UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

NATANAEL RODRIGO ZIMMER

Análise Comparativa de Desempenho de um Banco de Dados Relacional em Diferentes Linguagens de Programação

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Profa. Dra. Renata Galante

Porto Alegre
2023

“ We’re still pioneers, we barely begun. Our greatest accomplishments cannot be behind us, cause our destiny lies above us.” — JOSEPH A. COOPER

INTERSTELLAR

AGRADECIMENTOS

Quero expressar meu profundo agradecimento aos meus queridos pais, Tamara e Heleno. Eles sempre me mostraram que a educação é a chave para alcançar sonhos grandiosos. Um agradecimento especial à minha mãe, que nunca permitiu que eu desistisse dessa jornada e me apoiou de maneira incondicional em todas as escolhas que fiz.

À minha amada noiva Bibiana, quero agradecer por segurar minha mão e nunca soltá-la, me ajudando a levantar em cada queda da vida.

À minha orientadora Renata Galante, obrigado pela oportunidade e por sua paciência ao corrigir meus textos e responder as minhas dúvidas. Seu apoio foi de imensa valia.

Aos professores que fizeram parte dessa trajetória, meu muito obrigado. Vocês me deram a oportunidade de realizar meu sonho e sou imensamente grato por isso.

Por fim, quero expressar minha gratidão a todos os meus amigos e família que estiveram ao meu lado nessa jornada. Vocês me apoiaram e trouxeram momentos de descontração e alegria que fizeram toda a diferença.

RESUMO

A cada dia, novas linguagens de programação surgem, refletindo uma evolução constante nesse campo. Simultaneamente, a quantidade e diversidade de dados que precisamos processar aumentam substancialmente. Para se adaptar a essas mudanças, os sistemas de bancos de dados também evoluíram, visando fornecer um suporte mais eficaz aos desenvolvedores na criação de sistemas capazes de armazenar essas novas informações. Consequentemente, é de suma importância conduzir estudos comparativos entre as diversas linguagens para compreender os cenários nos quais cada linguagem se adapta melhor para acessar esses dados. Este trabalho tem como objetivo principal comparar o desempenho de duas linguagens de programação distintas: uma compilada e outra interpretada. A comparação é realizada no contexto do acesso a um banco de dados relacional. Aspectos como modelagem e design, criação do banco de dados, inserção de dados, consultas e a utilização de diferentes motores de banco de dados e *drivers* de conexão são considerados. Em termos gerais, as consultas não apresentaram diferenças de desempenho significativas. No entanto, ao avaliar outros aspectos, podem-se identificar discrepâncias que devem ser ponderadas ao decidir entre uma linguagem ou outra para uma aplicação específica.

Palavras-chave: banco de dados. modelagem relacional. desempenho. MySQL. C. python.

Comparative Performance Analysis of a Relational Database Across Different Programming Languages

ABSTRACT

Every day, new programming languages emerge, reflecting a constant evolution in this field. Simultaneously, the quantity and diversity of data we need to process substantially increase. To adapt to these changes, database systems have also evolved, aiming to provide more effective support to developers in creating systems capable of storing this new information. Consequently, it is of paramount importance to conduct comparative studies among various languages to understand the scenarios in which each language best adapts for accessing this data. The main objective of this work is to compare the performance of two distinct programming languages: one compiled and other interpreted. The comparison is conducted in the context of accessing a relational database. Aspects such as modeling and design, database creation, data insertion, queries, and the utilization of different database engines and connection drivers are considered. In general terms, queries did not exhibit significant performance differences. However, when evaluating other aspects, discrepancies can be identified that should be weighed when deciding between one language or another for a specific application.

Keywords: database, relational modeling, performance, MySQL, C, python.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SGBD	Sistema de Gerenciamento de Banco de Dados
IBM	International Business Machines
SGBDs	Sistemas de Gerenciamento de Banco de Dados
UBC	Universidade da Colúmbia Britânica
RDBMS	Relational Database Management System
SQL	Structured Query Language
ANSI	American National Standards Institute
ISO	International Organization for Standardization
ODBC	Open Database Connectivity
CGI	Common Gateway Interface
NoSQL	Not only SQL
INPE	Instituto Nacional de Pesquisas Espaciais
FRP	Fire Radiative Power
API	Application Programming Interface
GCC	GNU Compiler Collection

SUMÁRIO

1 INTRODUÇÃO

O Sistema Gerenciador de Banco de Dados (SGBD) passou a ser aplicado em meados da década de 1960, com fomento da IBM (*International Business Machines*), tendo o objetivo de flexibilizar o acesso aos dados e consultas (SILBERSCHATZ; KORTH; SUDARSHAN, 2010).

Existem no mercado vários tipos de SGBDs relacionais, ou seja, que estruturam os dados visualizados pelo usuário em tabelas ou relações; e não relacionais, que possuem configuração mais complexa (MARTINS FILHO, 2015).

Atualmente, ouvimos cada vez mais falar em *Big Data*, *Internet of Things*, computação em nuvem, e tudo cada dia envolve um maior volume e variedade de dados. Esses dados, sejam eles gerados em nuvem, dispositivos móveis ou até mesmo pelas redes sociais, necessitam de uma forma de modelagem e armazenamento. Como podemos ver em (HAN et al., 2011), com o avanço da tecnologia durante os últimos anos, principalmente nas áreas relacionadas a tratamento de dados, houve a necessidade de aprimorar os bancos de dados a fim de que eles fossem capazes de atender as novas demandas.

Com o advento das redes sociais e o crescente desenvolvimento do comércio eletrônico através da Internet, os Bancos de Dados têm experimentado um notável aumento em suas dimensões (PENTEADO et al., 2014). A título de exemplo, podemos mencionar o Twitter[1] e o Facebook[2], duas plataformas que oferecem serviços de recomendação de amigos próximos e inclusão de hashtags, operações de natureza complexa e morosa quando executadas em um ambiente de base de dados relacional (ROBINSON; WEBBER; EIFREM, 2015).

Dentre os bancos de dados mais utilizados, tem-se o MySQL, popular no mundo todo, possuindo compatibilidade com diversas linguagens de programação, portabilidade, facilidade de uso, excelente desempenho e estabilidade (ORACLE, 2023a). É um banco de dados multiprocessado capaz de gerenciar grandes quantidades de dados.

Diferentes linguagens de programação tem desempenhos diferentes devido aos diferentes paradigmas, métodos de gerenciamento de memória, tipagem, estratégias de execução, dentre outros (ALOMARI et al., 2015). Nesse trabalho, vamos analizar o impacto no desempenho de diferentes linguagens na execução de consultas ou inserções em um mesmo SGBD.

[1] <https://www.twitter.com>
[2] <https://www.facebook.com>

1.1 Problema de pesquisa

A utilização de diferentes linguagens de programação para a manipulação de grandes quantidades de dados utilizando um banco de dados relacional é algo comum no dia a dia dos desenvolvedores. Porém, mesmo que a execução das consultas ocorra no SGBD, podemos nos questionar: qual seria o impacto de desempenho ao realizar as mesmas consultas em diferentes linguagens de programação?

1.2 Objetivo

O objetivo deste trabalho é realizar uma comparação de desempenho entre a execução de consultas SQL utilizando as linguagens C e Python em um banco de dados relacional MySQL. As linguagens foram escolhidas por terem diferentes modos de execução, compilada e interpretada, além de que são as 2 primeiras linguagens no *ranking IEEE Spectrum - Top Programming Languages* de 2022[3].

O estudo foi realizado em condições equivalentes de manipulação, utilizando uma única instância do banco de dados, aproveitando os diferentes motores fornecidos pelo MySQL para verificar a eficiência relativa das linguagens nessas condições. Como objetivos buscou-se medir o impacto da linguagem e dos *drivers* na execução das consultas em uma mesma base de dados, que foi preparada de forma a ser utilizada em ambas as linguagens, garantindo resultados mais confiáveis ao trabalhar com dados semelhantes.

Durante os experimentos, foram utilizados recursos nativos do MySQL, como índices e suporte a diferentes motores, com o objetivo de melhorar o desempenho das consultas. Diversos cenários foram analisados, envolvendo consultas em diversos campos dos dados. Foram obtidos tempos de execução e calculadas estatísticas, permitindo a apresentação de um comparativo de desempenho nos diferentes cenários observados.

Ao realizar essa comparação, o estudo pretende fornecer indicativos sobre como os *drivers* e as linguagens C e Python se comportam em termos de eficiência ao executar consultas em um banco de dados relacional. As conclusões obtidas auxiliam que desenvolvedores e pesquisadores tomem decisões informadas sobre a escolha da linguagem mais adequada para suas necessidades específicas.

[3] <https://spectrum.ieee.org/top-programming-languages-2022>

1.2.1 Objetivo Geral

Realizar uma análise comparatória entre o desempenho das linguagens C e Python e dos *drivers* ao executar consultas no banco de dados relacional MySQL, trabalhando e manipulando dados em condições equivalentes.

1.2.2 Objetivos Específicos

Para realizar a análise comparatória entre o desempenho dos *drivers* nas linguagens C e Python ao executar consultas no banco de dados relacional MySQL, focando na recuperação e inserção de dados, podemos adotar as seguintes abordagens:

- Análise de recuperação de dados: construir um conjunto de consultas SQL que envolvam recuperação de dados em condições equivalentes, como seleção de registros, filtragem com cláusulas *WHERE*, junção de tabelas e ordenação.
- Análise de inserção de dados: criar uma estrutura de dados representando um conjunto de registros a serem inseridos no banco de dados.

Ao final da análise, será possível obter uma visão mais clara e detalhada do desempenho das linguagens C e Python junto aos *drivers* ao trabalhar com recuperação e inserção de dados no MySQL, bem como entender os efeitos do uso de transações em ambas as linguagens. Isso permitirá tomar decisões mais informadas ao escolher a linguagem mais adequada para um projeto específico.

1.3 Justificativa

A motivação para realizar a análise comparatória entre o desempenho dos *drivers* junto as linguagens C e Python ao executar consultas no banco de dados relacional MySQL é poder considerar o desempenho do *driver* de conexão utilizado, fundamentando-se na lacuna existente na literatura em relação a esse aspecto específico.

Como mencionado, a maioria dos testes de desempenho se concentra apenas na cobertura e comparação do desempenho dos bancos de dados em si, deixando de lado a avaliação do impacto do *driver* de conexão. No entanto, o *driver* de conexão é uma parte essencial do processo de acesso e manipulação dos dados em um banco de dados.

A escolha da linguagem de programação pode influenciar o desempenho do banco de dados, uma vez que diferentes linguagens possuem *driver*s de conexão distintos. Ao isolar o *driver* de conexão como a variável diferencial entre as linguagens, é possível identificar se existe alguma diferença significativa no desempenho do banco de dados devido à escolha da linguagem para fazer as consultas.

Ao realizar esses testes selecionados buscamos uma compreensão mais abrangente dos fatores que afetam o desempenho de bancos de dados relacionais em cenários reais de desenvolvimento. Essas descobertas podem levar a melhorias na seleção adequada de linguagens e *driver*s para cada projeto, visando alcançar maior eficiência e otimização nos sistemas.

1.4 Organização do texto

A estrutura do texto é organizada da seguinte maneira:

No Capítulo 2, é fornecida uma fundamentação teórica abrangente. Uma introdução concisa é oferecida ao leitor sobre os principais eventos históricos relacionados a bancos de dados, acompanhada por uma explicação sobre o funcionamento dos bancos de dados e as linguagens de programação envolvidas. Além disso, essa seção abrange também a revisão de trabalhos relacionados, comparando suas semelhanças e discrepâncias.

O Capítulo 3 é dedicado à apresentação da proposta de mapeamento dos dados, juntamente com a delimitação do seu domínio. Logo após, é exposta a modelagem dos dados, com foco na abordagem de banco de dados relacional.

O Capítulo 4 se concentra na realização de testes comparativos de desempenho no acesso ao SGBD através de duas linguagens de programação distintas. Essa análise é conduzida utilizando ambos os motores de armazenamento selecionados. Além disso, essa seção aborda eventuais desafios e problemas encontrados durante a execução dos testes.

