UNIVERSIDADE FEDERAL DO CEARÁ

***CAMPUS* SOBRAL**

DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO

CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

FRANCISCA JANNIELLY GARCIA DA COSTA

UMA ANÁLISE COMPARATIVA SOBRE O USO DE BANCOS DE DADOS RELACIONAL E NÃO RELACIONAL NO CONTEXTO DE APLICAÇÕES WEB

SOBRAL

2023

FRANCISCA JANNIELLY GARCIA DA COSTA

UMA ANÁLISE COMPARATIVA SOBRE O USO DE BANCOS DE DADOS RELACIONAL E NÃO RELACIONAL NO CONTEXTO DE APLICAÇÕES WEB

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação do *Campus* Sobral da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Fischer Ferreira

SOBRAL

2023

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

C872a Costa, Francisca Jannielly Garcia da.
Uma análise comparativa sobre o uso de bancos de dados relacional e não relacional no contexto de aplicações web / Francisca Jannielly Garcia da Costa. – 2023.
109 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Sobral, Curso de Engenharia da Computação, Sobral, 2023.
Orientação: Prof. Dr. Fischer Jonatas Ferreira.

1. Banco de dados. 2. Web. 3. SGBD. 4. MySQL. 5. MongoDB. I. Título.

CDD 621.39

FRANCISCA JANNIELLY GARCIA DA COSTA

UMA ANÁLISE COMPARATIVA SOBRE O USO DE BANCOS DE DADOS RELACIONAL E NÃO RELACIONAL NO CONTEXTO DE APLICAÇÕES WEB

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. Fischer Ferreira (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Carlos Elmano de Alencar e Silva
Universidade Federal do Ceará (UFC)

Prof. Me. Fernando Rodrigues de Almeida Júnior
Universidade Federal do Ceará (UFC)

Dedico este trabalho à minha amada avó Antônia *(in Memoriam)*, que não pôde estar presente fisicamente durante essa jornada, mas me criou e é base do que sou. Esta conquista também é sua, vó.

AGRADECIMENTOS

Primeiramente, quero expressar minha gratidão a Deus, cujo apoio constante tornou possível a realização desse sonho que planejamos juntos em oração.

À minha família, meu coração transborda de amor e reconhecimento, em especial à minha mãe, Iranete, cujo apoio foi além de financeiro, foi e é um alicerce de carinho e amor incondicional, assim como aos meus irmãos, Artur e Heitor, minha madrinha Isalena, meu padrinho Francisco e meu avô Osmar, que compartilharam cada passo dessa jornada.

Aos meus amigos e companheiros, quero que saibam que cada momento compartilhado tornou essa jornada mais emocionante e menos árdua. Nos momentos mais difíceis, vocês foram luz. Klayver, Gisely, Ana Lara, Priscila, Maria Rita, Jander e Douglas. Obrigada por tanto!

Quero expressar minha sincera gratidão ao meu professor orientador, Fischer, que acreditou em meu potencial, me guiou pelos desafios e sempre esteve disposto a me motivar e oferecer ajuda quando necessário.

Aos membros da banca, professores Elmano e Fernando, que aceitaram o convite e dedicaram seu valioso tempo para avaliar este trabalho, meu agradecimento.

A todos os professores que contribuíram para minha base de conhecimento, quero dizer que a engenheira de computação que sou hoje foi moldada por um pedaço do conhecimento de cada um de vocês. À Michelle, pela serenidade e disponibilidade de sempre. E, por fim, a UFC, pela estrutura e apoio que tornaram esta jornada possível.

Obrigada por fazerem parte desta jornada inesquecível.

RESUMO

CONTEXTO: Bancos de dados são estruturas que têm como intuito o armazenamento de informações sobre um determinado contexto. As informações armazenadas podem ser classificadas como relacionais, quando estruturadas, ou não relacionais, quando heterogêneas e semiestruturadas ou não estruturadas. A quantidade e o tipo dos dados dependem diretamente do ambiente de origem. Aplicações Web, por exemplo, tendem a gerar dados em grande quantidade e variedade de formatos. Para lidar com esses dados, é necessário a utilização de um software que permite a criação, organização e manipulação dos mesmos, que são os Sistemas de Gerenciamento de Banco de Dados (SGBDs). Atualmente, existe uma variedade de SGBDs no mercado, e para trabalhar com a alta demanda de armazenamento e processamento dos dados, é fundamental levar em consideração o SGBD a ser escolhido, suas especificidades e limitações. MOTIVAÇÃO: Entretanto, a partir da revisão bibliográfica feita concluiu-se que não há trabalhos recentes na literatura que levantem requisitos e características que auxiliem na escolha de SGBDs no contexto de aplicações Web. OBJETIVO: A fim de colaborar com o preenchimento dessa lacuna, o presente estudo tem como objetivo principal comparar os SGBDs mais utilizados em aplicações Web, com a finalidade de apresentar os resultados e suas principais características. Os SGBDs escolhidos foram: MySQL (banco de dados relacional) e MongoDB (banco de dados não relacional orientado a documentos). METODOLOGIA: Para obter os resultados, o estudo foi dividido em duas partes. Na primeira parte foi realizada uma comparação baseada em aspectos estáticos de bancos de dados. Na segunda parte, os Sistema de Gerenciamento de Banco de Dados (SGBD)s foram comparados dinamicamente, onde objetivou-se submetê-los as mesmas consultas, mesmo *hardware* e utilizando base de dados similares. Ao final das duas etapas, foi possível levantar características dos SGBDs e avaliar métricas, como performance e desempenho. RESULTADOS: Como resultados, pôde-se observar que o MySQL e o MongoDB apresentam respostas similares em relação a comparação estática, divergindo apenas na classificação do Teorema de CAP. Com Relação a adaptabilidade, o MongoDB possui suporte a um maior número de linguagens de programação, já o MySQL ganha em relação ao número de Sistema Operacional (SO)s suportados. Já na análise dinâmica, os resultados apontaram a eficiência do MongoDB, com melhor performance no consumo de *Random Access Memory* (RAM), Central Processing Unit (CPU) e espaço em disco dos dados armazenados, além de menor tempo de execução. BENEFICIADOS: Os resultados do trabalho em tela são úteis para profissionais da área de bancos de dados e desenvolvimento Web, como engenheiros e desenvolvedores de software, bem como empresas que utilizam.

Palavras-Chaves: Banco de dados, SGBD, Web, MySQL e MongoDB.

ABSTRACT

BACKGROUND: Databases are structures intended to store information about a given context. The stored information can be classified as relational, when structured, or non-relational, when heterogeneous and semi-structured or unstructured. The amount and type of data directly depend on the storage environment's origin. Web applications, for example, tend to generate data in large quantities and varieties. of formats. To deal with this data, it is necessary to use software that allows the creation, organization, and manipulation of them, which are the Management Systems of Database (DBMSs). Currently, there are a variety of DBMSs on the market, and for Working with the high demand for data storage and processing, it is essential to take into account the DBMS to be chosen, its specificities, and limitations. MOTIVATION: However, from the literature review carried out, it was concluded that there are no recent works in the literature that raise requirements and characteristics that help in the choice of DBMSs in the context of Web applications. OBJECTIVE: To help fill this gap, the main objective of this study is to compare the most used DBMSs in applications Web, to present the results and their main characteristics. The chosen DBMSs were: MySQL (relational database) and MongoDB (non-relational database-oriented documents). METHODOLOGY: To obtain the results, the study was divided into two parts. In the first part, a comparison was carried out based on static aspects of databases. In the second part, the Database Management System (DBMS) were dynamically compared, where the aim was to submit them to the same queries, even hardware and using similar databases. At the end of the two stages, it was possible to raise characteristics of DBMSs and evaluate metrics, such as performance and performance. RESULTS: As a result, it was observed that MySQL and MongoDB present similar responses about static comparison, differing only in the classification of the CAP Theorem. With Regarding adaptability, MongoDB supports a greater number of programming languages programming, MySQL wins about the number of Operating Systems (OS) supported. Already in the dynamic analysis, the results showed the efficiency of MongoDB, with better performance in the consumption of Random Access Memory (RAM), Central Processing Unit (CPU), and storage space disk of stored data, in addition to lower time execution. BENEFICIARIES: The results of the screen work are useful for professionals in the field of databases and Web development, such as software engineers and developers, as well as companies that use them.

Keywords: Database. DBMS. Web. MySQL. MongoDB.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Com a chegada da Web, a quantidade de usuários da Internet vem aumentando exponencialmente ao longo do anos e, concomitantemente, há também um crescimento no número e no formato dos dados presentes na mesma. Com o intuito de atender às necessidades de armazenamento e processamento, surgiram novas tecnologias associadas à criação de aplicações Web, como linguagens de programação, *frameworks* e tipos de bancos de dados (LÓSCIO *et al.*, 2011; JOSE; ABRAHAM, 2020).

Banco de dados pode ser descrito como todo e qualquer conjunto de informações que representam um domínio específico, armazenadas eletronicamente em um ou mais dispositivos e podendo ser classificadas como relacionais ou não relacionais (MICROSOFT, 2023b; ORACLE, 2023; AMAZON, 2023). Bancos de dados relacionais representam seus dados por meio de tabelas e esquemas predefinidos, onde colunas correspondem aos atributos e as linhas são os registros da tabela (MACÁRIO; BALDO, 2005; ORACLE, 2023). Já os não relacionais não possuem uma estrutura predefinida e podem variar de acordo com a necessidade da aplicação, podendo ser organizados como documentos, chave-valor, grafos ou colunas (JOSE; ABRAHAM, 2020; MACÁRIO; BALDO, 2005).

A forma com que os dados se comportam depende do ambiente no qual estão inseridos, e a maneira como são armazenados e manipulados, influencia diretamente no desempenho da aplicação (MANOVICH, 2015). Para que seja possível gerenciar as informações contidas em um banco de dados, é necessário utilizar uma ferramenta chamada de Sistema de Gerenciamento de Banco de Dados (SGBD). Este software permite gerenciar um banco de dados desde sua criação, facilitando a visualização, manipulação e garantindo consistência e permanência dos dados nele armazenados (GARCIA-MOLINA *et al.*, 2008). Para cada variação de banco de dados, existem um ou mais sistemas que atendem suas características. Por exemplo, o `MySQL` e o `PostgreSQL` são utilizados para bancos de dados relacionais, enquanto o `MongoDB` e o `Redis` são empregados para bancos de dados não relacionais (DBENGINES, 2023).

Como já citado, as aplicações Web tendem a gerar dados em grande quantidade e variedade de formatos. Para trabalhar com essa alta demanda de armazenamento e processamento, é fundamental levar em consideração o SGBD a ser escolhido, suas especificidades e limitações (CELESTI *et al.*, 2019). Entretanto, até onde se sabe, não há trabalhos recentes na literatura que levantem requisitos e características, e realizem comparações dinâmicas a fim de auxiliar na escolha SGBDs no contexto de aplicações Web. Diante dessa conclusão, o presente

trabalho possui como intuito principal realizar uma comparação dos SGBDs mais utilizados em aplicações Web, com base no ranking da plataforma *DB-Engines* [1], com a finalidade de levantar as principais características de cada um deles e expor os resultados. Os SGBDs escolhidos foram: `MySQL` (banco de dados relacional) e `MongoDB` (banco de dados não relacional orientado a documentos) (DBENGINES, 2023).

Para obter os resultados, houve a divisão duas etapas principais. Na primeira etapa, foi realizada uma comparação baseada em aspectos estáticos de bancos de dados, onde o objetivo é comparar cada um com base nos princípios relacionados às Transações ACID, Teorema de CAP, Suporte a Sistemas Operacionais e Linguagens de Programação, além da disponibilidade de Documentações Oficiais e Ferramentas de Gerenciamento Gratuitas. Já na segunda etapa, efetuou-se uma comparação dinâmica, onde os SGBDs foram submetidos as mesmas consultas, utilizando bases de dados semelhantes e mesmo hardware. Nesta fase, foi conduzido um teste de desempenho [2]. O teste foi elaborado para realizar requisições simultâneas em cada um dos SGBDs, abrangendo 12 cenários distintos. Estes cenários englobam diversas operações de *Create, Read, Update e Delete* (CRUD), com ênfase nas principais requisições HTTP (POST, GET, PUT e DELETE) (MOZILLA, 2023b). As requisições foram distribuídas em três blocos, cada um simulando diferentes cargas de usuários. O Bloco 1 simula 100 usuários realizando requisições simultâneas, enquanto o Bloco 2 envolve 1.000 usuários. Por fim, o Bloco 3 intensifica o teste com 10.000 usuários simultâneos.

Ao final das duas etapas, houve o levantamento e comparação das características dos SGBDs e avaliação das métricas, em relação ao tempo de execução e consumo de CPU e RAM, além do espaço em disco ocupado com o armazenamento dos dados. Diante dos resultados deste trabalho, foi possível expor as principais características de cada um dos SGBDs, junto com os resultados da comparação dinâmica. Os dados poderão ser úteis para profissionais da área de bancos de dados e desenvolvimento, e empresas que utilizam sistemas baseados em aplicações Web.

O restante dos capítulos da presente proposta de pesquisa está organizado da seguinte forma: no Capítulo 2 está a motivação para realização deste trabalho. No Capítulo 3 são expostos os objetivos deste trabalho, bem como as questões de pesquisa. O Capítulo 4 apresenta uma visão geral com foco na didática do conteúdo. O Capítulo 5 apresenta a metodologia, detalhando os procedimentos para a realização das análises. No Capítulo 6, são expostos os resultados e

[1] *DB - ENGINES*

[2] Conceito: Teste de Desempenho.

discussões. E por fim, no Capítulo 7, são as considerações finais e propostas de trabalhos futuros.

2 MOTIVAÇÃO

Com o crescimento exponencial do número de usuários da Internet e a diversificação dos dados presentes na Web, surgem desafios significativos de armazenamento e processamento. As tecnologias associadas ao desenvolvimento de aplicações Web, incluindo linguagens de programação e tipos de bancos de dados, se tornam essenciais nesse contexto.

Os bancos de dados, sejam eles relacionais ou não relacionais, representam conjuntos de informações eletronicamente armazenados. Enquanto os relacionais utilizam tabelas e esquemas predefinidos, os não relacionais oferecem estruturas flexíveis, como documentos ou grafos, adaptáveis às necessidades da aplicação. A maneira como os dados são gerenciados influencia diretamente o desempenho da aplicação. Neste contexto entram os SGBDs, responsáveis pela administração, manipulação e garantia da consistência dos dados, afetando diretamente a experiência do usuário e os custos operacionais.

No entanto, apesar da importância e da crescente demanda por armazenamento e processamento de dados em aplicações Web, diante de uma análise da literatura, tornou-se evidente a carência de estudos recentes que promovam uma comparação dinâmica e estática entre os SGBDs relacionais e não relacionais no âmbito das aplicações Web. Essa lacuna motiva este trabalho a realizar uma análise que abrange tanto aspectos estáticos quanto dinâmicos, buscando identificar comportamentos e características, e auxiliar na escolha do SGBD) mais adequado para aplicações Web.

3 OBJETIVOS

Neste capítulo são expostos os objetivos (geral e específicos) deste trabalho, além de introduzir as questões de pesquisa que nortearão a análise comparativa entre SGBDs `MySQL` e `MongoDB`.

3.1 Objetivo Geral

O objetivo geral deste trabalho é realizar uma comparação entre os SGBDs mais utilizados em aplicações Web, conforme classificação da plataforma *DB-Engines* (DBENGINES, 2023). O intuito é identificar e expor as principais características e comportamentos dos SGBDs `MySQL` (relacional) e `MongoDB` (não relacional orientado a documentos) diante dos parâmetros de comparação.

3.2 Objetivos Específicos

- Analisar aspectos estáticos dos bancos de dados, incluindo Transações ACID, Teorema de CAP, Suporte a Sistemas Operacionais e Linguagens de Programação, além da disponibilidade de Documentações Oficiais e Ferramentas de Gerenciamento Gratuitas;
- Comparar dinamicamente os SGBDs `MySQL` e `MongoDB` sob consultas semelhantes, utilizando bases de dados e hardware equivalentes. O foco é analisar o consumo de recursos (RAM, CPU e Disco) e o tempo de resposta.

3.3 Questões de Pesquisa

- RQ_1: Como os SGBDs se classificam dentro da métricas da análise estática?
- RQ_2: Em relação às características levantadas para a comparação estática, em quais delas os SGBDs `MySQL` e `MongoDB` divergem?
- RQ_3: Qual dos SGBDs demonstra maior flexibilidade em termos de suporte a linguagens de programação e sistemas operacionais?
- RQ_4: Como os SGBDs se comportam em relação ao consumo de RAM e CPU em cada um dos cenários estabelecidos?
- RQ_5: Qual é o espaço em disco real consumido por cada SGBD para armazenar a mesma quantidade de dados em diferentes configurações?

- RQ_6: Como os SGBDs respondem em termos de tempo durante os três blocos de testes?

4 BANCO DE DADOS PARA WEB

Esta seção tem como principal objetivo apresentar uma visão geral sobre conceitos usados e relacionados ao contexto teórico deste trabalho. O trabalho apresenta e discute: Aplicações Web (Seção 2.1), Uma visão geral sobre bancos de dados (Seção 2.2), Sistemas de Gerenciamento de Banco de Dados (Seção 2.3), MySQL (Seção 2.4), MongoDB (Seção 2.5), Redis (Seção 2.6) e Mapeamento objeto-relacional (Seção 2.7).

4.1 Aplicações Web

Criada pelo departamento de defesa dos Estados Unidos da América (EUA) em 1960, a *Advanced Research Projects Agency Network* (ARPANET) foi a primeira rede de computadores, na qual sua principal função era facilitar a comunicação em territórios com zona de guerra. Logo mais, na década de 70, surgiram os primeiros protocolos de comunicação, como o *Transmission Control Protocol* (TCP) e o *Internet Protocol* (IP), e foi a partir dessa época que a rede começou a ser chamada de Internet. O cientista da computação Tim Berners-Lee foi o responsável por desenvolver o primeiro buscador de informações na Internet. O navegador permitia o acesso a informações variadas por meio de um agrupamento de hiperlinks. Esse mecanismo foi o pontapé inicial para o que se conhece atualmente como Web (W3C, 2023).

Aplicações Web são *softwares* que executam diretamente em um navegador Web, permitindo que o usuário interaja e consuma o conteúdo por meio de uma Interface Gráfica do Usuário (GUI). Estas aplicações podem ser executadas em qualquer dispositivo com um navegador Web e uma conexão com a Internet. Sendo assim, é possível acessá-la por meio de computadores pessoais, notebooks, celulares e tablets.

Atualmente, o desenvolvimento de aplicações Web é dividido em duas vertentes, chamadas de *stack*s, as *stack front-end* e *stack back-end* (MOZILLA, 2023a). O *front-end* é o lado visual da aplicação, ou seja, a parte com a qual o usuário interage. Existe uma variedade de elementos que fazem parte do desenvolvimento *front-end*, como *layouts*, botões, *cards*, menus, formulários, textos e animações. As tecnologias-base desta *stack* são *HyperText Markup Language* (HTML), *Cascading Style Sheets* (CSS) e *JavaScript* (JS). O HTML é uma linguagem de marcação, e é usada para criar a estrutura da aplicação. O CSS é usado para estilizar a estrutura criada com HTML, dando cor, forma e outras configurações de estilo. Já o JS, por sua vez, é responsável por deixar a estrutura dinâmica e facilitar a interação com a página. Existem

outras linguagens que também são usadas na Web para *front-end* e uma infinidade de extensões e *frameworks* que facilitam a criação das aplicações (MILETTO; BERTAGNOLLI, 2014).

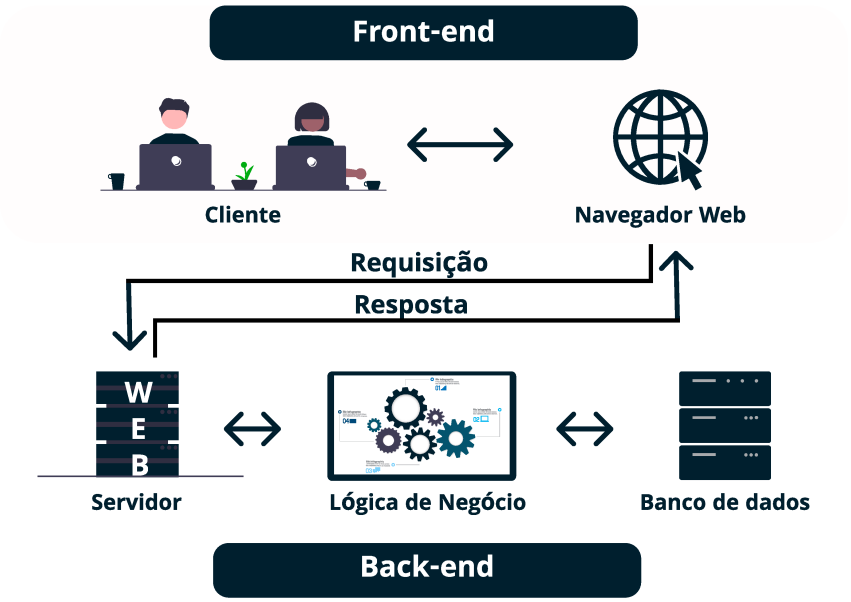
O *back-end* é o lado lógico da aplicação e é invisível ao usuário final. Geralmente, é a parte da infraestrutura em que são desenvolvidos os serviços a serem consumidos pelo *front-end* e na qual há comunicação e manipulação de informações quem vêm de bancos de dados. A interação entre a *stack front-end* e *back-end* se dá através de uma API, por meio da qual é possível estabelecer uma comunicação entre sistemas e serviços seguindo as lógicas de negócio e aplicação previstas na especificação do projeto. A comunicação em uma API é feita a partir de um conjunto de métodos, cada um com uma função específica, sendo que os métodos mais populares do protocolo *Hypertext Transfer Protocol* (HTTP) (MOZILLA, 2023b), que são:

- *GET*: utilizado para solicitar um ou mais recursos;
- *POST*: envia uma informação para ser armazenada;
- *PUT*: envia uma atualização para um recurso já armazenado;
- *DELETE*: remove uma informação/recurso armazenada.

Assim como no *front-end*, existe uma diversidade de ferramentas, linguagens e seus *frameworks* que são usados para o desenvolvimentos dos serviços *back-end*, como o próprio `JavaScript e TypeScript` com `Node`, `Java` com `Spring-Boot`, `C Sharp` com `.NET`, `Python` com `Django` e também tecnologias de bancos de dados, como bancos relacionais e não relacionais (IMASTER, 2023).

A Figura 1 apresenta um diagrama funcional de toda a infraestrutura envolvida em aplicações Web, do *front-end* até o *back-end*.

Figura 1 – Comunicação de aplicações Web



Fonte: Adaptado de (DICODING, 2023)

Pode-se observar no diagrama da Figura 1 que enquanto o *front-end* se localiza no lado do cliente, o *back-end* fica no lado do servidor, de tal forma que o cliente interage com o *front-end* da aplicação por meio do navegador, que executa localmente no dispositivo do usuário. O *front-end*, por sua vez, interage com o *back-end* através da API. Todo esse fluxo bidirecional de requisições e respostas é baseado no protocolo HTTP (MOZILLA, 2023b).

4.2 Uma visão geral sobre bancos de dados

Banco de dados pode ser descrito como todo e qualquer conjunto de informações que representam um domínio específico, armazenadas eletronicamente em um ou mais dispositivos e podem estar organizadas de maneira relacional ou não relacional. Essas informações podem variar de um simples dado a estruturas de maior complexidade, como por exemplo, arquivos de áudio e vídeo. O armazenamento destes dados pode ocorrer de maneira estática ou dinâmica, sendo assim, estarão disponíveis para o usuário visualizar e também para serem manipulados e gerenciados (REDIS, 2023), (MONGODB, 2023a), (MYSQL, 2023). A fim de garantir que a manipulação e gerenciamento dos dados ocorra de maneira precisa e eficiente, é necessário levar em consideração a forma como os dados são organizados. Foi com esse intuito que surgiu o processo de modelagem de dados.

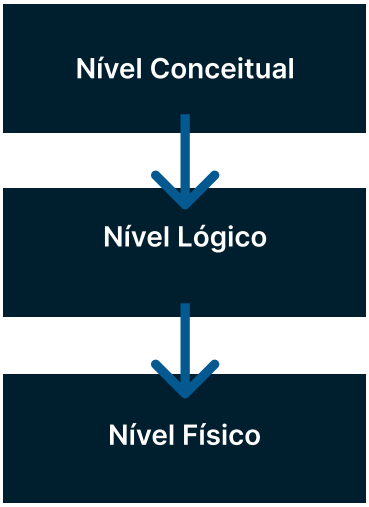
4.2.1 Modelagem de dados

A modelagem de dados é uma etapa indispensável no processo de criação de um banco de dados, pois consiste em uma sequência de passos que visam identificar o contexto no qual os dados estão inseridos e desenvolver um modelo abstrato que os represente adequadamente e funcionalmente. Esse modelo serve não somente para definir como os dados são organizados, mas também auxilia a garantir a integridade e consistência, já que com um modelo de dados bem definido fica mais fácil validar as informações de entrada e saída.

Existe uma hierarquia de três níveis em que a modelagem de dados pode ser categorizada, que são: conceitual, lógico e físico (MACORATTI, 2023). A Figura 2 representa de maneira visual como está organizado o fluxo dos níveis de modelagem de dados.

- Nível Conceitual: esse nível é o primeiro, e trata-se do mais abstrato, pois é nele que será feita uma análise e levantamentos dos requisitos, a fim de definir as estruturas em que os dados estarão armazenadas e/ou relacionados;
- Nível Lógico: nesse nível define-se de fato o modelo de dados que será utilizado, e para isso, é levado em consideração as informações levantadas e definidas no nível conceitual;
- Nível Físico: esse nível, como o próprio nome já diz, trata-se de aspectos físicos relacionados ao armazenamento das informações. Logo, é nele que é implementado o modelo de dados e as configurações que são definidas no nível lógico.

Figura 2 – Fluxo dos nível de modelagem.



Fonte: Adaptado de (LEYENDECKER, 2023)

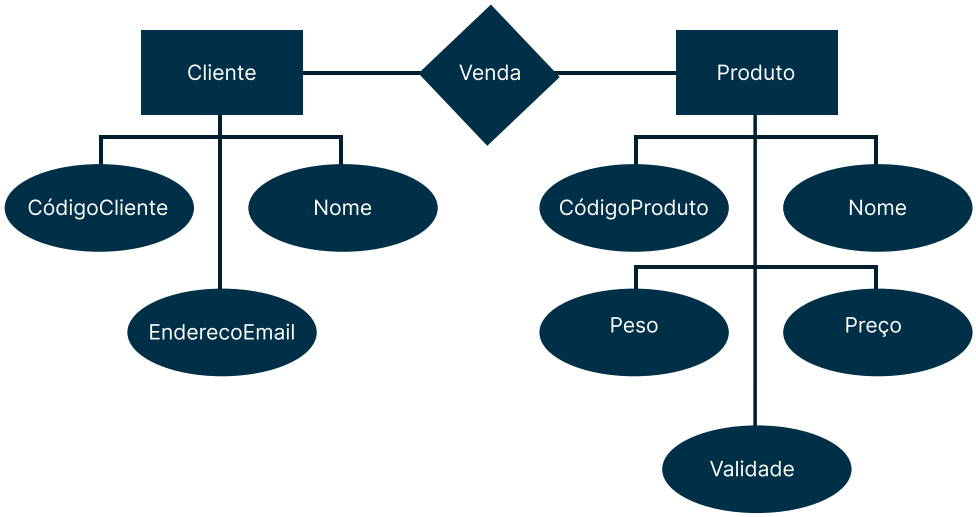
A nível conceitual, existe um modelo que é bastante difundido: o *Modelo Entidade*

Relacionamento (MER). O MER, por estar no primeiro nível da hierarquia, trata a organização de um projeto de banco com um alto nível de abstração. Para compreender como esse modelo funciona, é ideal entender as definições para os conceitos de: entidade, relacionamento, atributos e cardinalidade.

