INSTITUTO FEDERAL DE SANTA CATARINA

FILIPI VIRGILIO

Avaliação de Bancos de Dados para Séries Temporais: Um Estudo Comparativo entre MariaDB e InfluxDB

São José - SC

fevereiro/2025

AVALIAÇÃO DE BANCOS DE DADOS PARA SÉRIES TEMPORAIS: UM ESTUDO COMPARATIVO ENTRE MARIADB E INFLUXDB

Monografia apresentada ao Curso de Engenharia de Telecomunicações do campus São José do Instituto Federal de Santa Catarina para a obtenção do diploma de Engenheiro de Telecomunicações.

Orientador: Prof. Arliones Stevert Hoeller Junior, Dr.

São José - SC

fevereiro/2025

Filipi Virgilio

Avaliação de Bancos de Dados para Séries Temporais: Um Estudo Comparativo entre MariaDB e InfluxDB

São José - SC, 28 de fevereiro de 2025:

Documento assinado digitalmente
gov.br
ARLIONES STEVERT HOELLER JUNIOR
Data: 26/03/2025 11:07:19-0300
Verifique em https://validar.iti.gov.br

Prof. Arliones Stevert Hoeller Junior, Dr.
Orientador
Instituto Federal de Santa Catarina

Documento assinado digitalmente
gov.br
EDERSON TORRESINI
Data: 31/03/2025 12:44:51-0300
Verifique em https://validar.iti.gov.br

Prof. Ederson Torresini, M.Sc.
Instituto Federal de Santa Catarina

Documento assinado digitalmente
gov.br
RAMON HUGO DE SOUZA
Data: 31/03/2025 19:27:38-0300
Verifique em https://validar.iti.gov.br

Prof. Ramon Hugo Souza, Dr.
Instituto Federal de Santa Catarina

AGRADECIMENTOS

Agradeço, primeiramente, à minha mãe, Angela, e ao meu pai, Israel, pelo apoio incondicional desde sempre, por estarem ao meu lado em cada etapa da minha jornada e por me incentivarem no aprendizado.

À minha namorada, Júlia, pelo companheirismo, por compartilhar comigo os momentos bons e difíceis, sempre me motivando e acreditando em mim.

Aos meus amigos do IFSC e do Inova, agradeço pela amizade, pelo apoio e pelas inúmeras conversas que tornaram essa caminhada mais leve e enriquecedora.

Agradeço também a todos os meus professores, em especial ao meu orientador, Arliones, por sua dedicação e apoio, tanto nos projetos de pesquisa quanto na realização deste trabalho.

Muito obrigado a todos que, de alguma forma, contribuíram para a concretização deste momento.

RESUMO

O avanço da inteligência artificial e o aumento dos dispositivos da Internet das Coisas (IoT) geram a necessidade de armazenar grandes volumes de dados temporais. No entanto, bancos de dados relacionais tradicionais, como o MariaDB, enfrentam desafios ao lidar com séries temporais devido à sua estrutura, que pode impactar o desempenho em operações de inserção e consulta quando o volume de dados cresce significativamente. Este trabalho compara os tempos de inserção e recuperação de dados, e o espaço de armazenamento entre bancos de dados relacionais tradicionais e soluções especializadas em séries temporais, focando no MariaDB e InfluxDB. Definimos métricas para comparar tempo de resposta e eficiência de armazenamento, visando fornecer uma análise comparativa que destaque vantagens e desvantagens de cada solução para projetos que dependem de dados temporais. A análise mostra que o MariaDB tem limitações na gestão de grandes volumes de dados, especialmente quando o banco e seus índices não estão bem estruturados, enquanto o InfluxDB, projetado para séries temporais, oferece melhor gestão e otimização de espaço.

Palavras-chave: Séries Temporais. Desempenho de Bancos de Dados. Otimização de Armazenamentoa.

ABSTRACT

The advancement of artificial intelligence and the increase in Internet of Things (IoT) devices drive the need to store large volumes of temporal data. However, traditional relational databases, such as MariaDB, face challenges in handling time series due to their structure, which can impact performance in insertion and query operations as data volume grows significantly. This study compares insertion time, data retrieval, and storage space between traditional relational databases and time-series-specific solutions, focusing on MariaDB and InfluxDB. We define metrics to compare response time and storage efficiency, aiming to provide a comparative analysis that highlights the advantages and disadvantages of each solution for projects relying on temporal data. The analysis shows that MariaDB has limitations in managing large volumes of data, especially when the database and its indexes are not well-structured, whereas InfluxDB, designed for time-series data, offers better management and storage optimization.

Keywords: Time Series. Database Performance. Storage Optimization.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACID Atomicidade, Consistência, Isolamento e Durabilidade.

ANSI American National Standards Institute.

B-Tree Balanced Tree.

CSV Comma-Separated Values.

IoT Internet das Coisas.

LSM-Tree Log-Structured Merge Tree.

NoSQL Not Only Structured Query Language.

RAM Random Access Memory.

SGBD Sistema de Gerenciamento de Banco de Dados.

SQL Structured Query Language.

SWAP Swap Area for Memory Management.