A conclusão do trabalho é apresentada no Capítulo 5. Aqui, são sumarizados os resultados e conclusões obtidos ao longo da pesquisa. Além disso, esse capítulo oferece sugestões para trabalhos futuros, indicando direções para aprofundar o estudo ou explorar áreas relacionadas que possam contribuir para o avanço do conhecimento na temática tratada.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo trata dos fundamentos dos bancos de dados, bem como a história da sua criação e evolução, com enfoque especial nos bancos de dados relacionais e suas principais características. A Seção 2.1 apresenta uma visão panorâmica da história da criação dos bancos de dados até os dias atuais. Na Seção 2.2, a abordagem do Modelo Relacional é discutida em maior detalhe, enfatizando suas principais características e motivações, bem como os diferentes modelos propostos dentro dessa abordagem. A Seção 2.3 aborda o Sistema de Gerenciamento de Banco de Dados (SGBD) e as linguagens de programação selecionadas para este trabalho. Finalmente, na Seção 2.4, são apresentados alguns trabalhos relacionados, destacando seus principais pontos e estabelecendo conexões com o presente trabalho.

2.1 História

Os sistemas de gerenciamento de banco de dados modernos carregam consigo uma história que remonta as civilizações antigas. Atendendo às demandas de cada época, sumérios, egípcios e outros, desenvolveram técnicas de contabilidade, as quais lhes permitiam acompanhar e compreender detalhadamente os dados que permeavam seu cotidiano. A partir dessas iniciativas históricas, deu-se início a uma trajetória evolutiva que nos conduziu aos modernos sistemas de gerenciamento de bancos de dados que conhecemos hoje (BERG; SEYMOUR; GOEL, 2012).

Nos tempos modernos, antes da disseminação generalizada da computação, foram desenvolvidos sistemas elaborados de gerenciamento de dados para atender às necessidades específicas de escritórios governamentais, hospitais e bibliotecas. Um exemplo notável é o Sistema Decimal de Dewey, um sistema de classificação de documentos concebido pelo bibliotecário americano Melvil Dewey (DEWEY, 1876), que se tornou o sistema de classificação de bibliotecas mais amplamente utilizado em todo o mundo(OCLC, Inc.,). Embora tenha sido desenvolvido há muito tempo, alguns dos princípios fundamentais desse sistema ainda são aplicados nos softwares de banco de dados utilizados atualmente.

A história dos computadores e dos bancos de dados está intrinsecamente conectada. O surgimento e popularização dos bancos de dados modernos coincidiram com o desenvolvimento da computação, especialmente quando os computadores começaram a se tornar uma opção viável em termos de custo-benefício para organizações privadas.

À medida que o custo de armazenar dados em computadores diminuiu, a migração dos dados do papel para os sistemas computacionais tornou-se mais acessível e prática.

Na década de 1960, um dos primeiros Sistemas de Gerenciamento de Bancos de Dados (SGBDs) a alcançar sucesso comercial foi o SABRE, desenvolvido pela IBM. Esse sistema foi amplamente utilizado pela American Airlines para gerenciar os dados de reservas (Sabre GLBL Inc., 2017), representando um marco significativo no uso de tecnologias de banco de dados em escala empresarial. O SABRE possibilitou um novo nível de eficiência e organização na gestão das reservas de voos, demonstrando o potencial impacto positivo que os bancos de dados poderiam trazer para o gerenciamento de informações em diversas áreas de negócios.

O modelo relacional foi o primeiro modelo de organização de dados proposto teoricamente, baseado na lógica de predicados e na teoria dos conjuntos. Foi apresentado em 1970 por Edgar Frank Codd, um funcionário da IBM, no artigo intitulado "*A Relational Model of Data for Large Shared Data Banks*". Esse artigo teve um impacto significativo, alterando profundamente a perspectiva das pessoas em relação aos bancos de dados. No modelo relacional, o esquema do banco de dados, ou seja, a organização lógica dos dados em tabelas, relacionamentos e restrições, é dissociado do armazenamento físico das informações. Essa abordagem proporcionou maior flexibilidade e facilidade na manipulação dos dados. Consequentemente, essa separação tornou-se o princípio padrão adotado nos sistemas de banco de dados modernos.

Entre os anos de 1974 e 1977, dois notáveis protótipos de sistemas de banco de dados relacionais foram criados: o Ingres, desenvolvido na Universidade da Colúmbia Britânica (UBC), e o System R, criado nas instalações da IBM em San Jose (COUNCIL; BOARD; HISTORY, 1999). O Ingres adotou uma linguagem de consulta denominada QUEL, que posteriormente impulsionou o desenvolvimento de sistemas como Ingres Corp., MS SQL Server e Sybase, dentre outros. Por outro lado, o System R utilizou a linguagem de consulta SEQUEL e teve um papel fundamental no desenvolvimento de sistemas como SQL/DS, DB2, Allbase, Oracle e Non-Stop SQL. Foi nessa década que o termo "*Relational Database Management System*" (RDBMS) tornou-se amplamente reconhecido, marcando o advento e a consolidação dos sistemas de gerenciamento de bancos de dados relacionais. A contribuição desses protótipos e das linguagens de consulta QUEL e SEQUEL foi vital para a evolução dos bancos de dados relacionais e sua disseminação em diversos ambientes e aplicações, revolucionando a forma como os dados eram armazenados, acessados e manipulados nas organizações.

A linguagem de consulta estruturada, mais conhecida como SQL (Structured Query Language), conquistou o *status* de linguagem de consulta padrão, sendo selecionada pelo *American National Standards Institute* (ANSI) em 1986 e posteriormente adotada pela *International Organization for Standardization* (ISO) em 1987 (ISO, 1987). Essa padronização trouxe uma uniformidade essencial ao mundo dos bancos de dados, permitindo que os sistemas de gerenciamento de banco de dados relacionais pudessem interagir de maneira mais harmoniosa, independentemente do fornecedor ou plataforma utilizada. A escolha do SQL como linguagem de consulta padrão contribuiu para a disseminação ainda maior dos sistemas de banco de dados relacionais, consolidando sua posição como um dos pilares fundamentais da tecnologia de informação e do gerenciamento de dados em todo o mundo. Através do SQL, os desenvolvedores e usuários podem realizar consultas, atualizações e operações complexas de forma consistente e padronizada, tornando o acesso e manipulação de informações mais acessíveis e eficientes.

A década de 1990 desempenhou um papel de suma importância no avanço dos bancos de dados e do software relacionado. Assim como ocorreu na década de 1960, uma mudança cultural mais abrangente impulsionou novos desenvolvimentos na indústria de bancos de dados. Após um período de turbulência, as empresas sobreviventes passaram a comercializar produtos complexos de banco de dados a preços mais elevados (BERG; SEYMOUR; GOEL, 2012). Esse contexto também testemunhou o lançamento de novas ferramentas de cliente para o desenvolvimento de aplicativos, que existem até os dias de hoje, incluindo o Oracle Developer[1], PowerBuilder[2], Visual Basic[3] e outros, que permitiram um maior nível de sofisticação e agilidade na criação de aplicações. Paralelamente, diversas ferramentas de produtividade pessoal foram desenvolvidas, tais como ODBC (*Open Database Connectivity*) e a integração do Excel/Access com bancos de dados. Essas inovações contribuíram para a facilitação do acesso e manipulação de dados, permitindo aos usuários aproveitar ao máximo as informações armazenadas nos sistemas de bancos de dados. Ademais, o início dos anos 1990 viu o surgimento de protótipos para Sistemas de Gerenciamento de Bancos de Dados Orientados a Objetos. Esses sistemas buscavam aprimorar a maneira como os dados eram tratados, permitindo o armazenamento de objetos e suas relações diretamente no banco de dados, em vez de utilizar uma abordagem puramente relacional.

Em meados da mesma década, o advento da Internet provocou um crescimento ex-

[1] <https://docs.oracle.com/en/database/oracle/sql-developer/>
[2] <https://www.appeon.com/products/powerbuilder>
[3] <https://learn.microsoft.com/en-us/dotnet/visual-basic/>

ponencial da indústria de bancos de dados. Os usuários comuns de computadores pessoais passaram a utilizar sistemas de banco de dados cliente-servidor para acessar sistemas de computador que continham dados legados. Com o aumento do número de usuários adquirindo computadores pessoais e conectando-se à Internet, aprimorar os bancos de dados tornou-se uma necessidade premente.

No final dos anos 1990, o crescente investimento em negócios online resultou em uma demanda significativa por conectores de banco de dados voltados para a Internet, tais como Front Page, Active Server Pages, Java Servlets, Dream Weaver, ColdFusion, Enterprise Java Beans e Oracle Developer 2000. A utilização de tecnologias como CGI (Common Gateway Interface), MySQL, Apache e outros sistemas trouxe soluções de código aberto para a Internet, ampliando as opções e possibilidades disponíveis no mercado.

Concomitantemente, o aumento do uso da tecnologia de ponto de venda impulsionou o amadurecimento do processamento de transações online e do processamento analítico online, tornando essas áreas cada vez mais sofisticadas e eficientes no cenário digital.

Desde a década de 1980, o SQL assumiu o papel de linguagem de consulta padrão nos sistemas de banco de dados. No entanto, em 1998, Carlo Strozzi utilizou o termo "NoSQL" ao batizar seu banco de dados "Strozzi NoSQL" (LITH; MATTSSON, 2010). Nesse contexto inicial, a oferta ainda mantinha sua natureza relacional. Foi apenas em 2009 que o termo NoSQL ressurgiu no vocabulário da indústria, quando o desenvolvedor Johan Oskarsson organizou um evento dedicado à discussão de bancos de dados não relacionais, ressignificando o termo NoSQL para "Not only SQL". A partir desse ponto, o conceito de NoSQL ganhou popularidade e permaneceu em evidência no âmbito da tecnologia da informação. Atualmente, existem diversos bancos de dados que se enquadram nessa definição, abrangendo uma variedade de abordagens e tecnologias. Esses sistemas não relacionais oferecem alternativas e soluções inovadoras para o armazenamento, gerenciamento e análise de dados, o que contribuiu significativamente para a evolução do cenário de bancos de dados e sua adaptação às demandas contemporâneas.

Hoje, os bancos de dados desempenham um papel onipresente e essencial em nossa vida cotidiana, contribuindo para melhorar diversos aspectos de nossa rotina. Desde o armazenamento em nuvem pessoal até a previsão do tempo, muitos dos serviços que utilizamos atualmente são possíveis graças aos bancos de dados. O uso desses sistemas é diversificado e abrange uma ampla gama de aplicações que impactam significativamente o nosso dia a dia. Podemos observar uma crescente diversificação no mercado de bancos de

dados não relacionais, com diversos novos fornecedores oferecendo soluções específicas para atender a diferentes necessidades. Paralelamente aos tradicionais bancos de dados relacionais, como Oracle, MySQL e DB2, surgem novas tendências focadas em tornar a tecnologia mais acessível e poderosa para todos os usuários.

Essa abordagem mais acessível e voltada para o usuário tem o potencial de democratizar o uso de bancos de dados, permitindo que um público mais amplo explore o poder e os benefícios dessa tecnologia em diferentes contextos e aplicações.

2.2 Modelo Relacional

Conforme originalmente conceituado por Codd (CODD, 1970), um banco de dados relacional é uma estrutura que organiza os dados em tabelas, operadas por linguagens de programação que se fundamentam na teoria dos conjuntos matemáticos para consultar essas tabelas. A abordagem de Codd exerceu uma profunda influência sobre a construção de sistemas de banco de dados e, igualmente importante, sobre a maneira como esses bancos de dados são descritos em livros didáticos e ensinados aos alunos.

Bancos de dados representam estruturas de arquivos organizadas com o propósito fundamental de armazenar informações. Alguns bancos de dados possuem uma característica especial que substancialmente aprimora o desempenho das operações de escrita e leitura de dados. Essa particularidade reside na própria estrutura do banco de dados, que permite a associação de partes do banco com outras partes. Um banco de dados relacional organiza seus dados em relações. Cada relação pode ser visualizada como uma tabela, na qual cada coluna corresponde a atributos da relação, e as linhas representam as tuplas ou elementos da relação (RICARTE, 2002).

O Modelo Relacional é altamente flexível e adequado para resolver diversas questões no nível do projeto e da implementação do banco de dados. Os dados são organizados em um conjunto de tabelas com colunas e linhas. As tabelas têm a finalidade de armazenar informações sobre os objetos a serem representados no banco de dados. Cada coluna em uma tabela retém um tipo específico de dado, e cada campo armazena o valor do atributo correspondente. As linhas na tabela representam coleções de valores relacionados de um objeto ou entidade. Cada linha em uma tabela pode ser identificada de forma exclusiva por meio de uma chave primária. Além disso, as linhas entre várias tabelas podem ser relacionadas através do uso de chaves estrangeiras.