- Entidade: uma entidade busca representar um conceito do mundo real, como, por exemplo, um cliente ou produto.
- Atributo: é uma característica que uma determinada entidade possui, então se produto é uma entidade, seus atributos podem ser: identificador, nome, peso, data de validade, etc.
- Relacionamento: é o nome que se dá a associação entre duas ou mais entidades.
- Cardinalidade: é o indicativo de quantas instâncias de uma entidade estão associadas a outra, em um relacionamento. De maneira geral, a cardinalidade pode ser de três tipos:
  - Um para Um (1:1): onde uma instância de uma entidade só está relacionada a uma única instância de outra entidade em um relacionamento;
  - Um para Muitos (1:N): onde uma instancia de uma entidade pode se relacionar com várias instâncias de outra entidade em um relacionamento;
  - Muitos para Muitos(N:M): onde várias instâncias de uma entidade podem estar relacionadas a várias instancias de uma outra entidade em um relacionamento;

Existem alguns modelos visuais que auxiliam a compreender como os dados estão organizados dentro do modelo escolhido. Um dos mais populares é o diagrama de Entidade-Relacionamento, conhecido como Diagrama ER. Esse modelo consiste em representar, por meio de símbolos, os elementos que fazem parte do modelo de dados escolhido e organizá-los como um diagrama. Os retângulos representam as entidades, os losangos representam os relacionamentos e as elipses representam os atributos das entidades. A Figura 3 apresenta o relacionamento entre duas entidades. "Cliente", que possui 3 atributos (CódigoCliente, EnderecoEmail e Nome) e "Produto", que possui 5 atributos (CódigoProduto, Nome, Peso, Preço e Validade). O relacionamento entre as duas entidades é denominado "Venda".

Figura 3 – Representação do diagrama ER da relação de venda.



Fonte: autor.

A nível lógico, em banco de dados, existem dois grandes grupo em que os dados podem ser particionados, o modelo de dados relacional e o não relacional (GARCIA-MOLINA *et al.*, 2008).

4.2.2 Modelo de Dados Relacional

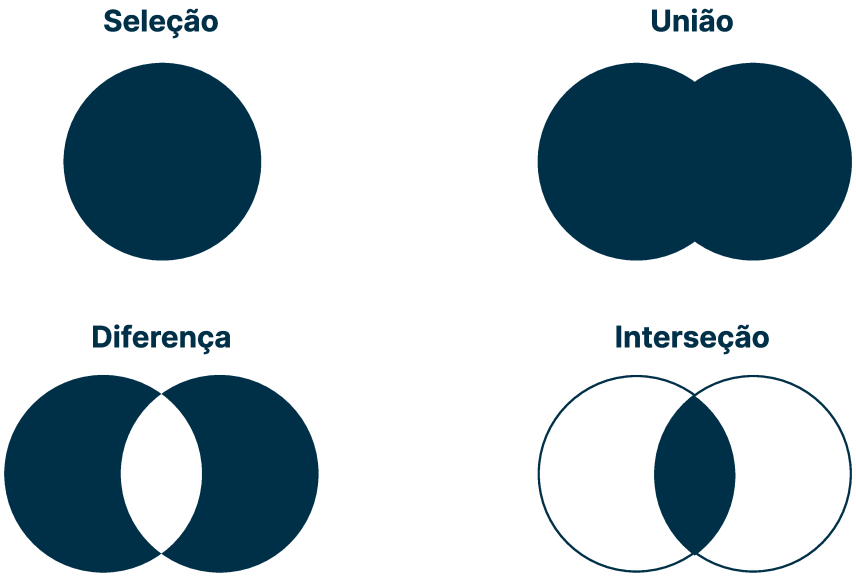
Criado por volta de 1969, com intuito de abstrair informações físicas (CODD, 1969), o modelo de dados relacional é um dos padrões de banco de dados mais utilizados. Sua popularidade se dá graças ao seu padrão rígido de organização das informações armazenadas, que facilita questões de consistência e segurança (JOSE; ABRAHAM, 2020). Para aprender como esse modelo funciona, é de suma importância entender os seguintes conceitos:

- Relação: em banco de dados, as relações, ou tabelas, são estruturas usadas para organizar os dados. Levando em consideração o MER, as tabelas estão fortemente ligadas ao conceito de entidades.
- Tupla: em banco de dados, as tuplas, são os registros de uma relação. Ou seja, as linhas de uma tabela;
- Coluna: em banco de dados, as colunas estão associadas as características ou atributos de uma relação (Entidade);
- Chave-Primária: refere-se a um ou mais atributos que servem para identificar cada tupla, tornando-as únicas dentro de um contexto.

- Chave-Estrangeira: representa uma chave primária de outra tabela e serve para estabelecer uma ligação entre duas tabelas;
- Restrições de Integridade: são as regras impostas sobre um determinado conjunto de dados a fim de assegurar a segurança e integridade.

A Álgebra Relacional é uma linguagem baseada em conceitos matemáticos que permite a realização de operações em um conjunto de dados. As operações podem variar entre seleção, união, interseção, diferença ou produto cartesiano. A Figura 4 representa a explicação visual das operações que podem ser realizadas através da Álgebra Relacional, tais: seleção, união, diferença e interseção.

Figura 4 – Operações de conjuntos



Fonte: Adaptado de (IONOS, 2018)

- Operação de seleção: capaz de realizar uma projeção dos dados.
- Operação de união: união de um ou mais conjuntos, podendo ou não estar condicionada a uma outra operação.
- Operação de interseção: seleção de dados que pertencem ao mesmo tempo aos conjuntos envolvidos.
- Operação de diferença: seleção de dados que estão em um conjunto mas não estão no outro.
- Operação de produto cartesiano: une um ou mais conjuntos, não se lavando em consideração as dimensões dos conjuntos envolvidos.

Tendo como base as operações da Álgebra Relacional, foi criada uma linguagem de consulta de banco de dados chamada de *Structured Query Language* (SQL). Essa linguagem possui palavras reservadas que facilitam realizar cada operação de união, seleção, interseção, diferença e produto cartesiano. Criada em 1970 pela IBM, o SQL tornou-se a linguagem padrão utilizada em bancos de dados relacionais e é frequentemente usada como modelo para outras linguagens de consulta (IBM, 2023).

4.2.3 Modelo de Dados Não Relacional

Diante da heterogeneidade e do grande volume de dados provenientes da popularidade de aplicações da Web, *Internet of Things* (IOT) e serviços em nuvem, bancos de dados relacionais acabaram não sendo uma alternativa suficiente para lidar com dados não estruturados. Essa problemática abriu margem para que organizações buscassem novas metodologias para solucionar esse entrave, e foi a partir disso que surgiram os SGBDs não relacionais, também conhecidos como *Not Only Structured Query Language* (NoSQL) (CELESTI *et al.*, 2019). NoSQL é um termo que visa representar modelos de dados que não usam a linguagem SQL como ferramenta principal para realizar consultas e operações. Atualmente, existe uma variedade de SGBDs no mercado, e o que diferencia cada um deles é o modelo de dado ao qual ele procura atender (RAMAKRISHNAN; GEHRKE, 2008). Os SGBDs não relacionais podem ser classificados em 4 tipos: Orientado a Documentos, Chave e Valor, Baseado em Colunas, e Orientado a Grafos (KHASAWNEH *et al.*, 2020).

4.2.3.1 Bancos de dados orientado a documentos

Um SGBD orientado a documentos armazena os dados em documentos no formato *Binary JSON* (BSON), que é uma variação do objeto JSON [1]. Ao contrário do documento textual, o BSON é armazenado em formato binário. Por ser flexível, essa forma de armazenamento possibilita um sistema de dados de maior escalabilidade, sendo ideal para dados não estruturados ou semi-estruturados. O Listing 4.2.1 é um documento que representa um estudante com os atributos: matrícula, nome, idade e as matérias às quais o mesmo está matriculado.

[1] Trabalhando com JSON

```
{
    "matricula": "123",
    "nome": "Jannielly",
    "idade": 22,
    "materias": [
        {
            "id": 1,
            "nome": "Ciências de Dados",
            "quantidade_de_horas": 64
        }
    ]
}

```

Listing 4.2.1 – Representação de um documento em formato *JavaScript Object Notation* (JSON)

4.2.3.2 Bancos de dados de chave e valor

Um SGBD orientado a chave e valor, como o próprio nome diz, armazena seus dados em termos de chave (que é responsável por identificar a propriedade) e o valor (que é o conteúdo). Não existe um formato fixo para o valor, portanto, ele pode variar entre uma simples variável, uma lista ou um objeto JSON. Os bancos de dados de chave e valor são reconhecidos por duas características importantes: seu modelo de acesso direto e sua estrutura otimizada, que juntas, possibilitam o rápido acesso aos valores e reduzem a necessidade de um processamento carregado. Esse tipo de armazenamento é voltado para aplicações que necessitam de maior velocidade de recuperação e manipulação dos dados. O Listing 2.2.2 representa uma estrutura de chave valor. A chave possui o identificador da estrurura, que é "pessoa123" e o valor é um objeto que possui algumas propriedades da "pessoa123"como: nome, altura e peso.

```
Chave: "pessoa123"
Valor: {
    "nome": "Jannielly",
    "altura": 159,
    "peso": "65kg"
}
```

Listing 4.2.2 – Representação da estrutura chave e valor

4.2.3.3 Bancos de dados baseados em colunas

Um SGBD baseado em colunas segue um princípio parecido com o modelo relacional, porém, ao invés dos dados serem armazenados em linhas, são armazenados em colunas. Essa forma de persistência de dados possibilita operações com maiores velocidades, já que as consultas que precisam de apenas algumas colunas podem ser executadas de maneira muito mais eficiente do que em bancos de dados relacionais tradicionais, que armazenam os dados por linha. Na Tabela 1, cada coluna representa as propriedades de um conjunto de pessoas, como nome, idade e sexo.

Nome	Jannielly	Ana Lara	Klayver	Jander
Idade	22	20	22	21
Sexo	Feminino	Feminino	Masculino	Masculino

Tabela 1 – Tabela colunar

4.2.3.4 Baseado em grafos

Um SGBD baseado em grafos armazena suas informações em grafos. Grafo é uma estrutura de dados que possui vértices e arestas. Essa estrutura é bastante difundida na área da computação, já que possibilita representar conexões entre objetos de maneira fácil de ser compreendida, sendo fortemente utilizada em aplicações que necessitam de uma sequência de ligações, como em redes sociais. No Listing 2.2.3 existe um nó central que representa uma pessoa chamada "Jannielly", a mesma possui 2 amigos conectados a ela por uma aresta do grafo.

```
{
  "nome": "Jannielly",
  "idade": 22,
  "amigos": [
    {
      "nome": "Ana Lara",
      "idade": 20
    },
    {
      "nome": "Gisely",
      "idade": 22
    }
  ]
}
```

Listing 4.2.3 – Representação de um grafo em formato JSON

4.3 Sistema de gerenciamento de banco de dados

SGBD é um software que tem como objetivo principal oferecer para o usuário um ambiente eficiente, organizado e seguro para fazer operações e transações com os dados. Com ele, é possível gerenciar um banco de dados desde sua criação, facilitando a visualização, manipulação e garantindo consistência e permanência de dados nele armazenados (GARCIA-MOLINA *et al.*, 2008). Em busca de facilitar a realização de consultas e manipulações em bancos de dados relacionais e não relacionais, foram criados uma diversidade de SGBDs, cada um com um propósito diferente, sendo os mais populares (DBENGINES, 2023):

- `Oracle Server`: Lançado em 1980 e desenvolvido pela *Oracle Corporation*, o `Oracle Server` é um SGBD que atende ao modelo de dados relacional e é de uso comercial. Suas principais características são o suporte a Transações ACID, multiplataforma, e a diversas linguagens de programação, como: `C`, `C Sharp`, `C++`, entre outras.
- `MySQL`: Lançado em 1995 e pertencente à *Oracle Corporation*, o `MySQL` é um SGBD que atende ao modelo de dados relacional e é open source. Suas principais características são o suporte a Transações ACID, multiplataforma, e a diversas linguagens de programação, como: `Ada`, `Java`, `JavaScript (Node.js)`, entre outras.
- `Microsoft SQL Server`: Lançado em 1989 e pertencente à *Microsoft Corporation*, o `Microsoft SQL Server` é um SGBD que atende ao modelo de dados relacional e de uso comercial. Suas principais características são o suporte a Transações ACID, aos sistemas operacionais Linux e Windows e a diversas linguagens de programação, como: `Delphi`,

Go, Java, entre outras.

- PostgreSQL: Lançado em 1989 e pertencente à *PostgreSQL Global Development Group*, o PostgreSQL é um SGBD que atende ao modelo de dados relacional e é *open source*. Suas principais características são o suporte a Transações ACID, diversos sistemas operacionais e a uma variedade de linguagens de programação, como: PHP, Python, Tcl, entre outras.
- MongoDB: lançado em 2007 e desenvolvido pela *MongoDB Corporation*, o MongoDB é um SGBD que atende ao modelo de dados não relacional e é *open source*. Suas principais características são a sua orientação a documentos, Transações ACID de vários documentos com isolamento instantâneo, ter suporte a diversos sistemas operacionais, e a uma variedade de linguagens de programação, como: Actionscript, C, C Sharp, entre outras.
- Redis: lançado em 2009 e pertencente à *Redis project core team*, o Redis é um SGBD que atende ao modelo de dados não relacional e *open source*. Suas principais características são sua orientação a chave e valor, ter suporte a diversos sistemas operacionais, e a diversas linguagens de programação, como: C, CSharp, C++, entre outras.
- Elasticsearch: lançado em 2010 e pertencente à *Elastic*, o Elasticsearch é um SGBD que atende ao modelo de dados não relacional e é *open source*. Suas principais características são seu mecanismo de pesquisa e análise distribuído e *RESTful* [2] moderno, suporte a todos os sistemas operacionais com uma *Virtual Machine* (VM) Java e a diversas linguagens de programação, como: .Net, Groovy, Java, entre outras.
- Cassandra: lançado em 2008 e pertencente à *Apache Software Foundation*, o Cassandra é um SGBD que atende ao modelo de dados não relacional e é *open source*. Suas principais características são o seu fator de replicação selecionável, suporte a diversos sistemas operacionais e a uma variedade de linguagens de programação, como: C Sharp, C++, Clojure, entre outras.

A qualidade de um SGBD depende de diversos fatores, por exemplo, o atendimento às limitações e especificidades do modelo de dados especificado para um dado projeto. Assim, existem algumas temáticas que são bastante difundidas quando se trata do modelo de SGBD a ser utilizado, e nas subseções seguintes são discutidas algumas dessas temáticas.

[2] *RESTful*

4.3.1 Transações ACID

A partir dos momento em que os bancos de dados começaram a ser usados em áreas mais críticas, como saúde e economia (UFRN, 2023), tornou-se essencial que os SGBDs pudessem garantir a confiabilidade e integridades dos dados. Diante dessa necessidade, um grupo de pesquisadores da IBM criou o conceito de estrutura ACID, sigla que faz referência aos requisitos fundamentais incorporados por esse conceito:

- Atomicidade: garante que todas as operações da transação sejam executadas sem erro, e caso ocorra, as demais operações devem ser revertidas;
- Consistência: garante que todos os comandos que devem ser realizados em uma operação sejam concluídos, a fim de deixar o banco sempre válido para futuras transações;
- Isolamento: garante que nenhuma transação possa interferir no funcionamento de outra, de maneira isolada;
- Durabilidade: garante que os dados armazenados estejam sempre disponíveis, mesmo em caso de falha de *hardware* ou *software*.

4.3.2 Teorema de CAP

O Teorema de CAP é outro quesito bastante difundido dentre os DBA's e desenvolvedores, já que com ele é possível escolher um conjunto de características e padrões do seu sistema de acordo com a necessidade. A sigla CAP representa três características, que são: consistência, disponibilidade (*Availability*) e partição.

- Consistência: capacidade de garantir que eventuais réplicas de dados em um sistema, devem estar atualizados simultaneamente;
- Disponibilidade: capacidade do sistema atender às necessidades requisições a ele enviadas;
- Partição: a capacidade do sistema de tolerar o particionamento dos dados.

O teorema afirma que é impossível se ter mais de duas das três definições acima e diante disso torna-se necessário analisar bem os requisitos do sistema de acordo com cada necessidade. As possibilidades são (SANTOS, 2017):

- CA: o sistema está sempre disponível e as réplicas de dados equivalentes entre si, mas sem tolerância a particionamento dos dados;
- AP: o sistema está sempre disponível e os dados podem ser particionados, porém, pode haver inconsistência entre réplicas de dados, já que podem estar em locais distintos.

- CP: a consistência entre réplicas de dados é garantida e os dados podem ser particionados, porém, há a possibilidade do sistema estar indisponível.

4.3.3 Cargas de trabalho e balanceamento de cargas

Cargas de trabalho (*workloads*) referem-se às tarefas a serem executadas, requisições a serem processadas ou dados a serem armazenados por sistemas de computação. Em aplicações Web, as cargas de trabalho podem variar entre buscas em bancos de dados, manipulações, processamento, transações, dentre outras, estando diretamente relacionadas ao número de acessos e ao foco da aplicação. Por exemplo, no contexto de redes sociais, em geral, a maioria das cargas de trabalho está relacionada ao armazenamento e processamento de dados, visto que possuem uma grande quantidade de usuários, conteúdos e um número alto de tempo de atividade (VOLPATO, 2023). Por outro lado, em aplicações relacionadas à venda, o foco pode estar mais ligado às transações e consultas de dados, uma vez que lojas realizam vendas sobre uma gama de produtos e/ou serviços.

O balanceamento de cargas tem o intuito de otimizar a distribuição de recursos e lidar de maneira eficiente com as requisições de usuários e aplicações. É nesse processo que é analisada a natureza da aplicação envolvida e, a partir disso, é identificada a melhor maneira de realizar as operações. Existem diversas maneiras de realizar o balanceamento e, como dito anteriormente, tudo dependerá do foco e tamanho da aplicação. No cenário de armazenamento de dados, as opções podem variar entre particionamento, distribuição geográfica ou uso de algoritmos criados especialmente para otimização de cargas de trabalho.

Elevadas cargas de trabalho em SGBDs, associadas à falta ou mau planejamento de balanceamento de cargas, gera ineficiência no uso dos recursos disponíveis, acarretando congestionamento, que implica lentidão e, no limite, recusa de atendimento às requisições, implicando em baixas eficiência e qualidade do *software*.

4.4 Uma visão geral sobre MySQL

Criado em 1995 pelos engenheiros suecos, Michael Widenius e David Axmarka (MYSQL, 2023), o `MySQL` está entre os mais populares sistemas de gerenciamento de banco de dados do tipo relacional (DBENGINES, 2023). Em sua concepção inicial, foi criado para ser um SGBD de uso pessoal, mas devido a sua facilidade de uso, simplicidade e disponibilidade, além

de ser um *software* de código aberto, logo foi comercializado e, em 2008, comprado pela *Oracle Corporation*, empresa a qual pertence até os dias de hoje.

O *software* está atualmente na versão 8.0 e desde sua criação foram implementadas diversas funcionalidades que justificam a sua popularidade atemporal (MYSQL, 2023). O `MySQL` possui suporte a *multithreading* e isso permite que diferentes *back-ends* acessem a mesma instância de um banco de dados e visualizem o mesmo conjunto de dados. No contexto de facilidades para uso em aplicações Web, as características que o diferenciam de outros SGBD's são: suporte à multiplataformas, recursos de gerenciamento de segurança, escalabilidade, suporte à Transações ACID e suporte à computação em nuvem. Além disso, existem alguns objetos que permitem a execução de ações específicas em resposta a consultas ou eventos, que podem ser procedimentos, funções, eventos, gatilhos ou *views*.

4.4.1 Ferramenta de Gerenciamento do MySQL

Existem alguns programas e recursos fornecidos pela *Oracle* a fim de facilitar a vida dos usuários `MySQL`, e o mais conhecido é o `MySQL Workbench`. O `MySQL Workbench` é um *software* de interface gráfica que possui diversas funcionalidades para simplificar e tornar intuitiva a implantação e gestão de bancos de dados ou servidores `MySQL` (MYSQL, 2023), sendo possível:

- Projetar: possibilita que o *Database Administrator* (DBA) projete, gere, configure e modele seu banco de dados de maneira visual. O *software* tem disponível vários tipos de modelagens em seu conjunto de ferramentas, que vão desde engenharia direta e reversa até mesmo modelos complexos de Entidade Relacionamento (ER);
- Desenvolver: fornece diversas ferramentas para fins de criação, otimização e execução de consultas, possuindo um editor SQL com algumas funcionalidades que facilitam o trabalho do desenvolvedor, tais como: preenchimento automático, histórico de execução, reutilização de código e realce de cores por sintaxe;
- Administrar: apresenta um menu com diversas funcionalidades ligadas diretamente à administração e configuração de bancos de dados e servidores, permitindo configurar servidores, administrar usuários, fazer *backup* e auditoria dos dados de maneira visual e simplificada.

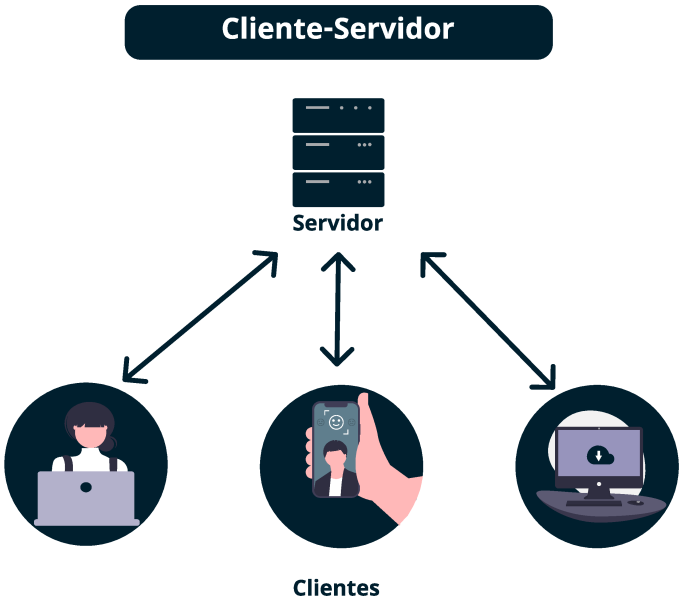
Além dessas funcionalidades, o `MySQL Workbench` possui uma gama de ferramentas que permite melhorar o desempenho dos aplicativos `MySQL` a partir de indicadores e migrar

facilmente um banco de dados de um outro SGBD relacional para o `MySQL`.

4.4.2 Arquitetura do MySQL

A arquitetura de um SGBD diz como os componentes deste *software* estão relacionados e como ocorre o funcionamento dos fluxos para o fornecimento dos serviços. O `MySQL` é baseado na arquitetura cliente-servidor, cuja principal característica é a divisão de funções em duas partes: a camada do cliente e a do servidor. A Figura 5 esboça o funcionamento de uma arquitetura cliente-servidor. Nesse exemplo, três clientes, através de requisições, solicitam serviços ou recursos ao servidor, que as processa e devolve as respostas aos clientes.

Figura 5 – Ilustração da arquitetura cliente-servidor



Fonte: Adaptado de (WIKIPéDIA, 2023)

A camada cliente tem a função de fornecer uma interface amigável para o usuário a fim de permitir que este possa realizar suas requisições para o servidor. É importante salientar que quando a consulta é realizada, existe um processo de validação de sintaxe e semântica, e caso tudo esteja correto, o resultado é obtido. Além disso, existe um conjunto de recursos e serviços nesta camada, que são: manipulação de conexão, autenticação e segurança.

- **Manipulação de conexão** é o processo em que o cliente estabelece uma comunicação com o servidor por meio de uma *thread*, e toda a troca de informação é realizada por meio desta;
- **Autenticação** é o processo realizado por parte do servidor para verificar se o cliente é

válido, e geralmente essa verificação é realizada por meio de *login* e senha;

- **Segurança** é o processo em que o servidor verifica se o usuário/cliente conectado tem permissão para realizar as consultas e manipulações solicitadas.

A camada servidor recebe as requisições do cliente, as processa e retorna à camada cliente o resultado demandado pelo usuário. E assim como na camada cliente, existe um conjunto de serviços que são realizados nesta camada, que são: tratamento de *thread, parser*, otimizador, cache de consulta, *buffer*, cache de metadados e cache da chave.

- **Tratamento de *thread*** é onde acontece o processo de fornecer e assegurar a conexão cliente-servidor por meio de uma *thread*.
- **Parser** é uma etapa que recebe a consulta a ser executada pelo SGBD e efetua uma análise semântica e sintática do código enviado pelo cliente. Caso esteja tudo correto, a consulta é separada em *tokens* que servirão para montar uma árvore de análise e a retornar para o próximo serviço;
- **Otimizador** recebe como entrada a árvore de análise e realiza algumas modificações, se necessárias, a fim de melhorar a consulta;
- **Cache de consulta** é um recurso que serve para verificar se a consulta que entrou para ser executada já foi realizada anteriormente. Caso já tenha sido, ela retorna a resposta sem a necessidade de passar pelas etapas de análise, *parse* e otimização;
- ***Buffer* e cache** são espaços de memória em que estão armazenados os dados frequentemente solicitados e acessados. No entanto, enquanto o *buffer* é usado para armazenar os dados que foram recentemente acessados, o cache armazena dados que provavelmente serão acessados em breve e para o armazenamento destes é levado em consideração as consultas realizadas anteriormente;
- **Cache de metadados** é um espaço de memória usado para rastrear informações de estruturas de armazenamento de dados, como bancos de dados e objetos. Ele contém informações como o nome das tabelas, colunas, índices e chaves primárias;
- **Cache de chave** é um tipo de identificador para uma estrutura de dados que está dentro do cache de metadados. Ele é usado para recuperar informações específicas sobre a estrutura de dados armazenada no cache de metadados.

4.4.3 *Instalação e Configuração*

Com intuito de demonstrar como o MySQL pode ser utilizado, será mostrada uma sequência de passos para a instalação e configuração do mesmo. Como já dito anteriormente, o MySQL é um SGBD multiplataforma, ou seja, pode ser instalado em qualquer SO, seja Windows, Linux ou MacOS (MYSQL, 2023). Nesta seção é realizada a instalação do MySQL Community e MySQL Workbench, além da configuração de um servidor MySQL numa máquina Windows.

1. Para efetuar o *download* do MySQL é preciso acessar o link [3] do site oficial. Ao acessar a página de *download*, é necessário escolher a versão do SO em que será instalado o SGBD, e após isso clicar em *download* conforme mostrado na Figura 6.

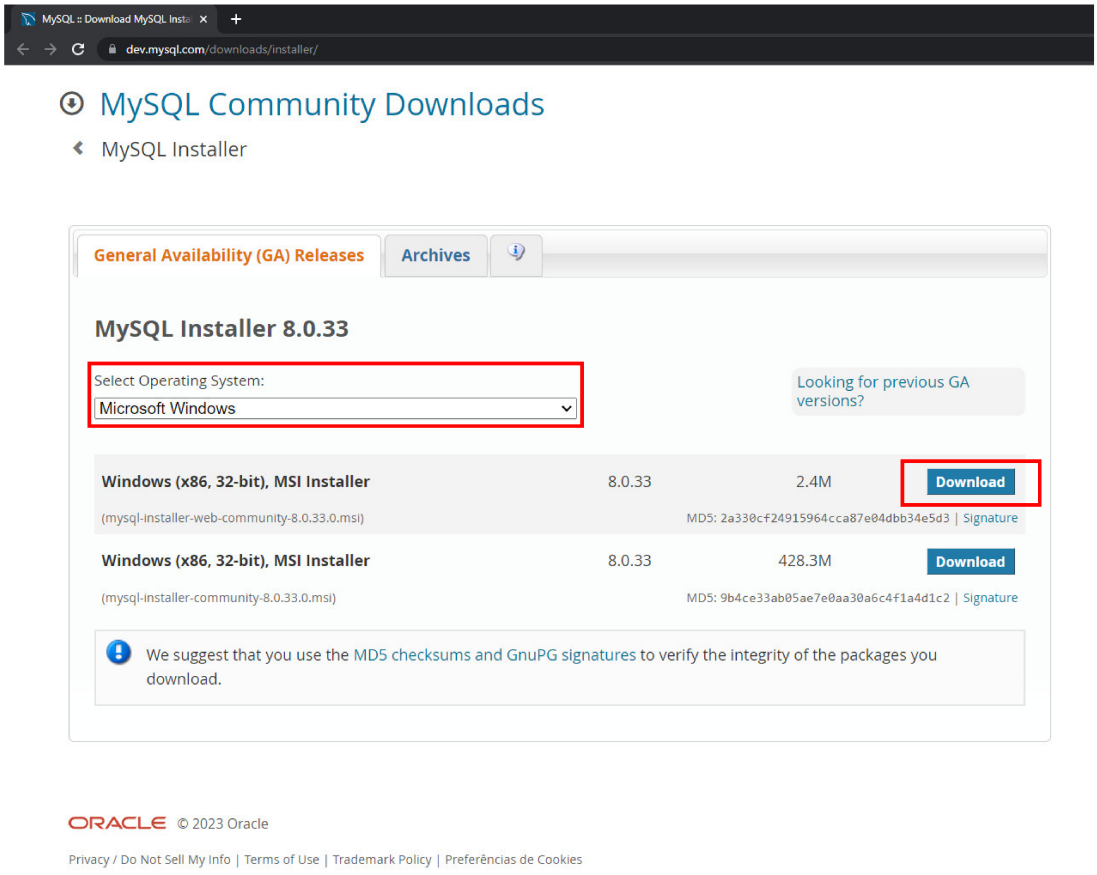


Figura 6 – Página inicial para download do MySQL

2. A Figura 7 demonstra a página de *login* para a conta *Oracle*. O *login* é opcional. Para continuar o *download* basta clicar em *No thanks, just start my download* (Não, obrigado, apenas inicie meu download).