TSM Time-Structured Merge Tree.

SUMÁRIO

1 INTRODUÇÃO

No cenário atual, impulsionado pelo rápido avanço da inteligência artificial e pela crescente utilização de redes neurais, a necessidade de uma estrutura de banco de dados eficiente para lidar com grandes volumes de séries temporais tornou-se crucial. A implementação dessas tecnologias exige grandes quantidades de dados ao longo do tempo para treinar e validar modelos de aprendizado de máquina, resultando na geração massiva de informações por dispositivos IoT, exigindo soluções de armazenamento capazes de lidar com essa quantidade de informações de forma eficaz.

Segundo Oliveira Junior e Schimiguel (2019), o crescimento exponencial na quantidade de dados e o contínuo avanço dos recursos para capturá-los têm desafiado as empresas a não apenas lidar, mas também a extrair significado e valor dessa imensa onda de dados brutos.

Com a necessidade de compreender e aprimorar as capacidades de armazenamento e manipulação de séries temporais, dadas as limitações dos bancos de dados relacionais tradicionais, como o MariaDB, diante do volume de dados voltados para séries temporais, a crescente demanda por sistemas mais eficientes para lidar com esse tipo de dados, especialmente em contextos de monitoramento, motiva a busca por soluções alternativas.

Conforme Carr et al. (2023), a modelagem de dados é um aspecto crítico para o sucesso da IoT, pois a qualidade e eficiência das aplicações dessas tecnologias dependem diretamente da adequada estruturação e organização dos dados. Compreender os desafios e soluções envolvidos nesse processo é fundamental para o desenvolvimento de sistemas mais eficientes.

Em comparação com sistemas relacionais tradicionais, a capacidade de lidar eficientemente com dados de séries temporais, especialmente em termos de velocidade de inserção e recuperação de dados, além do espaço de armazenamento necessário, se revela como um desafio significativo. Esse cenário se acentua ainda mais em ambientes de monitoramento, nos quais a geração de dados é constante e em grande volume.

A relevância deste estudo reside na necessidade de soluções eficientes para o armazenamento e manipulação de grandes volumes de dados de séries temporais, uma demanda crescente no contexto da IoT e da inteligência artificial. A escolha do tema justifica-se pela importância de identificar as limitações dos bancos de dados relacionais tradicionais, como o MariaDB, e comparar seu desempenho com soluções especializadas, como o InfluxDB. Este trabalho busca oferecer uma análise detalhada que auxilie na escolha do sistema de banco de dados mais adequado para projetos intensivos em dados de séries temporais.

1.1 Objetivos

1.1.1 Objetivo geral

Avaliar o desempenho dos sistemas gerenciadores de bancos de dados MariaDB e InfluxDB em termos de espaço ocupado em disco, tempo de inserção de dados e tempo recuperação de dados em diferentes configurações, no contexto de armazenamento de grandes volumes de dados de séries temporais.

1.1.2 Objetivos específicos

Com base no objetivo geral, os seguintes objetivos específicos foram elencados:

- Determinar os mecanismos tradicionais de configuração de bancos de dados relacionais e de séries temporais para aplicações de armazenamento de grandes volumes de dados de séries temporais.
- Definir a metodologia dos experimentos, incluindo a definição dos mecanismos de inserção e recuperação de dados e das técnicas para obtenção das métricas de desempenho a serem analisadas.
- Estudar e comparar o desempenho entre MariaDB e InfluxDB em termos de tempos de inserção e recuperação de dados, e de uso de espaço de armazenamento.

1.2 Organização do texto

O texto está organizado da seguinte forma: No Capítulo 1, apresenta-se o contexto do estudo, a justificativa, a definição do problema e os objetivos da pesquisa. No Capítulo 2, são discutidos os conceitos teóricos fundamentais, abordando Big Data, armazenamento de séries temporais, bancos de dados relacionais Structured Query Language (SQL), com foco no MariaDB, e bancos de dados especializados em séries temporais, exemplificados pelo InfluxDB. No Capítulo 3, são apresentados a proposta do trabalho, incluindo o ambiente de teste e as métricas de desempenho utilizadas. No Capítulo 4, são apresentados os resultados, já o Capítulo 5 apresenta a conclusão do estudo.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem a finalidade de fundamentar e explorar os conceitos e tecnologias utilizados neste trabalho, a fim de contextualizar o estudo e consolidar as motivações que justificam a comparação entre os bancos de dados MariaDB e InfluxDB.

2.1 Big Data

O rápido crescimento da quantidade de dados e os avanços nas tecnologias de captura têm desafiado as empresas a extrair valor dessa vasta quantidade de informações. Segundo o estudo de Oliveira Junior e Schimiguel (2019), a tecnologia permite a coleta de dados em uma velocidade impressionante, tanto em termos de volume quanto de variedade. Entre as principais fontes de dados no contexto de Big Data, destacam-se redes sociais, mídia, data warehouses e sensores.

Segundo Oliveira Junior e Schimiguel (2019), empresas há muito tempo investem em estruturas especializadas para armazenamento de dados, conhecidas como data warehouses. Esses armazéns de dados consistem em coleções de dados históricos que são mantidos e catalogados para recuperação fácil, seja para uso interno ou para cumprir exigências regulatórias.

