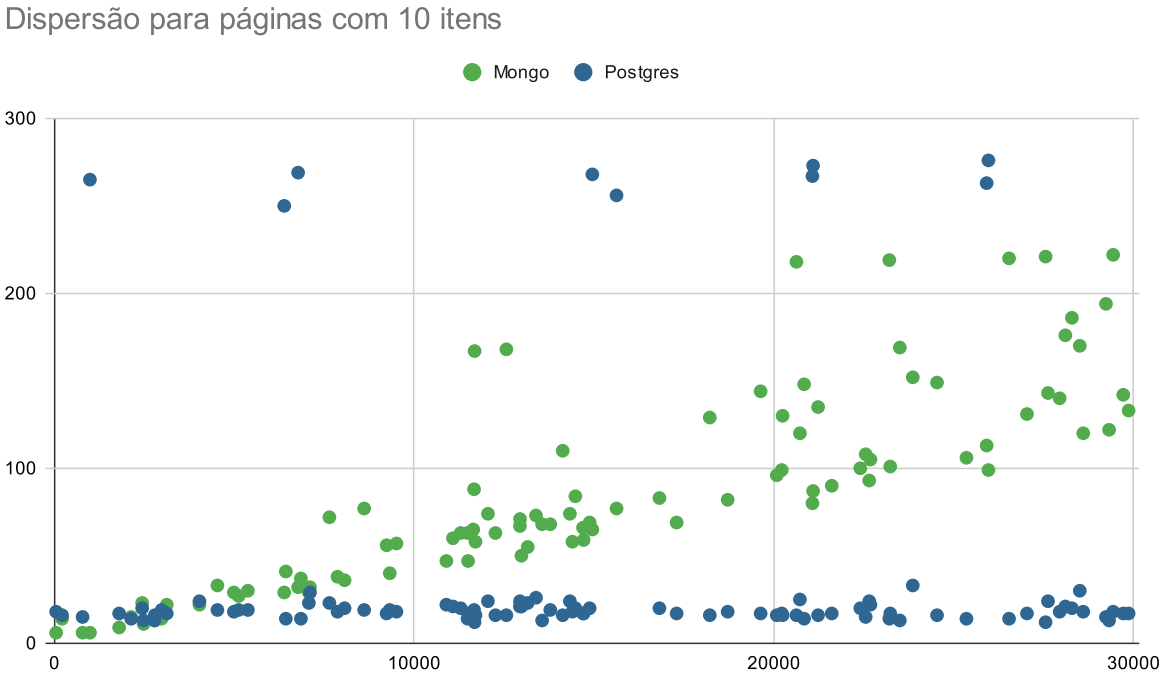

Figura 24 – Listagem paginada de funcionários com páginas de 10 itens

Fonte: Autor, 2023.