[3] Download MySQL

Figura 7 – Página de login *Oracle account*

3. Após o *download* finalizado, basta clicar no instalador baixado e iniciar o processo de instalação. A primeira etapa é a escolha do tipo de configuração. Recomenda-se clicar em *Custom*, para customizar os programas que serão instalados juntos ao `MySQL` e depois em *Next*.

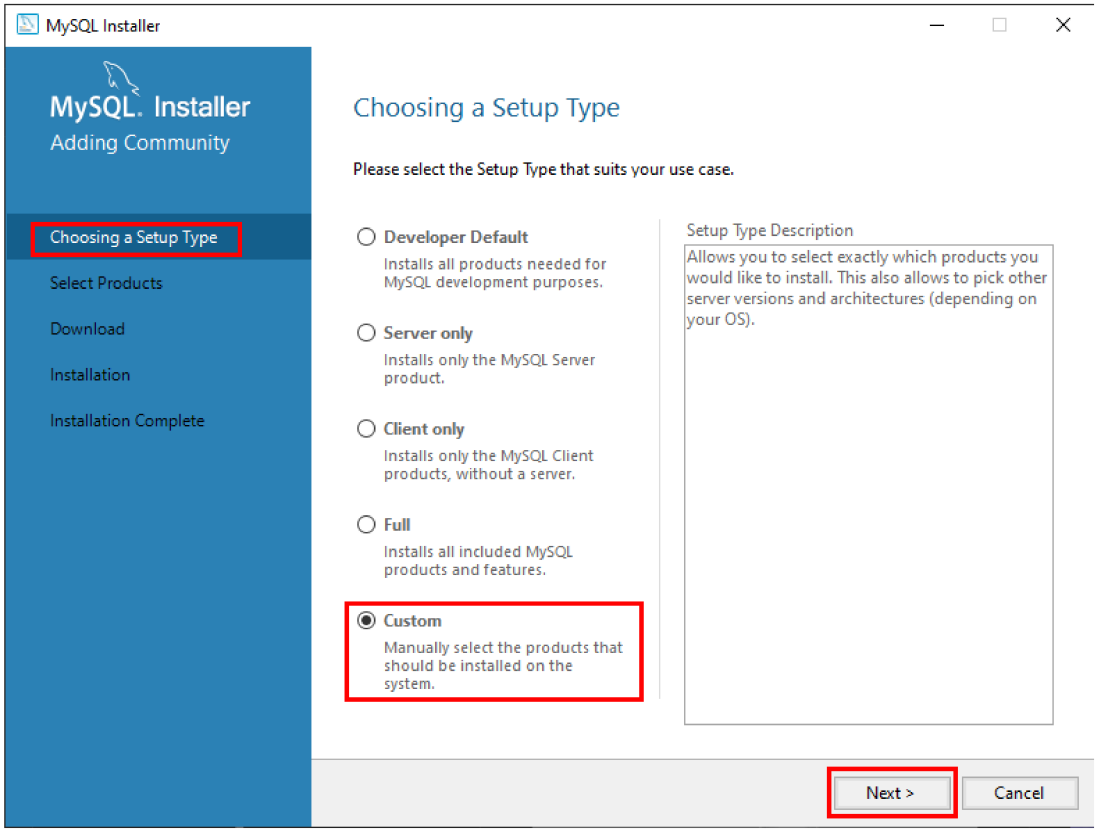


Figura 8 – Tela de escolha do tipo de instalação do `MySQL`

4. A Figura 9 demonstra a etapa em que ocorre a seleção dos programas e utilitários que serão instalados conjuntamente ao `MySQL`. É recomendado para iniciantes escolher a versão mais recente do `MySQL Server` e do `MySQL Workbench` . Em seguida, deve-se seguir a sequência de cliques: *Next* -> *Execute* -> *Next*.

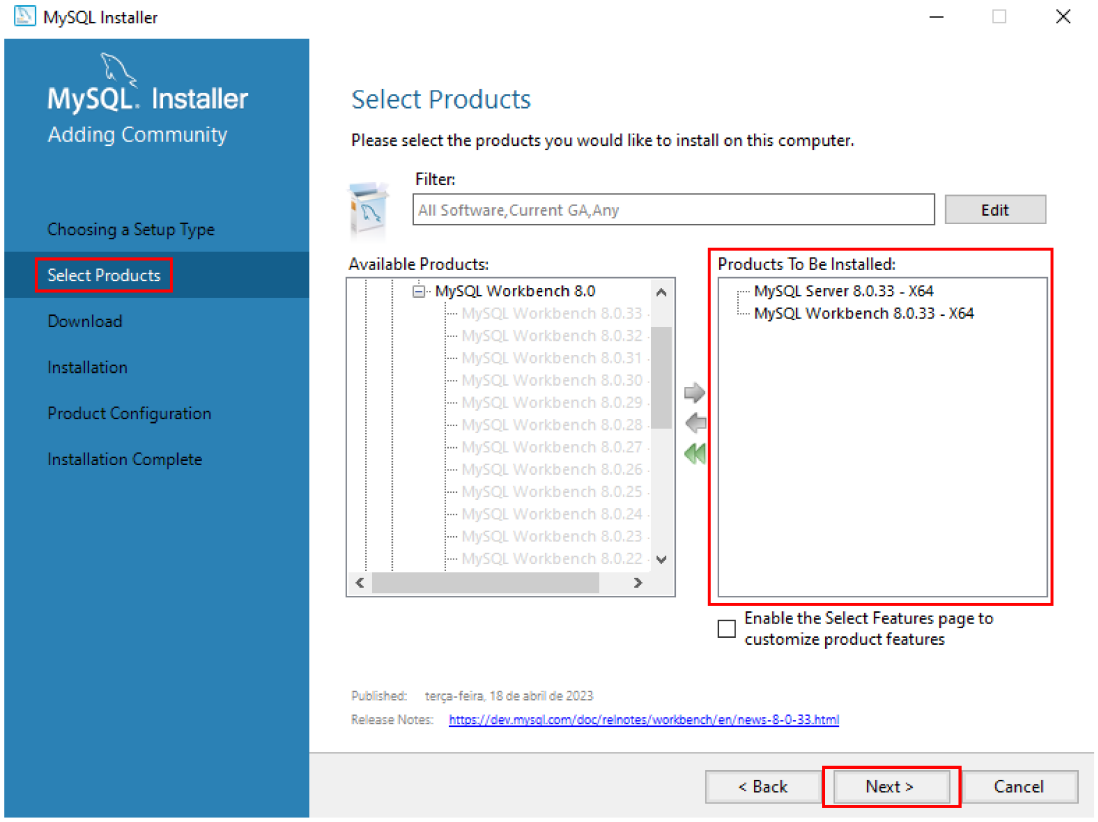


Figura 9 – Tela de escolha dos utilitários do MySQL

5. A Figura 10 representa a etapa de configurações de rede. Nesta, não é necessário modificar nada, apenas clicar em *Next*.

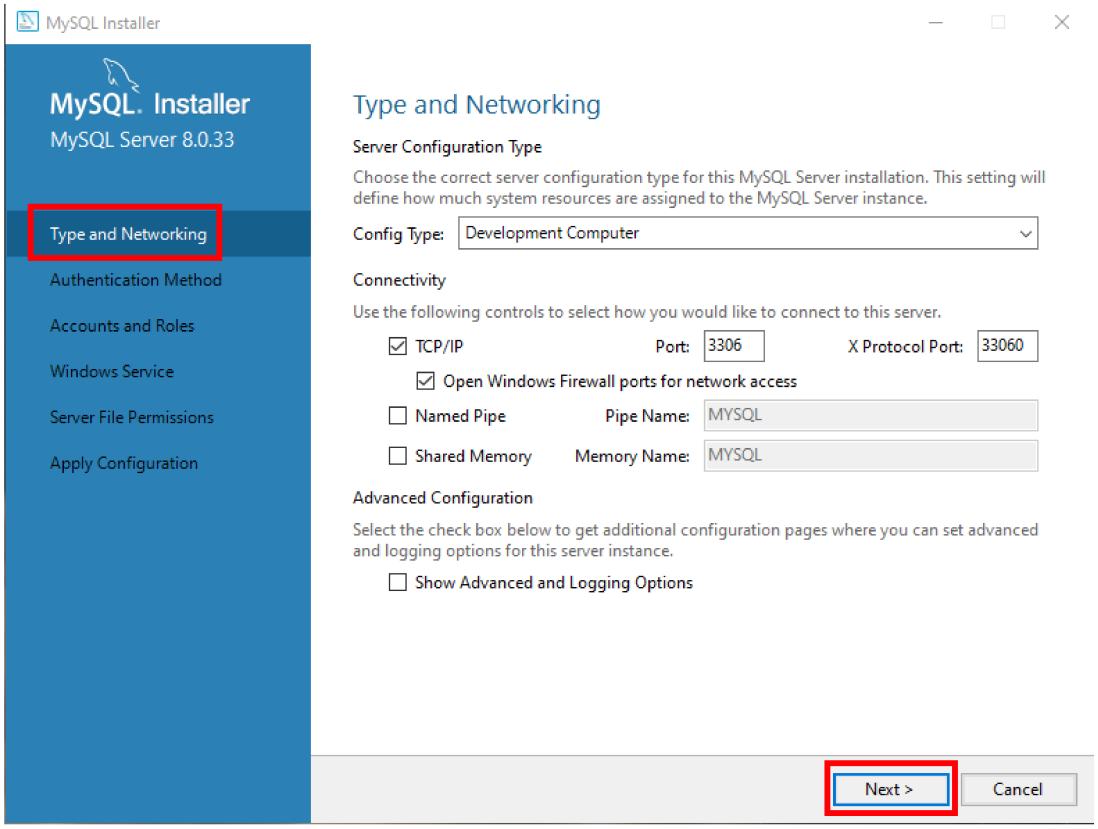


Figura 10 – Tela de configurações de rede do `MySQL`

6. Na etapa de *Authentication Method*, cuja tela é mostrada na Figura 11, são realizadas as configurações de autenticação. Nesta, o recomendado é marcar a primeira opção *Use strong Password Encryption Authentication* e, após, clicar em *Next*.

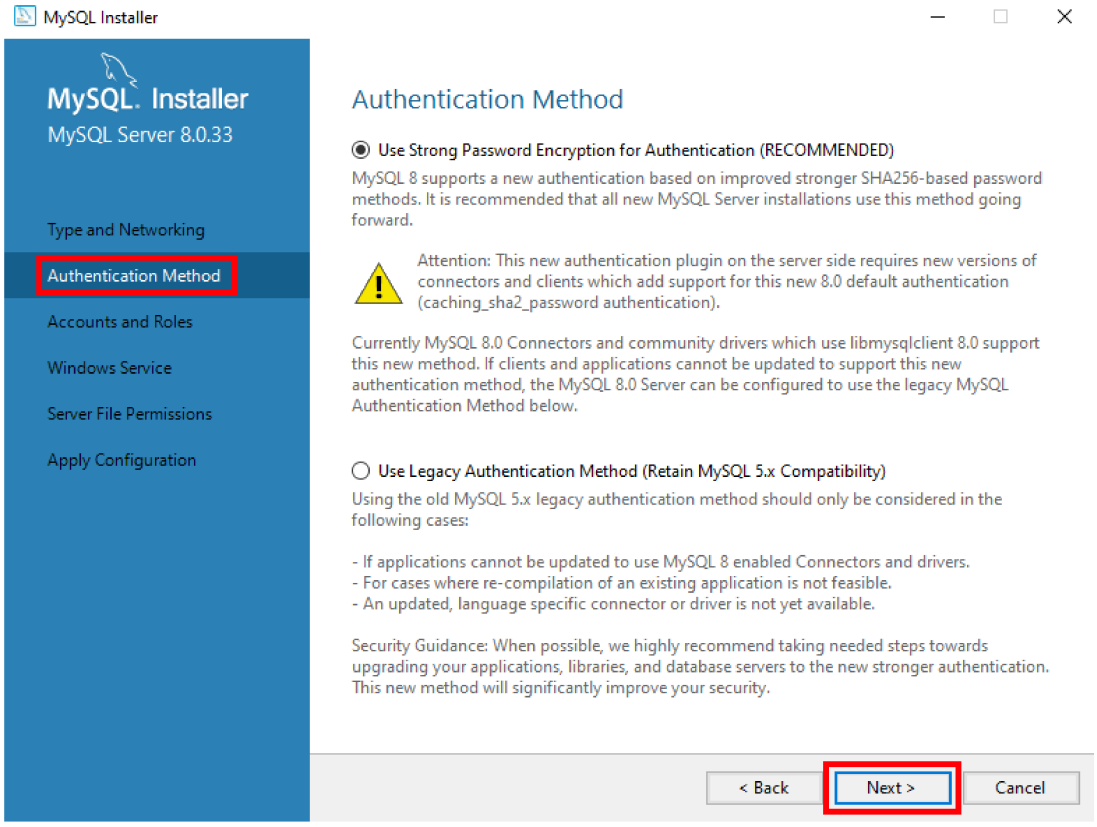


Figura 11 – Tela de configurações de autenticação do MySQL

7. Na etapa de *Accounts and Roles*, exemplificada pela Figura 11, são realizadas as configurações de usuário e senha. Nesta etapa é recomendado criar uma senha para o usuário *root*e, em seguida, clicar em *Next*. Nas próximas etapas não é necessário modificar nada, apenas confirmar as informações e continuar clicando em *Next* e *Execute* até finalizar.

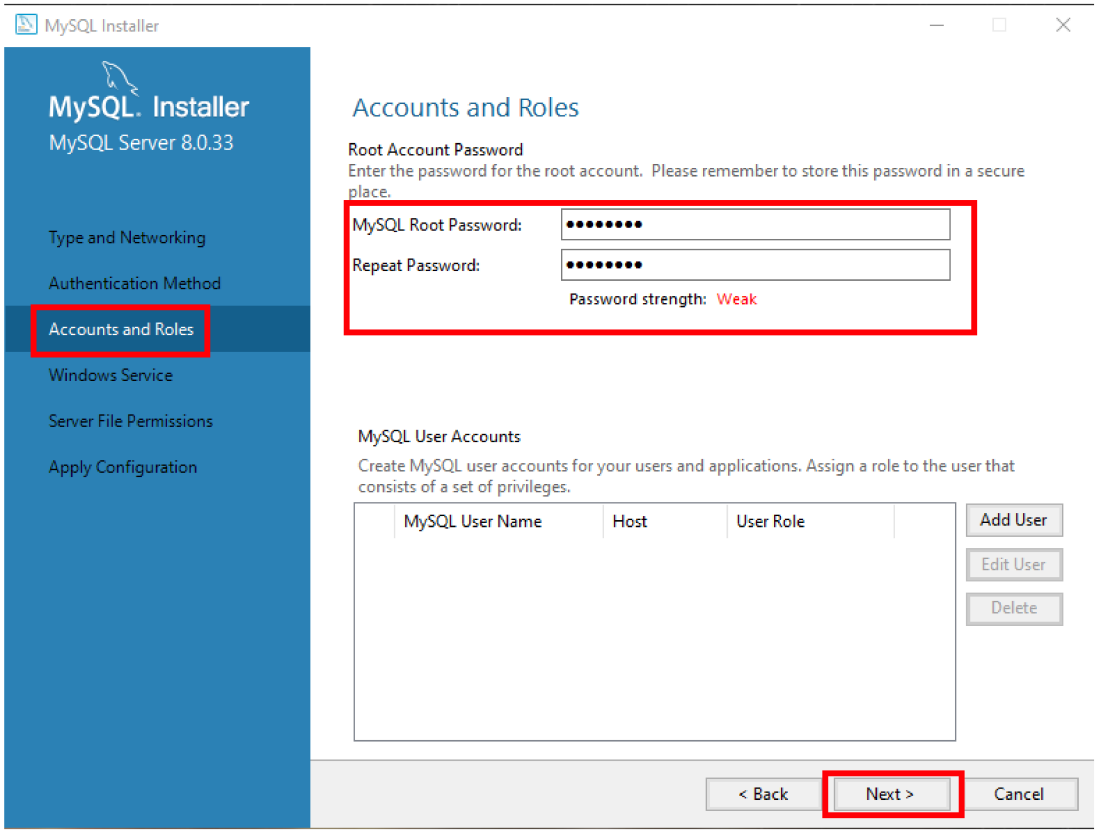


Figura 12 – Tela de configuração de contas e usuários do MySQL

8. Com o MySQL Workbench executado, a primeira ação é criar um servidor para o banco de dados. Para isto, basta clicar no ícone de 'mais' ao lado de MySQL *Connections*, conforme destacado na Figura 13.

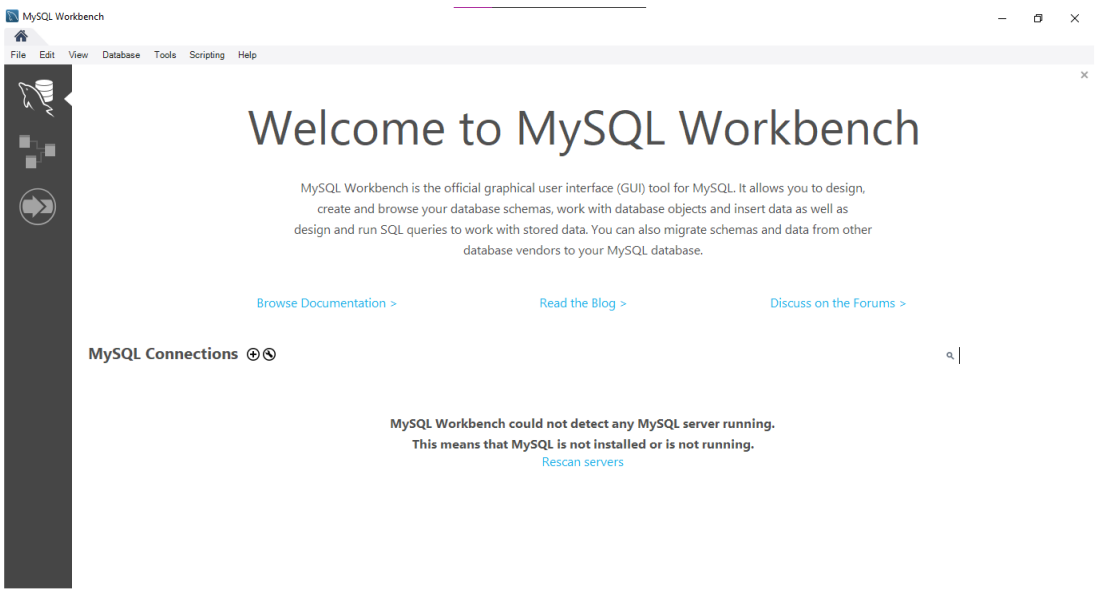


Figura 13 – Tela inicial MySQL Workbench

9. Para configurar o servidor basta preencher as lacunas de *Connection Name* e *HostName*, destacadas na Figura 13. Como o banco de dados será local, *HostName* deve ser preenchido com *localhost* e a senha será a chave já configurada no passo 7. Após preencher tudo, basta clicar em *Ok*.

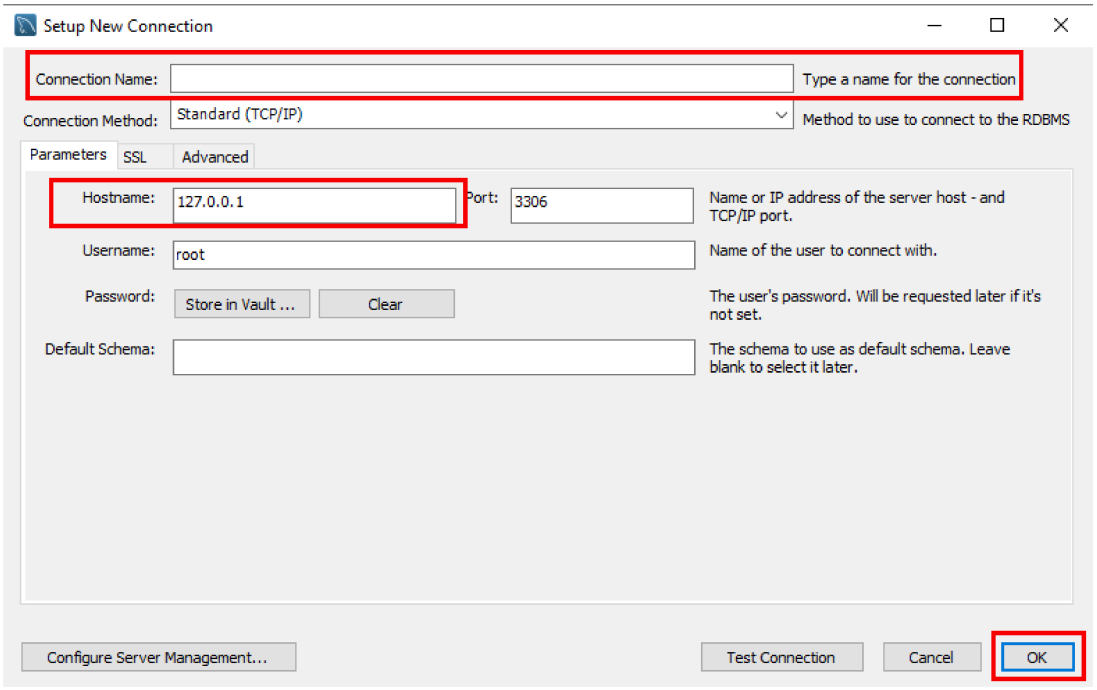


Figura 14 – Tela de configuração do servidor

10. Para acessar o servidor criado, basta, na página inicial do `MySQL Workbench` , clicar no servidor e preencher os dados de *login*.

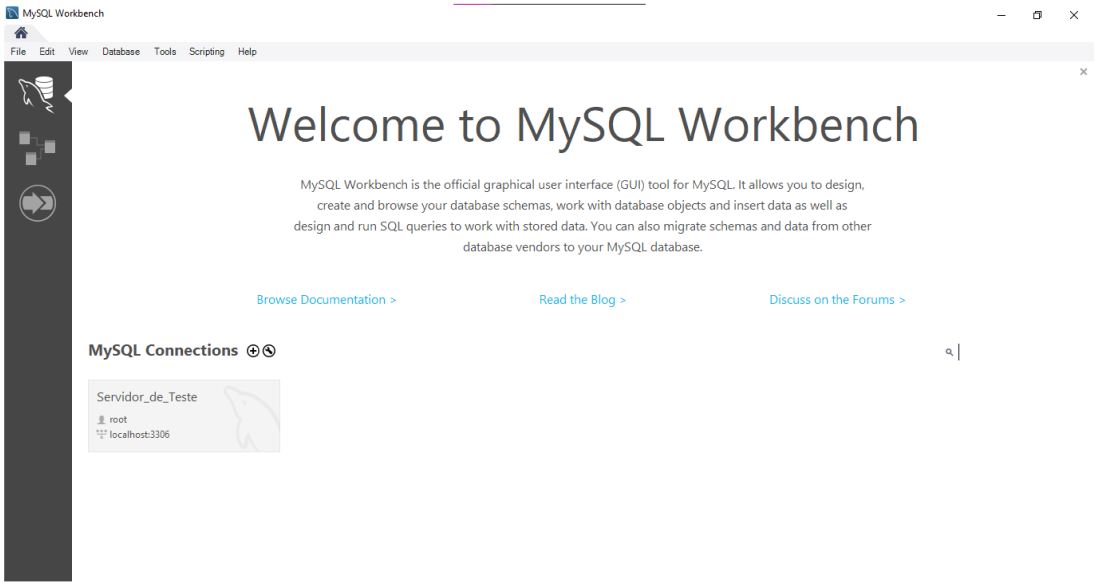


Figura 15 – Tela inicial MySQL Workbench com o servidor criado

11. A Figura 16 ilustra a tela de gerenciamento e criação de banco de dados do MySQL Workbench . É no *script* em que são escritas as consultas e manipulações.

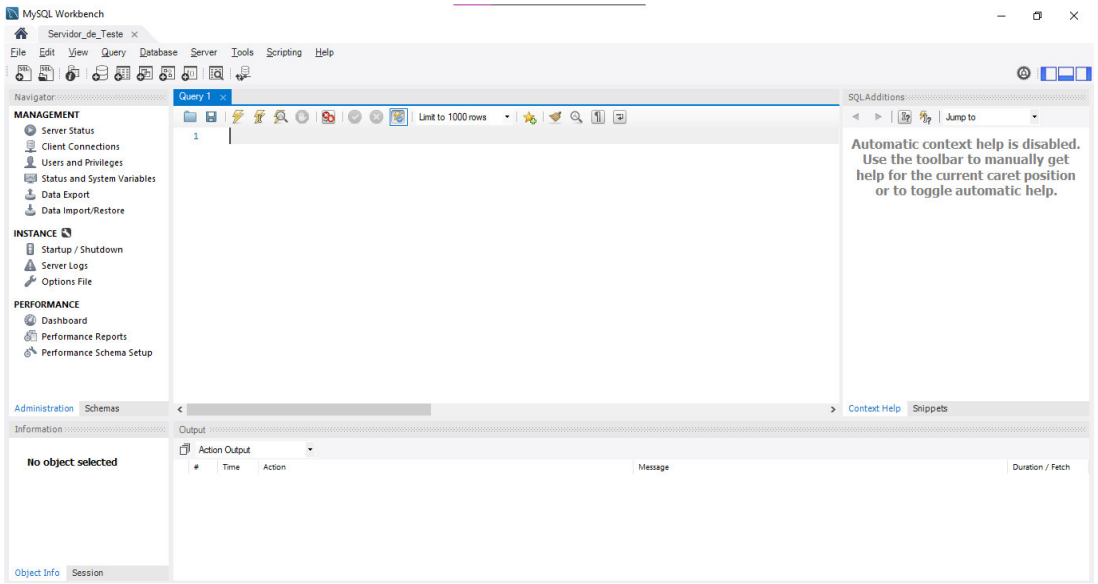


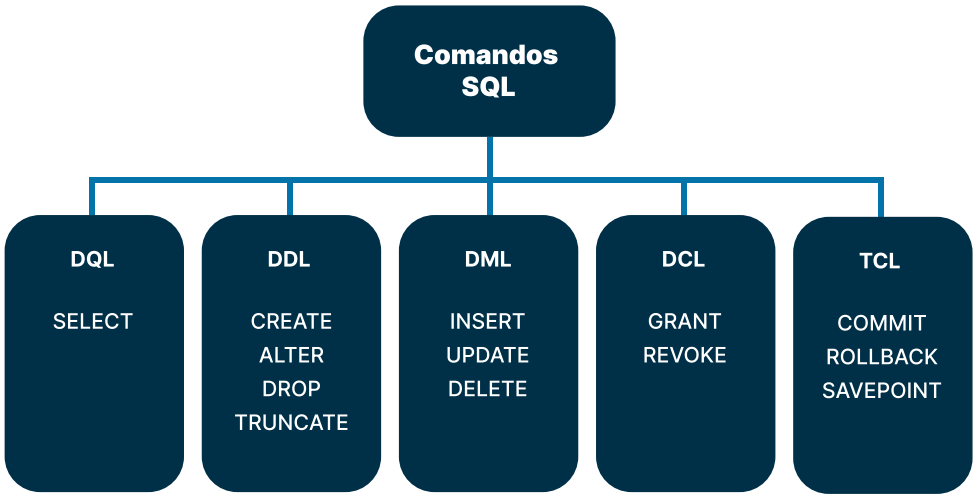
Figura 16 – Tela de gerenciamento de bancos de dados do MySQL Workbench

4.4.4 Linguagem de Consulta

Assim como outros SGBDs, o MySQL usa uma linguagem de consulta para realizar suas buscas, manipulações e configurações, e esta linguagem é o SQL. Como já citado na Seção 4.2, o SQL é uma linguagem de consulta projetada para bancos de dados relacionais e que

tem como base as operações da Álgebra Relacional. A linguagem é dividida em cinco blocos de comandos, conforme mostrado na Figura 17, no qual cada subdivisão possui um propósito, que será demonstrado na sequência.

Figura 17 – Comandos SQL.