Mais recentemente, a coleta de dados de dispositivos IoT e sensores tornou-se um aspecto crucial do Big Data. Tradicionalmente, sensores eram utilizados na indústria para monitorar medições como pressão, temperatura e umidade. Nesse sentido, ressalta o trabalho de Oliveira Junior e Schimiguel (2019), com a popularização de dispositivos portáteis, como smartwatches e aparelhos de casa inteligente, indivíduos agora podem gerar volumes de dados comparáveis aos de grandes instalações industriais. Essa mudança está transformando a maneira como os dados são coletados e utilizados, permitindo uma análise mais detalhada e em tempo real de diversas atividades e ambientes.

A proliferação de dispositivos conectados à IoT resulta em uma enorme quantidade de dados, variando em formato, velocidade de geração, volume e estrutura. A diversidade desses dados exige estratégias eficazes para lidar com sua heterogeneidade, garantindo a integridade, segurança e disponibilidade das informações. Conforme Venceslau (2021), dispositivos com sensores, atuadores e sistemas de comunicação são capazes de coletar dados do ambiente, processá-los e tomar ações baseadas nesses dados, criando ambientes inteligentes e autônomos que melhoram a eficiência, a tomada de decisões e a qualidade de vida.

A IoT transformou profundamente nossa interação com o mundo e as tecnologias

ao nosso redor. Esse ecossistema diversificado de dispositivos conectados, cada um com funcionalidades únicas, desafia os métodos tradicionais de modelagem de dados. Além disso, o enorme volume e a alta velocidade de dados gerados em tempo real pela IoT requerem que os sistemas sejam capazes de processar e armazenar essas informações de maneira eficiente, mantendo a agilidade e a capacidade de resposta.

Segundo Carr et al. (2023) deste modo, o que se tem é que a diversidade de dispositivos conectados na IoT apresenta desafios complexos para a modelagem de dados. A adaptação a essa heterogeneidade, o tratamento de grandes volumes de dados em tempo real, a segurança, a interoperabilidade e a escalabilidade são todos elementos essenciais para construir um ecossistema IoT eficaz.

2.2 Armazenamento de Séries Temporais

A necessidade crescente de armazenamento de dados devido ao avanço da inteligência artificial e IoT trouxe à tona a importância de bancos de dados eficientes. Tradicionalmente, bancos de dados relacionais como o MariaDB são amplamente utilizados, mas enfrentam limitações no gerenciamento de grandes volumes de dados de séries temporais.

Conforme o estudo de Oliveira Junior e Schimiguel (2019), o crescimento exponencial dos dados gerados por dispositivos IoT desafia a capacidade dos bancos de dados tradicionais em lidar com esses volumes de forma eficiente. A ingestão contínua de dados, por exemplo, é uma característica, onde o sistema deve ser capaz de processar e armazenar fluxos constantes de informações sem interrupções. Além disso, a necessidade de realizar agregações complexas em grandes volumes de dados, como cálculos de médias, máximos, mínimos e outras estatísticas ao longo de intervalos temporais variados, pode sobrecarregar bancos de dados tradicionais, comprometendo a eficiência.

Carr et al. (2023) também destacam a importância da modelagem de dados adequada para garantir a eficiência das aplicações de IoT. À medida que os volumes de dados aumentam exponencialmente, é necessário implementar estratégias de armazenamento que equilibrem a preservação de informações essenciais com a otimização do espaço e da velocidade de acesso, garantindo que consultas históricas não degradem o desempenho do sistema.

Segundo Lima (2023), os Sistema de Gerenciamento de Banco de Dados (SGBD) são programas de software que permitem armazenar, gerenciar e recuperar grandes volumes de dados de forma eficiente. Eles fornecem uma interface para os usuários interagirem com o banco de dados, permitindo a execução de consultas, inserções, atualizações e exclusões de dados. Esses sistemas são essenciais em várias aplicações, desde o gerenciamento de estoques até redes sociais, e podem ser classificados em diferentes tipos, como relacionais e não relacionais.

Desde a década de 60, com o desenvolvimento de um dos primeiros SGBD, o Integrated Data Store de Charles Bachman, evoluíram de tecnologias baseadas em fitas magnéticas e cartões perfurados para o uso de discos, permitindo consultas mais rápidas e flexíveis. Esse avanço teve grande importância para o gerenciamento de grandes volumes de dados e para a implementação de sistemas de informação em diversos setores.

Conforme o estudo de Elgrably (2015), a partir dos anos 70 o Modelo Relacional de Edgar Codd revolucionou os SGBD, resolvendo problemas de relacionamento entre entidades e trazendo maior flexibilidade na manipulação dos dados. Apesar de sua introdução em 1969, os primeiros sistemas relacionais só surgiram quase uma década depois, mas rapidamente se consolidaram no mercado. Hoje, SGBD relacionais como MySQL, MariaDB, PostgreSQL e Oracle são amplamente utilizados por organizações em todo o mundo devido à sua robustez e capacidade de atender às demandas modernas de gerenciamento de dados.