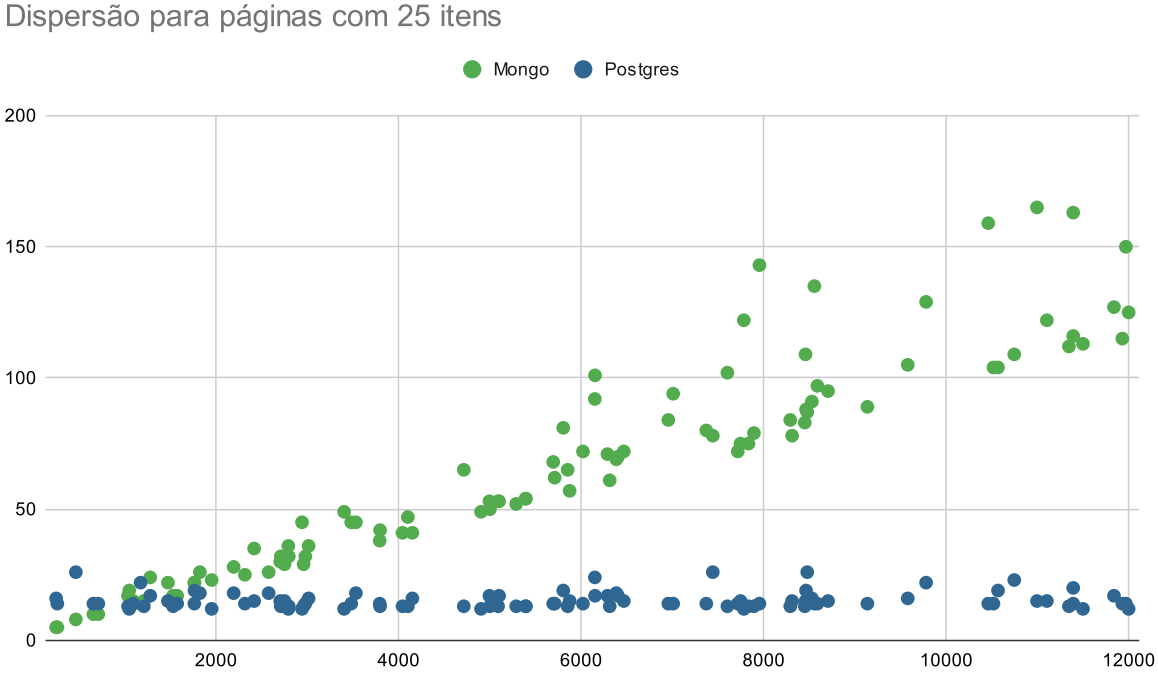

Figura 25 – Listagem paginada de funcionários com páginas de 25 itens

Fonte: Autor, 2023.

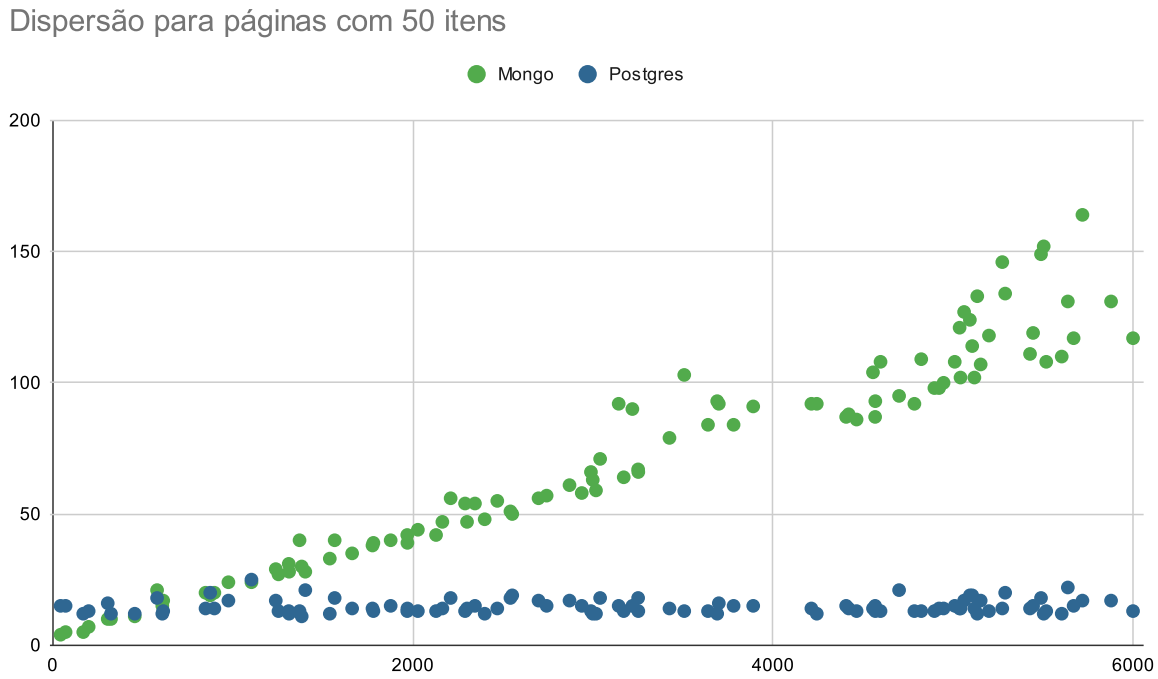

Figura 26 – Listagem paginada de funcionários com páginas de 50 itens

Fonte: Autor, 2023.