Fonte: Adaptado de (DICODING, 2023)

- ***Data Query Language* (DQL)**: é o subconjunto da linguagem SQL responsável por definir comandos para consulta de dados armazenados em um banco de dados. Este conjunto inclui apenas o comando *SELECT*, que é usado para resgatar dados de tabelas e outras fontes de dados. O Listing 4.4.1 exemplifica a seleção da coluna "alunos"dentro da tabela "estudantes".

```
1    SELECT aluno FROM estudantes;
```

Listing 4.4.1 – Exemplo de uso do *SELECT*

Como o SQL é uma linguagem voltada para bancos de dados relacionais, existem algumas operações e cláusulas que aproveitam essas relações para acessar informações mais precisas a partir do *SELECT*. Uma dessas cláusulas é o *JOIN*, que permite combinar colunas ou entidades com base em parâmetros específicos. Existem quatro variações principais de *JOIN*, cada uma com seu propósito e resultado (MYSQL, 2023). A seguir será introduzida uma breve explicação sobre cada uma delas, e as Tabelas 2 e 3 representam duas entidades a fim de facilitar o entendimentos dos exemplos a seguir:

empresas			
IdEmpresa	**IdCliente**	**Nome**	**Cidade**
1	1	Empresa1	Hidrolândia
2	3	Empresa2	Sobral
3	3	Empresa3	Ipueiras
4	4	Empresa4	Santa Quitéria

Tabela 2 – Tabela que representa a entidade *empresas*.

clientes		
IdCliente	**NomeCliente**	**Telefone**
1	Jannielly	(11) 11111-1111
2	Klayver	(22) 22222-2222
3	Ana Lara	(33) 33333-3333
4	Maria Rita	(44) 44444-4444

Tabela 3 – Tabela que representa a entidade *clientes*.

– *INNER JOIN*: essa cláusula seleciona somente os registros que correspondem as condições passadas no bloco de comando, ou seja, devolve como resposta da consulta apenas as informações que possuem os valores comuns nas tabelas envolvidas. O Listing 4.4.2 representa a junção das informações das 2 e 3 apenas onde as linhas corresponderem a condição da diretiva *ON*. A Tabela 4 ilustra o resultado da operação.

```
1  SELECT *
2  FROM empresas
3  INNER JOIN clientes
4  ON empresas.IdCliente = clientes.IdCliente;
```

Listing 4.4.2 – Exemplo de uso do *INNER JOIN*

IdEmpresa	IdCliente	Nome	Cidade	IdCliente	NomeCliente	Telefone
1	1	Empresa1	Hidrolândia	1	Jannielly	(11) 11111-1111
2	3	Empresa2	Sobral	3	Ana Lara	(33) 33333-3333
3	3	Empresa3	Ipueiras	3	Ana Lara	(33) 33333-3333
4	4	Empresa4	Santa Quitéria	4	Maria Rita	(44) 44444-4444

Tabela 4 – Resultado do *INNER JOIN*

– *CROSS JOIN*: essa cláusula realiza o produto cartesiano entre tabelas, ou seja, dada duas tabelas o resultado será um produto cartesiano entre as tabelas envolvidas. O Listing 4.4.3 representa o produto cartesiano entre as tabelas *clientes* e *empresas*. A Tabela 5 mostra o resultado da operação.

```
SELECT *
FROM empresas
CROSS JOIN clientes;
```

Listing 4.4.3 – Exemplo de uso do *CROSS JOIN*

IdEmpresa	IdCliente	Nome	Cidade	IdCliente	NomeCliente	Telefone
1	1	Empresa1	Hidrolândia	1	Jannielly	(11) 11111-1111
2	3	Empresa2	Sobral	1	Jannielly	(11) 11111-1111
3	3	Empresa3	Ipueiras	1	Jannielly	(11) 11111-1111
4	4	Empresa4	Santa Quitéria	1	Jannielly	(11) 11111-1111
1	1	Empresa1	Hidrolândia	2	Klayver	(22) 22222-22222
2	3	Empresa2	Sobral	2	Klayver	(22) 22222-22222
3	3	Empresa3	Ipueiras	2	Klayver	(22) 22222-22222
4	4	Empresa4	Santa Quitéria	2	Klayver	(22) 22222-22222
1	1	Empresa1	Hidrolândia	3	Ana Lara	(33) 33333-3333
2	3	Empresa2	Sobral	3	Ana Lara	(33) 33333-3333
3	3	Empresa3	Ipueiras	3	Ana Lara	(33) 33333-3333
4	4	Empresa4	Santa Quitéria	3	Ana Lara	(33) 33333-3333
1	1	Empresa1	Hidrolândia	4	Maria Rita	(44) 44444-4444
2	3	Empresa2	Sobral	4	Maria Rita	(44) 44444-4444
3	3	Empresa3	Ipueiras	4	Maria Rita	(44) 44444-4444
4	4	Empresa4	Santa Quitéria	4	Maria Rita	(44) 44444-4444

Tabela 5 – Resultado do *CROSS JOIN*

- *LEFT JOIN*: essa cláusula retorna todas as linhas da tabela que estão à esquerda e as linhas da tabela à direita que obedecerem à diretiva ON. Caso não haja nenhuma correspondência na tabela da direita com quaisquer tuplas na tabela da esquerda, tais tuplas aparecerão no resultado com valores *NULL* para os atributos da tabela da direita. O Listing 4.4.4 representa o uso do *LEFT JOIN* que retorna todas as linhas da tabela *empresas* e as linhas da tabela *clientes* que obedecerem à condição na diretiva *ON*. A Tabela 6 mostra o resultado da operação.

```
SELECT *
FROM empresas
LEFT JOIN clientes
ON empresas.IdCliente = clientes.IdCliente;
```

Listing 4.4.4 – Exemplo de uso do *LEFT JOIN*

IdEmpresa	IdCliente	Nome	Cidade	IdCliente	NomeCliente	Telefone
1	1	Empresa1	Hidrolândia	1	Jannielly	(11) 11111-1111
2	3	Empresa2	Sobral	3	Ana Lara	(33) 33333-3333
3	3	Empresa3	Ipueiras	3	Ana Lara	(33) 33333-3333
4	4	Empresa4	Santa Quitéria	4	Maria Rita	(44) 44444-4444

Tabela 6 – Resultado do *LEFT JOIN*

– *RIGHT JOIN*: essa cláusula retorna todas as linhas da tabela que estão a direita e as linhas da tabela a esquerda que obedecerem à diretiva ON. Caso não haja nenhuma correspondência na tabela da esquerda com quaisquer tuplas na tabela da direita, tais tuplas aparecerão no resultado com valores *NULL* para os atributos da tabela da esquerda. . O Listing 4.4.5 representa o uso do *RIGHT JOIN* que retorna todas as linhas da tabela empresas e as linhas da tabela clientes que obedecerem à condição na diretiva *ON*. Como resultado da operação pode-se obter a Tabela 7.

```
1  SELECT *
2  FROM empresas
3  RIGHT JOIN clientes
4  ON empresas.IdCliente = clientes.IdCliente;
```

Listing 4.4.5 – Exemplo de uso do *RIGHT JOIN*

IdEmpresa	IdCliente	Nome	Endereço	IdCliente	NomeCliente	Telefone
1	1	Empresa1	Hidrolândia	1	Jannielly	(11) 11111-1111
NULL	NULL	NULL	NULL	2	Klayver	(22) 22222-22222
2	3	Empresa2	Sobral	3	Ana Lara	(33) 33333-3333
3	3	Empresa3	Ipueiras	3	Ana Lara	(33) 33333-3333
4	4	Empresa4	Santa Quitéria	4	Maria Rita	(44) 44444-4444

Tabela 7 – Resultado da operação com *RIGHT JOIN*

Além das operações comuns e das junções, existem recursos avançados que podem ser utilizados dentro de um sistema de banco de dados, como as *PROCEDURES* e os *TRIGGERS*. As *PROCEDURES* são conjuntos de comandos que são executados de forma sequencial. Caso uma linha de comando não seja executada corretamente, as operações anteriores não são revertidas e a execução da *PROCEDURES* é encerrada. O Listing 4.4.6 e representa a estrutura de uma *PROCEDURE*.

```
1  CREATE PROCEDURE nome_dado_a_procedure
2  BEGIN
3      -- comandos SQL aqui
4  END;
```

Listing 4.4.6 – Exemplo de uso da *PROCEDURES*

Os *TRIGGERS* são objetos que estão diretamente relacionados a uma ou mais entidades. Estes objetos servem como "ouvintes" de um determinado evento e realizam alguma operação quando identificam o evento especificado. O Listing 4.4.7 representa uma *TRIGGER* que observa quando o evento de *INSERT* na tabela *clientes* é executado. Quando tal evento acontecer, a ação de adicionar este novo cliente a tabela *clienteEmpresa* é executada.

```
1  CREATE TRIGGER adc_cliente_empresa
2  AFTER INSERT ON clientes
3  FOR EACH ROW
4  BEGIN
5      INSERT INTO clienteEmpresa (id_cliente, nome_cliente)
6      VALUES (NEW.id_cliente, NEW.nome_cliente);
7  END;
```

Listing 4.4.7 – Exemplo de uso da *TRIGGER*

- ***Data Definition Language* (DDL)**: é o subconjunto da linguagem SQL responsável por fazer a definição do banco de dados. Este conjunto inclui quatro comandos: *CREATE, ALTER, DROP e TRUNCATE.*
  - *CREATE*: usado para criar estruturas dentro de banco de dados. O Listing 4.4.8 representa a criação de uma tabela chamada "estudantes".

```
1  CREATE TABLE Estudantes (
2  ID INT PRIMARY KEY AUTO_INCREMENT,
3  Nome VARCHAR(255),
4  Idade INT,
5  Endereco VARCHAR(255);
```

Listing 4.4.8 – Exemplo de uso do *CREATE*

  - *ALTER*: usado para alterar estruturas dentro de banco de dados. O Listing 4.4.9 representa uma alteração na tabela "estudantes". Nesse caso está sendo adicionada

uma nova coluna chamada "telefone".

```
1  ALTER TABLE estudantes ADD telefone VARCHAR(20);
```

Listing 4.4.9 – Exemplo de uso do *ALTER*

- *DROP*: usado para apagar estruturas dentro de banco de dados. O Listing 4.4.10 representa a deleção da estrutura da tabela "estudantes".

```
1  DROP TABLE estudantes;
```

Listing 4.4.10 – Exemplo de uso do *DROP*

- *TRUNCATE*: usado para excluir os dados de estruturas dentro de banco de dados de forma permanente. O Listing 4.4.11 representa uma alteração na tabela estudantes. Nesse caso todos os dados da tabela "estudantes"estão sendo deletados.

```
1  TRUNCATE TABLE estudantes;
```

Listing 4.4.11 – Exemplo de uso do *TRUNCATE*

- ***Transaction Manipulation Language* (DML)**: é o subconjunto da linguagem SQL responsável por manipular as informações de banco de dados. Este conjunto inclui três comandos: *INSERT, UPDATE e DELETE.*
  - *INSERT*: usado para inserir novos dados em uma ou mais tabelas. O Listing 4.4.12 demonstra a inserção de um novo dado a tabela de "estudantes".

```
1  INSERT INTO estudantes (matricula, nome_completo, telefone)
2  VALUES (1234, Jannielly, 8899999999);
```

Listing 4.4.12 – Exemplo de uso do *INSERT*

  - *UPDATE*: usado para atualizar dados em uma ou mais tabelas. O Listing 4.4.13 representa a atualização do atributo "nome" na tabela de "estudantes" com base na condição WHERE.

```
1  UPDATE estudantes
2  SET nome = 'Francisca Jannielly Garcia da Costa'
3  WHERE matricula = 66666;
```

Listing 4.4.13 – Exemplo de uso do *UPDATE*

- *DELETE*: usado para deletar dados de uma estrutura. Diferentemente do *DROP* que apaga a estrutura e seus dados ou do *TRUNCATE* que deleta os dados de maneira permanente, o comando *DELETE*, permite a recuperação dos dados. O Listing 4.4.14 representa a exclusão dos dados da tabela "estudantes".

```
1  DELETE FROM estudantes;
```

Listing 4.4.14 – Exemplo de uso do *DELETE*

- ***Data Control Language* (DCL)**: é o subconjunto da linguagem SQL relacionado a segurança do banco de dados. Este conjunto inclui 2 comandos: G*RANT e REVOKE*.
  - *GRANT*: usado para conceder permissão de acesso ou manipulação a um objeto do banco de dados para um usuário específico. O Listing 4.4.15 representa a permissão para o usuário "Priscila"consultar a tabela "estudantes".

```
1  GRANT SELECT ON estudantes TO Priscila;
```

Listing 4.4.15 – Exemplo de uso do *GRANT*

  - *REVOKE*: usado para revogar a permissão de acesso ou manipulação a um objeto do banco de dados para um usuário. O Listing 4.4.16 representa a permissão para o usuário "Jander"consultar a tabela estudantes.

```
1  REVOKE SELECT ON estudantes FROM Jander;
```

Listing 4.4.16 – Exemplo de uso do *REVOKE*

- ***Transaction Control Language* (TCL)**: é o subconjunto da linguagem SQL relacionado a transações em um banco de dados. Este conjunto inclui três comandos: *COMMIT, ROLLBACK e SAVEPOINT*.
  - *COMMIT*: usado para certificar que as alterações foram realizadas em uma transação, tornando as mudanças persistentes no banco de dados. O Listing 4.4.17 representa

a transação de deletar o estudante com a matrícula 12345, neste caso, o *COMMIT* garante que as alterações da transação foram realizadas.

```
BEGIN TRANSACTION
    DELETE FROM estudantes
    WHERE matricula = 12345;
COMMIT;
```

Listing 4.4.17 – Exemplo de uso do *COMMIT*

– *ROLLBACK*: usado para reverter uma transação, ou seja, ao usar *ROLLBACK* é recuperado o estado do banco de dados até o último *COMMIT* que foi realizado. O Listing 4.4.18 representa o *ROLLBACK* para reverter uma determinada transação dado que houve um erro.

```
-- suponha que aqui houve um erro
ROLLBACK;
```

Listing 4.4.18 – Exemplo de uso do *ROLLBACK*

– *SAVEPOINT*: usado para definir um ponto na transação em que os dados e alterações são salvas. Este comando é bem útil para garantir o versionamento dos dados. O Listing 4.4.19 representa a configuração de um ponto de salvamento dentro de uma transação, ou seja, tudo feito antes está armazenado e denominado de "*ponto1*".

```
BEGIN TRANSACTION;
    GRANT SELECT ON estudantes TO Priscila;
    SAVEPOINT ponto1;
    DELETE FROM clientes WHERE id = 2;
COMMIT
```

Listing 4.4.19 – Exemplo de uso do *SAVEPOINT*

4.5 Uma visão geral sobre MongoDB

`MongoDB` é um SGBD orientado a documentos e é o mais utilizado quando se trata de banco de dados não relacionais para aplicações Web (DBENGINES, 2023). Criado em 2007 pela então empresa *10gen*, hoje chamada de `MongoDB`, teve como primeira versão o modelo de plataforma em nuvem para banco de dados, e sua idealização deu-se pela necessidade de

trabalhar com os diferentes tipos de dados que a empresa lidava (MONGODB, 2023a). Tendo o banco de dados relacional como exemplo de comparação, podemos entender que os dados são armazenados conforme mostra a Tabela 8:

Banco de Dados Relacional	**Banco de Dados Não Relacional**
Base de dados	Base de dados
Tabela	Coleção
Registro	Documento
Coluna	Atributo

Tabela 8 – Comparação Banco de dados relacionais vs. Banco de dados não relacionais (MongoDB)

O documento é uma estrutura formada por pares de campo e valor, sendo bem similar a objetos JSON. Seus campos podem ser preenchidos por diversos tipos de dados, que podem ser desde um simples valor inteiro a um vetor de outros documentos. O Listing 4.5.1 apresenta um exemplo de um arquivo no formato especificado pelo MongoDB.

```
1   {
2       "nome": "Francisca Jannielly Garcia da Costa",
3       "idade":22,
4       "escolaridade": "Ensino Superior incompleto",
5       "caracteristicasFisicas":{
6           "altura": 159,
7           "peso":65,
8           "raca":"Preta"
9       }
10  }
```

Listing 4.5.1 – Representação de um documento em MongoDB

O formato JSON oferece vantagens para o uso em documentos MongoDB. A primeira vantagem é o fato do JSON ser o formato nativo para armazenamento de dados na maioria das linguagens de programação. Além disso, essa estrutura possibilita que algumas operações e técnicas sejam aplicadas, como junções e polimorfismo fluente (MONGODB, 2023a).

Diferente do MySQL, que usa SQL para realizar consultas e operações com os dados do banco, o MongoDB possui sua própria linguagem de consulta baseada na linguagem JavaScript. Embora possuam similaridades, em alguns aspectos a linguagem de consulta do MongoDB e a SQL se diferenciam totalmente, já que no MongoDB a sintaxe é baseada em objetos JSON e não

em Tabelas.

4.5.1 Ferramenta de Gerenciamento do MongoDB

O MongoDB *Compass* é um *software* com interface gráfica que permite realizar operações e configurações em bancos de dados e servidores MongoDB. Este *software* é completamente gratuito e está disponível para execução em macOS, Windows e distribuições Linux. Assim como outros softwares de gerenciamento de banco de dados, com o MongoDB *Compass* é possível criar e gerenciar seu banco de dados de maneira simplificada, esteja ele em sua máquina local ou em servidores na nuvem (MONGODB, 2023a).

Caso possua um banco de dados do tipo relacional, com o MongoDB *Compass* é possível realizar a transferência de seu banco de dados para um modelo não relacional do tipo MongoDB, para isto, basta exportar seus dados em arquivos do tipo JSON e/ou *Comma-separated values* (CSV) e importar para o MongoDB *Compass* realizar a configuração. Com os dados já importados o banco de dados já está pronto para ser utilizado, sendo possível realizar buscas configuradas, inserções, atualizações e deleções.

O pipeline de agregação do MongoDB possibilita a realização de operações de agregação e de consultas avançadas. Estes recursos permitem extrair e manipular os dados armazenados no banco de dados de maneira eficiente. Para realizar as consultas e agregações, o MongoDB *Compass* possibilita que o usuário do MongoDB possa executar seus comandos diretamente em um ambiente JavaScript interativo, conhecido tecnicamente como (*shell*).

4.5.2 Arquitetura do MongoDB

A arquitetura do MongoDB também é baseada na arquitetura cliente-servidor. Isso significa que há uma separação entre o cliente (usuários), que solicita os dados, e o servidor (MongoDB), que é responsável por gerenciar, processar e armazenar esses dados. Para entender mais sobre o modelo de arquitetura cliente-servidor, basta acessar a subseção 2.4.2.

4.5.3 Instalação e Configuração

Com intuito de demonstrar como o MongoDB pode ser utilizado, nesta seção será dada uma sequência de passos para a instalação e configuração do mesmo. Como já dito anteriormente, o MongoDB é um SGBD multiplataforma, ou seja, pode ser instalado em qualquer

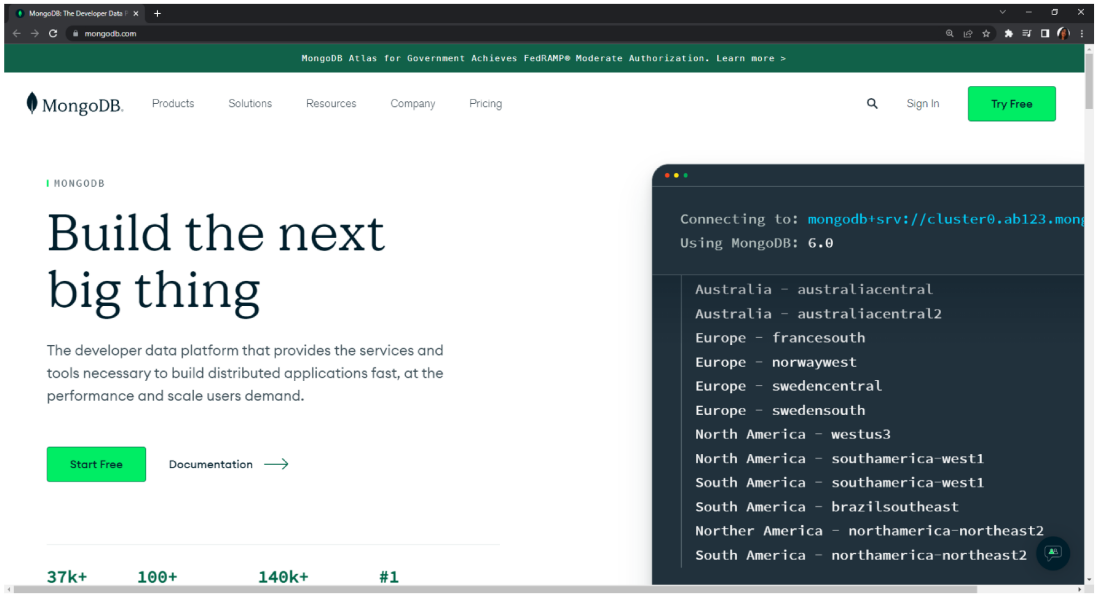


Figura 18 – Página inicial MongoDB

SO, seja Windows, Linux ou MacOS (MONGODB, 2023a). Para esta seção será usado o MongoDB *Community Server*, será realizada a instalação do MongoDB *Compass* e configuração do seu primeiro servidor numa máquina Windows.

1. Para fazer o *download* do MongoDB é preciso acessar o link [4] do site oficial. Ao acessar a página de *download* será possível escolher o SO em que ocorrerá a instalação. Após a escolha, basta clicar em *download* para baixar o MongoDB. A Figura 18 representa a tela inicial do site oficial do MongoDB.
2. Para ir para a página de *download* do MongoDB é necessário acessar, no menu superior, a aba *products* e clicar em *Community Server*, conforme mostra a Figura 19.

[4] *Página de download do MongoDB*

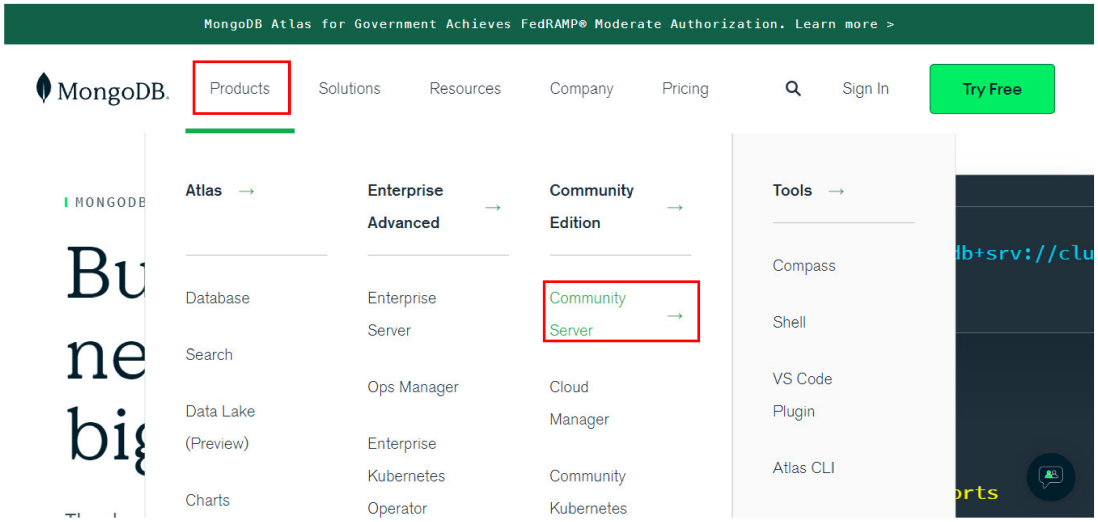


Figura 19 – Página inicial do MongoDB com ênfase no menu

3. A Figura 20 representa a página de *download* do MongoDB. No final da página, é necessário selecionar a versão do MongoDB, o SO e o formato do instalador. Após preencher os três campos, basta clicar em *download*.

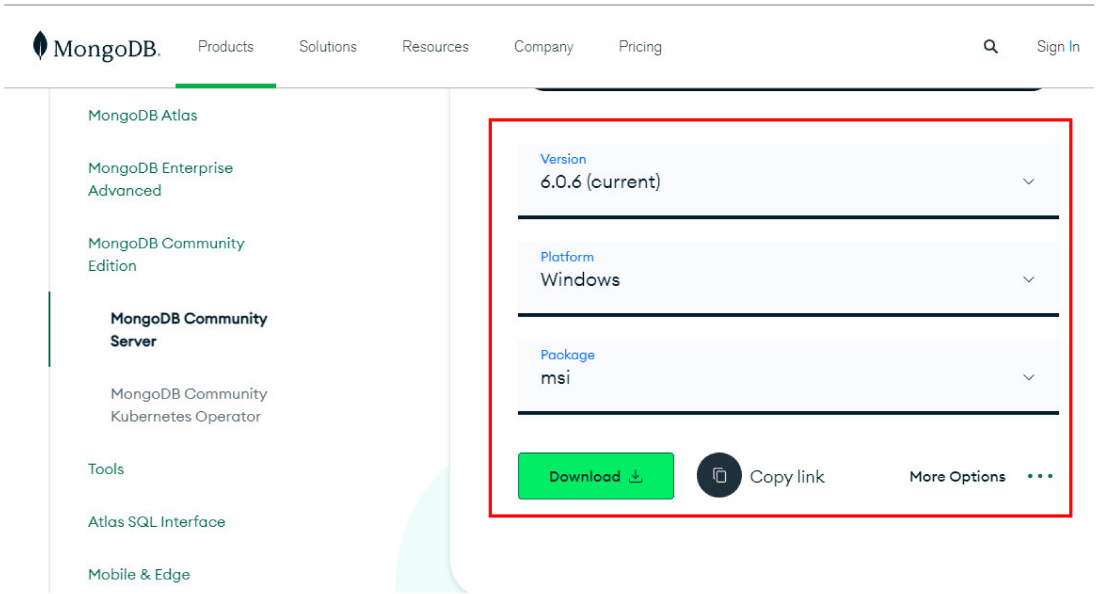


Figura 20 – Página de download do MongoDB Server

4. Quando o *download* finalizar, basta clicar no instalador. A Figura 21 mostra a primeira tela do processo de instalação. Nesta, basta clicar em *Next*.

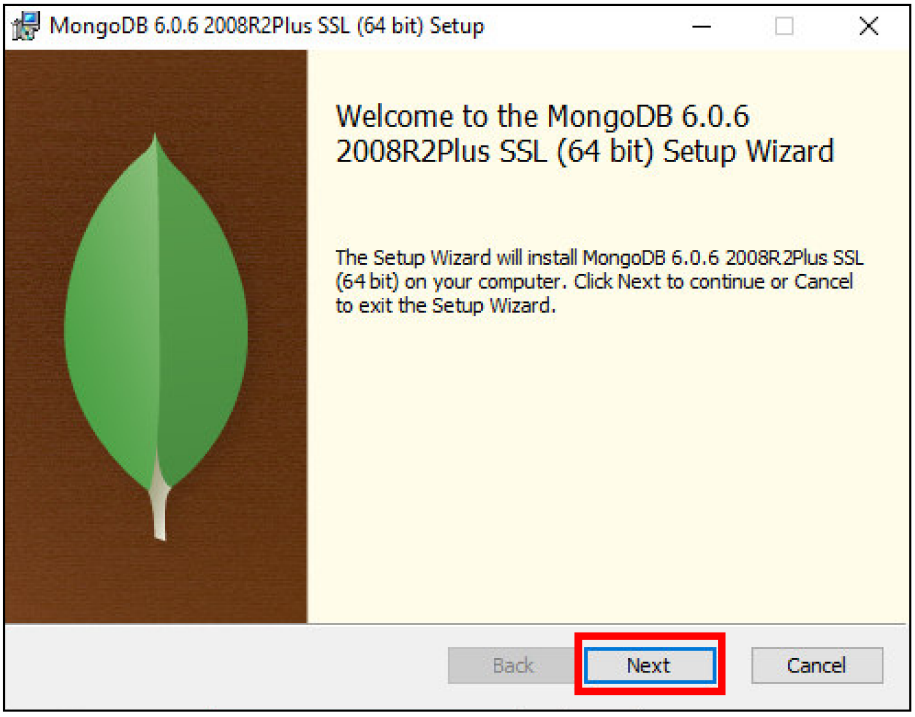


Figura 21 – Primeiro passo da instalação do `MongoDB`

5. A Figura 22 representa a segunda tela do processo de instalação do `MongoDB`. Nesta tela, estão os termos de instalação. Para aceitar, basta marcar a caixinha *I accept the terms in the license Agreement*.