Esses dados podem ser acessados de várias maneiras diferentes, sem que seja

necessário reorganizar as próprias tabelas do banco de dados. O modelo de banco de dados relacional é amplamente utilizado para armazenar dados em bancos de dados. Esse modelo baseia-se no armazenamento de dados em tabelas, permitindo que as tabelas se interliguem entre si (SILBERSCHATZ; KORTH; SUDARSHAN, 2010).

Sua estrutura é composta por relações (tabelas). Cada relação é formada por um ou mais atributos (campos) que definem o tipo de dados a serem armazenados. Cada instância do esquema (linha) é denominada tupla (registro). O modelo relacional não possui caminhos pré-definidos para acessar os dados, diferentemente dos modelos que o precederam. Para trabalhar com as tabelas, algumas restrições foram impostas no modelo relacional a fim de evitar aspectos indesejáveis, como repetição de informação, incapacidade de representar partes da informação e perda de dados. Essas restrições incluem a integridade referencial, chaves e integridade das junções entre relações.

Os dados são independentes dos programas aplicativos, uma vez que o Sistema Gerenciador de Banco de Dados tem a capacidade de distinguir as visões físicas e lógicas, permitindo que os programas utilizem os dados de um banco de dados compartilhado. Todas as operações realizadas em um banco de dados são intermediadas pelo SGBD. Ele é responsável por salvar os dados no disco rígido, manter em memória os dados mais frequentemente acessados, gerenciar a conexão entre dados e metadados, sendo encarregado de todo o gerenciamento do banco de dados. O acesso às informações e o modo de uso tornam-se mais simples, uma vez que o banco de dados estabelece relacionamentos entre diferentes registros e aplicações que podem ser acessados e combinados. A organização dos arquivos e dos bancos de dados influencia significativamente a forma como as informações podem ser fornecidas, tornando essencial que o projeto de um banco de dados seja realizado com muita atenção (LAUDON; LAUDON, 2011).

2.3 MySQL

O MySQL foi criado na Suécia por David Axmark, Allan Larsson e o finlandês Michael Widenius, tendo o projeto iniciado em 1980. Trata-se de um Sistema de Gerenciamento de Banco de Dados (SGBD) que utiliza a linguagem SQL como interface. É amplamente reconhecido como um dos sistemas gerenciadores de banco de dados mais utilizados pelas empresas, destacando-se como um dos melhores em sua categoria. Ele oferece opções de licença tanto paga quanto livre, o que o torna atraente para empresas de pequeno, médio e grande porte, proporcionando vantagens competitivas em relação a

seus concorrentes (MILANI, 2007).

Destacam-se como características do MySQL sua escalabilidade e estabilidade, bem como seu alto desempenho na inserção e recuperação de dados. Além disso, o sistema é elogiado pela facilidade de escalonamento e manutenção de bases de dados, auxiliado por diversos softwares que simplificam esse processo (NEWMAN, 2006).

Uma das características distintivas do MySQL é o suporte que ele oferece a diversas plataformas, incluindo Linux, UNIX e Windows. Além disso, ele é capaz de fornecer suporte para uma variedade de tipos de aplicações, independentemente de serem aplicações web, desktop ou embarcadas, e também não importa a linguagem de programação utilizada. Em termos de autenticação do banco de dados, o MySQL disponibiliza mecanismos poderosos para assegurar que somente usuários autorizados tenham acesso.

Considerado como um banco de dados robusto, o MySQL atende a todos os requisitos necessários para uma utilização altamente profissional. Atualmente, está presente em praticamente todos os servidores de hospedagem ao redor do mundo. Optar pelo MySQL significa utilizar um SGBD profissional, flexível, seguro, robusto e com custo zero para a empresa. O MySQL possui um extenso portfólio de clientes e usuários, contando com centenas de grandes projetos bem-sucedidos em todo o mundo. Um exemplo notável é o uso da Nasa, que comprova a confiança e a eficácia do sistema em projetos de grande porte (LOBO, 2008).

O servidor MySQL foi originalmente projetado para lidar com bancos de dados muito grandes de maneira mais rápida do que as soluções existentes, e sua eficácia tem sido comprovada em ambientes de alta demanda por muitos anos. Uma característica distintiva do MySQL é a disponibilidade de vários tipos de motores para armazenar dados, cada um com suas próprias características. Essa variedade de motores oferece a flexibilidade de escolher o tipo adequado para cada situação específica. Enquanto alguns tipos priorizam a velocidade de acesso, outros são mais adequados para armazenar grandes volumes de dados (MILANI, 2007).

O MySQL é extremamente poderoso, com alto poder de execução e armazenamento, sendo capaz de lidar com mais de um bilhão de consultas por dia ou processar milhares de transações por minuto. Essa capacidade é fundamental para empresas que constantemente inserem e consultam informações (NEWMAN, 2006).

Para utilizar o MySQL, é necessário instalar um servidor e uma aplicação cliente. O servidor é responsável por armazenar os dados, responder às requisições, controlar a consistência dos dados e executar transações concorrentes, entre outras tarefas. O cliente,

por sua vez, se comunica com o servidor através da linguagem SQL. A versão gratuita do MySQL é conhecida como Edição da Comunidade e inclui tanto o servidor quanto uma interface gráfica cliente.

2.4 Linguagem C

Com toda a sua história e impacto significativo no mundo do desenvolvimento de software, a linguagem C se consolidou como uma das mais importantes e influentes na computação. Sua origem no Bell Labs, com Dennis Ritchie, para criar o sistema operacional Unix, evidencia sua capacidade de gerenciar com eficiência os recursos de *hardware* e sua portabilidade em diferentes plataformas.

Como uma linguagem procedural, o C oferece acesso direto a endereços de memória e operações de baixo nível, o que o torna uma opção ideal para programação de sistemas. Sua padronização, começando com o ANSI C e mais tarde com o padrão ISO C, garantiu compatibilidade e consistência entre diferentes compiladores e plataformas.

A rica biblioteca padrão do C contém uma ampla variedade de funções para manipulação de *strings*, operações de entrada/saída, cálculos matemáticos e muito mais. Além disso, os desenvolvedores podem criar suas próprias bibliotecas personalizadas para estender ainda mais a funcionalidade da linguagem.

A natureza de baixo nível do C, aliada ao acesso direto aos recursos de *hardware*, proporciona a capacidade de escrever códigos altamente eficientes. Isso torna o C uma escolha ideal para aplicativos de desempenho crítico, como sistemas operacionais, mecanismos de jogos e simulações científicas.

Devido à sua abordagem procedural e portabilidade, o C influenciou inúmeras outras linguagens de programação, tornando-se uma base sólida para o desenvolvimento de novas tecnologias e sistemas. Com todas essas características e seu histórico notável, o C continua sendo uma ferramenta poderosa e valiosa para a comunidade de desenvolvedores, comprovando sua importância ao longo dos anos.

2.5 Python

Python é de uma linguagem de programação notável, amplamente apreciada por sua simplicidade e sintaxe clara. Seu design eficiente inclui estruturas de dados de alto ní-

vel, como listas, dicionários, e manipulação de data/hora, tornando-a uma escolha versátil para diversos propósitos. Além disso, sua tipagem dinâmica permite uma maior flexibilidade durante a execução do código, tornando-a ideal para desenvolvimento rápido de aplicações em várias áreas.

Sendo uma linguagem interpretada, Python permite a interação direta com o código, o que agiliza a escrita e execução de scripts. É também uma linguagem orientada a objetos, permitindo o uso de herança múltipla, exceções e tratamento de erros de maneira eficiente. Isso torna o desenvolvimento de software mais robusto e organizado.

Python é conhecido por sua vasta coleção de módulos prontos para utilização, que são extensões pré-escritas que permitem a realização de tarefas específicas de forma rápida e fácil. Esses módulos abrangem várias áreas, desde o desenvolvimento de interfaces gráficas até a manipulação de arquivos e redes, facilitando o trabalho do desenvolvedor.

Outra característica marcante é a ampla interface com bibliotecas externas e sistemas de janela, o que aumenta a capacidade de integração com outras tecnologias e plataformas.

O suporte aos paradigmas de programação orientada a objeto e procedural dá aos desenvolvedores a flexibilidade de escolher a abordagem mais adequada para suas necessidades.

2.6 Trabalhos Relacionados

Nesta seção, empreenderemos uma análise dos trabalhos correlatos, abordando alguns dos pontos de destaque em cada pesquisa e estabelecendo conexões com o presente estudo. notável que a maior parte dos estudos previamente publicados concentra-se na comparação direta entre Sistemas de Gerenciamento de Bancos de Dados (SGBDs), frequentemente deixando à margem a consideração de que o desempenho é influenciado pelo *driver* de conexão ou pela linguagem de programação utilizada.

No estudo conduzido por Carriconde (2016), encontramos uma comparação direta entre dois Sistemas de Gerenciamento de Bancos de Dados (SGBDs) relacionais: o Postgres e o MySQL. O foco central dessa pesquisa reside na análise do desempenho desses SGBDs em dois sistemas operacionais distintos. Para atingir esse objetivo, o autor investiga a eficiência dos SGBDs em operações fundamentais, como inserção, exclusão, atualização e seleção, utilizando conjuntos de dados sintéticos gerados pelo próprio autor. Os testes executados possibilitaram ao autor a conclusão de que, em ambos os sistemas

operacionais, o Postgres demonstrou um desempenho superior em todas as operações avaliadas, quando comparado ao MySQL.

No estudo realizado por Oliveira (2019), foi empreendida uma análise comparativa entre Sistemas Gerenciadores de Bancos de Dados (SGBDs) relacionais e SGBDs orientados a grafos, com foco em aplicações que possuem um elevado grau de interdependência entre os dados. Para alcançar tal objetivo, a autora escolheu o MySQL e o Neo4j como representantes desses dois tipos de SGBDs. Os resultados dessa pesquisa evidenciam que a utilização de SGBDs orientados a grafos oferece vantagens para buscas em conjuntos de dados fortemente interligados, especialmente quando as consultas envolvem exploração em ramos dos grafos ou em profundidade, categorizadas como consultas estruturais neste estudo. Contudo, nos casos em que as consultas requerem a busca obrigatória em todas as entidades do conjunto de dados, denominadas como consultas analíticas, o modelo relacional demonstrou maior eficiência. Dessa forma a autora sugere que a integração entre esses dois modelos de SGBD representa uma solução interessante para as necessidades atuais de consultas de dados. Adotar um único mecanismo de banco de dados para abordar as diferentes demandas de consultas pode conduzir a soluções de desempenho inferior, e a coexistência de ambos os modelos pode proporcionar flexibilidade e eficiência, dependendo das especificidades de cada cenário.

Em Lau (2021) o autor fez uma comparação entre distintos sistemas de bancos de dados, tanto NoSQL quanto SQL, abordando seu desempenho em relação a uma base de dados contendo cerca de 1,2 milhões de registros de focos de incêndio obtidos da fonte DBQueimadas. Nesse trabalho diversos resultados foram observados, englobando aspectos como a ocupação de espaço em disco, além os tempos requeridos para operações de leitura e de escrita em cada sistema de gerenciamento de banco de dados avaliado. Ao término da pesquisa, o autor expõe que para cada um dos aspectos investigados, distintos níveis de desempenho foram identificados, com a tendência de que os bancos de dados NoSQL se mostrem mais adaptáveis para armazenar informações de formatos variados, enquanto os bancos de dados SQL demonstram maior eficiência no tratamento de dados estruturados.

Pode-se observar, portanto, que o campo dos bancos de dados está em um estado contínuo de evolução, com diversas oportunidades de pesquisa e desenvolvimento em uma ampla gama de Sistemas Gerenciadores de Banco de Dados (SGBDs). É evidente que não existe uma solução única que possa ser aplicada de maneira ideal a todos os tipos de problemas no cenário atual. A busca por um SGBD adequado requer uma aná-

lise criteriosa das características específicas de cada aplicação, uma vez que não existe uma fórmula padronizada para determinar o modelo ideal. Cada cenário demanda uma avaliação individualizada.