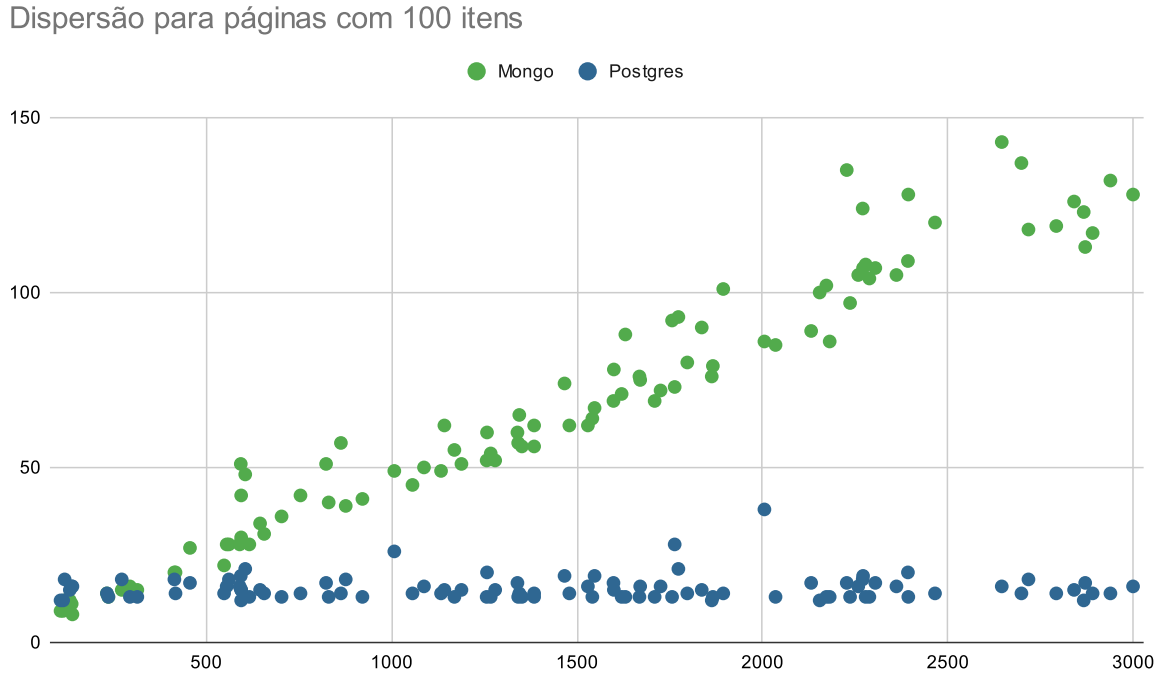

Figura 27 – Listagem paginada de funcionários com páginas de 100 itens

Fonte: Autor, 2023.