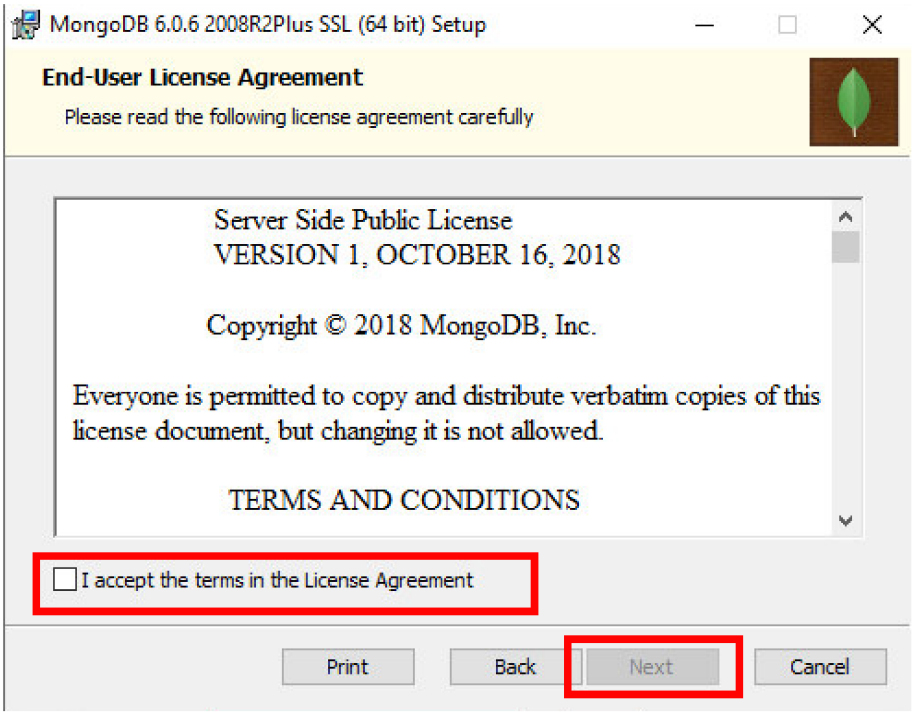


Figura 22 – Tela de termos e licenças do `MongoDB`

6. A Figura 23 mostra a etapa de escolha do tipo de configuração. Nesta tela, haverá duas opções: a versão completa, que irá instalar todas as ferramentas disponíveis no `MongoDB`; e a versão customizada, na qual o usuário pode baixar apenas as ferramentas que necessita. Para este trabalho, a escolha será a versão completa.

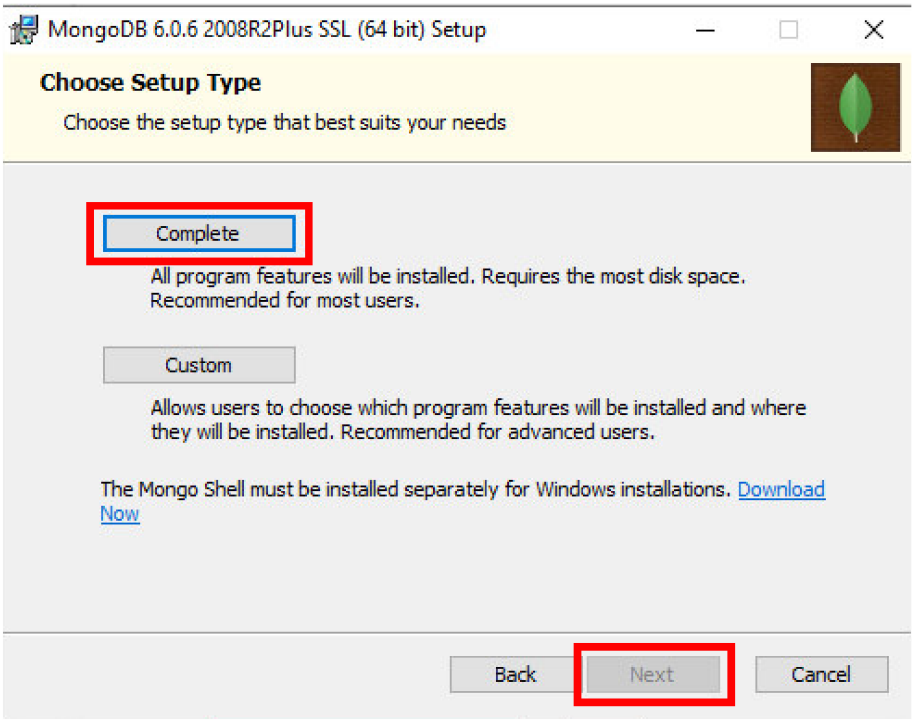


Figura 23 – Tela de escolha do tipo de instalação do `MongoDB Server`

7. A Figura 24 representa a tela onde ocorre as configurações do serviço Mongo, como adição de usuário, configuração de rede e local de instalação. Nesta etapa, basta clicar em *Next*.

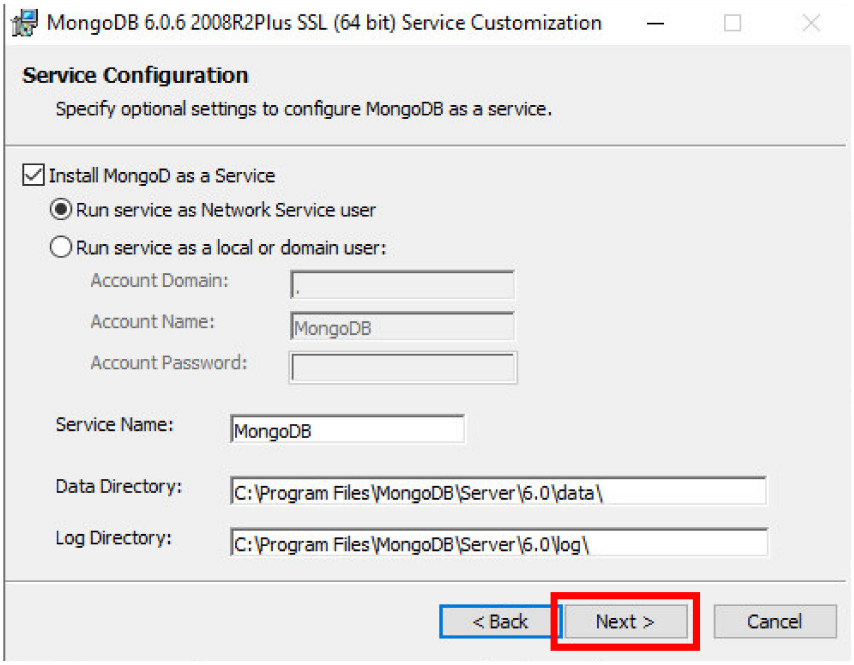


Figura 24 – Tela de configuração de rede e usuário para o `MongoDB Server`

8. A etapa mostrada na Figura 25 é de bastante importância, pois é nela que será confirmada a instalação do `MongoDB` *Compass*, ferramenta já introduzida nesta seção. Após confirmar a instalação, basta clicar em *Next* e, em seguida, em *Finish*.

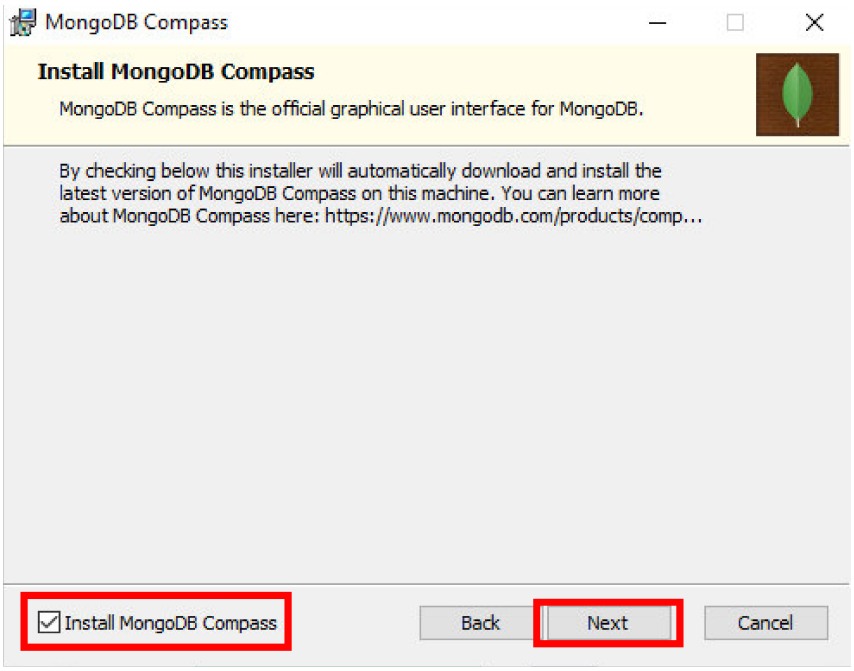


Figura 25 – Tela de instalação do `MongoDB Compass`

9. A página inicial do `MongoDB` *Compass* dá acesso ao login no servidor ao qual está o banco

de dados. Para casos de uso de um servidor local, o caminho será: 'MongoDB://localhost:27017/'. Para conectar, basta clicar em *Connect*. A Figura 26 representa a tela de início do MongoDB *Compass*.

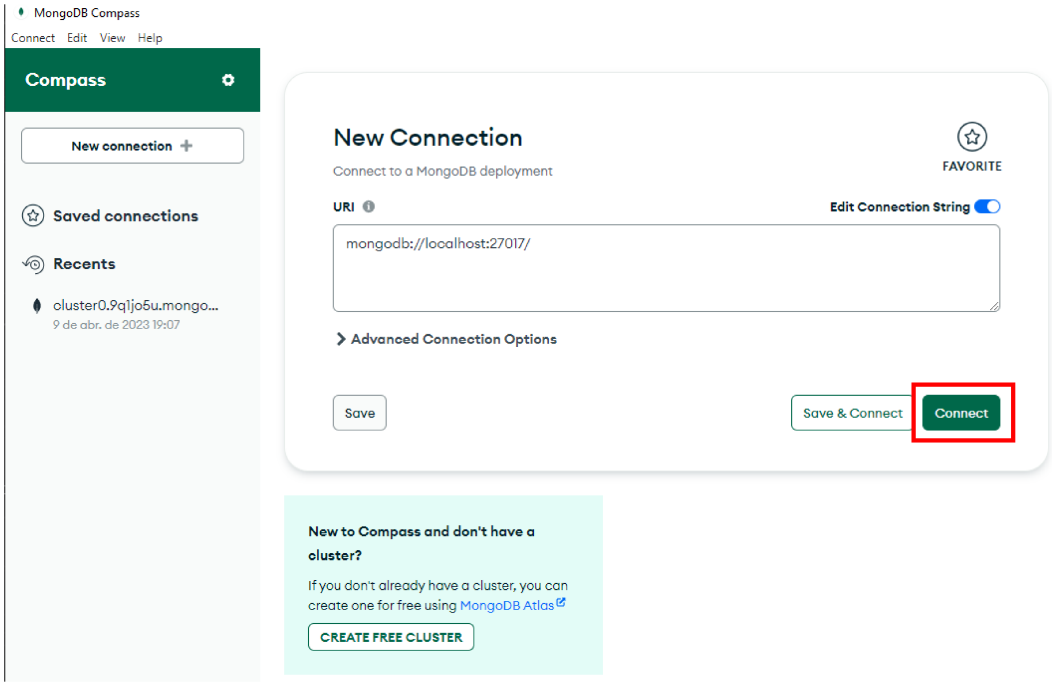


Figura 26 – Tela inicial do MongoDB Compass

10. A Figura 27 mostra o ambiente do servidor conectado. Nesta página é possível criar bancos de dados, realizar consultas e fazer configurações de performance. A opção *create database* possibilita a criação de um banco de dados MongoDB.

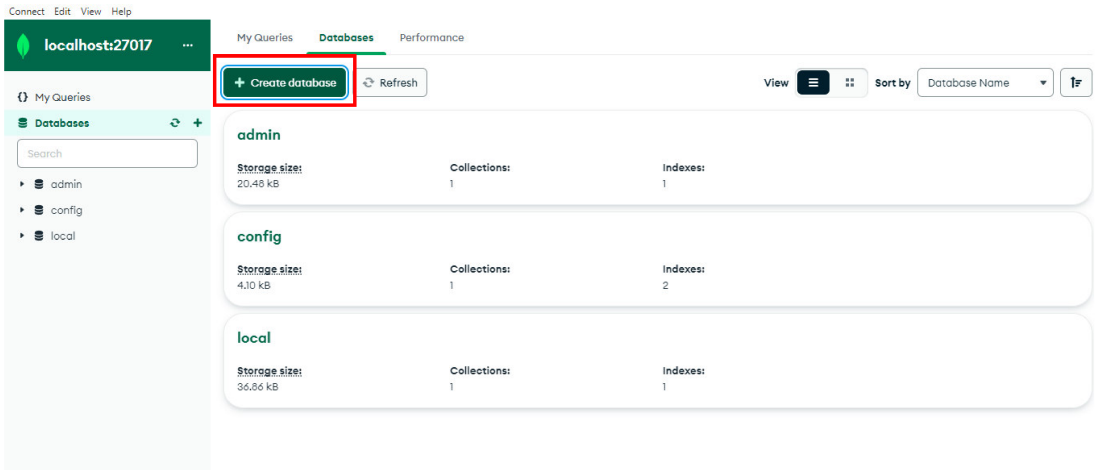


Figura 27 – Tela de gerenciamento dos bancos de dados no MongoDB Compass

11. A Figura 28 mostra a tela de criação de um banco de dados usando o MongoDB *Compass*.

Para criar, basta preencher os campos de *Database Name* e *Collection Name* e, em seguida, clicar em *Create Database*.

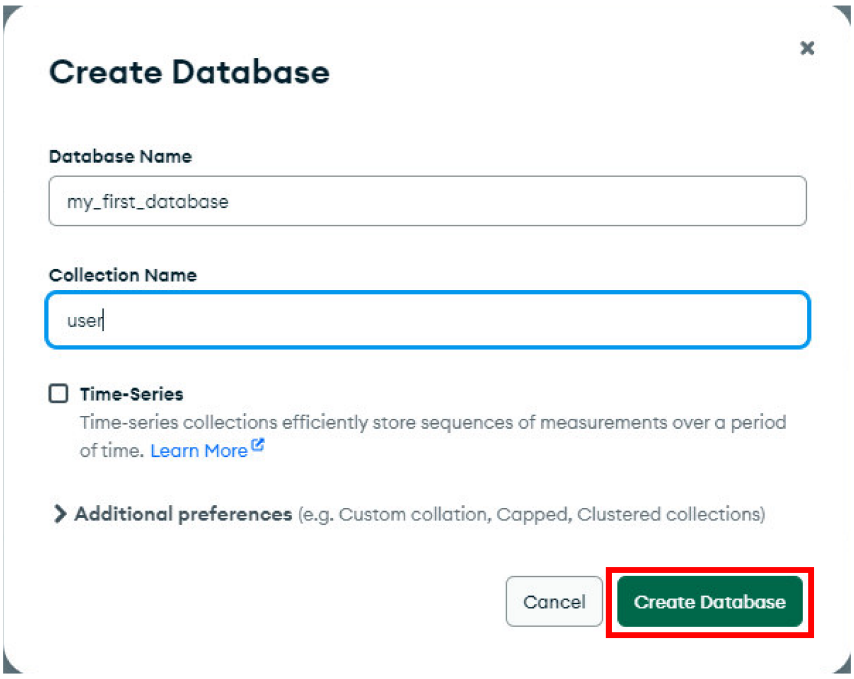


Figura 28 – Tela de criação de um banco de dados no `MongoDB Compass`

12. A tela representada pela Figura 29 dá acesso ao banco de dados criado. Nessa tela, é possível importar, criar, manipular e consultar dados.

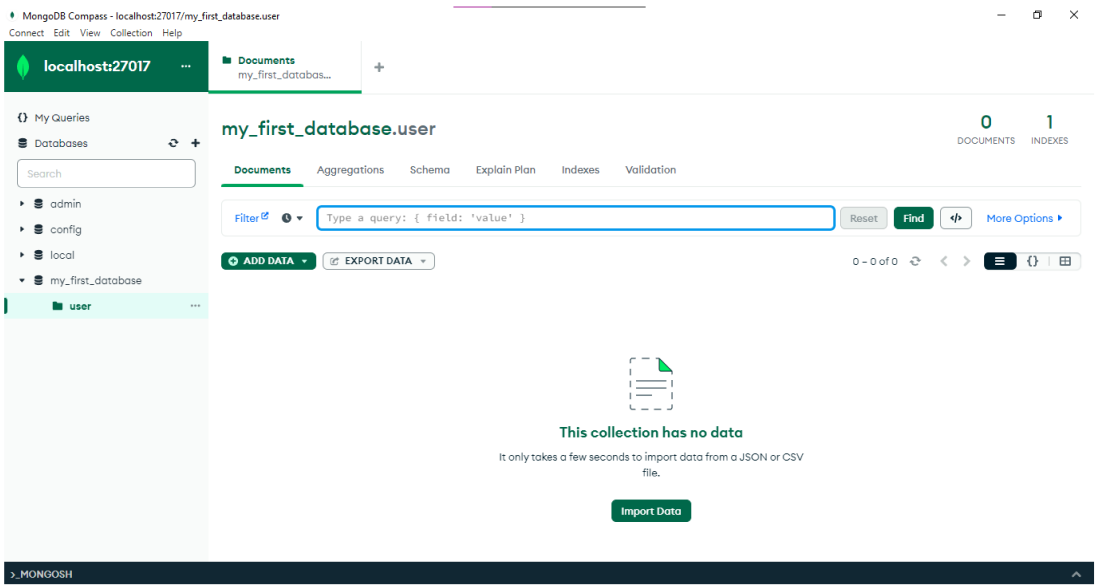


Figura 29 – Tela de gerenciamento do banco de dados criado

4.5.4 Linguagem de Consulta

O MongoDB tem suporte a diversas linguagens de programação, como C Sharp, JavaScript e C++, mas a linguagem que é comumente utilizada ao utilizar este software é uma linguagem de consulta própria, que é baseada em JavaScript. Como SQL já foi introduzida na Seção 2.4, nesta seção será apresentado uma relação entre as duas linguagens a fim de entender como funcionam as consultas e cláusulas no MongoDB.

- O banco de dados utilizado para as seguintes explicações será denominado "db". Para criar uma nova coleção, que seria como uma tabela em um banco de dados relacional, usa-se a função *createCollection()*. O Listing 2.5.2 representa a criação de uma coleção de users.

```
1   db.createCollection(users)
```

Listing 4.5.2 – Exemplo de uso do comando *createCollection*

- Para mostrar todas as coleções associadas ao banco de dados usa-se o comando *show*. O Listing 2.5.3 mostra todas as coleções associadas aos *users*.

```
1   show collections
```

Listing 4.5.3 – Exemplo de uso do comando *show*

- Para adicionar um documento a coleção *users*, que seria como adicionar uma linha em uma tabela, usa-se a função *insert()*. O Listing 2.5.4 representa a adição dos atributos 'nome', 'idade' e 'sexo' à tabela *users*.

```
1   db.users.insert({
2       "name": "Jannielly",
3       "idade": 22,
4       "sexo": "Feminino"
5   })
```

Listing 4.5.4 – Exemplo de uso do comando *insert*

- A função *insertMany()* realiza a inserção de mais de um documento por vez. O Listing 2.5.5 representa a adição múltipla dos atributos 'nome', 'idade' e 'sexo' à tabela *users*.

```
db.users.insertMany({
    "name": "Jannielly",
    "idade": 22,
    "sexo": "Feminino"
},
{
    "name": "Ana Lara",
    "idade": 20,
    "sexo": "Feminino"
})
```

Listing 4.5.5 – Exemplo de uso do comando *insertMany*

- O MongoDB gera automaticamente uma chave de identificação para cada novo item adicionado ao banco de dados. Para deleção de um documento pode-se utilizar a função *remove()*. O Listing 2.5.6 representa a remoção de um documento pelo seu identificador.

```
db.users.remove({_id:xxxx1111})
```

Listing 4.5.6 – Exemplo de uso do comando *remove()*

- Para edição de um documento usa-se a função *update()*. O Listing 2.5.6 representa a alteração do atributo 'idade' de um usuário da coleção *users*.

```
db.users.update({_id:xxxx1111},
{$set: {idade: 23})
```

Listing 4.5.7 – Exemplo de uso do comando *update*

- Para filtrar usa-se a diretiva *find()*. O Listing 2.5.8 representa a seleção de todos os documentos da coleção *users*.

```
db.users.find()
```

Listing 4.5.8 – Exemplo de uso do comando *find*

- Para filtrar usa-se a diretiva *find()* com uma condição. O Listing 2.5.9 representa a seleção de todos os documentos da coleção *users* com idade 22.

```
db.users.find({"idade": 22})
```

Listing 4.5.9 – Exemplo de uso do comando *find* com uma condição

- Para deletar uma coleção usa-se a função *drop()*. O Listing 2.5.10 representa a deleção da coleção *users*.

```
db.users.drop()
```

Listing 4.5.10 – Exemplo de uso do comando *drop*

- No MongoDB não existe nenhuma cláusula que faça exatamente o quê o comando *join* faz, mas existe uma função que pode realizar uma operação similar, que é o comando *$lookup*. Supondo que existam duas coleções, *users* e *products*:

```
{
    "_id": ObjectId("z8z8z8z8z8z8"),
    "username": "Jannielly",
    "age": 22
    },
{
   "_id": ObjectId("a9a9a9a9a9a"),
    "username": "Ana Lara",
    "age": 20
}
```

Listing 4.5.11 – Coleção *users*

```
{
    "_id": ObjectId("sadhsa"),
    "user_id": ObjectId("z8z8z8z8z8z8"),
    "Produto_Name": "Computador"
},
{
    "_id": ObjectId("60a02e063f457f2ec02c1e12"),
    "user_id": ObjectId("a9a9a9a9a9a"),
    "Produto_Name": "Smartphone"
}
```

Listing 4.5.12 – Coleção *products*

Para fazer a junção das duas coleções, será usado o comando *$lookup*. O Listing 2.5.13

representa a junção das coleções *users* e *products*.

```
   db.products.aggregate([
  {
    $lookup: {
      from: "users",
      localField: "user_id",
      foreignField: "_id",
      as: "user_info"
    }
  }
])
```

Listing 4.5.13 – Exemplo de uso do comando *aggregate*

O resultado da operação será a junção das duas coleções com base no identificador.

```
    {
        "_id": ObjectId("sadhsa"),
        "user_id": ObjectId("z8z8z8z8z8z8"),
        "Produto_Name": "Computador",
        user_info: [
        {
            "_id": ObjectId("z8z8z8z8z8z8"),
            "username": "Jannielly",
            "age": 22
        }
        ]
    },
   {
        "_id": ObjectId("sadhsa"),
        "user_id": ObjectId("z8z8z8z8z8z8"),
        "Produto_Name": "Computador",
        user_info: [
        {
            "_id": ObjectId("a9a9a9a9a9a"),
            "username": "Ana Lara",
            "age": 20
        }
        ]
    }
```

Listing 4.5.14 – *aggregate*, *users* e *products*

Existem outras funções e variações das funções já apresentadas, cada uma atendendo a uma

determinada especificidade. Para consultar outros comandos ou cláusulas é recomendado acessar a documentação do `MongoDB` no site oficial [5].

4.6 Mapeamento objeto-relacional

Object Relational Mapping (ORM) é uma técnica que possibilita a comunicação entre a camada lógica e o banco de dados utilizado na aplicação. Essa técnica realiza o mapeamento e abstração de complexidades das operações de banco de dados e facilita a manipulação, e persistência dos dados através de uma espécie de tradução entre a linguagem de programação e a estrutura da linguagem de consulta usada no banco de dados. Além de realizar o mapeamento entre objetos e tabelas, existem recursos que fazem do ORM uma técnica tão utilizada. Com ele, podemos gerar esquemas de estruturas automaticamente, gerenciar modelos de dados, acessar e gerenciar cache de objetos e ter controle sobre as transações efetuadas, todos esses recursos sem a necessidade de geração de códigos em linguagens de consulta, como SQL (CHEN TSE-HSUN E SHANG, 2016).

Apesar das vantagens, também existem desvanteagens em se utilizar um ORM em sua aplicação, tais como perda de desempenho e performance, bloqueios gerados por plataformas de gerenciamento de bancos de dados, além do aumento da complexidade da aplicação. Existe uma variedade de ORM, geralmente um para cada linguagem de programação, e cada qual com suas especificidades, como, por exemplo, o `Entity Framework` usado no `C Sharp/.NET`.

4.6.1 Entity Framework

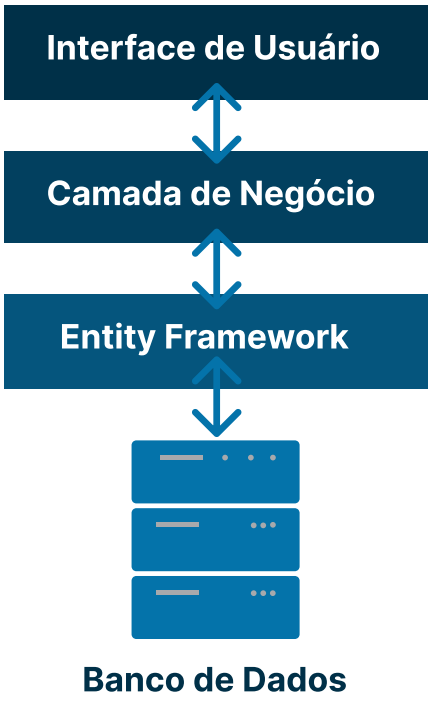
`Entity Framework` é um ORM com suporte da `Microsoft` usado em aplicações Web do tipo `ASP.NET`, sendo um componente que fornece ferramentas e recursos para a realização do mapeamento entre aplicação e bancos de dados relacionais. Com o `Entity Framework` é possível realizar operações que vão desde um CRUD simples, até operações mais complexas, como *join* entre tabelas, rastreamento de mudanças, gerenciamento e otimização de consultas e consultas *Language Integrated Query* (LINQ).

O `Entity Framework` possui suporte ao chamado *"code first"*. Essa técnica consiste em realizar a criação, via código, dos modelos (entidades) e todas as suas configurações. O *code first* permite que os desenvolvedores foquem seu tempo na lógica do desenvolvimento enquanto o `Entity Framework`, após os modelos criados e lógica concluída, possa realizar as

[5] *Documentação do*

configurações de banco de dados. Assim como todos os ORMs, o `Entity Framework` possui comandos próprios para realizar a criação de instâncias, configurações e realizar consultas. Cada mudança que é feita no banco de dados, seja estrutural ou atualização de dados, é registrada pelas *migrations*. *Migration* ou migração, é uma abordagem que tem intuito de controlar as versões e moderar as mudanças realizadas no banco de dados de maneira a facilitar o trabalho em grupo e evitar possíveis perdas ou falhas.A Figura 30 representa o fluxo de comunicaçaõ em aplicações que usam ORM.

Figura 30 – Comunicação usando `Entity Framework`



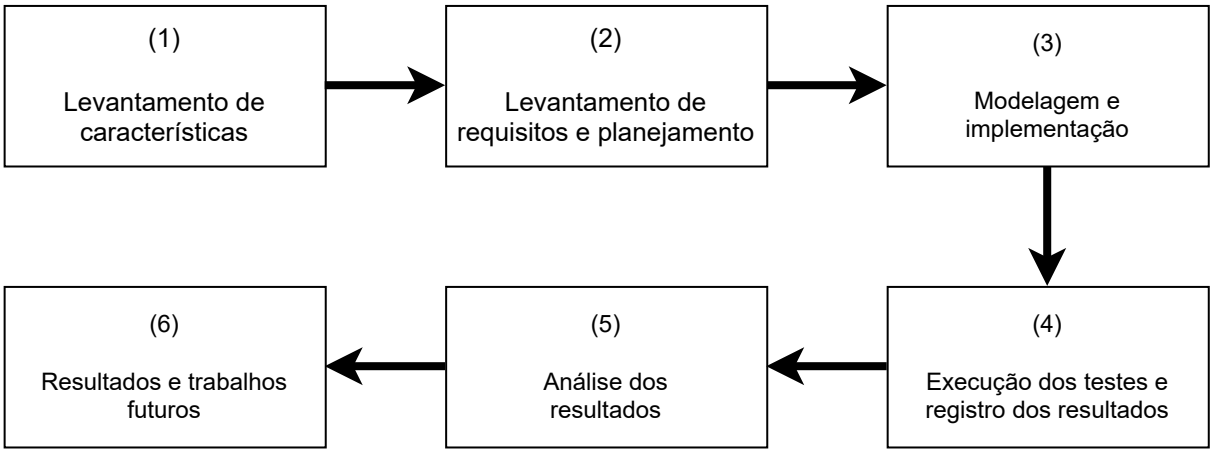
Fonte: Adaptado de (MATEUS, 2023)

Apesar de ser uma ferramenta que possui diversas funcionalidades, existem ainda algumas limitações diminuem sua popularidade do mesmo, tais como o fato de não ter suporte para *multithread* em um único contexto e sua perda de desempenho em aplicações que exigem uma complexidade mais alta nas consultas e aplicações (MICROSOFT, 2023a).