A necessidade de lidar com distintos tipos de dados, modelos de consulta e requisitos de desempenho exige uma abordagem mais flexível e adaptável. Um SGBD que se mostra eficiente em um contexto pode não ser a melhor escolha em outro. Isso enfatiza a importância de conduzir estudos de caso específicos para determinar qual SGBD melhor se alinha às características de cada aplicação.

Vale também ressaltar que, apesar dos esforços significativos de pesquisa na área de SGBDs, uma lacuna se faz presente no que diz respeito à avaliação de desempenho desses sistemas quando empregados em conjunto com diferentes linguagens de programação. Essa é uma oportunidade que ainda não foi abordada de forma abrangente. A comparação do desempenho dos SGBDs em contextos variados de linguagens de programação pode gerar informações valiosas sobre como diferentes combinações impactam na eficiência global do sistema.

Os trabalhos relacionados apresentados foram escolhidos de forma a explicitar os diferentes pontos que podem ser explorados em um trabalho desta área. Enquanto (CARRICONDE, 2016) compara dois SGBDs relacionais diretamente, Oliveira (2019) e Lau (2021) comparam diferentes modelos de SGBDs entre si, sendo esse tipo de comparações entre SGBDs diferentes o tipo mais comum de pesquisa encontrada na área. Em outro aspecto de avaliação, Carriconde (2016) compara a diferença de desempenho em sistemas operacionais diferentes, que é outro ponto que pode causar uma grande variação nos resultados. Pelos trabalhos de Oliveira (2019) e Lau (2021) é possivel ver que para dados que tem uma boa modelagem em modelos relacionais, geralmente esses ainda são a melhor opção, enquanto que modelos NoSQL são bons para documentos e SGBDs orientados a grafos são eficientes em conjuntos fortemente interligados.

Tabela 2.1 – Comparação entre os trabalhos

Foco	CARRICONDE	OLIVEIRA	LAU	Este Trabalho
SGBD Relacional	X	X	X	X
SGBD NoSQL		X	X	
SGBD orientado a grafos		X		
Diferentes sistemas operacionais	X			
Diferentes motores de um SGBDs				X
Diferentes linguagens de programação				X
Diferentes *drivers* de conexão				X

Fonte: Autor

O principal ponto em que esse trabalho se diferencia dos trabalhos relacionados citados é que em nenhum deles a linguagem de programação e o *driver* de conexão utilizados para acessar o SGBD são considerada. Na Tabela 2.1 podemos ver os pontos em que esse trabalho se difere em relação aos trabalhos relacionados aqui apresentados.

3 MAPEAMENTO DOS DADOS

O presente capítulo apresenta e descreve a abordagem proposta para o mapeamento lógico e físico do BDQueimadas (INPE, 2019), que abrange um conjunto de dados fornecido pelo Instituto Nacional de Pesquisas Espaciais (INPE), conjunto esse que representa os focos de incêndios e queimadas obtidos por meio de imagens via satélite de diferentes países.

A escolha do modelo relacional, em detrimento de outros modelos, foi realizada com o propósito de possibilitar a comparação de desempenho de diferentes linguagens de programação, utilizando um modelo mais tradicional e popular entre as aplicações. Essa escolha permitirá analisar a eficiência dos sistemas implementados e fornecer *insights* sobre o desempenho ao trabalhar com dados desse contexto específico. Dessa forma, o capítulo visa contribuir para a compreensão e avaliação da aplicação do modelo relacional no contexto do BDQueimadas.

O capítulo inicia com uma descrição detalhada do domínio do problema, que envolve os dados relacionados aos focos de incêndio e queimadas de determinadas regiões. São apresentadas informações sobre a natureza dos dados, sua origem e a relevância para a pesquisa ou aplicação em questão.

Posteriormente, é especificado o modelo lógico para a abordagem proposta. Esse modelo descreve a estrutura e a organização dos dados de forma independente do sistema de gerenciamento de banco de dados (SGBD) ou de qualquer implementação física específica. O objetivo do modelo lógico é representar as entidades, seus atributos e os relacionamentos entre elas de maneira clara e abstrata. Essa abordagem é fundamental para garantir a consistência e a coerência dos dados, facilitando a compreensão do sistema e o desenvolvimento de soluções adequadas para as necessidades do domínio em questão.

Com a especificação do modelo lógico, estabelecem-se as bases para a criação e o desenvolvimento do banco de dados, possibilitando a posterior implementação física em um SGBD específico. Essa etapa é de suma importância para a correta estruturação e manipulação dos dados.

3.1 Dominio

O domínio selecionado para este estudo é o monitoramento de queimadas e incêndios fornecido pelo Instituto Nacional de Pesquisas Espaciais (INPE). O sistema BD-

Queimadas do INPE não apenas fornece os dados relevantes sobre focos de incêndios e queimadas em diferentes regiões, mas também oferece um conjunto de ferramentas de visualização disponíveis em seu site. Essas ferramentas possibilitam que os usuários explorem e analisem os dados de forma interativa e visualmente significativa, proporcionando *insights* valiosos sobre o cenário de queimadas e incêndios florestais.

Para cada foco de incêndio registrado, são coletadas informações específicas, tais como a data e a hora da detecção, o satélite responsável pela identificação, as coordenadas geográficas precisas que indicam a localização do foco (latitude e longitude), além de dados relacionados ao país, estado e município onde ocorreu o incêndio. Além desses dados básicos, são armazenadas informações adicionais para cada foco de incêndio, como o número de dias sem chuva que antecederam a detecção, o risco de fogo previsto para o dia da detecção, o valor da precipitação registrada até o momento da detecção e o *Fire Radiative Power* (FRP), que representa a taxa média de calor emitida pelo incêndio.

Essas informações detalhadas e as ferramentas de visualização oferecidas pelo sistema BDQueimadas contribuem para uma compreensão mais intuitiva e eficiente dos dados, tornando-os acessíveis não apenas para pesquisadores e cientistas, mas também para tomadores de decisão e o público em geral.

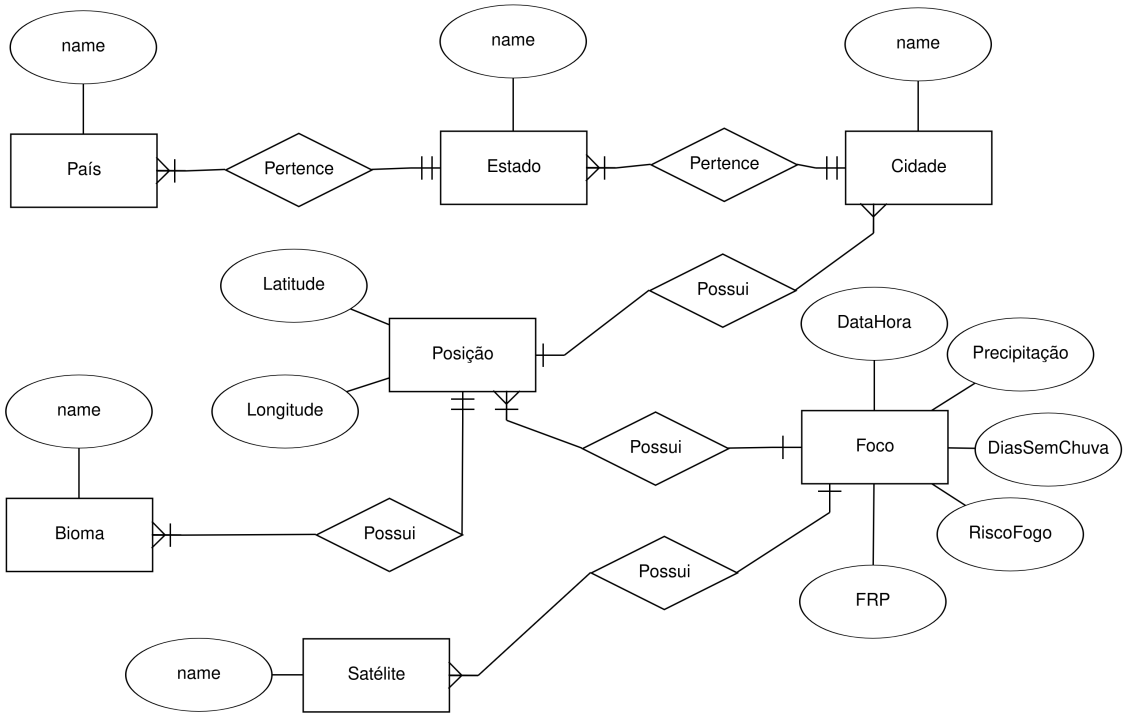
3.2 Modelagem

Para representar os dados na abordagem relacional, é necessário construir um modelo que contenha as principais entidades capazes de representar todo o conjunto de dados fornecido pelo INPE, composto por diversas informações provenientes dos satélites. O objetivo desta seção é realizar a modelagem dos dados para a abordagem relacional por meio de um diagrama entidade-relacionamento, que demonstrará de forma clara e objetiva como essas entidades se relacionam. Além disso, serão abordadas as restrições de integridade aplicadas ao modelo proposto.

Inicialmente, é construído o modelo conceitual, no qual se identificam as principais entidades e seus relacionamentos para possibilitar a criação posterior de um banco de dados relacional, permitindo o armazenamento de informações e a realização de consultas. O modelo conceitual pode ser visualizado na Figura 3.1. Com base nesse modelo conceitual, desenvolveu-se o modelo lógico, representado na Figura 3.2.

O modelo lógico proposto consiste nas seguintes entidades:

Figura 3.1 – Modelo conceitual para o domínio dos dados de focos de incêncido.



Fonte: Autor

- Pais: Representa o país em que cada foco está localizado.
- Estado: Representa os estados de cada país em que cada foco está localizado.
- Municipio: Representa os municípios de cada estado em que cada foco está localizado.
- Bioma: Representa o bioma da região do foco.
- Posicao: Representa a posição em que o foco está localizado, assim como características relacionadas a esta posição.
- Satelite: Representa os satélites utilizados para o monitoramento e identificação dos focos.
- Foco: Representa os focos e suas características.

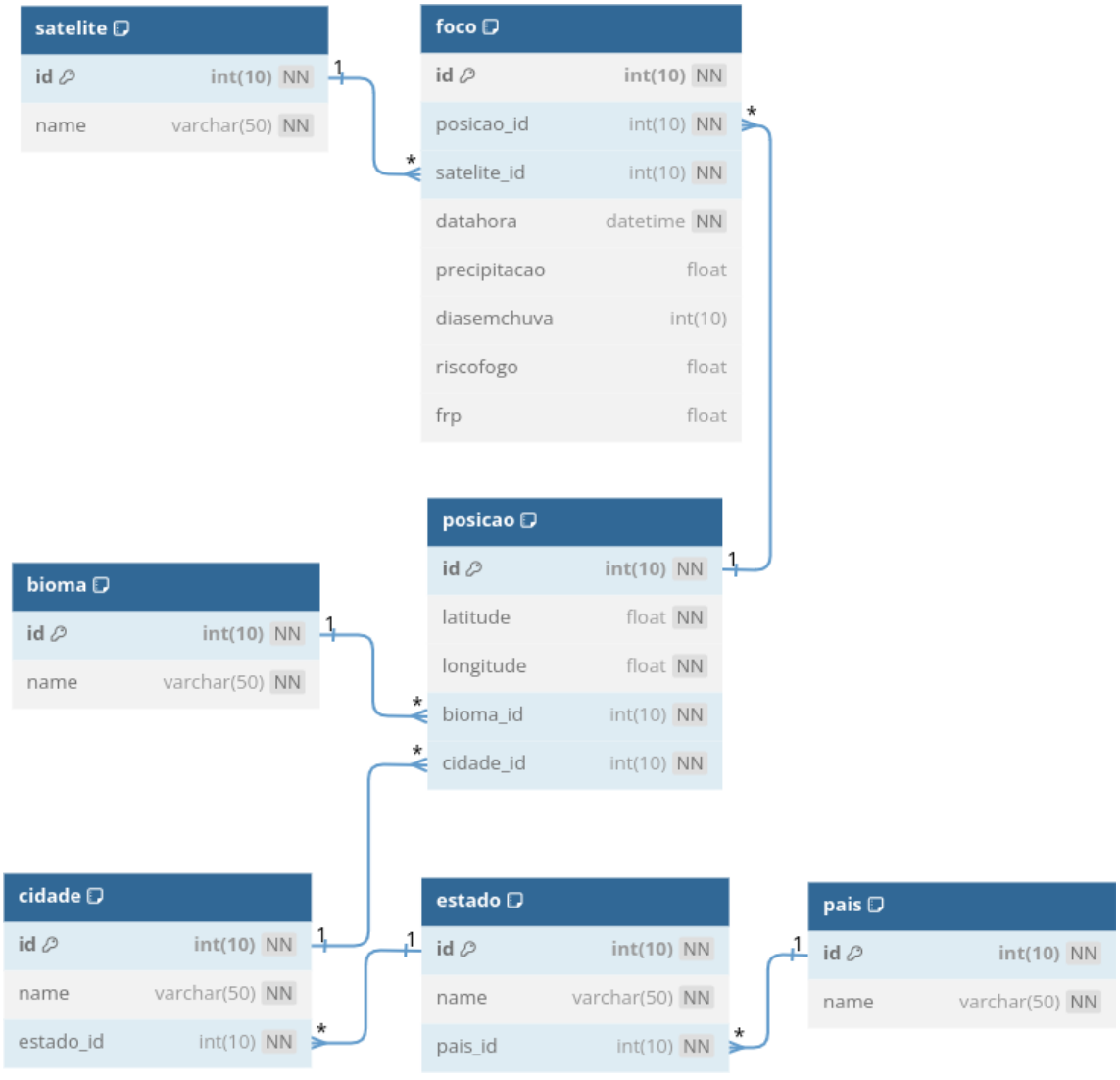
O modelo proposto também inclui restrições de integridade essenciais para garantir a qualidade e a consistência dos dados armazenados:

- Integridade referencial: A integridade referencial é assegurada no modelo de banco de dados por meio da utilização de chaves estrangeiras, cada uma correspondendo a um valor de uma chave primária existente na tabela de origem. Essa prática visa garantir a validade e consistência de todas as referências a outras entidades. Um

exemplo concreto é o atributo “estado” na tabela “Municipio”, que referencia a chave primária “nome” na tabela “Estado”, assegurando que cada município esteja sempre associado a um estado existente e válido.

- Integridade de entidade (chave): Em todo o modelo de banco de dados, todas as tabelas são equipadas com chaves primárias que fornecem identificação única para cada elemento contido na tabela. Essas chaves primárias asseguram que cada registro seja exclusivo e não duplicado, garantindo, assim, a integridade da entidade e evitando qualquer redundância de dados.
- Integridade de domínio: Os atributos das tabelas são definidos com tipos de dados específicos e tamanhos, estabelecendo conjuntos de valores permitidos para cada um. Essa integridade de domínio assegura que apenas valores válidos possam ser inseridos nos atributos, evitando a presença de dados inconsistentes ou inválidos. Adicionalmente, alguns atributos têm a capacidade de conter valores nulos, o que permite que campos específicos não tenham valores atribuídos. Essa flexibilidade foi incorporada ao modelo para tratar situações em que algumas informações podem não estar disponíveis ou não serem aplicáveis.

Figura 3.2 – Diagrama entidade-relacionamento para o domínio dos dados de focos de incêncido.



Fonte: Autor

4 AVALIAÇÃO EXPERIMENTAL

Neste capítulo, realizamos uma análise comparando os *drivers* e as duas linguagens de programação selecionadas para este estudo: C e Python. A Seção 4.1 contem uma breve descrição das ferramentas e linguagens utilizadas nesse estudo e o motivo pelos quais foram escolhidos. Na Seção 4.2 fornecemos uma descrição do ambiente computacional empregado para a condução dos experimentos. Aspectos como *hardware*, sistema operacional e versões das linguagens e do SGBD foram considerados, visando assegurar uma avaliação justa e replicável dos resultados.

Posteriormente, na Seção 4.3, elucidamos a origem, o período e o motivo dessa seleção para os dados utilizados, enquanto que na Seção 4.4 temos os procedimentos adotados para a preparação dos dados utilizados nos modelos físicos previamente delineados no Capítulo 3. Essa etapa foi realizada com o objetivo de garantir que os dados estivessem representados de forma apropriada para cada linguagem e que pudessem ser manipulados de maneira equivalente.

Os resultados oriundos da execução de algumas operações de inserção de dados em ambos os ambientes, C e Python, são apresentados na Seção 4.6. Analisamos os tempos de execução e examinamos possíveis disparidades de desempenho entre as linguagens nesse contexto específico.

Já na Seção 4.7, descrevemos o planejamento das demais consultas efetuadas e apresentamos os resultados referentes ao tempo de execução de cada consulta em ambas as linguagens. Essa análise foi fundamental para compreender como as linguagens se comportam ao realizar operações mais complexas no banco de dados.

Por fim, na Seção 4.9, efetuamos uma síntese dos resultados gerais obtidos nos experimentos, enfatizando as principais observações e conclusões. Também discutimos possíveis limitações do estudo e apresentamos sugestões para pesquisas futuras, visando aprimorar e ampliar a investigação em outras áreas do desenvolvimento de software.

4.1 Materiais e métodos

Para o desenvolvimento do estudo utilizaremos um SGBD relacional com suporte a mais de um motor de armazenamento de dados, no caso o MySQL. Essa escolha se deve a ele ser Open-Source além de ter suporte a diversos motores de armazenamento. Ao avaliar a quantidade de produtos e empresas que utilizam o MySQL podemos ver que

eles são alguns dos banco de dados open source mais popular do mundo. Estando por trás de muitas das aplicações mais acessadas, incluindo Facebook[1], Twitter[2], Netflix[3], Uber[4], Airbnb[5], Shopify[6], e Booking.com[7] (ORACLE, 2023b). Para a estruturação do banco de dados relacional será utilizada o DBeaver, ferramenta de banco de dados multiplataforma altamente versátil e gratuíta, projetada para atender às necessidades de desenvolvedores, administradores de banco de dados e analistas (DBEAVER, 2023). Com ela foi desenvolvida a modelagem do banco de dados e foram geradas a figura do diagrama Entidade-Relacionamento.

As linguagens de programação escolhidas, como dito anteriormente, foram o C e o Python. Os principais motivos que levaram a essa escolha é elas serem as 2 primeiras colocadas no *ranking IEEE Spectrum - Top Programming Languages* de 2022[8], a diferença de paradigma entre elas, com o C sendo uma linguagem compilada e o Python uma linguagem interpretadas, e também porque os *drivers* mais comunmente utilizados nessas linguagens serem diferentes. A linguagem C utiliza o *driver* de API (*application programming interface*) do MySQL para conexão enquanto o Python utiliza o *driver* Connector do MySQL para conexão. Isso faz com que o código em C utilize chamadas diretas a API do MySQL enquanto em Python temos uma camada extra de abstração, utilizando cursores para percorrer os dados recebidos na consulta.

4.2 *Hardware* e *Software*

Os experimentos foram conduzidos em um sistema com as seguintes especificações de *hardware*: processador Intel Core i7-7700HQ com frequência de 2.8 GHz, contendo 4 núcleos e 8 threads. A capacidade de memória RAM é de 32GB, operando em dual Channel a 1600MT/s. Além disso, o sistema é equipado com um SSD NVME de 1TB, apresentando uma taxa de transferência de leitura sequencial de 3500MB/s e uma taxa de escrita sequencial de 3300MB/s. O sistema operacional utilizado para a execução dos experimentos foi o ArchLinux, com *kernel* 6.1.39-1-lts.

O sistema de gerenciamento de banco de dados (SGBD) selecionado para os ex-

[1] <https://www.facebook.com>
[2] <https://www.twitter.com>
[3] <https://www.netflix.com>
[4] <https://www.uber.com>
[5] <https://www.airbnb.com>
[6] <https://www.shopify.com>
[7] <https://www.booking.com>
[8] <https://spectrum.ieee.org/top-programming-languages-2022>

perimentos foi o MySQL 8.0, o qual foi executado por meio de uma imagem Docker. A escolha de utilizar o Docker permitiu uma maior facilidade na replicação do ambiente, possibilitando a execução da mesma versão do SGBD em diferentes locais de forma mais simples e consistente.

A versão do Docker utilizada para esse propósito foi a 24.0.5, build ced0996600, assegurando o funcionamento do MySQL em seu ambiente virtualizado.

Quanto às linguagens de programação, o código em C foi compilado utilizando a versão 13.1.1 20230714 do GCC (GNU Compiler Collection), enquanto o Python foi interpretado na versão 3.11.3. Essas escolhas foram feitas visando a utilização de versões atualizadas e confiáveis das linguagens, a fim de garantir a consistência e precisão dos resultados obtidos nos experimentos.

4.3 Base de dados - INPE DBQueimadas

A fonte de dados utilizada para este estudo foi a disponibilizada pelo INPE através da aplicação BDQueimadas (INPE, 2023), que é responsável por monitorar e fornecer informações visuais sobre os focos de queimadas na América do Sul. Para garantir uma avaliação abrangente e com grande volume de dados durante os testes, optou-se por obter uma quantidade significativa de dados não filtrados do sistema. Dessa forma, a base de dados sobre os focos de queimadas foi obtida sem restrições, contendo registros desde 01/10/2022 até 31/03/2023.

O propósito dessa abordagem foi permitir a recuperação de uma maior quantidade de dados, facilitando o acesso a eles por outros pesquisadores que possam se interessar em realizar trabalhos futuros. No entanto, a seleção irrestrita de dados pode levar a inconsistências e valores nulos, especialmente em relação a dados que não se referem ao Brasil.

É importante destacar que, para o objetivo deste trabalho, o foco está na verificação do desempenho do banco de dados após a saída do *driver*. Nesse contexto, o processo de transferência de dados, mesmo considerando registros nulos ou inconsistentes, representa um método válido para testar o desempenho do sistema.

Embora a presença de dados nulos e inconsistências possa parecer um problema em um primeiro momento, essa abordagem permite avaliar a capacidade do banco de dados de lidar com diferentes situações e verificar sua eficiência em relação ao processamento de uma grande quantidade de informações. Portanto, apesar das possíveis li-

mitações da seleção irrestrita, essa metodologia contribui para uma análise abrangente e representativa do desempenho do banco de dados em estudo.

4.4 Mapeamento para o MySQL

Para efetuar a inserção dos dados no modelo concebido para o banco de dados MySQL, foi conduzida uma etapa de pré-processamento dos dados. Nesse procedimento, as colunas contidas no arquivo CSV original foram cuidadosamente convertidas em objetos que representam as diversas entidades e suas relações, mantidos em memória. Posteriormente, esses objetos foram salvos em arquivos SQL, possibilitando, assim, a carga dos dados nas tabelas criadas no modelo proposto, tal como apresentado no Capítulo 3.

Essa abordagem proporciona a versatilidade de carregar os dados por meio de qualquer ferramenta capaz de estabelecer conexão com o banco de dados. O teste de inserção de dados, mencionado anteriormente, foi executado utilizando esses arquivos SQL gerados.

O algoritmo de pré-processamento empregado foi desenvolvido em Python 3, utilizando uma conexão com o banco de dados de destino para garantir o correto *escape* de *strings* a serem inseridas no banco de dados de maneira segura. Essa medida é adotada para mitigar possíveis problemas de segurança relacionados à inserção de dados.

4.5 Metodologia

Cada teste será realizado em 10 iterações independentes, com início sempre em uma nova execução do programa, nunca ocorrendo dentro da mesma instância de processo e, no caso das inserções, com um banco de dados limpo, somente com as tabelas sem dados. Essa abordagem visa assegurar que os resultados sejam obtidos de forma consistente e sem interferências entre as iterações dos testes.

Para obter o tempo médio, faremos uma média aritmética simples desses conjuntos, desconsiderando o maior e o menor tempo de cada grupo. Essa medida tem o propósito de eliminar possíveis anomalias ou valores atípicos que possam ter ocorrido durante o processo de teste, proporcionando uma análise mais robusta e confiável dos resultados.

Com o intuito de mitigar possíveis efeitos advindos da leitura de dados do disco e de qualquer tratamento de *strings* durante a execução dos testes, adotaremos a seguinte

metodologia:

1. Estabelecer a conexão com o banco de dados.
2. Carregar todas as consultas que devem ser executadas para a memória, armazenando-as em uma lista de *strings*. Cada item da lista representa um *insert* (consulta).
3. Registrar o instante inicial em nanosegundos.
4. Executar sequencialmente todas as consultas contidas na lista.
5. Iterar sobre os retornos, caso existam
6. Registrar o instante final em nanosegundos.
7. Calcular o tempo decorrido para a execução de todas as consultas.