A principal limitação dos bancos de dados relacionais em relação ao armazenamento de séries temporais reside na sua estrutura e nos mecanismos de índice. Conforme Dlugokenski (2016), uma das possíveis soluções para essa dificuldade que se está perseguindo é a adoção de soluções não tradicionais, como bancos de dados não relacionais, ou ainda, bancos especializados em determinado tipo de dados, no caso específico, bancos de séries temporais. Bancos de dados especializados, como o InfluxDB, são projetados para séries temporais, otimizando operações e proporcionando inserções e consultas mais rápidas e eficientes.

No estudo de Carr et al. (2023), é enfatizada a importância de uma estrutura capaz de lidar com o grande volume, variedade e velocidade dos dados das aplicações IoT, destacando os desafios enfrentados pelos sistemas tradicionais, como a dificuldade de lidar com dados em tempo real e a necessidade de alta disponibilidade. Em contrapartida, bancos de dados especializados, como os focados em séries temporais, oferecem soluções otimizadas para armazenar, consultar e analisar grandes volumes de dados, garantindo a integridade e acessibilidade contínua das informações.

2.2.1 Bancos Relacionais

O Modelo Relacional transformou a forma como os usuários interagiam com bases de dados ao introduzir uma linguagem de alto nível que abstraía a representação física dos dados. Isso aumentou a produtividade por não precisar mais lidar com apontadores físicos ou modificar seus softwares sempre que a estrutura física dos dados mudava, permitindo maior flexibilidade no gerenciamento de bancos de dados.

Nesse modelo, os dados são organizados em tabelas que implementam o conceito de relação. Cada banco de dados é composto por várias tabelas, definidas por colunas

que formam seu esquema. Para manipular essas tabelas, foi criada a linguagem SQL, baseada na álgebra relacional, que permite executar uma ampla variedade de consultas de maneira eficiente.

Conforme o estudo de Elgrably (2015), a linguagem SQL, desenvolvida pela IBM nos anos 70 como parte do projeto System R, foi criada para simplificar a interação entre usuários e bancos de dados relacionais, sendo inicialmente chamada de Sequel. Com o tempo, evoluiu para SQL e se tornou o padrão global para manipulação de dados nesses sistemas. Embora sua implementação tenha variado entre diferentes produtos no início, a SQL foi posteriormente padronizada pela American National Standards Institute (ANSI), garantindo maior uniformidade. Hoje, é amplamente utilizada para executar diversas tarefas, como inserir, modificar e consultar dados, criar objetos no banco de dados, gerenciar usuários e controlar transações, sendo essencial para a operação de SGBD.

Segundo Silberschatz (2016), o modelo relacional tornou-se o principal modelo para aplicações comerciais de processamento de dados, em grande parte devido à sua simplicidade e flexibilidade. Permitindo manipular dados de forma mais intuitiva, utilizando tabelas relacionadas, o que facilita a modelagem e a consulta dos dados. Diferentemente dos modelos de rede e hierárquico, que exigiam maior controle sobre as estruturas físicas e relações mais rígidas entre os dados, o Modelo Relacional abstrai a complexidade, oferecendo uma interface de alto nível para operações de consulta e gerenciamento. Essa combinação de simplicidade e eficiência fez com que o modelo relacional se tornasse a escolha dominante no mercado.

Os bancos de dados relacionais utilizam a linguagem SQL para a manipulação e consulta de dados, proporcionando uma interface padronizada e poderosa para interagir com os dados armazenados. Segundo Oliveira et al. (2018), banco de dados relacional é todo banco cujo armazenamento é realizado em relações de tabelas. Cada relação é composta de tuplas (ou registros) e atributos (ou colunas). Cada registro na tabela é identificado por um campo chave que contém um valor único, assim a capacidade de realizar operações de forma eficiente e consistente faz com que os bancos de dados relacionais sejam amplamente adotados em diversas indústrias.

Entretanto, com o crescimento exponencial dos dados, especialmente os gerados por dispositivos IoT, surgem desafios significativos para os bancos de dados relacionais. A natureza dos dados de séries temporais, caracterizada por um fluxo contínuo e ordenado de registros com carimbos de data e hora, exige um modelo de armazenamento que possa lidar eficientemente com inserções rápidas e consultas de grandes intervalos de tempo.

A estrutura rígida dos bancos de dados relacionais pode levar a problemas de desempenho e escalabilidade quando aplicada a cenários de séries temporais em larga escala. Conforme Dlugokenski (2016), bancos relacionais têm características notáveis como a obrigatoriedade de uma estrutura previamente montada (statically typed schema)

e fortes garantias de Atomicidade, Consistência, Isolamento e Durabilidade (ACID).

Para mitigar essas limitações, várias abordagens têm sido exploradas, como o uso de índices baseados em intervalos de tempo e particionamento de tabelas. Essas técnicas visam melhorar o desempenho das operações de leitura e escrita em bancos de dados relacionais quando utilizados para armazenamento de séries temporais. No entanto, essas soluções muitas vezes introduzem complexidade adicional no gerenciamento e manutenção do banco de dados.