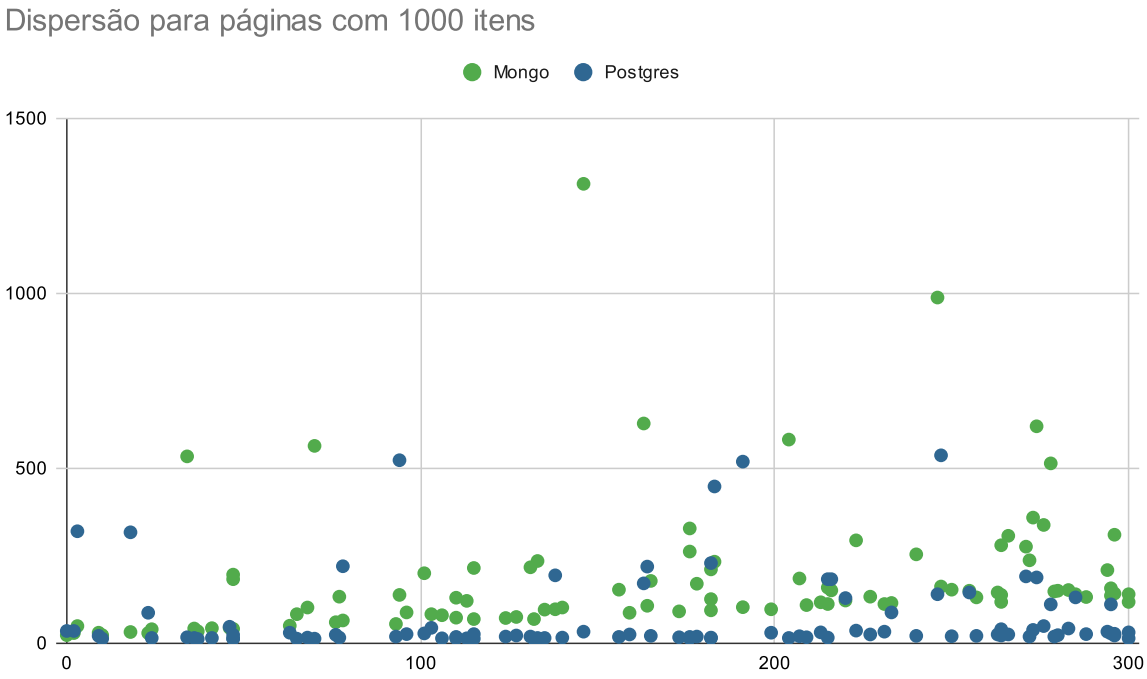

Figura 28 – Listagem paginada de funcionários com páginas de 1000 itens

Fonte: Autor, 2023.

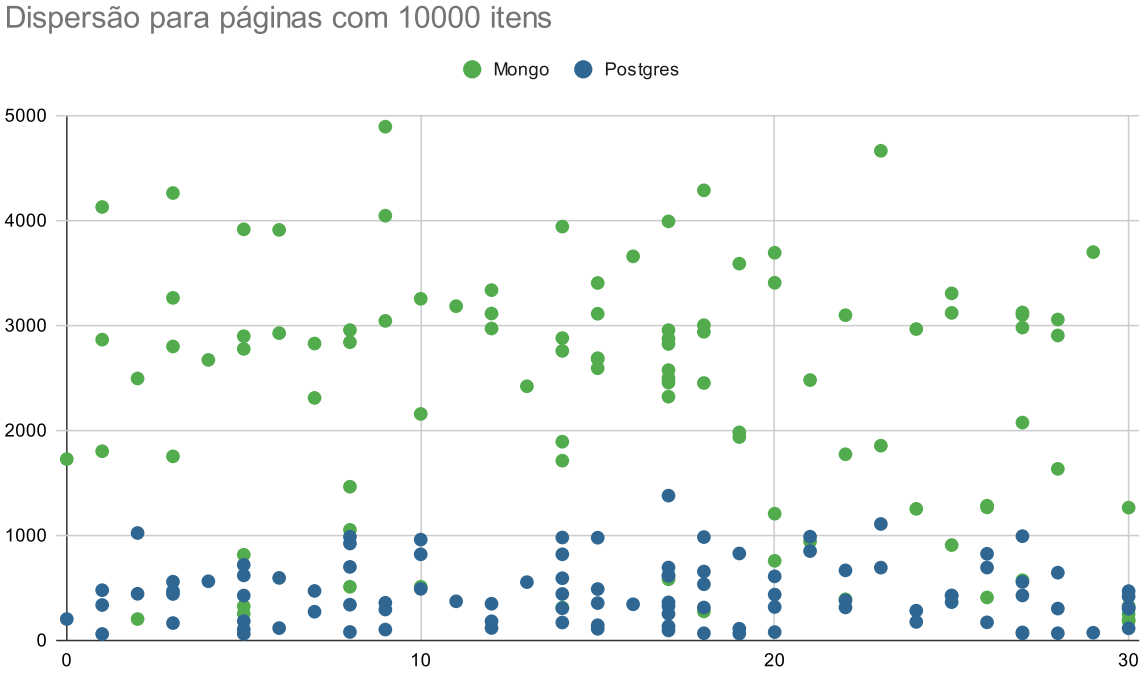

Figura 29 – Listagem paginada de funcionários com páginas de 10000 itens

Fonte: Autor, 2023.