Durante a execução das consultas em ambas as linguagens, não serão realizadas quaisquer chamadas de funções desnecessárias, exceto aquelas inerentes à API de execução do banco de dados e manipulação da memória.

4.6 Inserção dos dados

Para realizar os testes de inserção de dados, serão utilizados três tamanhos distintos para os dados de entrada. Cada inserção estará limitada a 100 linhas por vez. No primeiro teste, o tamanho de cada tabela será limitado a 100.000 linhas, no segundo teste, a 1.000.000 de linhas e, por fim, no terceiro teste, a 50.000.000 de linhas.

Após a execução de somente uma iteração do teste de inserção de 100.000 de linhas foi obtido um resultado inesperado. A execução do código em C levou 11,03 segundos, enquanto o código em Python levou apenas 3,51 segundos.

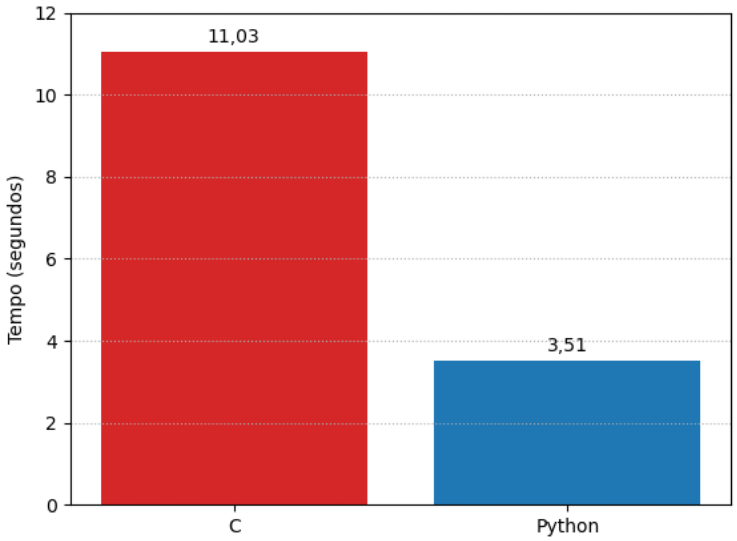
O resultado desta insersão foi inesperado, mas isso se deve a uma diferença de funcionamento dos *drivers* utilizados. O *driver* de Python inicia automaticamente transações no banco de dados, e isso afeta muito a inserção de dados no MySQL. Na listagem 4.1, podemos ver o código em C responsável pela execução das inserções no banco.

Para tornarmos o comportamento dos drivers de ambas as linguagens equivalentes o código foi alterado, adicionando a inicialização e finalização de uma transação com o banco, o qual pode ser visto na listagem 4.2.

A quesito de comparação, na listagem 4.3 está o código Python equivalente utilizado para as inserções.

Após realizar a modificação do código do programa em C para iniciar manual-

Figura 4.1 – Tempo de execução para inserção com tabelas limitas a 100.000 linhas e sem transações.



Fonte: Autor

Listagem 4.1 – Código em C para insersão

```
clock_gettime(CLOCK_REALTIME, &timer_start);
for (int i = 0; i < num_queries; i++) {
    if (mysql_real_query(conn, queries[i], queries_len[i]) !=
        0) {
        return 0;
    }
}
clock_gettime(CLOCK_REALTIME, &timer_end);
```

mente transações, obtivemos um resultado condizente com o esperado. Em qualquer quantidade de linhas inseridas a diferença entre C e Python foi muito pequena, chegando a pouco mais de 3 segundos com as tabelas limitas a 5 milhões de linhas. Todos os tempos podem ser vistos na Figura 4.2.

Os resultados obtidos dos testes de inserção de dados revelam que, apesar de uma das linguagens ser interpretada (Python) e a outra ser compilada (C), a diferença de desempenho entre elas é mínima. Mesmo com o aumento significativo na quantidade de dados, o tempo de execução entre as linguagens não apresenta um crescimento proporcional.

Essa constatação pode ser relevante e surpreendente, uma vez que geralmente se espera que linguagens compiladas sejam mais eficientes em termos de tempo de execução do que as interpretadas. No entanto, os resultados obtidos demonstram que ambas as

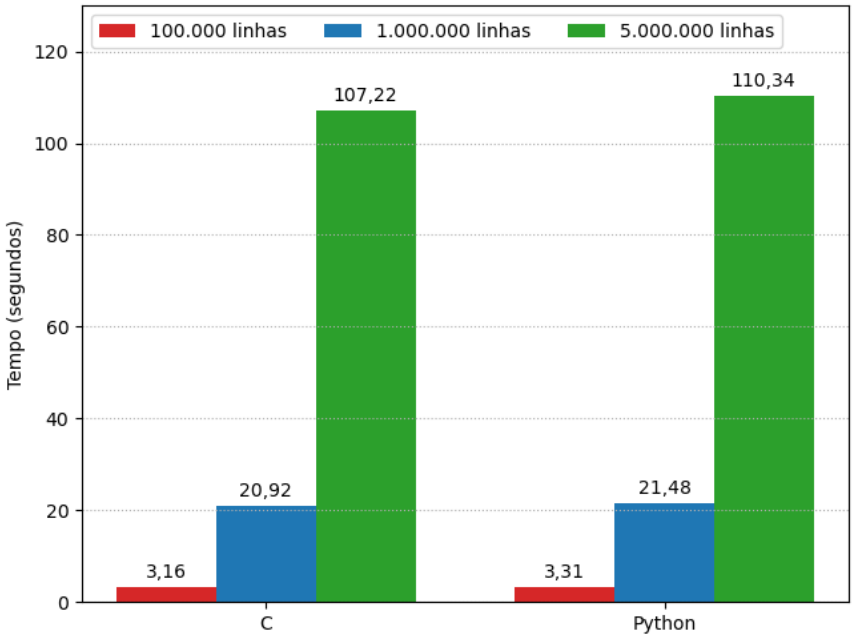
Listagem 4.2 – Código em C para insersão com transações

```c
clock_gettime(CLOCK_REALTIME, &timer_start);
mysql_query(conn, "start transaction;");
for (int i = 0; i < num_queries; i++) {
    if (mysql_real_query(conn, queries[i], queries_len[i]) !=
        0) {
        return 0;
    }
}
mysql_query(conn, "commit;");
clock_gettime(CLOCK_REALTIME, &timer_end);
```

Listagem 4.3 – Código em Python para inseção

```python
start_time = time.time_ns()
cursor = connection.cursor()
for query in queries:
    cursor.execute(query)
connection.commit()
end_time = time.time_ns()
```

Figura 4.2 – Tempos de execução para inserção de dados.



Fonte: Autor

linguagens, C e Python, são capazes de lidar com inserção de grandes volumes de dados de forma eficiente, sem que a diferença entre elas seja substancial, desde que seja avaliado somente o tempo para inserção dos dados.

4.7 Consultas

Para a avaliação das consultas, os testes seguirão uma abordagem similar à dos testes de inserção de dados. No entanto, um aspecto diferencial é que, em ambas as linguagens (C e Python), será necessário ler os dados recebidos para a memória, visto que isso faz parte do processo de acesso aos dados propriamente dito. Adicionalmente, um ponto relevante é que as mesmas consultas serão executadas em dois motores diferentes do MySQL:

- InnoDB: Motor de uso geral, sendo ele o motor padrão do MySQL.
- MyISAM: Motor otimizado para consultas, não tendo suporte a transações.

O objetivo desses testes é verificar o desempenho relativo de cada linguagem ao realizar consultas em ambientes distintos do MySQL. A utilização de motores diferentes permite analisar como as linguagens interagem e aproveitam os recursos fornecidos por cada motor para otimizar o desempenho das consultas.

Será adotada a mesma metodologia empregada nos testes de inserção de dados, com o cálculo do tempo médio de execução por meio da simples média aritmética, desconsiderando os maiores e menores tempos de cada grupo de execução. A realização dos testes será repetida 10 vezes, em instâncias de processos independentes, para garantir resultados mais confiáveis e consistentes.

4.7.1 Consulta 1: Listagem das cidades e o itendificador do estado correspondente

A consulta em questão tem como objetivo obter uma lista contendo o nome de todos os municípios presentes na base de dados, bem como o ID do estado ao qual cada município está vinculado. A consulta utilizada para essa finalidade é bastante simples:

```
1    SELECT nome, estado_id
2    FROM cidade;
```

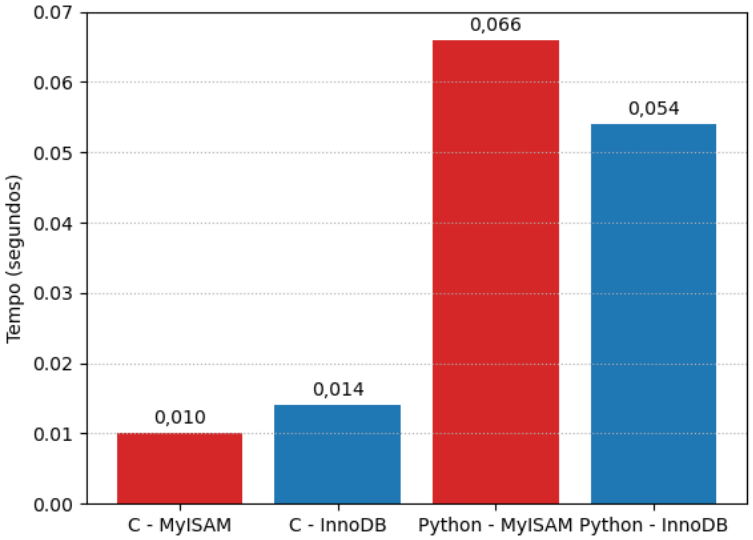
Dado o caráter singelo da consulta, a aparente grande diferença de tempo entre as linguagens não é algo perceptível na vida real, não passando de 50 milissegundos como apresentado na Figura 4.3. A simplicidade, baixa complexidade da operação e pequena quantidade de resultados retornados permitem que ambas as linguagens desempenhem de forma eficiente na realização da consulta, resultando em tempos de execução muito

baixos.

Esse cenário é corroborado pelo fato de que a consulta em si não envolve operações computacionalmente intensivas ou manipulação complexa de dados. O processo de recuperação dos dados da tabela “cidade”, com apenas a extração do nome e o ID do estado associado, é facilmente executado, independentemente da linguagem de programação utilizada.

Também é possível ver que para consultas que envolvem poucos resultados não é possível perceber praticamente nenhuma diferença entre o motor InnoDB e a motor MyISAM, e para um caso com tão poucos resultados o InnoDB pode acabar sendo até mais performatico que o MyISAM, como ocorreu com o teste em Python.

Figura 4.3 – Tempo de execução da consulta 1.



Fonte: Autor

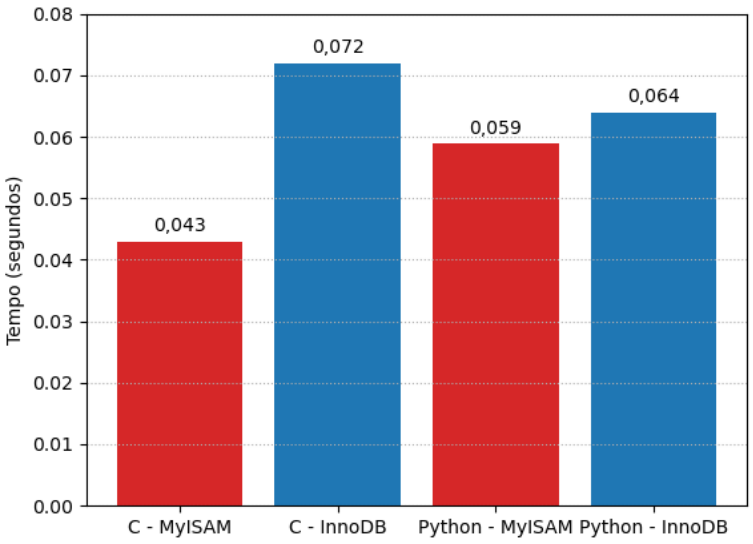
4.7.2 Consulta 2: Listagem das cidades juntamente do nome de seu estado e país

A segunda consulta tem o propósito de buscar informações mais abrangentes sobre cada cidade, incluindo o seu nome, o nome do estado ao qual pertence e o nome do país ao qual o estado está vinculado. A consulta requer o uso de operações JOIN, embora ainda não apresente filtros ou condições mais complexas. Essa consulta possui uma complexidade ligeiramente maior em comparação com a anterior, mas ainda é simples.

```
1    SELECT c.name, e.name, p.name
```

```
    FROM cidade AS c
    JOIN estado AS e ON c.estado_id = e.id
    JOIN pais AS p ON e.pais_id = p.id;
```

Figura 4.4 – Tempo de execução da consulta 2.