2.2.1.1 MariaDB

O MySQL e o MariaDB estão entre os sistemas de gerenciamento de banco de dados relacional mais amplamente utilizados. Conforme Souza e Oliveira (2019), são reconhecidos por sua escalabilidade, flexibilidade e desempenho robusto, sendo uma escolha popular para uma vasta gama de aplicações, desde pequenos sites até grandes sistemas de *e-commerce* e aplicações empresariais.

Segundo Elgrably (2015), MariaDB é um sistema gerenciador de banco de dados desenvolvido como uma alternativa ao MySQL, criado por alguns dos autores originais do mesmo após a venda deste para a Oracle. Preocupados com a continuidade do projeto como código aberto, decidiram fazer um fork do MySQL, originando o MariaDB, com o apoio da comunidade de software livre. Seu objetivo é oferecer uma plataforma aberta e gratuita, preservando a compatibilidade com o MySQL. A adoção de MariaDB por grandes empresas e organizações tem crescido. O Google, por exemplo, migrou suas bases de dados do MySQL para MariaDB, buscando maior controle sobre o desenvolvimento e evitando a dependência da Oracle. Empresas que prestam serviços para base de dados de nuvem têm colaborado para facilitar a migração de aplicações do MySQL para MariaDB, reforçando o papel do sistema como uma solução de banco de dados eficiente e flexível.

Conforme o estudo de Tavares (2021), a MariaDB Foundation oferece ferramentas avançadas para diferentes motores de armazenamento, que trazem flexibilidade ao adaptar o banco de dados a diversas estratégias de armazenamento de forma dinâmica. Esses motores podem ser instalados como *plug-ins*, otimizando leitura, escrita ou armazenamento conforme as necessidades de cada tabela.

A seguir, são apresentados alguns dos principais motores de armazenamento, conforme descrito na documentação oficial do MariaDB (2024):

- InnoDB: O motor de armazenamento padrão tanto do MariaDB quanto do MySQL. É projetado para ser generalista, oferecendo estabilidade e desempenho equilibrado em uma ampla gama de cenários. Sua flexibilidade o torna uma escolha clássica para sistemas com operações tanto de leitura quanto de escrita.

- MyRocks: Este motor é otimizado para operações de escrita. Utiliza técnicas como o enfileiramento de dados em memória para aumentar a taxa de escrita, o que o torna especialmente indicado para aplicações que lidam com grandes volumes de inserções, como as de séries temporais.
- Aria: Projetado para cenários de leitura intensiva, o Aria é recomendado para sistemas onde o volume de leituras é significativamente maior do que o de escritas.
- ColumnStore: Ideal para cargas de trabalho analíticas e híbridas. Otimizada para Big Data e análise de grandes volumes de dados.
- Memory: Este motor armazena os dados diretamente na memória, oferecendo acesso extremamente rápido, porém volátil.
- MyISAM: Antigo motor padrão do MySQL, otimizado para leitura. Embora ainda esteja disponível, o MyISAM é menos usado em novos projetos devido às suas limitações quanto a transações e integridade.
- S3: Um motor de armazenamento otimizado para leitura em nuvem, usado para armazenar grandes volumes de dados em sistemas de arquivos de somente leitura, como o Amazon S3.

Conforme a documentação a engine ColumnStore é a mais indicada para armazenar series temporais no MariaDB, ela é otimizada para Big Data e análise de grandes volumes de dados, o que é essencial para o gerenciamento de séries temporais. Outra possibilidade seria a MyRocks, que é otimizada para escritas intensivas, sendo relevante para escrita frequente de grandes volumes de dados e menos em consultas analíticas.

O MariaDB ColumnStore é um mecanismo de armazenamento projetado para análise de grandes volumes de dados. Erdelt (2022) apontam que sua arquitetura baseada em colunas permite consultas analíticas otimizadas, reduzindo a necessidade de leitura de dados irrelevantes e melhorando a eficiência em análises de séries temporais. Esse modelo o torna uma alternativa viável para cenários de Big Data e Business Intelligence, onde operações como agregações e cálculos estatísticos são frequentes.

Embora o MariaDB ColumnStore seja otimizado para análises complexas de dados estruturados, ele pode não ser tão eficiente quanto bancos de séries temporais, como o InfluxDB, para inserções contínuas e frequentes. Erdelt (2022) destacam que, para cargas de trabalho que exigem recuperação rápida de séries temporais, bancos especializados como InfluxDB apresentam melhor desempenho devido à sua indexação nativa baseada em tempo.

Já para aplicações que exigem um alto volume de inserções contínuas, MyRocks surge como uma alternativa eficiente. Esse mecanismo de armazenamento utiliza a estru-

tura Log-Structured Merge Tree (LSM-Tree), que otimiza gravações sequenciais e reduz a fragmentação do armazenamento. No entanto, conforme Lee, An e Lee (2023), o MyRocks pode sofrer com um padrão de desempenho irregular ao longo do tempo, causado pelo processo de compactação periódica dos níveis da LSM-Tree. Essa reorganização dos dados gera um comportamento oscilatório no desempenho, caracterizado por períodos de alta eficiência seguidos por quedas abruptas.