- Paginação de itens;
- Pesquisa por itens a partir de atributos da tabela;
- Contagem de itens pertencentes a outro item, e
- Consulta com junção a outras tabelas.

Por outro lado, o MongoDB, com índices, obteve vantagem de desempenho em consultas como:

- Contagem de itens;
- Contagem de itens aplicáveis a uma regra;
- Inserção de novo item;
- Atualização de item existente, e
- Consulta de itens ordenados por atributo.

Quadro 6 – Banco de dados mais adequado para cada funcionalidade e proporção

Funcionalidade	Banco	Proporção
Contagem de departamentos	MongoDB	4,08
Listagem paginada de funcionários	Postgres	3,62
Contagem de funcionários por departamento	MongoDB	2,28
Contagem de contratações em um mesmo período do ano em dois anos seguidos	MongoDB	21,95
Contagem de funcionários	MongoDB	1,03
Inserção de um novo funcionário	MongoDB	6,21
Busca por funcionário pelos campos nome e sobrenome	Postgres	2,39
Atualização de um funcionário	MongoDB	2,96
Consulta das 5 últimas contratações	MongoDB	13,0
Consulta das 6 últimas promoções	Postgres	6,06
Contagem de salários	Postgres	4,70
Contagem de títulos	Postgres	9,4

Fonte: Autor, 2023.

7 Conclusão

Conforme discutido no Capítulo 1, atualmente existem diversas formas de armazenar dados em bancos de dados. Considerando esta diversidade, e a necessidade em construir aplicações com bom desempenho, se faz necessário escolher a melhor, ou a mais adequada, alternativa de banco de dados.

Este trabalho busca abordar as principais diferenças, em termos de desempenho, entre os tipos de bancos de dados mais utilizados, detalhando suas vantagens e desvantagens. E, além disso, o trabalho tem o objetivo de produzir insumos suficientes para que, diante de uma aplicação web moderna, e um conjunto de funcionalidades definido, escolher o tipo de banco de dados mais adequado.

Como a literatura, disponível e utilizada como base para este trabalho, foca em operações atômicas, se torna pertinente uma análise mais detalhada sobre funcionalidades presentes em aplicações web comuns.

7.1 Status do Trabalho

Desenvolveu-se o presente trabalho em duas etapas igualmente importantes:

- Definição e comparação, com base em literaturas, do desempenho entre bancos de dados relacionais e não relacionais; e
- Construção e análise de resultados de uma aplicação que compara estes dois tipos de banco de dados, os mais utilizados atualmente.

O desenvolvimento foi pautado por um objetivo geral, de estudo exploratório, subdividido em doze objetivos específicos. Embora os objetivos específicos se assemelhem a um passo a passo, estes foram úteis para o entedimento do andamento do trabalho, assim como uma métrica, para avaliação do êxito em alcançar o objetivo principal.

Na primeira etapa do presente trabalho, foram alcançados os seguintes objetivos específicos:

- Identificar as principais diferenças entre um banco de dados relacional e um não relacional;
- Abordar exemplos de bancos de dados relacionais e não relacionais;
- Descrever a base de dados exemplo escolhida;

- Modelar a base de dados exemplo, considerando um banco de dados relacional;
- Modelar a base de dados exemplo, considerando um banco de dados não relacional, e
- Apresentar as operações comuns em aplicações web modernas.

Nesta segunda etapa, foram alcançados os demais objetivos específicos:

- Realizar o tratamento necessário sobre a base de dados exemplo para cada uma das abordagens a serem comparadas;
- Construir o protótipo básico para o frontend da aplicação;
- Desenvolver o frontend responsável por solicitar as operações e mostrar as comparações de desempenho;
- Desenvolver as operações sobre cada banco de dados;
- Desenvolver os middlewares de monitoramento de tempo de resposta de cada requisição, e
- Analisar os resultados sobre os dados coletados durante o monitoramento da aplicação.