5 METODOLOGIA

Nesse capítulo será abordada a metodologia utilizada para realização do presente trabalho. Como citado, o objetivo principal deste trabalho é realizar uma análise comparativa, tanto estática quanto dinâmica, dos SGBDs mais populares no contexto de aplicações Web, sendo eles: `MySQL e MongoDB`. A Figura 31 apresenta graficamente o fluxo das etapas metodológicas de maneira cronológica. O fluxo é composto por: levantamento de características, levantamento de requisitos e planejamento, modelagem e implementação, execução dos testes e registro de resultados, análise dos resultados e, por fim, resultados e trabalhos futuros.

Figura 31 – Etapas Metodológicas



Fonte: Elaborada pelo autor.

5.1 Levantamento de características

A primeira fase deste trabalho consistiu em uma revisão bibliográfica no campo dos bancos de dados. Nesta fase, foram examinadas documentações oficiais, trabalhos acadêmicos e livros, o que desempenhou um papel crucial na base de conhecimento para esta pesquisa. Inicialmente, foi realizada uma análise aprofundada dos conceitos subjacentes ao campo de banco de dados, abrangendo a exploração da definição e do propósito dos bancos de dados no contexto de aplicações Web. Além disso, essa etapa abordou a distinção entre bancos de dados relacionais e não relacionais, incluindo uma exposição dos diversos tipos de bancos de dados não relacionais, como os orientados a chave-valor, baseados em documentos, orientados a colunas e orientados a grafos.

Posteriormente, foi realizado um estudo detalhado de três ferramentas de gerenciamento de banco de dados selecionadas para este estudo comparativo. O critério de escolha

para as ferramentas foi a popularidade no cenário de desenvolvimento Web e gratuidade de uso. A plataforma *DB-Engines*[1] fornece um *ranking* de ferramentas de gerenciamento de banco de dados, e possui diversos parâmetros, sendo um deles a popularidade. A partir deste parâmetro, os SGBDs escolhidos, nos tópicos: relacional e não relacional, foram: `MySQL e MongoDB`.

Após a conclusão do estudo de cada uma das ferramentas, foi possível realizar a comparação estática, considerando os seguintes aspectos como parâmetros de análise:

1. **Transações ACID:** foi verificada a aderência dos SGBDs aos conceitos de Transações ACID. Esses conceitos são essenciais para serem avaliados em um sistema, especialmente em sistemas de maior porte, como as aplicações Web (UFRN, 2023).
2. **Teorema de CAP:** faz-se importante avaliar como os SGBDs se comportam relação ao Teorema de CAP, visto que, o entendimento do mesmo, possibilita que hajam tomadas de decisões que atendam aos requisitos em contextos específicos (KLEPPMANN, 2017).
3. **Sistemas operacionais:** o suporte a multiplataformas torna o software mais flexível, sendo acessível para um público mais amplo. Essa facilidade de adaptação torna-se um grande diferencial em ambientes diversificados e competitivos, permitindo que o software atenda às demandas variadas do mercado (MOONEY, 2004). Diante disto, este foi um dos critérios de avaliação escolhidos para a análise comparativa.
4. **Linguagens de programação suportadas:** foi considerado a quantidade de linguagens de programação que os SGBDs possuem suporte. Esse parâmetro possibilita a identificação do sistema que possui maior versatilidade de soluções para resolver problemas em diversas vertentes (SEBESTA, 2018).
5. **Documentação:** outro critério analisado foi a disponibilidade de documentação fornecida pelas plataformas oficiais de cada um dos SGBDs abordados neste estudo. A existência de uma boa documentação facilita a comunicação, desenvolvimento e compreensão do software por parte de quem desenvolve e utiliza (FORWARD ANDREW E LETHBRIDGE, 2002).
6. **Ferramenta de gerenciamento gratuita:** foi verificada a existência de versões de ferramenta de gerenciamento gratuita para cada um dos SGBDs. Ferramentas de gerenciamento possuem interfaces que possibilitam organizar e realizar ações sem a necessidade realizar todas as operações por meio de linha de código, o que pode facilitar o gerenciamento e desenvolvimento do software.

[1] *DB-Engines Ranking*

5.2 Levantamento de requisitos e planejamento

Foi aplicado um teste de desempenho[2] com o objetivo de avaliar o comportamento, em relação ao uso de recursos de *hardware* e tempo de execução, dos SGBDs diante dos cenários que serão apresentados mais adiante nesta seção. Em relação ao comportamento, nesta prática, foram considerados os seguintes tópicos como critério de avaliação:

1. **Tempo de resposta:** o tempo de resposta diz sobre quanto tempo um software devolve o resultado de uma determinada ação para o usuário. Com este, pode-se analisar quando e qual SGBD é mais eficiente em relação a tempo.
2. **Consumo de RAM:** o consumo de RAM foi o segundo critério, pois diante desta métrica, pode-se avaliar como cada sistema utiliza a memória para otimizar o desempenho e a capacidade de resposta durante os teste.
3. **Consumo de CPU:** o consumo de CPU, que indica o processamento utilizado por um processo para realizar suas operações, foi escolhido como critério de avaliação, pois assim como a memória RAM, é um critério relevante para a avaliação do desempenho do software.
4. **Espaço em disco ocupado pelos dados:** usar o espaço de disco de maneira eficiente, é essencial, visto que este fator afeta o desempenho e os custos operacionais do sistema. Diante disso, o terceiro parâmetro de comparação deste trabalho foi a medição de espaço utilizado por cada um dos SGBDs para o armazenamento dos dados nas operações de POST.

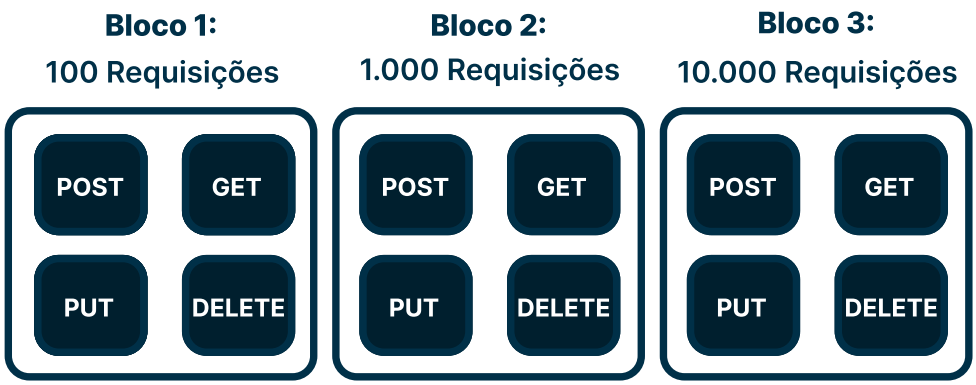
Para esta análise foram criados três blocos de testes. O Bloco 1 simula 100 usuários acessando simultaneamente o banco de dados. No Bloco 2, o número de usuários é 1.000, e, no terceiro bloco, o número de usuários é 10.000. A escolha do número de usuários em cada bloco foi determinada pela necessidade de avaliar o desempenho do sistema sob diversas cargas. Iniciou-se com 100 usuários para analisar uma carga inicial, aumentou-se para 1.000 para um teste intermediário, chegando a 10.000 para examinar a capacidade do sistema em condições de carga mais elevada. Essa progressão graduada permite compreender a resposta do sistema em diferentes níveis de demanda.

Cada um dos blocos de teste possui 4 operações diferentes, baseadas nas requisições mais conhecidas do método HTTP: POST, GET, PUT e DELETE, conforme detalhado no Capítulo4. A Figura 32 mostra como os blocos de teste estão organizados. O bloco 1 simula 100

[2] Conceito: Teste de Desempenho

usuários que farão requisições simultâneas de cada uma das operações já mencionadas. O Bloco 2 simula 1.000 usuários realizando requisições simultâneas. E por fim, o bloco 3 representa 10.000 usuários realizando requisições simultâneas aos SGBDs.

Figura 32 – Blocos de teste



Fonte: Elaborada pelo autor.

Respectivamente, as operações (POST, GET, PUT e DELETE) dentro de cada um dos blocos foram executadas individualmente. Isso significa que, para cada bloco, foram realizados quatro testes, resultando em um total de 12 testes para cada SGBD. Para facilitar a exposição das informações, a partir deste ponto, cada operação, em cada um dos blocos, será referido como um cenário de teste. A Tabela 9 exemplifica todos os 12 cenários de testes e detalha as ações realizadas em cada um deles. Por exemplo: no cenário de teste 1 é avaliado o comportamento dos SGBDs em relação a operação de POST com 100 usuários simultâneos.

	100	**1.000**	**10.000**
POST	1	2	3
GET	4	5	6
PUT	7	8	9
DELETE	10	11	12

Tabela 9 – Cenários de teste

Para facilitar a manipulação dos bancos de dados e assegurar a consistência nos testes, foram desenvolvidas duas APIs. Ambas as versões dessas APIs compartilham o mesmo contexto e propósito, que é a execução de operações CRUD relacionadas a produtos. No entanto, a distinção entre as duas versões reside na maneira como foram modeladas, adaptando-se às características específicas dos dois SGBDs selecionados. A escolha de utilizar a API visa

simular um ambiente mais próximo ao real, uma vez que é bastante utilizada em aplicações Web, estabelecendo comunicação entre a interface e o banco de dados (REDHAT, 2023).

A fim de simular os usuários realizando as requisições, foram escritos testes para cada cenário. Para este trabalho, os testes foram escritos na linguagem `Python`, com o uso das bibliotecas: *requests*[3] e *ThreadPoolExecutor*[4]. Essas duas bibliotecas, respectivamente, permitem: realizar solicitações HTTP e tarefas simultâneas por meio de *threads*[5].

Os testes foram executados em um notebook `Lenovo` modelo `ideapadS145` com 8GB de RAM, `SSD` de 250GB e processador `AMD Ryzen`. As medições de tempo, consumo de RAM e CPU foram conduzidas em um ambiente `Linux`, na versão `Ubuntu LTS 22.04`. Para quantificar o espaço ocupado pelas operações de criação (*create*), empregaram-se comandos específicos para cada um dos SGBDs, os quais serão apresentados detalhadamente na Seção 5.4.

5.3 Modelagem e implementação

Nesta etapa, ocorreu a modelagem e implementação das APIs juntamente com a criação dos testes correspondentes a cada um dos cenários previamente discutidos.

5.3.1 Implementação e modelagem das APIs

Optou-se por desenvolver uma aplicação que permite a criação, recuperação, exclusão e edição de produtos. A Figura 33 mostra a estrutura de produto, contendo os seguintes atributos: id (*Primary Key*), nome, preço por unidade, quantidade e data de validade. Esse modelo foi utilizado para a implementação da aplicação.

[3] Biblioteca *Requests* em `Python`: Guia Completo
[4] *ThreadPoolExecutor in Python: The Complete Guide*
[5] O que é Thread?

Figura 33 – Modelo Produto

Fonte: Elaborada pelo autor.

Como já exposto nesta seção, as APIs utilizadas serviram de intermédio entre usuário e o SGBD. Por este motivo, foi necessário implementar duas versões da aplicação, onde a principal diferença está na maneira na qual as consultas são realizadas e na conexão com os respectivos bancos de dados. Foi utilizada a ferramenta `Visual Studio Community 2022`[6] para a codificação das aplicações. Para realizar as consultas em cada uma das aplicações foram usados frameworks e ferramentas especificas do `.NET`, para `MySQL` e `MongoDB`, usou-se, respectivamente, os seguintes pacotes: `Entity Framework` e `MongoDB Driver`[7].

1. **Implementação versão MySQL**:
    - O Listing 5.3.1 mostra o código implementado para o método GET. Uma ação assíncrona[8] é definida para obter e retornar uma lista de produtos. A lista é recuperada da base de dados utilizando o `Entity Framework`.

```
1 public async Task<ActionResult<List<Produto>>> Get()
2 {
3     List<Produto> produtos = await _conexao.Produto.ToListAsync();
4      return Ok(produtos);
5 }
```

Listing 5.3.1 – Implementação do método GET - `MySQL`

    - O Listing 5.3.2 mostra o código implementado para o método POST. Neste caso, é adicionado um novo produto ao banco de dados utilizando o `Entity Framework`.

[6] `Visual Studio Community 2022`
[7] `MongoDB Driver`
[8] Programação assíncrona

```
public async Task<ActionResult<Produto> >  Post[FromBody] Produto
 ↪  produto)
{
    await _conexao.Produto.AddAsync(produto);
    await _conexao.SaveChangesAsync();
    return Ok(produto);
}
```

Listing 5.3.2 – Implementação do método POST - MySQL

- O Listing 5.3.3 mostra o código implementado para o método PUT. Neste caso, são atualizadas as informações de um produto existente, pelo seu ID, no banco de dados. O código possui tratamento de exceções para caso o produto não seja encontrado.

```
public async Task<ActionResult>  Put[FromBody] Produto produto,
↪  [Required] int id)
{
    try
    {
        var produtoProcurado = await
        ↪  _conexao.Produto.FirstOrDefaultAsync(p =>  p.Id== id);

        if (produtoProcurado == null)
        {
            return NotFound();
        }

        produtoProcurado.Nome = produto.Nome;
        produtoProcurado.PrecoPorUnidade = produto.PrecoPorUnidade;
        produtoProcurado.Quantidade = produto.Quantidade;
        produtoProcurado.DataValidade = produto.DataValidade;

        _conexao.Produto.Update(produtoProcurado);
        await _conexao.SaveChangesAsync();

        return NoContent();
    }
    catch (Exception)
    {
        return StatusCode(500, "Ocorreu um erro interno. Por favor,
        ↪  tente novamente mais tarde.");
    }
}
```

Listing 5.3.3 – Implementação do método PUT - MySQL

- O Listing 5.3.4 mostra o código implementado para o método DELETE. O método busca o produto pelo ID, remove-o se encontrado. O código possui tratamento de exceções para caso o produto não seja encontrado.

```
public async Task<ActionResult> DeleteById(int id)
{
    try
    {
        var produtoProcurado = await
        ↪ _conexao.Produto.FirstOrDefaultAsync(p => p.Id== id);

        if (produtoProcurado == null)
        {
            return NotFound();
        }

        _conexao.Produto.Remove(produtoProcurado);
        await _conexao.SaveChangesAsync();

        return Ok();
    }
    catch (Exception)
    {
        return StatusCode(500, "Ocorreu um erro interno. Por favor,
        ↪ tente novamente mais tarde.");
    }
}
```

Listing 5.3.4 – Implementação do método DELETE - `MySQL`

2. **Implementação versão MongoDB**:
   - O Listing 5.3.5 mostra o código implementado para o método GET. Os produtos são obtidos a partir do método assíncrono *ToListAsync()*, nativo do `MongoDB Driver`.

```
public async Task<ActionResult<List<Produto> > > Get
{
    var produtos = await _conexao.Produtos.ToListAsync();
    return Ok(produtos);
}
```

Listing 5.3.5 – Implementação do método GET - `MongoDB`

- O Listing 5.3.6 mostra o código implementado para o método POST. Neste caso, é inserido um novo produto no banco de dados `MongoDB`, com o método `InsertOneAsync()`.

```
public async Task<ActionResult<Produto> >  Post([FromBody] Produto
↪  produto)
{
    await _conexao.Produtos.InsertOneAsync(produto);
    return Ok(produto);
}

```

Listing 5.3.6 – Implementação do método POST - MongoDB

- O Listing 5.3.7 mostra o código implementado para o método PUT. Neste caso, são atualizadas as informações de um produto existente, pelo seu ID, no banco de dados, através do método *UpdateOneAsync()*.

```
public async Task<ActionResult>  Put(string id, [FromBody] Produto
↪  produto)
{
    try
    {
        var filter = Builders<Produto> .Filter.Eq("_id",
        ↪  ObjectId.Parse(id));
        var update = Builders<Produto> .Update
            .Set("Nome", produto.Nome)
            .Set("PrecoPorUnidade", produto.PrecoPorUnidade)
            .Set("Quantidade", produto.Quantidade)
            .Set("DataValidade", produto.DataValidade);

        var result = await _conexao.Produtos.UpdateOneAsync(filter,
        ↪  update);

        if (result.ModifiedCount == 0)
        {
            return NotFound();
        }

        return NoContent();
    }
    catch (Exception)
    {
        return StatusCode(500, "Ocorreu um erro interno. Por favor,
        ↪  tente novamente mais tarde.");
    }
}

```

Listing 5.3.7 – Implementação do método PUT - MongoDB

- O Listing 5.3.8 mostra o código implementado para o método DELETE. O método busca o produto pelo ID, remove-o se encontrado, através do método *DeleteOneAsync()*. O código possui tratamento de exceções para caso o produto não seja encontrado.

```
1  public async Task<ActionResult>  DeleteById(string id)
2  {
3      try
4      {
5          var filter = Builders<Produto> .Filter.Eq("_id",
           ↪  ObjectId.Parse(id));
6          var result = await _conexao.Produtos.DeleteOneAsync(filter);
7
8          if (result.DeletedCount == 0)
9          {
10             return NotFound();
11         }
12
13         return Ok();
14     }
15     catch (Exception)
16     {
17         return StatusCode(500, "Ocorreu um erro interno. Por favor,
           ↪  tente novamente mais tarde.");
18     }
19 }
20
```

Listing 5.3.8 – Implementação do método DELETE - MySQL

5.3.2 Implementação dos testes

Como já exposto, o trabalho pretende comparar, em relação à prática, os SGBDs em cenários de CRUD para 100, 1.000 e 10.000 requisições simultâneas, conforme exemplificado na Tabela 9. A implementação dos testes segue uma lógica específica para cada cenário, garantindo uma abordagem personalizada para avaliar diferentes casos. A principal variação entre os cenários está no número de requisições e na URL de requisição.

O Listing 5.3.9 apresenta o teste para os cenários que realizam requisições POST. Nas linhas de 1 a 3, ocorre a importação das bibliotecas mencionadas na Seção 5.2. Da linha 4 à 8, é realizada a implementação da função *send_post_request()*, responsável por efetuar a requisição HTTP POST, configurando a URL de requisição e o corpo, que inclui os campos do Produto

(Id, Nome, Preço por unidade, Quantidade e Data de Validade). Das linhas 9 a 27, encontra-se a função *__main__*. Nela, a lógica principal é executada, configurando a quantidade de threads (número de usuários), definindo o limite de requisições, tratando exceções e retornando os resultados.

```
1  import requests
2  from concurrent.futures import ThreadPoolExecutor
3
4  def send_post_request(_):
5      url = "http://localhost:porta_do_servidor/api/Produto"
6      data = { <json da requisição aqui>}
7      response = requests.post(url, json=data)
8      return response
9
10 if __name__ == "__main__":
11     successful_responses = 0
12     with ThreadPoolExecutor(max_workers=<numero de usuários>) as
       ↪  executor:
13         results = list(executor.map(send_post_request, range(<limite de
           ↪  requisições>)))
14
15     for result in results:
16         if result.status_code == 200:
17             successful_responses += 1
18
19     success_percentage = (successful_responses / len(results)) * 100
20
21     print(f"Porcentagem de sucesso: {success_percentage:.2f}%")
22
23     for result in results:
24         if result.status_code == 200:
25             print("Produto criado com sucesso!")
26         else:
27             print("Falha ao criar o produto.")
```

Listing 5.3.9 – Teste para POST em Python

Já no Listing 5.3.10, é apresentado o código de teste para cenários que executam requisições GET. A lógica de implementação é semelhante ao código anterior, diferindo na função responsável pela requisição HTTP, chamada de *send_get_request()*. Nesta função, é configurada a requisição com a URL e o tratamento da resposta.

```python
import requests
from concurrent.futures import ThreadPoolExecutor

def send_get_request(_):
    url = "http://localhost:porta_do_servidor/api/Produto"
    response = requests.get(url)
    return response

if __name__ == "__main__":
    successful_responses = 0

    with ThreadPoolExecutor(max_workers=<numero de usuários>) as executor:
        results = list(executor.map(send_get_request, range(<limite de requisições>)))

    for result in results:
        if result.status_code == 200:
            successful_responses += 1

    success_percentage = (successful_responses / len(results)) * 100

    print(f"Porcentagem de sucesso: {success_percentage:.2f}%")

    for result in results:
        if result.status_code == 200:
            print("Requisição GET bem-sucedida!")
        else:
            print("Falha na requisição GET.")

```

Listing 5.3.10 – Teste para GET em Python

Nos testes PUT (Listing 5.3.11) e DELETE (Listing 5.3.12), a lógica varia um pouco. Ambas as requisições exigem o parâmetro "ID" para serem efetuadas. Diante disso, é inicialmente realizada uma requisição GET que retorna todos os IDs presentes no banco. Posteriormente, a requisição principal do teste é executada (PUT ou DELETE). Neste contexto, a quantidade de usuários realizando as requisições dependerá da quantidade de produtos armazenados, em que cada usuário edita ou deleta um produto. Portanto, torna-se necessário verificar a quantidade de dados no banco antes de realizar as requisições. Para os testes do Bloco 1, é necessário existir 100 dados, para o Bloco 2, 1.000 usuários, e assim por diante.

```python
import requests
from concurrent.futures import ThreadPoolExecutor

def get_all_product_ids():
    url = "http://localhost:porta_do_servidor/api/Produto"
    response = requests.get(url)
    if response.status_code == 200:
        products = response.json()
        return [product["id"] for product in products]
    else:
        return []

def send_put_request(product_id):
    url = f"http://localhost:porta_do_servidor/api/Produto"
    data = {  <json da requisição aqui> }
    response = requests.put(url, json=data)
    return response

if __name__ == "__main__":
    successful_responses = 0

    product_ids = get_all_product_ids()

    with ThreadPoolExecutor(max_workers=<número de usuários>) as
    ↪  executor:
        results = list(executor.map(send_put_request, product_ids))

    for result in results:
        if result.status_code == 204:
            successful_responses += 1

    success_percentage = (successful_responses / len(results)) * 100

    for result in results:
        if result.status_code == 204:
            print("Requisição PUT bem-sucedida!")
        else:
            print("Falha na requisição PUT.")
            print("Resposta:", result.text)

    print(f"Porcentagem de sucesso: {success_percentage:.2f}%")
```

Listing 5.3.11 – Teste para PUT em Python

```python
import requests
from concurrent.futures import ThreadPoolExecutor

def get_all_product_ids():
    url = "http://localhost:porta_do_servidor/api/Produto"
    response = requests.get(url)
    if response.status_code == 200:
        products = response.json()
        return [product["id"] for product in products]
    else:
        return []

def send_delete_request(product_id):
    url = f"http://localhost:porta_do_servidor/api/Produto"
    response = requests.delete(url)
    return response

if __name__ == "__main__":
    successful_responses = 0

    product_ids = get_all_product_ids()

    with ThreadPoolExecutor(max_workers=<número de usuários>) as
    ↪  executor:
        results = list(executor.map(send_delete_request, product_ids))

    for result in results:
        if result.status_code == 200:
            successful_responses += 1

    success_percentage = (successful_responses / len(results)) * 100

    for result in results:
        if result.status_code == 200:
            print("Requisição DELETE bem-sucedida!")
        else:
            print("Falha na requisição DELETE.")
```

Listing 5.3.12 – Teste para DELETE em Python

Nestes testes, a biblioteca *ThreadPoolExecutor* é empregada para simular usuários, onde cada usuário é representado por uma *thread*. Esses usuários virtuais executam a função *send_<método>_request*, e o número máximo de usuários simultâneos é controlado pelo parâmetro *max_workers*. O parâmetro *range*(<limite de requisições>) determina quantas requisições

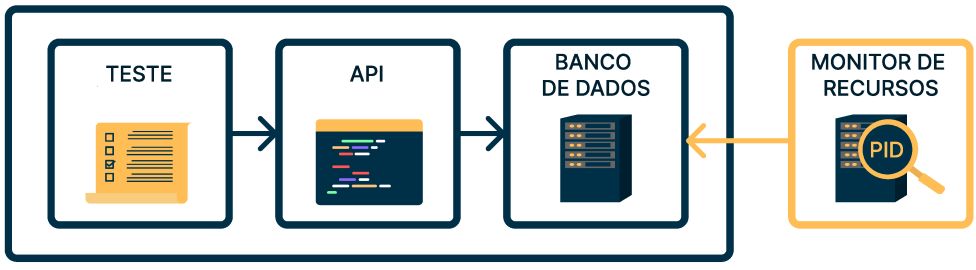
cada usuário realiza de maneira concorrente. Essa configuração permite avaliar como o sistema se comporta com a simulação de múltiplos usuários realizando operações simultaneamente.

5.4 Execução dos testes e registro de resultados

Após a modelagem e configurações necessárias para a realização dos testes, conforme detalhado na Seção 5.3, a execução foi iniciada. Foi preciso ajustar alguns limites, uma vez que tanto o `MySQL` quanto o `MongoDB` têm, por padrão, um limite de requisições simultâneas. Nesse sentido, foram realizados ajustes nas configurações. Para o `MySQL`, foi necessário editar seu arquivo de configuração, modificando o parâmetro *max_connections*[9] para 10.000. Já no `MongoDB`, uma alteração semelhante foi feita, mas focando no parâmetro *maxIncoming-Connections*[10]. Outra configuração crucial envolveu o número máximo de arquivos abertos simultaneamente no sistema `Ubuntu`, sendo alterado através do parâmetro *ulimit*[11]. Esses ajustes garantiram um ambiente propício para a execução eficiente dos testes.

A Figura 34 ilustra a organização do fluxo de teste em todos os cenários. O processo se inicia com os testes, que simulam os usuários, realizando requisições à API. Esta, por sua vez, efetua as requisições ao SGBD no banco de dados para recuperar ou modificar dados. O monitor de recursos, representado pelo comando `top`[12], observa os recursos consumidos pelo SGBD por meio do `PID`[13] associado a ele. Esse monitoramento deve ser iniciado antes dos testes, razão pela qual está posicionado fora do padrão na imagem. Essa estrutura proporciona uma visão do comportamento do sistema durante a execução dos testes.

Figura 34 – Modelo geral do experimento



Fonte: Adaptada de (The Beginner's Guide to REST API, 2022)

[9] *MySQL 8.0 Reference Manual - Too many connections*
[10] *Configuration File Settings and Command-Line Options Mapping*
[11] *Ulimit, Soft Limits and Hard Limits in Linux*
[12] Processos Linux
[13] Comandos `Linux` – Comando `top`

Em cada bloco de teste, as requisições seguiram a ordem: POST, GET, PUT e DELETE. Para garantir a consistência, inicou-se cada bloco com bancos vazios, assegurando a quantidade exata de dados após a execução do POST. É importante notar que, para cada bloco, foi verificado e garantido a existência do número preciso de dados correspondente ao cenário.

Para todos os testes, independentemente do cenário e SGBD, foi seguido o seguinte passo a passo:

1. Rodar a API com o comando do Listing 5.4.1:

```
dotnet run
```

Listing 5.4.1 – Comando para executar a API .NET

2. Obter o PID relacionado ao banco a ser testado com o comando do Listing 5.4.2:

```
lsof -i :"numero da porta"
```

Listing 5.4.2 – Comando para obter o PID relacionado à porta do banco de dados

3. Rodar o comando "top" para fazer a leitura dos recursos do banco com o PID obtido e realizar a escrita dos dados em um arquivo, conforme mostrado no Listing 5.4.3. Neste caso, os recursos utilizados pelo SGBD com o PID 760 estão sendo monitorados e gravados no arquivo "nome-do-log-do-top.txt".

```
top -b -d 1 -p 760 >> ~caminho/nome-do-log-do-top.txt
```

Listing 5.4.3 – Comando para monitorar recursos com o comando "top"

4. Rodar o teste específico para o cenário, como exemplificado no Listing 5.4.4.

```
python nome_do_teste.py
```

Listing 5.4.4 – Comando para executar o teste Locust

Para obter o tamanho em disco ocupado pelo armazenamento em cada bloco, foram utilizados comandos específicos dentro do servidor de cada um dos SGBDs, no terminal do Linux. No caso do `MySQL`, foi empregado o comando do Listing 5.4.5 e para o `MongoDB`, o comando do Listing 5.4.6.

```
SELECT
  ROUND(data_length / 1024 / 1024, 2) AS "Tamanho da Tabela (MB)"
FROM information_schema.tables
WHERE table_schema = 'produtosDB'
  AND table_name = 'Produto';
```

Listing 5.4.5 – Comando para obter tamanho alocado com armazenamento dos dados num banco `MySQL`

```
db.Produtos.stats()
```

Listing 5.4.6 – Comando para obter tamanho alocado com armazenamento dos dados num banco `MongoDB`

Posteriormente à realização dos testes, procedeu-se à análise dos resultados para cada um dos Sistemas de Gerenciamento de Banco de Dados alvos deste trabalho. Isso envolveu a leitura e avaliação dos recursos captados cada teste e, subsequentemente, a geração de gráficos. Esses dados desempenharam um papel essencial na etapa que se concentra nos resultados obtidos e discussões. A fim de obter resultado mais precisos cada teste foi executado três vezes e os resultados obtidos são fruto da média dos dados das três execuções.