Por outro lado, o InnoDB, amplamente utilizado em bancos relacionais, enfrenta desafios distintos. Sua estrutura baseada em Balanced Tree (B-Tree) pode levar a uma amplificação de escrita, devido ao espaço subutilizado nos nós da árvore. Isso impacta negativamente a eficiência do armazenamento e pode gerar uma sobrecarga de operações. Para mitigar essas limitações, técnicas como redistribuição de dados entre nós adjacentes e reserva dinâmica de espaço podem ser aplicadas, reduzindo a necessidade de operações excessivas de escrita e melhorando a eficiência geral do banco de dados (LEE; AN; LEE, 2023).

De acordo com o estudo de Souza e Oliveira (2019), os bancos relacionais apresentam um alto poder de armazenamento e desempenho. No entanto, o custo de armazenamento pode variar dependendo de diversos fatores, incluindo o volume de dados, a escolha do mecanismo de armazenamento e a indexação.

Para diminuir as limitações naturais dos bancos relacionais em cenários de séries temporais, várias técnicas de otimização podem ser aplicadas. Conforme Weiland (2016), o particionamento de tabelas, permite dividir grandes volumes de dados em segmentos menores. Isso melhora significativamente o desempenho de consultas que filtram dados por períodos específicos, reduzindo o tempo de leitura. Outra técnica é o uso de índices de cobertura, que permitem que todas as colunas necessárias para uma consulta estejam contidas no índice, minimizando o acesso à tabela base e acelerando a execução da consulta.

2.2.2 Bancos de Dados para Séries Temporais

Os bancos de dados de séries temporais são projetados para armazenar e gerenciar dados que são indexados ao longo do tempo. Esses sistemas são eficientes para lidar com grandes volumes de dados que são gerados continuamente, como aqueles provenientes de sensores IoT, transações financeiras e monitoramento de redes. Segundo Vasile, Avolio e Soloviev (2020), a característica essencial desses bancos de dados é sua capacidade de realizar inserções rápidas e consultas eficientes, facilitando a análise de dados temporais em tempo real.

Conforme Naqvi, Yfantidou e Zimányi (2017), uma das principais vantagens dos bancos de dados de séries temporais é a sua capacidade de compactar e armazenar dados

de maneira eficiente. Técnicas avançadas de compressão permitem que grandes volumes de dados sejam armazenados sem ocupar muito espaço, o que é vital para aplicações que geram dados em alta frequência. Além disso, a indexação baseada em tempo otimiza as consultas, permitindo que grandes intervalos de dados sejam recuperados rapidamente. Esse desempenho é fundamental para a análise de tendências e padrões ao longo do tempo, o que é frequentemente necessário em áreas como finanças, saúde e monitoramento ambiental.

Enquanto bancos de dados relacionais, como o MariaDB, são amplamente utilizados para armazenamento estruturado, eles podem não ser ideais para cargas de trabalho envolvendo séries temporais. Garg et al. (2020) apontam que bancos como InfluxDB, projetados especificamente para armazenar séries temporais, apresentam otimizações que permitem inserções rápidas e consultas eficientes, diferentemente de bancos relacionais que podem enfrentar gargalos devido à necessidade de transações ACID e indexação tradicional.

2.2.2.1 Influx

Segundo Gimeno-Sales et al. (2020), o InfluxDB tem sido amplamente utilizado em sistemas de monitoramento IoT devido à sua capacidade de armazenar e consultar grandes volumes de dados de séries temporais de forma eficiente. Sua arquitetura otimizada permite inserções rápidas e indexação eficiente por tempo, o que o torna uma escolha adequada para aplicações em tempo real, como monitoramento de sensores IoT e análise de dados.

Conforme o estudo de Lima (2023), o InfluxDB é um banco de dados de séries temporais de código aberto, altamente escalável e otimizado para lidar com dados de séries temporais em tempo real. Ele oferece recursos avançados de consulta, armazenamento e agregação de dados, além de suporte a integrações com ferramentas populares de visualização e análise.

Segundo Venceslau (2021), o InfluxDB foi desenvolvido para armazenar e gerenciar dados temporais, como tendências, ciclos, sazonalidades e variações irregulares. Projetado especificamente para atender às necessidades de aplicações que lidam com grandes volumes de dados de séries temporais, destacando-se pelo seu alto desempenho e capacidade de escalabilidade.

Além disso, conforme Lima (2023), o InfluxDB é implementado usando Not Only Structured Query Language (NoSQL). Diferentemente de outros modelos de banco de dados, os bancos de dados de série temporal não possuem uma estrutura fixa, podendo adotar diferentes modelos dentro do mesmo sistema. Essa flexibilidade permite que eles lidem de maneira eficiente com grandes volumes de dados ordenados cronologicamente e com inserções rápidas e frequentes.