Ao final do desenvolvimento da primeira versão do produto de software, percebeu-se que a comparação apresentava ganhos, na maioria dos cenários, para o banco de dados relacional. Com o objetivo de tornar a comparação mais acirrada, foi seguida a sugestão da banca examinadora, na primeira etapa deste trabalho, em buscar alternativas que aproximassem o tipo de banco de dados com pior desempenho, do tipo de banco de dados com melhor desempenho. A utilização de índices, para o banco de dados não relacional, tornou a análise de resultados mais interessante, se aproximando bastante do resultado esperado pelo trabalho.

Em resumo, o MongoDB, utilizando índices, na base de dados exemplo utilizada, obteve vantagens em operações de:

- Seleção por categoria;
- Seleção ordenada por atributo;
- Inserção de um novo item;
- Atualização de um item existente, e

- Contagem de itens em tabelas/esquemas com poucos elementos. Para a base de dados exemplo, menos de 300 mil itens.

Enquanto o Postgres, para a mesma base de dados exemplo, obteve vantagens em operações de:

- Seleção paginada;
- Seleção por atributo que aplique a determinada regra;
- Seleção de itens pertencentes a um item pai;
- Seleção de itens, aplicáveis a uma regra, pertencentes a um item pai, e
- Contagem de itens em tabelas/esquemas com muitos elementos. Para a base de dados exemplo, mais de 300 mil itens.

Já o MongoDB, em sua implementação sem índices, só obteve vantagens, em relação ao Postgres, em operações de:

- Inserção de um novo item, e
- Contagem de itens em tabelas/esquemas com poucos elementos. Para a base de dados exemplo, 9 itens.

7.2 Trabalhos Futuros

Com base nos resultados obtidos, e a experiência adquirida pelo autor no desenvolvimento deste trabalho, vislumbra-se os seguintes trabalhos futuros:

- Estender a comparação não somente a tipos de bancos de dados, mas diversos bancos de dados diferentes, dentro de ambos os tipos e, até mesmo, a outros tipos de bancos de dados;
- Buscar otimizações, para ambos os tipos de bancos de dados, com objetivo de superar a alternativa mais adequada, em cada funcionalidade;
- Realizar as mesmas comprações e análises sobre bases de dados relativamente menores, assim como bases de dados ainda maiores das utilizadas no presente trabalho, e
- Aumentar o conjunto de funcionalidades com funcionalidades mais específcas, ou menos comuns, porém de cunho estratégico nas comparações.

Embora, neste trabalho, tenham sido aplicadas otimizações somente ao MongoDB, também existem otimizações aplicáveis ao PostgreSQL:

- Utilização de índices;
- Utilização do comando *VACUUM* para recuperação de espaço em disco e otimização de desempenho;
- Utilização do comando *ANALYZE* para a coleta de estatísticas do banco de dados para utilização na execução de comandos, e
- Criação de *clusters*. Esta alternativa também está disponível para o MongoDB.

Referências

APOLINáRIO, F. *Dicionário de metodologia científica: um guia para a produção do conhecimento científico*. 2. ed. [S.l.]: Grupo GEN, 2011. Citado na página 39.

ARORA, R.; AGGARWAL, R. R. Modeling and querying data in mongodb. *International Journal of Scientific and Engineering Research*, v. 4, n. 7, p. 141–144, 2013. Citado 2 vezes nas páginas 41 e 48.

BANKER, K. *MongoDB in Action*. [S.l.: s.n.], 2011. Citado na página 51.

DIANA, M. D.; GEROSA, M. A. Nosql na web 2.0: Um estudo comparativo de bancos não-relacionais para armazenamento de dados na web 2.0. In: *Workshop de Teses e Dissertações de Bancos de Dados do Simpósio Brasileiro de Bancos de Dados WTDBD2010*. [S.l.: s.n.], 2010. Citado 5 vezes nas páginas 23, 24, 30, 41 e 45.