Fonte: Autor

Podemos ver na Figura 4.4 que os tempos de execução entre as linguagens C e Python continuaram com diferenças muito pequenas, inclusive são proporcionalmente menores que na consulta 1, porém, com um exame mais cuidadoso, é possível observar uma pequena melhora de desempenho do motor MyISAM em relação a InnoDB para a linguagem Python nesse resultado que contem uma quantidade um pouco maior de dados.

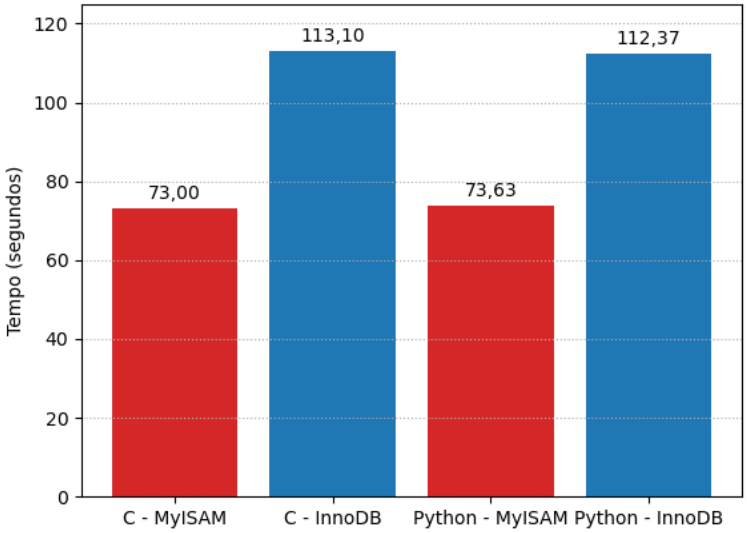
Um ponto interessante que podemos perceber é que com o motor InnoDB a consulta em Python foi ligeiramente mais eficiente que a execução em C, ainda assim, essa diferença é sutil e não chega a ser significativa.

4.7.3 Consulta 3: Contagem dos números de foco de incêndio por estado

A terceira consulta tem como objetivo encontrar o número de focos de queimadas em cada estado. Essa consulta é mais complexa em termos de estrutura da consulta SQL, mas resulta em uma quantidade de dados muito menor em comparação com a consulta anterior. Nela, utilizamos operações de agregação, como a função COUNT, para calcular a quantidade de focos de queimadas em cada estado.

```
SELECT COUNT(f.id) AS NumeroDeFocos, e.name
FROM foco f
INNER JOIN posicao p ON f.posicao_id = p.id
INNER JOIN cidade c ON p.cidade_id = c.id
INNER JOIN estado e ON c.estado_id = e.id
GROUP BY e.id;
```

Figura 4.5 – Tempo de execução da consulta 3.



Fonte: Autor

Conforme a Figura 4.5 podemos ver tempo total de execução dessa consulta aumentou muito em ambas as linguagens em comparação com as consultas anteriores, mas é importante observar que a diferença entre elas diminuiu. Aqui começamos a ver diferença em relação ao motor, como muito mais dados são consultados, o tempo de execução utilizando MyISAM é aproximadamente 60% do tempo com InnoDB, mesmo que a quantidade de dados retornados pela consulta seja menor que na consulta 2. Aqui se repete a pequena vantagem da linguagem Python sobre a linguagem C quando utilizamos InnoDB, mas ainda é uma diferença muito pequena.

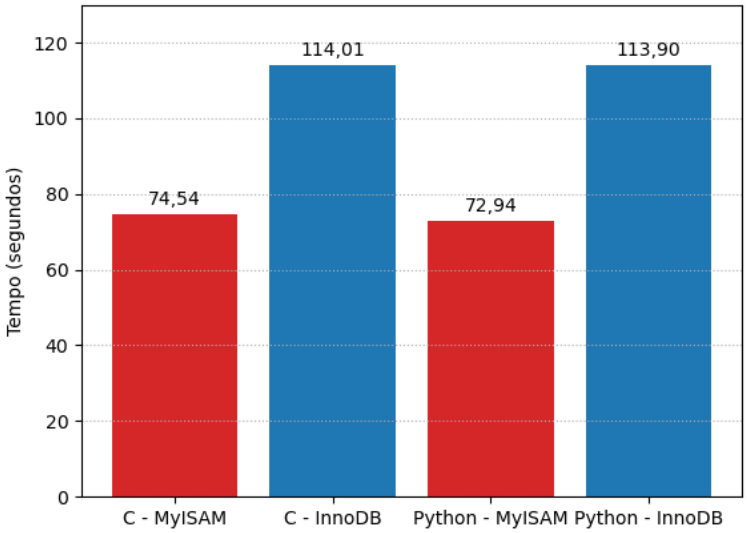
4.7.4 Consulta 4: Contagem dos números de foco de incêndio por país

A quarta consulta tem como objetivo contar o número de focos de queimadas por país. Nela, adicionamos mais um join em relação à consulta anterior para buscar

informações sobre o país associado a cada foco de queimada. Apesar do aumento na complexidade da consulta, a quantidade de dados retornada é ainda menor que a consulta anterior.

```
1  SELECT COUNT(f.id) AS NumeroDeFocos, pa.name
2  FROM foco f
3  INNER JOIN posicao p ON f.posicao_id = p.id
4  INNER JOIN cidade c ON p.cidade_id = c.id
5  INNER JOIN estado e ON c.estado_id = e.id
6  INNER JOIN pais pa ON e.pais_id = pa.id
7  GROUP BY pa.id;
```

Figura 4.6 – Tempo de execução da consulta 4.



Fonte: Autor

Ao executar essa consulta, percebemos que o tempo total aumentou ligeiramente em ambas as linguagens, mas a diferença entre elas também diminuiu um pouco como mostrado na Figura 4.6. Isso reforça a observação anterior de que a quantidade de dados retornados deve afetar a diferença de tempo entre as linguagens, enquanto a complexidade da consulta em si tem o mesmo impacto em ambas. Nesta consulta novamente o Python se saiu ligeiramente melhor que o C, porem essa diferença ocorreu em ambos os motores, e não somente com InnoDB.

O que alguns testes mostraram é que isso se deve a alguma otimização da linguagem Python que não está sempre fazendo uma cópia dos dados para uma variavel local durante a consulta. Como mostrado nas Listagems 4.4 e 4.5, o código em C faz a cópia

da memória para uma *string* utilizando *sprintf* enquanto em Python é feita uma atribuição concatenada para uma *string*.

Listagem 4.4 – Código em C para consultas

```c
clock_gettime(CLOCK_REALTIME, &time_start);
if (mysql_real_query(conn, queries[0], queries_len[0]) != 0) {
    return 0;
}
MYSQL_RES *result = mysql_store_result(conn);
if (result == NULL) {
    fprintf(stderr, "mysql_store_result failed\n");
    mysql_close(conn);
    return 1;
}
MYSQL_ROW row;
while ((row = mysql_fetch_row(result))) {
    unsigned long *lengths = mysql_fetch_lengths(result);
    for (int i = 0; i < mysql_num_fields(result); i++) {
        sprintf(temp, "%.*s.", (int)lengths[i], row[i] ? row[i]
            : "NULL");
    }
}
clock_gettime(CLOCK_REALTIME, &time_end);
```

Listagem 4.5 – Código em Python para consultas

```python
start_time = time.time_ns()
cursor = connection.cursor()
cursor.execute(query)
for linha in cursor:
    for i in colrange:
        temp = f'{linha[i]}.'
cursor.close()
end_time = time.time_ns()
```

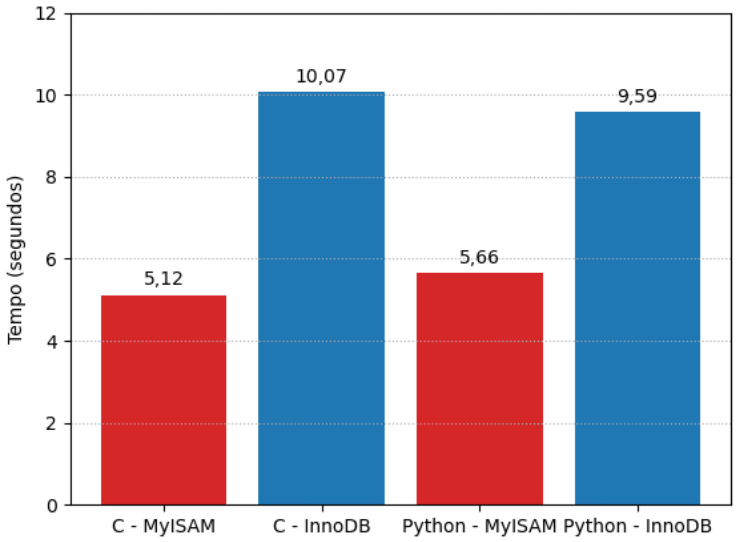
4.7.5 Consulta 5: Recuperação de todos os dados relacionados a um foco de incendio

A consulta em questão tem como objetivo recuperar todos os dados recebidos do DBQueimadas, contendo todos os campos possíveis. Embora possa parecer complexa à primeira vista, sua lentidão não está diretamente relacionada à complexidade da consulta em si, mas sim à quantidade de dados recebidos, o que acaba afetando o tempo de execução. Nesse caso esperamos observar uma diferença mais significativa de tempo de

execução entre as linguagens, uma vez que trazer uma grande quantidade de dados para a memória levaria mais tempo em Python.

```
1  SELECT f.datahora, s.name, pa.name, e.name, c.name, b.name,
       f.diasemchuva, f.precipitacao, f.riscofogo, p.latitude,
       p.longitude, f.frp
2  FROM foco f
3  JOIN satelite s ON f.satelite_id = s.id
4  JOIN posicao p ON f.posicao_id = p.id
5  JOIN bioma b ON p.bioma_id = b.id
6  JOIN cidade c ON p.cidade_id = c.id
7  JOIN estado e ON c.estado_id = e.id
8  JOIN pais pa ON e.pais_id = pa.id
9  LIMIT 1000000;
```

Figura 4.7 – Tempo de execução da consulta 5.



Fonte: Autor

No entanto ao analizarmos a Figura 4.7 fica evidente que a diferença se manteve pequena e novamente a linguagem Python foi mais rápida que a linguagem C utilizando o InnoDB. O único motivo para essa consulta ter tido um tempo de execução menor do que as consultas 3 e 4 é a limitação da quantidade de linhas retornadas. Mesmo que essa quantidade seja muito maior do que as consultas anteriores, o fato de ela não ter nenhum filtro ou agrupamento faz com que o SGBD não tenha que consultar toda a tabela para coletar as informações, sendo que a quantidade de dados lida do disco e processada pelo

SGBD é muito menor que nas consultas 3 e 4.

4.8 Limitações e ameaças aos experimentos

O que qualquer experimentação entre as linguagens C e Python costuma constatar é que um código em C equivalente a um código em Python tem um tempo de execução menor, entretanto não foi isso que encontramos neste estudo. O comportamento observado pode ser explicado em parte devido as *strings* em Python serem imutáveis. Isso faz com que ao executarmos a atribuição do valor recebido pelo *driver* do banco de dados para a *string* local, muitas vezes, ter o mesmo conteúdo, por exemplo os nomes das cidades, a linguagem Python otimiza essa atribuição e não faz a cópia do dado de um local da memória para o outro. Na linguagem C, como utilizamos a função *sprintf*, é garantido que o conteúdo será copiado de um local da memória para o outro.

Foram feitos outros testes com diferentes métodos de cópia de dados, mas nenhum surtiu o efeito desejado. Alem disso foram efetuados testes nos quais os resultados da consulta eram salvos em disco ou enviados para */dev/null*, mas devido ao modo como as linguagens trabalham com os *Buffers* e salvamento de arquivos serem diferentes, a medição do tempo de execução não seria somente da consulta ao banco de dados.