De acordo com Lima (2023), a estrutura do InfluxDB é baseada em quatro conceitos principais:

- Fields: Colunas que armazenam dados importantes em formato chave-valor. Cada field tem uma Field Key (nome da coluna) e um Field Value (valor armazenado). Fields não são indexados, o que pode tornar as consultas mais lentas.
- Tags: Pares chave-valor usados para indexar e filtrar dados. São armazenados como metadados e são indexados, facilitando consultas rápidas.
- Timestamps: Valores que indicam quando o ponto de dados foi registrado, armazenados em nanossegundos desde a época Unix (1 de janeiro de 1970).
- Measurements: Equivalentes a tabelas em bancos de dados relacionais. Agrupam tags e fields e organizam os dados armazenados.

Essa combinação permite ao InfluxDB processar e consultar dados temporais de forma ágil e eficiente, aproveitando a indexação por Tags para consultas rápidas e a capacidade de adicionar dados rapidamente por Fields.

Diferentemente dos bancos relacionais, que requerem otimizações manuais para lidar com grandes volumes de dados temporais, o InfluxDB adota uma estrutura otimizada chamada Time-Structured Merge Tree (TSM), permitindo compactação eficiente e alta taxa de ingestão (GRZESIK; MROZEK, 2020). Essa abordagem reduz o uso de armazenamento e melhora o desempenho de consultas temporais, tornando o InfluxDB mais adequado para aplicações IoT.

Conforme descrito na documentação oficial do InfluxData (2024), o InfluxDB utiliza o formato de armazenamento TSM, que permite armazenar grandes volumes de dados de maneira eficiente. A arquitetura TSM também é responsável por gerenciar o processo de gravação e leitura de dados, permitindo a ingestão de dados em tempo real. Além disso, o InfluxDB implementa políticas automáticas de compactação, que reduzem o uso de armazenamento e impactam a performance de leitura de dados temporais

Outro recurso relevante é o uso de consultas contínuas, que permite que agregações e cálculos sejam feitos automaticamente em intervalos regulares. Essas consultas contínuas facilitam o processamento de dados em tempo real, sem sobrecarregar o banco de dados com cálculos frequentes.

3 METODOLOGIA

A proposta deste trabalho é realizar uma comparação entre dois sistemas de gerenciamento de banco de dados: MariaDB, um banco de dados relacional, e InfluxDB, um banco de dados especializado em séries temporais. O objetivo é avaliar o desempenho de ambos em termos de tempo de inserção e recuperação de dados, bem como o espaço de armazenamento.

3.1 Criação das Tabelas

Serão testados quatro bancos de dados relacionais e um banco de dados especializado em séries temporais. Os bancos relacionais serão configurados no MariaDB, versão 11.6.2, com diferentes mecanismos de armazenamento: o primeiro utilizará a engine padrão do MariaDB sem otimizações, o segundo contará com a mesma engine porém com otimizações, o terceiro implementará o MyRocks também com otimizações, e o quarto utilizará o mecanismo ColumnStore. Já o banco de dados não relacional, representado pelo InfluxDB, usará a engine padrão TSM, na versão 2.7.11.

Para este estudo, cada banco de dados foi organizado em uma única tabela, decisão fundamentada em dois principais motivos. Primeiro, conforme destacado por MariaDB (2025), tabelas particionadas não podem conter chaves estrangeiras nem ser referenciadas por elas, isso impede a criação de relacionamentos entre tabelas, limitando a modelagem relacional tradicional. Segundo, para evitar o uso de operações JOIN, dado que estamos lidando com um volume elevado de registros, o que poderia impactar negativamente o desempenho das consultas.

Quadro 1 – Listagem das estruturas dos bancos.

Banco	**Engine**	**Estrutura**
Maria	InnoDB	Vanilla
Maria	InnoDB	Índice + partição
Maria	MyRocks	Índice + partição
Maria	ColumnStore	Fechada
Influx	TSM	Fechada

Fonte: Elaborada pelo autor.

Conforme mostrado no Quadro 1 para os bancos de dados MyRocks e InnoDB, a criação de índices é realizado, uma vez que a indexação torna as buscas mais rápidas. Para gerenciar o crescimento contínuo dos dados, o particionamento permite dividir as operações de leitura e escrita em segmentos menores, melhorando a performance geral do banco de dados.

Por outro lado, as engines ColumnStore e InfluxDB possuem arquiteturas mais fechadas e menos flexíveis para customizações estruturais, pois foram projetadas para cenários específicos. O ColumnStore é otimizado para cargas analíticas e estruturado de forma a não suportar diretamente a criação de índices ou particionamento manual, mas é indicado para processamento de grandes volumes de dados. Já o InfluxDB, com sua engine TSM, segue um padrão próprio para organização de dados temporais, utilizando conceitos como tags, fields e timestamps, otimizando tanto a ingestão quanto as consultas em séries temporais.

Quadro 2 – Listagem das colunas.

Banco	**Engine**	**timestamp**	**value**	**sensor**	**year**
Maria	InnoDB	Yes	Yes	Yes	No
Maria	InnoDB	Yes	Yes	Yes	Yes
Maria	MyRocks	Yes	Yes	Yes	Yes
Maria	ColumnStore	Yes	Yes	Yes	No
Influx	TSM	Yes	Yes	Yes	No

Fonte: Próprio Autor

Já o Quadro 2 detalha as colunas disponíveis em cada configuração testada. Nos bancos relacionais MariaDB com as engines InnoDB e MyRocks, foi estruturada coluna year adicional, para melhorar o desempenho e a organização dos dados, essas coluna facilita a criação de índices compostos e a aplicação de particionamento para escalabilidade.