ELMASRI;NAVATHE. *Sistemas de banco de dados*. 6. ed. [S.l.]: Pearson, 2011. Citado 2 vezes nas páginas 28 e 41.

FONSECA, J. *Metodologia da pesquisa científica*. [S.l.]: UEC, 2002. Citado 2 vezes nas páginas 39 e 40.

GERHARDT T. E.; SILVEIRA, D. T. *Metodologia Científica*. [S.l.]: Plageder, 2009. Citado na página 39.

GIL, A. C. *Como elaborar projetos de pesquisa*. 4. ed. [S.l.: s.n.], 2002. Citado na página 40.

GYORÖDI, C.; GYORÖDI, R.; SOTOC, R. A comparative study of relational and non-relational database models in a web-based application. *International Journal of Advanced Computer Science and Applications*, Science and Information (SAI) Organization Limited, v. 6, n. 11, p. 78–83, 2015. Citado 6 vezes nas páginas 25, 41, 46, 47, 54 e 66.

HE, C. Survey on nosql database technology. *Journal of Applied Science and Engineering Innovation Vol*, v. 2, n. 2, p. 50–54, 2015. Citado 2 vezes nas páginas 31 e 42.

JATANA, N. et al. A survey and comparison of relational and non-relational database. *International Journal of Engineering Research & Technology*, v. 1, n. 6, p. 1–5, 2012. Citado 6 vezes nas páginas 24, 30, 32, 33, 41 e 42.

LEONARD, A. *Pro Hibernate and MongoDB*. [S.l.]: Apress, 2013. Citado 3 vezes nas páginas 42, 46 e 51.

SILBERSCHATZ, A. *Sistema de Banco de Dados*. 7. ed. [S.l.: s.n.], 2020. Citado 4 vezes nas páginas 28, 29, 41 e 45.

TRUICA, C.-O.; BOICEA, A.; TRIFAN, I. Crud operations in mongodb. In: ATLANTIS PRESS. *2013 International Conference on Advanced Computer Science and Electronics Information (ICACSEI 2013)*. [S.l.], 2013. p. 347–350. Citado 2 vezes nas páginas 42 e 51.

VALENTIM, N. M. C.; SILVA, W.; CONTE, T. Avaliando a experiência do usuário ea usabilidade de um aplicativo web móvel: Um relato de experiência. In: *Cibse*. [S.l.: s.n.], 2015. p. 788. Citado na página 46.

Apêndices

APÊNDICE A – Código de exemplo de middleware

```
1 var express = require('express');
2 var app = express();
3
4 app.get('/', function (req, res, next) {
5   next();
6 });
7
8 app.listen(3000);
```

APÊNDICE B – Esquema do banco de dados MongoDB

```
{
    "employees": {
        "_id": "objectid",
        "emp_no": "int32",
        "gender": "string",
        "birth_date": "date",
        "hire_date": "date",
        "first_name": "string",
        "last_name": "string",
        "salaries": [{
            "salary": "float",
            "from_date": "date",
            "to_date": "date"
        }],
        "titles": [{
            "title": "string",
            "from_date": "date",
            "to_date": "date"
        }]
    },
    "departments": {
        "_id": "objectid",
        "dept_no": "int32",
        "dept_name": "string"
    },
    "dept_manager": {
        "_id": "objectid",
        "emp_no": "int32",
        "dept_no": "int32",
        "from_date": "date",
        "to_date": "date"
    },
    "dept_emp": {
```

```
34          "_id": "objectid",
35          "emp_no": "int32",
36          "dept_no": "int32",
37          "from_date": "date",
38          "to_date": "date"
39      }
40  }
```

APÊNDICE C – Código da função de transformação de instruções DML SQL para JSON

```
1  const parseTuples = (sql, fields) => {
2    const documents = [];
3    const sqlString = sql.replace(/INSERT\sINTO\s.+\sVALUES\s/g, '').replace(/\n/g, '
4    const tuples = sqlString.split('),');
5    tuples.forEach((tuple) => {
6      const values = tuple.replace(')', '').replace('(', '').replace(/'/g, '').split(
7      const document = {};
8      values.forEach((value, index) => {
9        document[fields[index]] = value;
10     });
11     documents.push(document);
12   });
13   return documents;
14 };
```