5.5 Análise dos resultados

Para orientar o estudo, as questões de pesquisa foram formuladas para alcançar o objetivo proposto. Essas questões foram delineadas no Capítulo 3, na Seção 3.3. Nesta seção, serão discutidas as questões de pesquisa com o intuito de explicar o que cada questão busca alcançar e como os resultados foram obtidos.

- A "**RQ_1: Como os SGBDs se classificam dentro da métricas da análise estática?**" busca responder, com base nas características apresentadas na Seção 5.1, como os SGBDs se comportam. Isso permite analisar a adequação de cada sistema a essas características. A resposta a essa questão foi obtida por meio da análise da Tabela 10, a qual foi elaborada a partir do estudo e análise dos conceitos relacionados a bancos de dados apresentados no Capítulo 4;
- A "**RQ_2: Em relação às características levantadas para a comparação estática, em**

quais delas os SGBDs `MySQL` e `MongoDB` divergem?" busca responder quais são as principais diferenças entre os dois SGBDs em relação as características levantadas neste estudo. Para responder esta questão foi necessário analisar a mesma tabela usada para responder a "RQ_1";

- A "**RQ_3: Qual dos SGBDs demonstra maior flexibilidade em termos de suporte a linguagens de programação e sistemas operacionais?**" busca responder qual dos SGBD ganha no quesito flexibilidade em termos de suporte a linguagens de programação e sistemas operacionais. Para responder esta questão foi necessário analisar a mesma tabela usada para responder a "RQ_1";
- A "**RQ_4: Como os SGBDs se comportam em relação ao consumo de RAM e CPU em cada um dos cenários estabelecidos?**" busca responder qual o comportamento dos SGBDs abordados neste estudo ao término de cada cenário de teste, analisando os consumos de recursos e discutindo os resultados. Para abordar essa questão, foi necessário executar cada um dos cenários de testes e analisar os gráficos gerados a partir dos dados resultantes;
- A "**RQ_5: Qual é o espaço em disco real consumido por cada SGBD para armazenar a mesma quantidade de dados em diferentes configurações?**" busca responder quais os tamanhos, em MB, dos espaços utilizados para o armazenamento dos dados em cada um dos blocos de teste propostos neste trabalho. Para responder esta questão foi necessário executar os comando mostrados na Seção 5.4;
- A "**RQ_6: Como os SGBDs respondem em termos de tempo durante os três blocos de testes?**" busca responder qual o tempo total que cada um dos SGBDs usa para executar cada um dos cenários de teste propostos neste trabalho. O método para chegar no resultado desta pergunta foi o mesmo da "RQ_4";

5.6 Resultados e trabalhos futuros

Concluídos os procedimentos necessários para a comparação, tanto dinâmica quanto estática, foram realizadas análises com base nos resultados obtidos. Nesta fase, foi possível realizar observações e conclusões com base nos parâmetros apresentados na Seção 5.5. Além disso, ao final das comparações foi possível observar possíveis melhorias para obter resultados mais completos em trabalhos futuros.

6 RESULTADOS E DISCUSSÕES

Neste capítulo, apresentam-se os resultados alcançados nas duas etapas deste estudo, e também serão discutidas as questões de pesquisa propostas no Capítulo 3. Inicialmente, na Seção 6.1, serão respondidas as questões de pesquisa referentes à análise estática. Em seguida, na Seção 6.2, abordam-se as respostas para as questões relacionadas à análise dinâmica.

6.1 Resultados da comparação teórica dos SGBDs alvos do estudo

A Tabela 10 apresenta os resultados da comparação de características teóricas dos dois SGBDs envolvidos neste trabalho: `MySQL` e `MongoDB`.

Características		**MySQL**	**MongoDB**
Transações ACID		SIM	SIM
Teorema de CAP	Consistência (Consistency)	SIM	SIM
	Disponibilidade (Availability)	SIM	NÃO
	Particionamento (Partition Tolerance)	NÃO	SIM
Sistemas Operacionais	Windows	SIM	SIM
	MacOs	SIM	SIM
	Linux (Incluindo as Distribuições)	SIM	SIM
Linguagens de Programação	JavaScript	SIM	SIM
	C#(C Sharp)	SIM	SIM
	Python	SIM	SIM
	PHP	SIM	SIM
	Java	SIM	SIM
Documentação		SIM	SIM
Ferramenta de Gerenciamento Gratuitas		SIM	SIM

Tabela 10 – Resultado da comparação estática entre os SGBDs

6.1.1 Resposta da RQ_1: Como os SGBDs se classificam dentro da métricas da análise estática?

- **Transações ACID:** em relação as transações ACID, o `MySQL`, por ser um banco de dados relacional, oferece suporte nativo a transações ACID, conforme sua documentação (CLOUD, 2023). Isso significa que o `MySQL` garante a atomicidade, consistência, isolamento e durabilidade das transações, o que é essencial para manter a confiabilidade dos dados em aplicações críticas.

  Por outro lado, o `MongoDB` introduziu o suporte para transações ACID de vários documentos na versão 4.0, em 2018, e estendeu esse suporte para transações ACID de vários documentos distribuídas na versão 4.2, em 2019 (MONGODB, 2023c). Isso significa que o `MongoDB` oferece recursos para garantir a atomicidade, consistência, isolamento e durabilidade em transações que envolvem vários documentos, tornando-o adequado para casos de uso em que a escalabilidade e a flexibilidade são necessárias.
- **Teorema de CAP:** em relação ao Teorema de CAP, o `MySQL`, por padrão, oferece consistência e disponibilidade, mas não é tolerante à partição (CLOUD, 2023). Isso significa que todas as instâncias/nós do banco de dados manterão os dados no mesmo estado, permanecendo sempre atualizados, independentemente de alterações. Além disso, garante que os dados estejam sempre disponíveis para consultas e manipulações, independentemente de falhas de rede ou outros problemas que possam ocorrer.

  Para o `MongoDB`, a situação é diferente, de acordo com sua documentação. Por padrão, este sistema é consistente e tolerante à partição. Isso implica que todos os nós/servidores podem funcionar individualmente em caso de falhas, e as informações contidas em todos os nós estarão sempre atualizadas (MONGODB, 2023b). Vale ressaltar que essas são as configurações padrão de cada um dos SGBDs, que podem ser adaptadas de acordo com as necessidades do sistema.
- **Sistemas Operacionais:** tanto o `MySQL` quanto `MongoDB` possuem suporte para três SO's mais utilizados na atualidade: `Windows`, `MacOS` e `Linux` (LISBOA, 2023). Para mostrar as diversas ferramentas suportadas, a Tabela 11 mostra os SGBDs e seus respectivos SO suportados de acordo com as documentações (MYSQL, 2023), (MONGODB, 2023a).

Sistemas Operacionais	MySQL	MongoDB
Linux	X	X
Windows	X	X
macOS	X	X
Solaris	X	
FreeBSD	X	
OpenBSD	X	
AIX	X	
HP-UX	X	
Other Unix-like	X	X

Tabela 11 – Suporte dos SGBD's a Sistemas Operacionais (SO).

- **Linguagens de Programação:** a Tabela 12 mostra quais linguagens de programação são suportadas por cada um dos SGBDs, e as informações estão em conformidade com as descrições em suas respectivas documentações(MYSQL, 2023; MONGODB, 2023a). Analisando a tabela pode-se notar que o sistema que se destaca claramente nesse o `MongoDB`, que demonstra um suporte substancial a linguagens de programação, com um total de 37 linguagens suportadas. E por último, o `MySQL`, que oferece suporte a um conjunto de 14 linguagens de programação.

Linguagem	MySQL	MongoDB
Actionscript		X
Ada		
C	X	X
C#	X	X
C++	X	X
Clojure		X
ColdFusion		X
Crystal		X
Dart		X
Delphi	X	X
Eiffel		
Elixir		X
Erlang	X	X
Fancy		X
Go		X
Groovy		
Haskell	X	X
Haxe		
Java	X	X
JavaScript		X
Kotlin		X
Lisp		X
Lua		X
MatLab		X
Objective-C	X	X
OCaml	X	X
Pascal		
Perl	X	X
PHP	X	X
PowerShell		X
Prolog		X
Pure Data		X
Python	X	X
R		X
Rebol		X
Ruby	X	X
Rust		X
Scala		X
Scheme		X
Smalltalk		X
Swift		X
Tcl	X	X
Visual Basic		

Tabela 12 – Panorama de Linguagens de Programação Suportadas por cada SGBD

- **Documentação:** no que se refere à documentação, é relevante observar que tanto o `MySQL`

quanto o MongoDB possuem documentações oficiais disponibilizadas. Isso assegura que os usuários tenham acesso a recursos detalhados e informações precisas para aproveitar ao máximo esses SGBD e garantir a eficiência e a confiabilidade em seus projetos (MYSQL, 2023; MONGODB, 2023a).

- **Ferramenta de Gerenciamento Gratuitas:** a presença de uma interface amigável para gerenciar as operações de um sistema de gerenciamento é de grande importância, pois facilita a compreensão e torna o uso mais acessível. Além disso, a disponibilidade de uma opção gratuita é um detalhe relevante a ser considerado, tornando o sistema acessível a um público mais amplo. Conforme discutido na fundamentação, pode-se observar que o MySQL oferece o MySQL Workbench, o MongoDB disponibiliza o MongoDB Compass, ambas ferramentas com versões gratuitas e eficazes para gerenciamento de banco de dados.

6.1.2 Resposta da RQ_2: Em quais características específicas os SGBDs MySQL e MongoDB divergem, e quais são as implicações práticas dessas diferenças?

Segundo a Tabela 10, a única divergência ocorre no tópico do Teorema de CAP. O MySQL, por padrão, busca oferecer consistência e disponibilidade, portanto, em casos de particionamento, não consegue garantir tolerância à falhas. Em contraste, o MongoDB, por padrão, prioriza consistência e tolerância à partição, o que significa que, em situações de particionamento, se um nó se tornar inconsistente devido a falhas, o sistema o tornará indisponível. Em termos gerais, essa divergência entre os dois SGBDs indica que, em casos de falhas, os dois sistemas terão comportamentos distintos, sendo necessário avaliar quais dos três critérios são mais importantes para o contexto do projeto a ser implementado.

6.1.3 Resposta da RQ_3: Qual dos SGBDs demonstra maior flexibilidade em termos de suporte a linguagens de programação e sistemas operacionais?

Segundo as Tabelas 11 e 12 juntas à análise para responder à RQ_1, observa-se que, em relação as linguagens analisadas, o MongoDB oferece suporte a um maior número de linguagens de programação, totalizando 37, enquanto o MySQL suporta apenas 14. Em contrapartida, no que diz respeito ao suporte nos sistemas operacionais, o MySQL é o que apresenta um maior número, totalizando 9, ao passo que o MongoDB oferece suporte a apenas 4. Diante desses resultados, compreende-se que o MongoDB possui uma gama mais ampla de linguagens de programação às quais pode ser conectado, revelando-se mais flexível nesse aspecto. Por outro lado, o MySQL

destaca-se pela flexibilidade em relação aos sistemas operacionais. A escolha entre os dois SGBDs dependerá, portanto, das prioridades específicas do projeto, considerando as necessidades de linguagens de programação e sistemas operacionais.

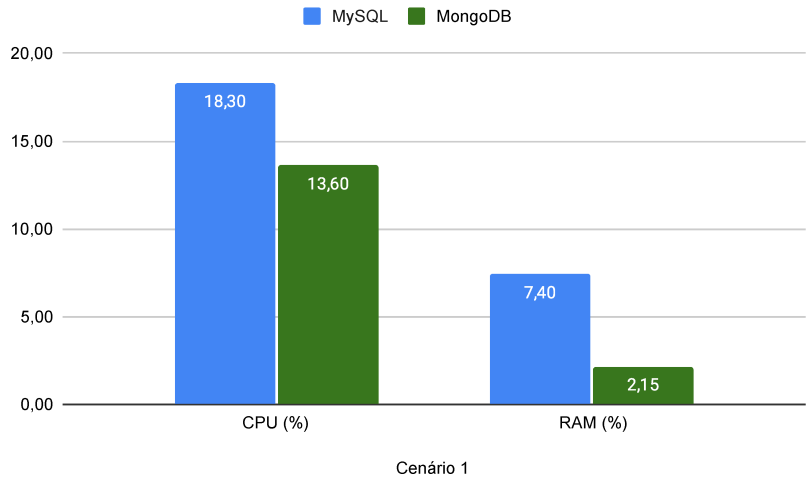
6.2 Resultados da comparação prática dos SGBDs alvos do estudo

Nesta seção, serão apresentados e discutidos os resultados da comparação prática com base nos testes realizados, conforme documentado na seção 5.4. Os resultados serão abordados com base nas questões de pesquisa relacionadas a parte dinâmica desta comparação.

6.2.1 Resposta da RQ_4: Como os SGBDs se comportam em relação ao consumo de RAM e CPU em cada um dos cenários estabelecidos?

- **Cenário 1 - 100 requisições "POST" simultâneas:**

Figura 35 – Resultados do Cenário 1

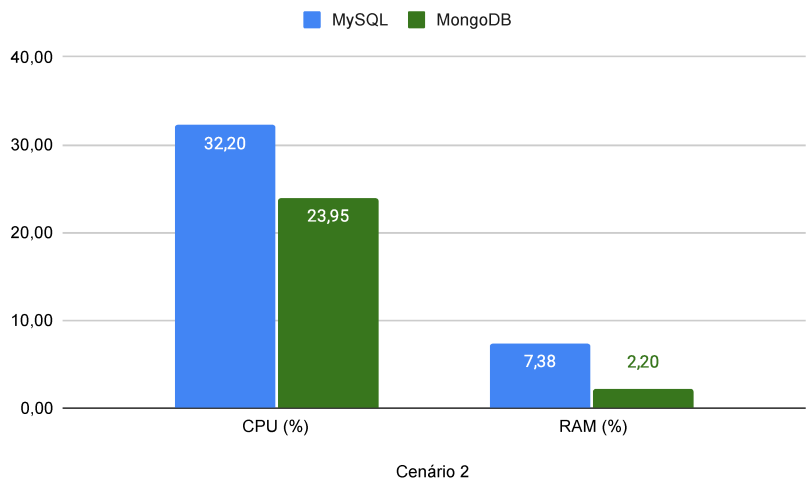


Fonte: Elaborada pelo autor

Analisando a Figura 35, nota-se que o `MySQL` consome 18,30% da CPU e 7,40% da RAM, enquanto o `MongoDB` utiliza 13,60% da CPU e 2,15% da RAM. A diferença percentual no consumo de CPU e RAM entre o `MySQL` e o `MongoDB` é aproximadamente de 25,68% e 70,94%, respectivamente. Com base nesses números, conclui-se que, neste cenário, o `MongoDB` apresenta um melhor desempenho em ambas as métricas em comparação com o `MySQL`.

- **Cenário 2 - 1.000 requisições "POST" simultâneas:**

Figura 36 – Resultados do Cenário 2

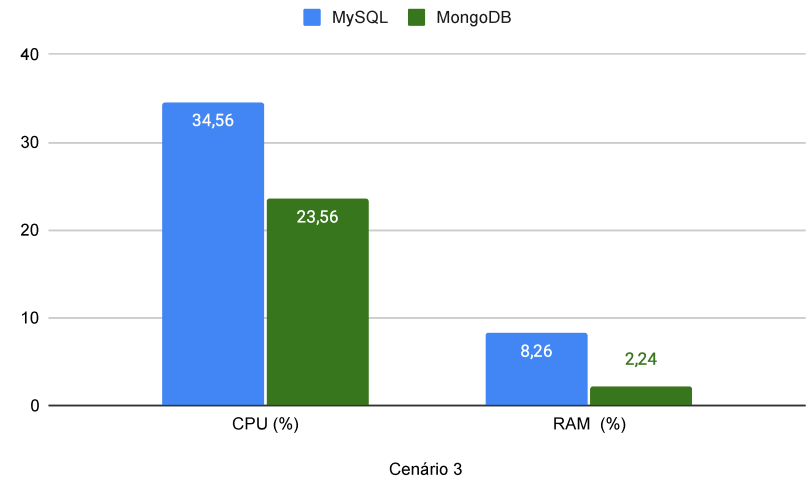


Fonte: Elaborada pelo autor

Analisando a Figura 36, nota-se que o MySQL consome 32,20% da CPU e 7,38% da RAM, enquanto o MongoDB utiliza 23,95% da CPU e 2,20% da RAM. A diferença percentual no consumo de CPU e RAM entre o MySQL e o MongoDB é aproximadamente de 25,62% e 70,18%, respectivamente. Com base nesses números, conclui-se que, neste cenário, o MongoDB apresenta um melhor desempenho em ambas as métricas em comparação com o MySQL.

- **Cenário 3 - 10.000 requisições "POST" simultâneas:**

Figura 37 – Resultados do Cenário 3

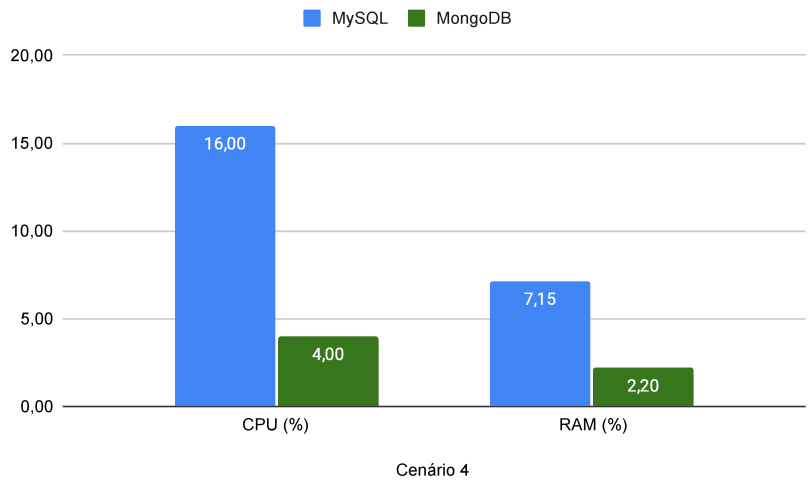


Fonte: Elaborada pelo autor

Analisando a Figura 37, nota-se que o MySQL consome 34,56% da CPU e 8,26% da RAM, enquanto o MongoDB utiliza 23,56% da CPU e 2,24% da RAM. A diferença percentual no consumo de CPU e RAM entre o MySQL e o MongoDB é aproximadamente de 31,82% e 72,88%, respectivamente. Com base nesses números, conclui-se que, neste cenário, o MongoDB apresenta um melhor desempenho em ambas as métricas em comparação com o MySQL.

- **Cenário 4 - 100 requisições "GET" simultâneas:**

Figura 38 – Resultados do Cenário 4

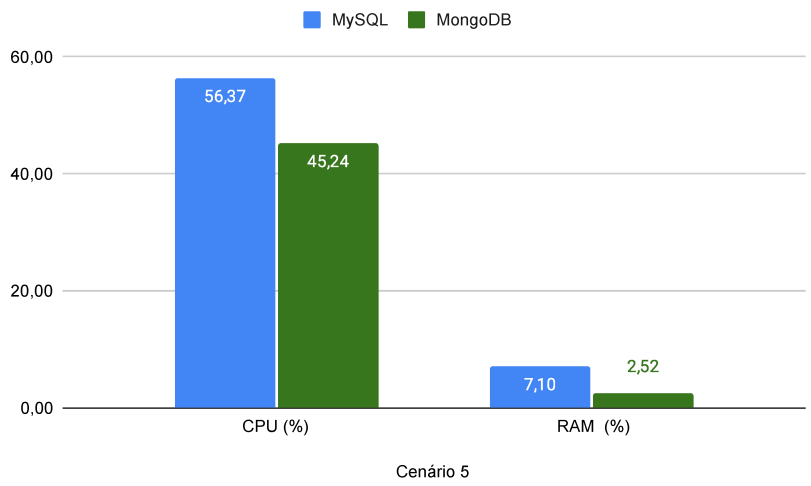


Fonte: Elaborada pelo autor

Analisando a Figura 38, percebe-se que o MySQL consome 16,00% da CPU e 7,15% da RAM, enquanto o MongoDB utiliza 4,00% da CPU e 2,20% da RAM. A diferença percentual no consumo de CPU e RAM entre o MySQL e o MongoDB é aproximadamente de 75,00% e 69,23%, respectivamente. Com base nesses números, conclui-se que, neste cenário, o MongoDB demonstra um desempenho superior em ambas as métricas em comparação com o MySQL.

- **Cenário 5 - 1.000 requisições "GET" simultâneas:**

Figura 39 – Resultados do Cenário 5

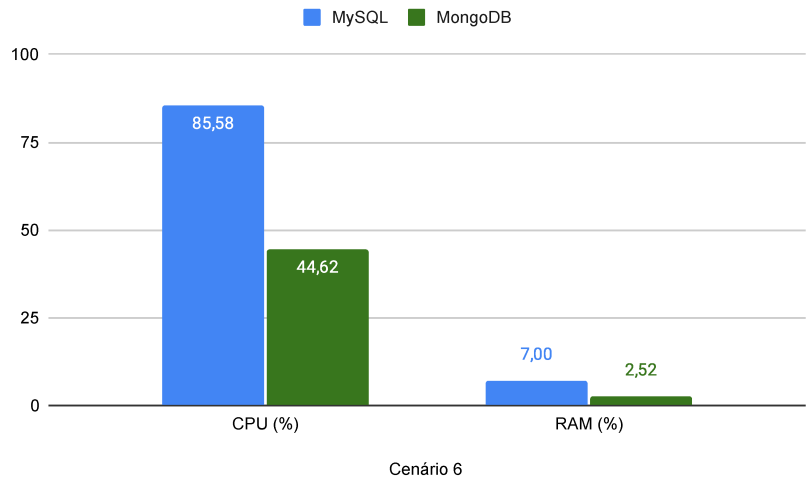


Fonte: Elaborada pelo autor

Analisando a Figura 39, percebe-se que o `MySQL` consome 56,37% da CPU e 7,10% da RAM, enquanto o `MongoDB` utiliza 45,24% da CPU e 2,52% da RAM. A diferença percentual no consumo de CPU e RAM entre o `MySQL` e o `MongoDB` é aproximadamente de 19,74% e 64,50%, respectivamente. Com base nesses números, conclui-se que, neste cenário, o `MongoDB` demonstra um desempenho superior em ambas as métricas em comparação com o `MySQL`.

- **Cenário 6 - 10.000 requisições "GET" simultâneas:**

Figura 40 – Resultados do Cenário 6

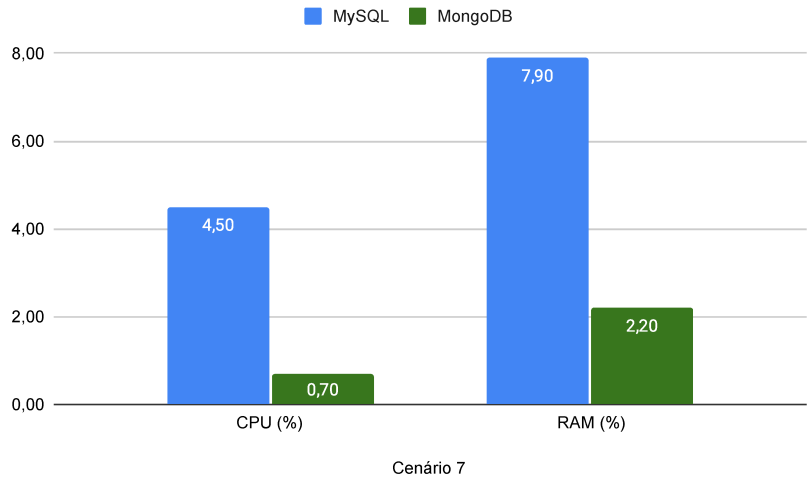


Fonte: Elaborada pelo autor

Analisando a Figura 40, nota-se que o MySQL consome 85,58% da CPU e 7,00% da RAM, enquanto o MongoDB utiliza 44,62% da CPU e 2,52% da RAM. A diferença percentual no consumo de CPU e RAM entre o MySQL e o MongoDB é aproximadamente de 47,85% e 64,00%, respectivamente. Com base nesses números, conclui-se que, neste cenário, o MongoDB apresenta um melhor desempenho em ambas as métricas em comparação com o MySQL.

- **Cenário 7 - 100 requisições "PUT" simultâneas:**

Figura 41 – Resultados do Cenário 7

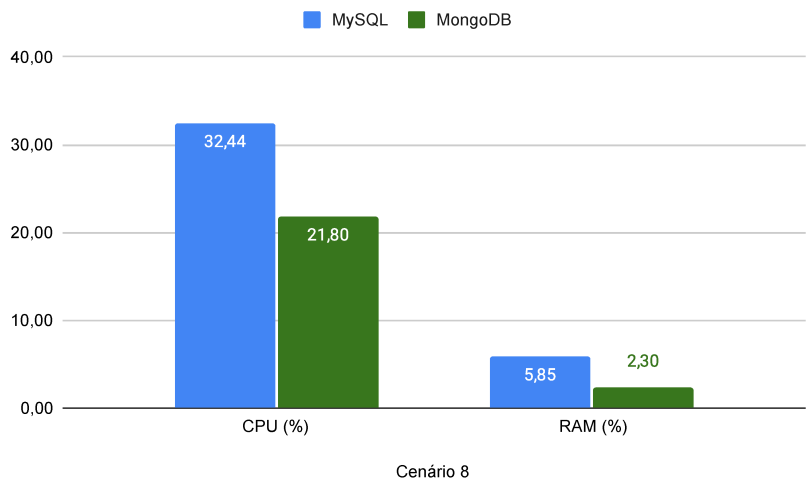


Fonte: Elaborada pelo autor

Analisando a Figura 41, observa-se que o MySQL consome 4,50% da CPU e 7,90% da RAM, enquanto o MongoDB utiliza 0,70% da CPU e 2,20% da RAM. A diferença percentual no consumo de CPU e RAM entre o MySQL e o MongoDB é aproximadamente de 84,55% e 72,15%, respectivamente. Com base nessas métricas, conclui-se que, neste cenário, o MongoDB demonstra um desempenho significativamente superior em ambas as métricas em comparação com o MySQL.

- **Cenário 8 - 1.000 requisições "PUT" simultâneas:**

Figura 42 – Resultados do Cenário 8

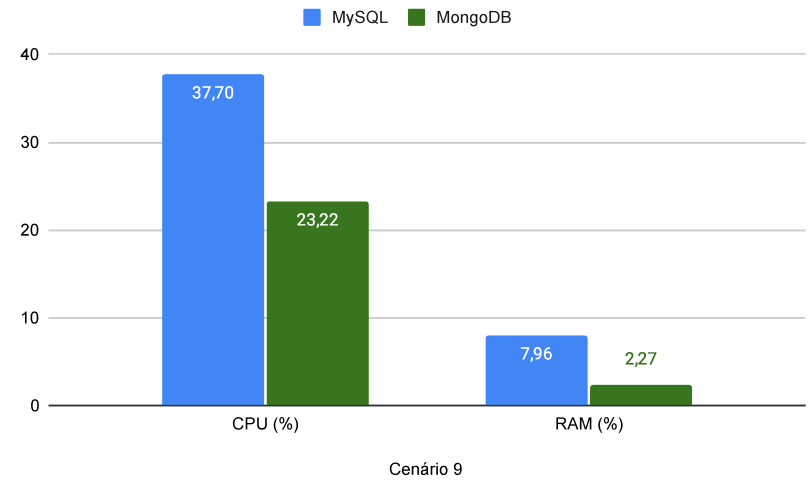


Fonte: Elaborada pelo autor

Analisando a Figura 42, verifica-se que o `MySQL` consome 32,44% da CPU e 5,85% da RAM, enquanto o `MongoDB` utiliza 21,80% da CPU e 2,30% da RAM. A diferença percentual no consumo de CPU e RAM entre o `MySQL` e o `MongoDB` é aproximadamente de 32,79% e 60,68%, respectivamente. Com base nesses números, conclui-se que, neste cenário, o `MongoDB` apresenta um desempenho superior em ambas as métricas em comparação com o `MySQL`.