Dessa forma os resultados aqui apresentados tem um viés que beneficia o Python em detrimento do C. Entretanto, tendo o conhecimento desse viés é possível chegarmos a uma conclusão justa dos resultados observados.

4.9 Avaliação Geral

Durante os testes foi possível perceber que a diferença de desempenho entre as linguagens não foi significativa. Tanto o código em C quanto o em Python levaram praticamente o mesmo tempo em todos os testes de inserções.

Nos experimentos de consultas as diferenças, da mesma forma, continuaram bem pequenas e em alguns momentos o Python teve um desempenho melhor do que o C. Como avaliado durante os testes e na seção anterior, isso possivelmente se deve a otimizações feitas pelo Python em relações a *strings* duplicadas.

Uma das maiores diferenças que podemos ver na comparação entre as linguagens, inclusive nos pequenos trechos de código listados no decorrer desse trabalho, é que o

desenvolvimento de um código em Python é muito mais simples que um código em C.

Em relação aos motores foi possível perceber que, para consultas, o MyISAM é realmente muito superior que o InnoDB. Isso era esperado devido a esse motor ser otimizado para consultas. Devido ao primeiro teste de inserção ter demonstrado que o uso de transações faz muita diferença no desempenho desse tipo de consulta, e como o motor MyISAM não tem suporte a essa funcionalidade, ele nem foi testado para comparações maiores em relação ao InnoDB.

Um outro ponto importante que podemos perceber é que *drivers* de acesso ao SGBD tem comportamentos diferentes. Enquanto o *driver* Connector inicia automaticamente uma transação, o *driver* de API requer que o programador faça isso se julgar necessário, e essa é uma questão que se deve ter em mente quando uma escolha de linguagem e *driver* estiver sendo feita.

De forma resumida, alguns dos principais pontos a serem considerados ao escolher uma linguagem de programação e um Sistema de Gerenciamento de Banco de Dados (SGBD) para um projeto são:

- **Avaliação de Desempenho**: É fundamental analisar as exigências de desempenho do projeto. Isso implica determinar quão rápido o sistema precisa ser para atender às necessidades.
- **Manutenção do Código**: Deve-se avaliar a probabilidade de alterações frequentes no código. Algumas linguagens e SGBDs podem ser mais adequados para projetos que exigem atualizações regulares.
- **Frequência de Uso**: Analise com que frequência a aplicação será utilizada. Para sistemas altamente utilizados, a eficiência torna-se crucial.
- **Manipulação de Dados pela Linguagem de Programação**: Leve em consideração o grau de manipulação de dados que a linguagem de programação precisa realizar, pois isso pode impactar a eficiência e a complexidade do código.
- **Local de Execução**: Considere onde o código será executado. Leve em conta fatores como ambiente de servidor, plataformas de implantação e compatibilidade.
- **Proporção entre Inserções e Consultas**: Avalie o equilíbrio entre operações de inserção e recuperação de dados. Alguns SGBDs podem se destacar em uma dessas operações.
- **Comportamento do Driver**: Pesquise o comportamento padrão do driver de conexão entre a linguagem de programação e o SGBD. Isso pode afetar o desempenho

e a funcionalidade.

5 CONCLUSÃO

Este estudo apresentou uma análise abrangente de desempenho entre duas linguagens de programação utilizando o mesmo sistemas de gerenciamento de bancos de dados, no caso o MySQL. Esta avaliação abordou diversas métricas, englobando o tempo necessário para carregar os dados, o volume de registros armazenados, e o tempo de execução de consultas específicas. A base de dados empregada para essa comparação é proveniente de registros de focos de incêndio, gentilmente disponibilizada pelo Instituto Nacional de Pesquisas Espaciais (INPE) através da aplicação web BDQueimadas.

Os resultados obtidos neste estudo permitiram uma visualização de algumas diferenças de desempenho das linguagens e seus respectivos *drivers* no acesso ao SGBD. As diferenças observadas incluem, mas não se limitam a:

Complexidade do código: um grande diferencial entre as linguagens é a complexidade para construir o código que executa as consultas no banco de dados. Python é uma linguagem muito mais amigável nesse quesito e somado a isso a diferença não tão grande de desempenho entre as linguagens faz com que ele pareça ser uma boa opção de linguagem quando se fala de acesso a SGBDs.

Paradigma da linguagem: mesmo considerando que o Python é uma linguagem interpretada, os resultados obtidos nos diversos cenários testados neste estudo demonstraram que a discrepância de paradigma entre uma linguagem interpretada e outra compilada não se manifestou de forma tão ampla quanto se esperava. Tradicionalmente, em muitos *benchmarks*, o Python é tido como uma linguagem mais de 40 vezes mais lenta em relação ao C. Entretanto, tal discrepância é mitigada em grande medida pelo fato de que o driver de conexão com o SGBD é programado em uma linguagem compilada, enquanto o código Python utilizado para acesso ao SGBD não passa de uma camada de interface que invoca o código compilado. Portanto, é possível inferir que a diferença substancial de desempenho reside essencialmente na sobrecarga gerada pela camada em Python do driver e na subsequente operação do código responsável por processar os resultados provenientes do banco de dados.

Comportamento padrão: nem todos os *drivers* de conexão a um SGBD apresentam as mesmas configurações e comportamentos em suas configurações padrão. Foi notável a constatação de que o driver Connector do Python, por exemplo, já inclui as transações ativadas por padrão, o que pode exercer uma influência considerável na operação de inserção de dados.

Um ponto importante a ser levado em conta nessa aparente equivalência das linguagens é que os dados não estavam sendo realmente utilizados pelas códigos. A única utilização era uma tentativa de cópia para um outro local de memória dos dados recebidos, o que em caso de dados duplicados, no caso do Python, essa cópia era otimizada e não executada, enquanto em C ela era sempre executada. Em testes utilizando gravação em disco foi percebido que o tempo de execução do Python ficava em torno de 40% mais lento que o código em C, mas como não foi possível medir outros impactos causados pelo gerenciamento dos arquivos pela linguagens esses testes foram descartados.

Através desses resultados, torna-se evidente que ambas as linguagens apresentam suas próprias vantagens e desvantagens, e vários fatores devem ser considerados ao tomar uma decisão entre uma ou outra. Python se destaca por sua facilidade e velocidade no desenvolvimento, tornando-o particularmente atrativo para prototipagem, processamento de dados pontuais e inserção de grandes quantidades de dados. Enquanto isso, C é uma escolha sólida para elaborar versões definitivas de protótipos construídos em Python, bem como para outros códigos que não estejam sujeitos a mudanças substanciais, devido à sua complexidade de codificação. Além disso, sua capacidade de throughput é significativamente superior, o que o torna uma opção valiosa para códigos que demandem alto desempenho.

Quanto a possíveis melhorias deste trabalho e sugestões para trabalhos futuros, seria interessante considerar a comparação com outros sistemas de gerenciamento de bancos de dados, como MongoDB ou outros sistemas NoSQL, a fim de verificar a validade e aplicabilidade dos resultados obtidos. Além disso, a segurança dos códigos em ambas as linguagens também poderia ser investigada, especialmente no que diz respeito a vulnerabilidades como buffer overflow em códigos escritos em C. Abordar essa questão se torna ainda mais relevante, uma vez que códigos nessa linguagem que operam em strings são suscetíveis a ataques desse tipo, tornando o desenvolvimento em C ainda mais desafiador.

REFERÊNCIAS

ALOMARI, Z. et al. Comparative studies of six programming languages. abs/1504.00693, 2015. Disponível em: <http://dblp.uni-trier.de/db/journals/corr/corr1504.html\#AlomariHSP15>.

BERG, K.; SEYMOUR, D. T.; GOEL, R. History of databases. **International Journal of Management & Information Systems (IJMIS)**, v. 17, p. 29, dez. 2012.

CARRICONDE, A. B. Postgresql x mysql: Crud performance comparison windows vs ubuntu. In: . [s.n.], 2016. Disponível em: <https://www.liphscience.com/submissoes/nBDzsBXfQeLw3cLS.pdf>.

CODD, E. F. A relational model of data for large shared data banks. **Commun. ACM**, v. 13, n. 6, p. 377–387, 1970. Disponível em: <https://doi.org/10.1145/362384.362685>.

COUNCIL, N.; BOARD, C.; HISTORY, C. **Funding a Revolution: Government Support for Computing Research**. National Academies Press, 1999. 162–165 p. ISBN 9780309062787. Disponível em: <https://nap.nationalacademies.org/read/6323/chapter/8\#162>.

DBEAVER. **DBeaver Community - Free Universal Database Tool**. 2023. Disponível em: <https://dbeaver.io>. Acesso em: 30.7.2023.

DEWEY, M. **A Classification and Subject Index for Cataloguing and Arranging the Books and Pamphlets of a Library**. [s.n.], 1876. Disponível em: <https://www.gutenberg.org/ebooks/12513>.

HAN, J. et al. Survey on nosql database. In: **2011 6th International Conference on Pervasive Computing and Applications**. [S.l.: s.n.], 2011. p. 363–366.

ISO, I. O. for S. **Information processing systems – Database language – SQL**. 1987. Disponível em: <https://www.iso.org/standard/16661.html>. Acesso em: 23.7.2023.

LAU, N. D. A. Análise comparativa de performance e modelagem entre um banco de dados relacional e um de documentos para a base de dados do bdqueimadas (inpe). In: . [S.l.: s.n.], 2021.

LAUDON, K.; LAUDON, J. **Sistemas de informação gerenciais**. Pearson Prentice Hall, 2011. ISBN 9788576059233. Disponível em: <https://books.google.com.br/books?id=e4ARYAAACAAJ>.

LITH, A.; MATTSSON, J. Investigating storage solutions for large data. In: . [s.n.], 2010. Disponível em: <https://publications.lib.chalmers.se/records/fulltext/123839.pdf>.

LOBO, E. **Curso prático de MySQL**. [S.l.]: Universo dos Livros Editora, 2008.

MARTINS FILHO, M. A. P. SQL x NoSQL: Análise de desempenho do uso do MongoDB em relação ao uso do PostgreSQL. In: . [s.n.], 2015. Disponível em: <https://www.cin.ufpe.br/~tg/2014-2/mapmf.pdf>.

MILANI, A. **MySQL - Guia do Programador**. Novatec Editora, 2007. ISBN 9788575221037. Disponível em: <https://books.google.com.br/books?id=81EwMDA-pC0C>.

NEWMAN, C. **Sams Teach Yourself MySQL in 10 Minutes**. Sams, 2006. (Safari Books Online). ISBN 9780672328633. Disponível em: <https://books.google.com.br/books?id=XoAjxEeHUyYC>.

OCLC, Inc. **Dewey Services**. Disponível em: <https://www.oclc.org/en/dewey.html>. Acesso em: 23.7.2023.

OLIVEIRA, F. D. S. Análise comparativa de um banco de dados relacional e um banco de dados em grafos. In: . [s.n.], 2019. Disponível em: <http://hdl.handle.net/123456789/3950>.

ORACLE. **About MySQL**. 2023. Disponível em: <https://www.mysql.com/about/>. Acesso em: 23.7.2023.

ORACLE. **What is MySQL**. 2023. Disponível em: <https://www.oracle.com/mysql/what-is-mysql/>. Acesso em: 30.7.2023.

PENTEADO, R. R. M. et al. Um estudo sobre bancos de dados em grafos nativos. In: . [s.n.], 2014. Disponível em: <https://turing.pro.br/anais/ERBD-2014/artigos_aceitos/trilha_pesquisa/124468_1.pdf>.

RICARTE, I. L. M. **Programação Orientada a Objetos: Uma abordagem com Java**. 2002. Disponível em: <https://www.dca.fee.unicamp.br/cursos/PooJava/javadb/bdrel.html>. Acesso em: 30.7.2023.

ROBINSON, I.; WEBBER, J.; EIFREM, E. **Graph Databases**. 2. ed. [S.l.]: O'Reilly, 2015. ISBN 978-1-4919-3089-2.

Sabre GLBL Inc. **The Sabre History**. 2017. Disponível em: <https://www.sabre.com/files/Sabre-History-rev2017.pdf>. Acesso em: 23.7.2023.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Database system concepts**. 6. ed. [S.l.]: McGraw-Hill, 2010.