APÊNDICE D – Instruções DML SQL para inserção na tabela *departments*

```sql
INSERT INTO `departments` VALUES
('d001','Marketing'),
('d002','Finance'),
('d003','Human Resources'),
('d004','Production'),
('d005','Development'),
('d006','Quality Management'),
('d007','Sales'),
('d008','Research'),
('d009','Customer Service');
```

APÊNDICE E – JSON resultante das operações de substituição


```
[
    {
        "dept_no": "d001",
        "dept_name": "Marketing"
    },
    {
        "dept_no": "d002",
        "dept_name": "Finance"
    },
    {
        "dept_no": "d003",
        "dept_name": "Human Resources"
    },
    {
        "dept_no": "d004",
        "dept_name": "Production"
    },
    {
        "dept_no": "d005",
        "dept_name": "Development"
    },
    {
        "dept_no": "d006",
        "dept_name": "Quality Management"
    },
    {
        "dept_no": "d007",
        "dept_name": "Sales"
    },
    {
        "dept_no": "d008",
        "dept_name": "Research"
    },
```

```
    {
        "dept_no": "d009",
        "dept_name": "Customer Service"
    }
]
```

APÊNDICE F – Script em python para coleta de resultados de todas as rotas

```python
import requests
import threading
import time
from random import randint, shuffle

host = 'http://localhost/api'

routes = [
  'departments/count?',
  'employees/count?',
  'titles/count?',
  'salaries/count?',
  'employees/by/year?',
  'employees/by/department?',
  'last/hirings?',
  'last/promotions?',
  'employees/search?search=search&',
  'employees/update?',
  'employees/new?'
]

def get_random_employee():
  return requests.get(f'{host}/employee/get').json()

def get_employee():
  return requests.get(f'{host}/employee/random').json()

def target ():
  indexes = [i for i in range(len(routes))]
  shuffle(indexes)
  update = get_random_employee()
  search = update['first_name']
  employee = get_employee()
```

```
34    for i in indexes:
35      dbs = ['mongo', 'postgres']
36      shuffle(dbs)
37      for db in dbs:
38        s = randint(5, 10)
39        time.sleep(s)
40        if 'new' in routes[i]:
41          requests.post(f'{host}/{db}/{routes[i]}id=test', json=employee)
42        elif 'search' in routes[i]:
43          requests.get(f'{host}/{db}/{routes[i]}id=test'.replace('=search', f'={search}'))
44        elif 'update' in routes[i]:
45          requests.put(f'{host}/{db}/{routes[i]}id=test', json=update)
46        else:
47          requests.get(f'{host}/{db}/{routes[i]}id=test')
48
49
50  if __name__ == "__main__":
51      threads = [threading.Thread(target=target) for t in range(100) ]
52      for t in threads:
53        t.start()
```

APÊNDICE G – Script em python para coleta de resultados da rota da funcionalidade de paginação

```python
import requests
import threading
import time
from random import randint, shuffle

host = 'http://localhost/api'

route = 'employees/all?page=page&limit=limit&'

limits = [10, 25, 50, 100, 1000, 10000]
def get_random_page(id, limit):
  count = requests.get(f'{host}/postgres/employees/count?id={id}').json()['response
  page = randint(0, int(count/limit))
  return page

def get_employee():
  return requests.get(f'{host}/employee/random').json()

def target ():
  for limit in limits:
    page = get_random_page('test', limit)
    employee = get_employee()
    dbs = ['mongo', 'postgres']
    shuffle(dbs)
    for db in dbs:
        s = randint(5, 10)
        time.sleep(s)
        requests.get(f'{host}/{db}/{route}id=test'.replace('=page', f'={page}').rep


if __name__ == "__main__":
```

```
32      threads = [threading.Thread(target=target) for t in range(100) ]
33      for t in threads:
34        t.start()
```