- **Cenário 9 - 10.000 requisições "PUT" simultâneas:**

Figura 43 – Resultados do Cenário 9

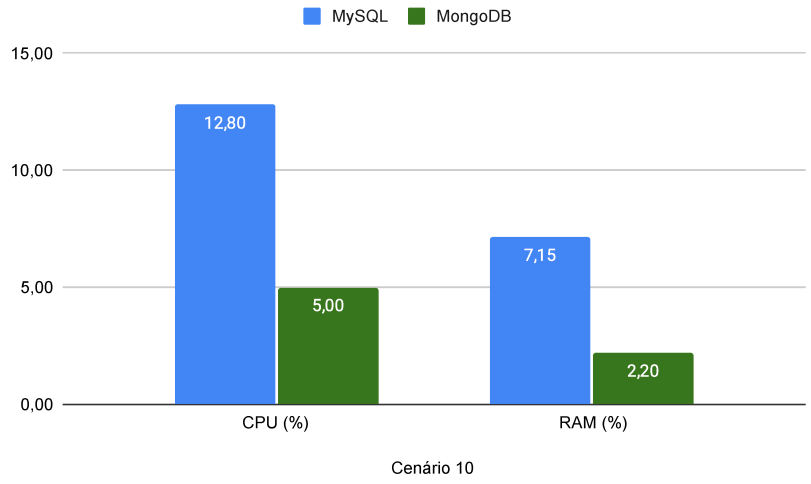


Fonte: Elaborada pelo autor

Analisando a Figura 43, constata-se que o MySQL consome 37,70% da CPU e 7,96% da RAM, enquanto o MongoDB utiliza 23,22% da CPU e 2,27% da RAM. A diferença percentual no consumo de CPU e RAM entre o MySQL e o MongoDB é aproximadamente de 38,40% e 71,48%, respectivamente. Com base nessas métricas, conclui-se que, neste cenário, o MongoDB demonstra um desempenho superior em ambas as métricas em comparação com o MySQL.

- **Cenário 10 - 100 requisições "DELETE" simultâneas:**

Figura 44 – Resultados do Cenário 10

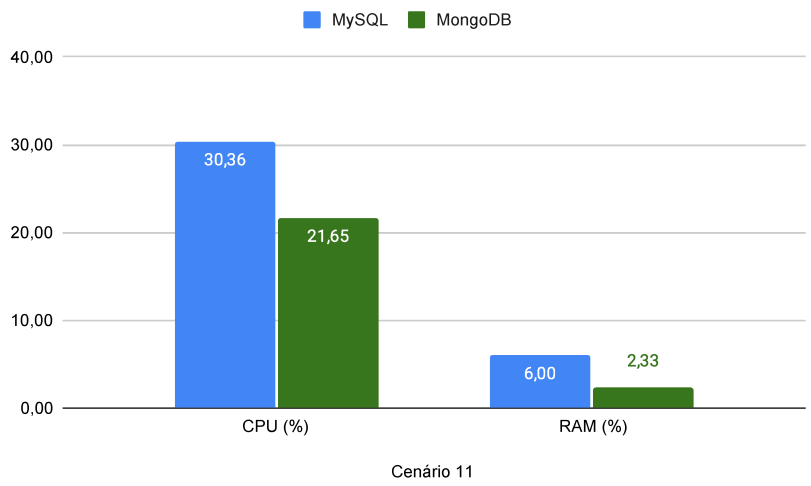


Fonte: Elaborada pelo autor

Analisando a Figura 44, percebe-se que o MySQL consome 12,80% da CPU e 7,15% da RAM, enquanto o MongoDB utiliza 5,00% da CPU e 2,20% da RAM. A diferença percentual no consumo de CPU e RAM entre o MySQL e o MongoDB é aproximadamente de 60,93% e 69,23%, respectivamente. Com base nessas métricas, conclui-se que, neste cenário, o MongoDB demonstra um desempenho superior em ambas as métricas em comparação com o MySQL.

- **Cenário 11 - 1.000 requisições "DELETE" simultâneas:**

Figura 45 – Resultados do Cenário 11

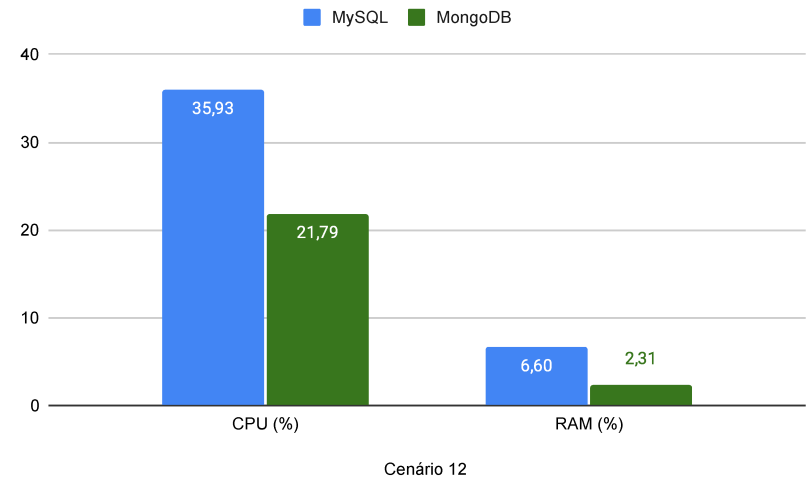


Fonte: Elaborada pelo autor

Analisando a Figura 45, nota-se que o MySQL consome 30,36% da CPU e 6,00% da RAM, enquanto o MongoDB utiliza 21,65% da CPU e 2,33% da RAM. A diferença percentual no consumo de CPU e RAM entre o MySQL e o MongoDB é aproximadamente de 28,68% e 61,16%, respectivamente. Com base nessas métricas, conclui-se que, neste cenário, o MongoDB apresenta um desempenho superior em ambas as métricas em comparação com o MySQL.

- **Cenário 12 - 10.000 requisições "DELETE" simultâneas:**

Figura 46 – Resultados do Cenário 12

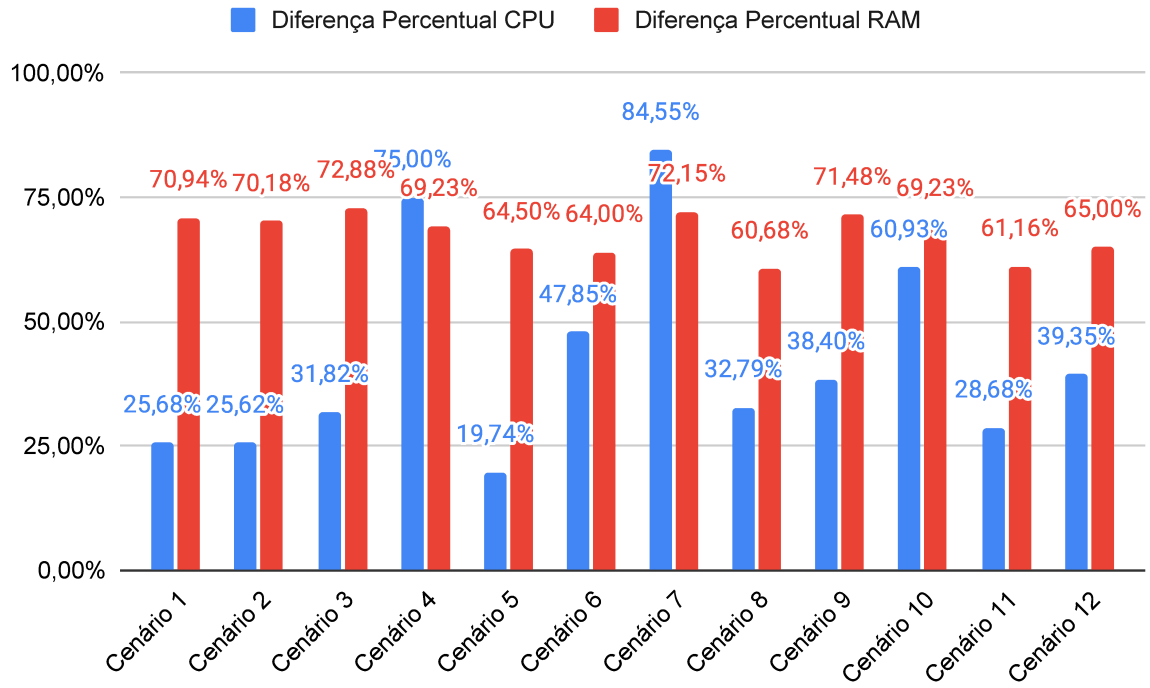


Fonte: Elaborada pelo autor

Analisando a Figura 45, observa-se que o MySQL consome 35,93% da CPU e 6,60% da RAM, enquanto o MongoDB utiliza 21,79% da CPU e 2,31% da RAM. A diferença percentual no consumo de CPU e RAM entre o MySQL e o MongoDB é aproximadamente de 39,35% e 65,00%, respectivamente. Com base nessas métricas, conclui-se que, neste cenário, o MongoDB apresenta um desempenho superior em ambas as métricas em comparação com o MySQL.

A partir da análise em cada um dos cenários de teste e do gráfico na Figura 47 pode-se notar que em todos os casos o MongoDB possui um melhor desempenho, tanto em relação ao consumo de CPU quanto de RAM, fazendo com que, se a eficiência no uso de recursos (CPU e RAM) for uma prioridade, os resultados indicam que o MongoDB pode ser uma escolha mais otimizada em termos de desempenho.

Figura 47 – Análise geral de todos os cenários em relação aos recursos CPU e RAM

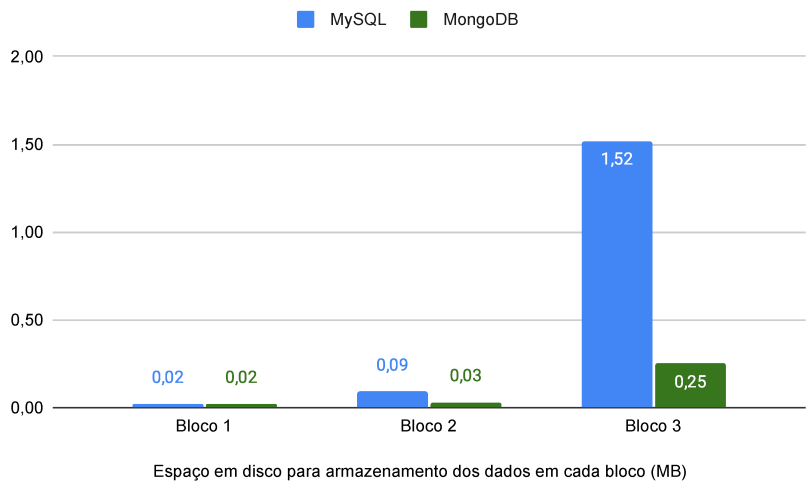


Fonte: Elaborada pelo autor

6.2.2 *Resposta da RQ_5: Qual é o espaço em disco real consumido por cada SGBD para armazenar a mesma quantidade de dados em diferentes configurações?*

Em relação ao espaço em disco ocupado pelos diferentes blocos de teste propostos neste trabalho, constata-se, com base no gráfico da Figura 48, que o `MySQL` demanda um espaço substancial em comparação ao `MongoDB`. No primeiro bloco, ambos ocupam 0,02 MB. No entanto, no segundo cenário, observa-se uma diferenciação: o `MySQL` ocupa 0,09 MB, enquanto o `MongoDB` utiliza apenas 0,03 MB. O terceiro bloco apresenta a diferença mais notável, visto que o `MySQL` consome 1,52 MB para os 10.000 dados, enquanto o `MongoDB` ocupa apenas 0,25 MB. Essas discrepâncias evidenciam que, em termos de eficiência no espaço de armazenamento, o `MongoDB` se destaca.

Figura 48 – Resultados - Espaço em Disco



Fonte: Elaborada pelo autor

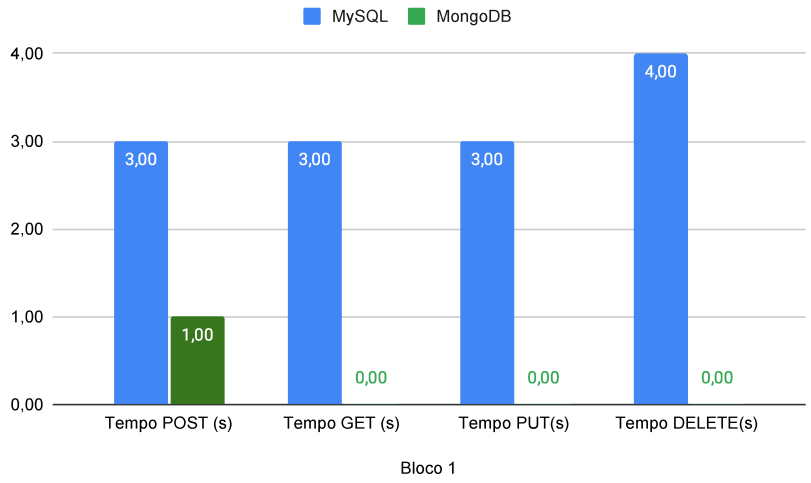
6.2.3 *Resposta da RQ_6: Como os SGBDs respondem em termos de tempo durante os três blocos de testes?*

Com base nos resultados obtidos e representados nos gráfico das Figuras 49, 50 e 51, pode-se destacar algumas observações. No contexto do `MySQL`, no Bloco 1, o tempo de resposta para as operações POST, GET e PUT é de 3 segundos, mantendo-se constante, exceto na operação DELETE, que demanda 4 segundos. No Bloco 2, os tempos aumentam para 5 segundos nas operações POST, PUT e DELETE, mas se elevam para 7 segundos na operação GET. Já no Bloco 3, os tempos de resposta são mais longos, atingindo 27 segundos para POST,

86 segundos para GET, 33 segundos para PUT e 23 segundos para DELETE.

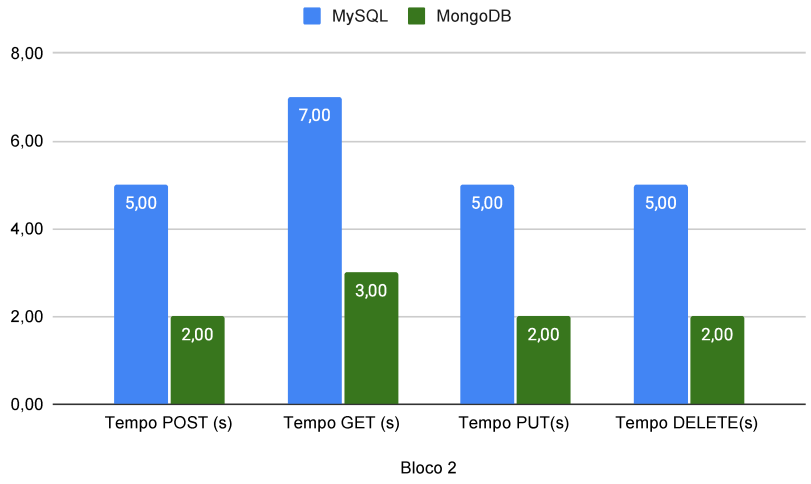
No caso do MongoDB, os tempos de resposta para POST, GET, PUT e DELETE no Bloco 1 são respectivamente 1 segundo, 0 segundo, 0 segundo e 0 segundo. No Bloco 2, esses tempos se elevam para 2 segundos, 3 segundos, 2 segundos e 2 segundos. No Bloco 3, observamos 24 segundos para POST, 51 segundos para GET, 25 segundos para PUT e 23 segundos para DELETE.

Figura 49 – Resultados relativos ao tempo de execução - Bloco 1



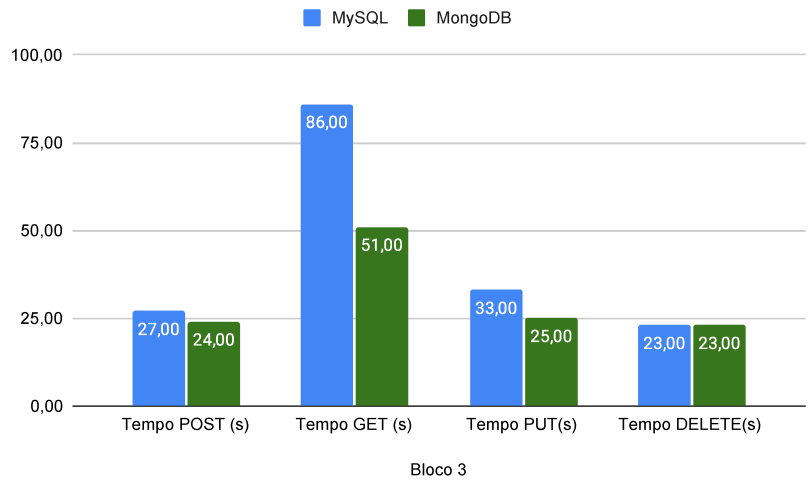
Fonte: Elaborada pelo autor

Figura 50 – Resultados relativos ao tempo de execução - Bloco 2



Fonte: Elaborada pelo autor

Figura 51 – Resultados relativos ao tempo de execução - Bloco 3



Fonte: Elaborada pelo autor

Uma informação importante a ser destacada é que tempos abaixo de 1 segundo foram considerados como 0, pois o comando `top`, que realiza o monitoramento dos recursos, foi configurado para fazer leituras a cada 1 segundo, e como a requisição levou menos tempo que isso, foi registrado como 0 segundos.

Ao comparar esses resultados, percebe-se que o `MongoDB` geralmente apresenta tempos de resposta mais curtos em comparação com o `MySQL` em todas as operações e blocos avaliados. Essa eficiência no tempo de resposta pode ser um fator crucial ao considerar o desempenho e a escolha entre esses dois sistemas de gerenciamento de banco de dados em diferentes cenários de uso.

7 CONCLUSÃO E TRABALHOS FUTUROS

A seleção do Sistema de Gerenciamento de Banco de Dados é uma escolha crítica em qualquer projeto de desenvolvimento de software. O SGBD desempenha um papel fundamental na eficiência e desempenho de uma aplicação, afetando diretamente a experiência do usuário e os custos operacionais. Tendo isto em vista, o presente trabalho teve como objetivo a realização de comparações dinâmicas e estáticas de Sistemas de Gerenciamento de Banco de Dados, a fim de identificar caraterísticas e analisar o comportamento dos sistemas envolvidos, em diferentes cenários. Para alcançar tal objetivo, o trabalho foi dividido em duas partes. A primeira parte tratou de levantar as características relevantes aos SGBDs e compará-los diante das mesmas. Na segunda, foram realizados testes em diferentes cenários, cenários estes, que variavam de acordo com o número de requisições e o tipo de operação a ser realizada (CRUD), conforme descrito na metodologia deste trabalho.

Na análise comparativa estática entre SGBDs, destaca-se a diferença principal em relação ao Teorema de CAP: o `MySQL`, por padrão, assegura Consistência e Disponibilidade (CD), enquanto o `MongoDB` opera por padrão com Consistência de Dados e Tolerância a Particionamento (CP). Além disso, considerando o suporte a diferentes linguagens de programação, o `MongoDB` lidera a classificação, seguido pelo `MySQL`. Quanto à compatibilidade com SO, o `MySQL` suporta uma maior quantidade.

Na análise comparativa dinâmica, ao examinar os resultados em vários cenários, observa-se que, em todos os cenários, independente da quantidade de requisições ou do método HTTP realizado, o desempenho se mantém equivalente. Conclui-se que em relação ao consumo de RAM, CPU, tempo de execução e espaço em disco ocupados pelos dados, o `MongoDB` se destaca como a opção mais eficiente.

Esses resultados destacam a importância de considerar múltiplos aspectos ao escolher um SGBD, alinhando-os com os objetivos do projeto, o orçamento e o ambiente de desenvolvimento. Portanto, a seleção não deve se basear em apenas um critério, mas sim em uma análise abrangente de todas as necessidades do contexto. É fundamental avaliar cada aspecto separadamente, seja ele dinâmico ou estático, para tomar a decisão mais apropriada.

Como próximos passos, deseja-se ampliar a comparação para incluir um maior número de SGBDs não relacionais e considerar realizar consultas de complexidade maior. Isso proporcionará resultados mais próximos da realidade e uma maior variedade de opções na escolha de SGBDs.

REFERÊNCIAS

AMAZON. **O que é um bancos de dados?** 2023. Acessado em 05-junho-2023, disponível em <https://www.oracle.com/br/database/what-is-database/>.

CELESTI, A.; FAZIO, M.; VILLARI, M. A study on join operations in mongodb preserving collections data models for future internet applications. **Future Internet**, v. 11, n. 4, 2019. ISSN 1999-5903. Disponível em: <https://www.mdpi.com/1999-5903/11/4/83>.

CHEN TSE-HSUN E SHANG, W. e. Y. J. e. H. A. E. e. G. M. W. e. N. M. e. F. P. Um estudo empírico sobre a prática de manutenção de código de mapeamento objeto-relacional em sistemas java. In: . [S.l.: s.n.], 2016. Acessado em 05-maio-2023, disponível em <https://dl.acm.org/doi/abs/10.1145/2901739.2901758>.

CLOUD, G. **O que é o MySQL?** 2023. Acessado em 05-setembro-2023, disponível em <https://cloud.google.com/mysql?hl=pt-br>.

CODD, E. F. **A Relational Model of Data for Large Shared Data Banks**. 1969. Acessado em 07-setembro-2023, disponível em <https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf>.

DBENGINES. **DB-Engines Rankin**. 2023. Acessado em 25-abril-2023, disponível em <https://db-engines.com/en/ranking>.

DICODING. **O que são serviços da Web? Junto com definições e exemplos**. 2023. Acessado em 13-junho-2023, disponível em <https://www.dicoding.com/blog/apa-itu-web-service/>.

FORWARD ANDREW E LETHBRIDGE, T. C. **A relevância da documentação, ferramentas e tecnologias de software: uma pesquisa**. 2002. Acessado em 29-agosto-2023, disponível em <https://dl.acm.org/doi/abs/10.1145/585058.585065>.

GARCIA-MOLINA, H.; ULLMAN, J. D.; WIDOM, J. **Database Systems: The Complete Book**. 2nd. ed. [S.l.]: Pearson, 2008. ISBN 9780131873254.

IBM. **SQL Documentation**. 2023. Acessado em 05-maio-2023, disponível em <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview>.

IMASTER. **Confira quais foram as linguagens de programação mais usadas em 2022**. 2023. Acessado em 25-maio-2023, disponível em <https://imasters.com.br/noticia/confira-quais-foram-as-linguagens-de-programacao-mais-usadas-em-2022>.

IONOS, D. G. **SQL JOIN: consultas a varias tablas de datos**. 2018. Acessado em 07-setembro-2023, disponível em <https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/sql-join/>.

JOSE, B.; ABRAHAM, S. Performance analysis of nosql and relational databases with mongodb and mysql. **Materials Today: Proceedings**, v. 24, p. 2036–2043, 2020. ISSN 2214-7853. International Multi-conference on Computing, Communication, Electrical Nanotechnology, I2CN-2K19, 25th 26th April 2019. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2214785320324159>.

KHASAWNEH, T. N.; AL-SAHLEE, M. H.; SAFIA, A. A. **2020 11th International Conference on Information and Communication Systems (ICICS)**. [S.l.: s.n.], 2020. Acessado em 04-maio-2023, disponível em <https://ieeexplore.ieee.org/abstract/document/9078970?casa_token=Vx-Q4r5mySkAAAAA:HmOm018kf_Ff2SCdNEtDu3Y-WQsBpiPg1q7442uALW7NAUVjk8p8YmI5YAhXwEjm1Xf4wV45EA>.

KLEPPMANN, M. **Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems**. [S.l.]: "O'Reilly Media, Inc.", 2017.

LEYENDECKER, D. **Os três níveis da modelagem de dados: conceitual, lógico e físico**. 2023. Acessado em 07-setembro-2023, disponível em <https://www.dio.me/articles/os-tres-niveis-da-modelagem-de-dados-conceitual-logico-e-fisico>.

LISBOA, A. **Qual o sistema operacional de PC mais usado do mundo?** 2023. Acessado em 28-setembro-2023, disponível em <https://www.terra.com.br/byte/qual-o-sistema-operacional-de-pc-mais-usado-do-mundo,d8982ba2f7924362a52083bb8828d757q18ztp5f.html>.

LÓSCIO, B. F.; OLIVEIRA, H. d.; PONTES, J. d. S. Nosql no desenvolvimento de aplicações web colaborativas. **VIII Simpósio Brasileiro de Sistemas Colaborativos**, sn, v. 10, n. 1, p. 11, 2011.

MACÁRIO, C. G. d. N.; BALDO, S. M. O modelo relacional. **Instituto de Computação Unicamp. Campinas**, p. 1–15, 2005.

MACORATTI. **Conceitos Básicos de modelagem de dados**. 2023. Acessado em 11-maio-2023, disponível em <https://www.macoratti.net/cbmd1.htm>.

MANOVICH, L. Banco de dados. **Revista ECO-Pós**, v. 18, n. 1, p. 7–26, 2015.

MATEUS, L. **Quem é melhor? ORM ou SQL puro?** 2023. Acessado em 07-setembro-2023, disponível em <https://dev.to/lucas_jdev/quem-e-melhor-orm-ou-sql-puro-kjb>.

MICROSOFT. **Entity Framework Documentation**. 2023. Acessado em 05-maio-2023, disponível em <https://learn.microsoft.com/en-us/ef/>.

MICROSOFT. **O que são bancos de dados?** 2023. Acessado em 05-junho-2023, disponível em <https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/what-are-databases/>.

MILETTO, E. M.; BERTAGNOLLI, S. de C. **Desenvolvimento de Software II: Introdução ao Desenvolvimento Web com HTML, CSS, JavaScript e PHP-Eixo: Informação e Comunicação-Série Tekne**. [S.l.]: Bookman Editora, 2014.

MONGODB. **MongoDB Documentation**. 2023. Acessado em 28-maio-2023, disponível em <https://www.mongodb.com/docs/>.

MONGODB. **Replicação - MongoDB**. 2023. Acessado em 05-junho-2023, disponível em <https://www.mongodb.com/docs/manual/replication/>.

MONGODB. **What are ACID Properties in Database Management Systems?** 2023. Acessado em 28-setembro-2023, disponível em <https://www.mongodb.com/basics/acid-transactions>.

MOONEY, J. D. **Desenvolvendo software portátil**. 2004.

MOZILLA. **Introdução a WEB**. 2023. Acessado em 25-maio-2023, disponível em <https://developer.mozilla.org/pt-BR/docs/Learn/Getting_started_with_the_web>.

MOZILLA. **Uma visão Geral do HTTP**. 2023. Acessado em 05-maio-2023, disponível em <https://www.ibm.com/docs/en/i/7.2?topic=programming-sql>.

MYSQL. **MySQL Documentation**. 2023. Acessado em 28-maio-2023, disponível em <https://dev.mysql.com/doc/>.

ORACLE. **O que é um bancos de dados?** 2023. Acessado em 05-junho-2023, disponível em <https://www.oracle.com/br/database/what-is-database/>.

RAMAKRISHNAN, R.; GEHRKE, J. **Database Management Systems**. [S.l.]: McGraw-Hill Education, 2008. v. 3.

REDHAT. **O que é API?** 2023. Acessado em 28-outubro-2023, disponível em <https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces>.

REDIS. **Redis Documentation**. 2023. Acessado em 28-maio-2023, disponível em <https://redis.io/docs/>.

SANTOS, L. N. **O teorema de CAP – Utilizado para auxiliar um servidor de sistema distribuídos**. 2017. Acessado em 25-abril-2023, disponível em <https://medium.com/@lucasnsantos/o-teorema-de-cap-utilizado-para-auxiliar-um-servidor-de-sistema-distribu%C3%ADdos-f73d6a4c5d9>.

SEBESTA, R. W. **Conceitos de Linguagens de Programação-11**. [S.l.]: Bookman Editora, 2018.

UFRN, I. M. D. **História dos bancos de dados**. 2023. Acessado em 05-maio-2023, disponível em <https://materialpublic.imd.ufrn.br/curso/disciplina/4/56/1/3>.

VOLPATO, B. **Ranking: as redes sociais mais usadas no Brasil e no mundo em 2023, com insights, ferramentas e materiais**. 2023. Acessado em 25-maio-2023, disponível em <https://resultadosdigitais.com.br/marketing/redes-sociais-mais-usadas-no-brasil/>.

W3C. **WEB. Como tudo começou**. 2023. Acessado em 26-maio-2023, disponível em <https://www.w3.org/2004/Talks/w3c10-HowItAllStarted/?toc=true>.

WIKIPéDIA. **Modelo Cliente-Servidor**. 2023. Acessado em 07-setembro-2023, disponível em <https://pt.wikipedia.org/wiki/Modelo_cliente%E2%80%93servidor>.