3.1.1 Banco de Dados MariaDB com InnoDB (Configuração Padrão)

A primeira configuração testada utilizou a engine InnoDB padrão da versão 11.6.2 do MariadDB, sendo um banco relacional tradicional sem otimizações específicas para séries temporais. A tabela foi criada conforme a seguinte estrutura de colunas:

- **`event_timestamp`**: Registro do momento do evento, do tipo `TIMESTAMP`, definido como `NOT NULL`.
- **`temperature`**: Valor da temperatura registrada, armazenado como `FLOAT(4)`, definido como `NOT NULL`.
- **`sensor_name`**: Nome do sensor responsável pela medição, armazenado como `VARCHAR(10)`, definido como `NOT NULL`.

3.1.2 Banco de Dados MariaDB com InnoDB e MyRocks Otimizado

A segunda configuração utilizou a engine InnoDB padrão da versão do MariaDB, enquanto a terceira empregou a engine MyRocks, na versão 8.9.1. Ambas mantiveram a mesma estrutura de tabelas, com otimizações voltadas para a recuperação de séries

temporais. Nessa configuração, foram adicionados índices para acelerar as consultas e implementado um particionamento baseado no ano do registro, proporcionando uma organização mais eficiente dos dados.

Diferentemente do InnoDB, a engine MyRocks não vem habilitada por padrão no MariaDB. Para utilizá-la, é necessário baixar o plugin e instalá-lo manualmente antes de ativá-lo no banco de dados.

As tabelas foram criadas com as seguintes estruturas de colunas:

- **`event_timestamp`**: Registro do momento do evento, do tipo `TIMESTAMP`, definido como `NOT NULL`.
- **`temperature`**: Valor da temperatura registrada, armazenado como `FLOAT(4)`, definido como `NOT NULL`.
- **`sensor_name`**: Nome do sensor, armazenado como `VARCHAR(10)`, definido como `NOT NULL`.
- **`year_number`**: Ano correspondente ao evento, extraído do **`event_timestamp`**, armazenado como `INT` e utilizado para o particionamento dos dados.

Além da estrutura básica, essa configuração incluiu índices adicionais para otimizar as consultas temporais:

- **`idx_sensor`**: Índice sobre a coluna `sensor_name`, acelerando buscas filtradas por sensor.
- **`idx_event_timestamp`**: Índice sobre a coluna `event_timestamp`, otimizando consultas por intervalo de tempo.
- **`idx_year`**: Índice sobre a coluna `year_number`, facilitando consultas por períodos anuais.
- **`idx_sensor_event`**: Índice composto (`sensor_name, event_timestamp`), acelerando buscas que envolvem um sensor específico em um intervalo de tempo.
- **`idx_year_sensor_event_timestamp`**: Índice composto (`year_number`, `sensor_name`, `event_timestamp`), otimizando buscas filtradas por ano, sensor e tempo.

A chave primária foi redefinida para incluir múltiplas colunas. No MariaDB, a coluna utilizada como referência para particionamento deve fazer parte da chave primária, pois o mecanismo de particionamento exige que cada partição contenha chaves primárias distintas para evitar inconsistências nos dados.

- **`PRIMARY KEY`**: Definida sobre as colunas `(year_number, sensor_name, event_timestamp)` para otimizar a segmentação dos dados.

Além dos índices, a tabela foi particionada por ano para distribuir os dados e melhorar as consultas em longo prazo:

- **`PARTITION BY RANGE(year_number)`**: Criado para dividir os dados em diferentes partições baseadas no ano do evento. Cada partição contém registros pertencentes a um intervalo de tempo específico.

Essa abordagem melhora a eficiência das consultas, pois permite que o banco de dados acesse apenas as partições relevantes, evitando varreduras completas na tabela. No entanto, essa otimização tem um custo: o tempo de inserção pode ser maior devido à necessidade de manter a estrutura de particionamento e atualizar múltiplos índices durante cada nova entrada de dados.

3.1.3 Banco de Dados MariaDB com ColumnStore

A quarta configuração testada utilizou a engine ColumnStore na versão 23.02.4, que implementa um modelo de armazenamento orientado a colunas, diferentemente das engines tradicionais baseadas em linhas, como InnoDB e MyRocks. Essa abordagem é otimizada para consultas analíticas de grandes volumes de dados, especialmente aquelas que realizam agregações e filtros em colunas específicas.

A principal vantagem do ColumnStore em relação às outras engines é a capacidade de processar consultas em larga escala de forma mais eficiente, armazenando colunas separadamente para minimizar o carregamento desnecessário de dados. Entretanto, essa engine não suporta particionamento manual ou a criação de índices tradicionais, pois sua estrutura interna já é otimizada para busca e compressão de colunas.

Como o MyRocks, a engine ColumnStore não vem habilitada por padrão no MariaDB. Para utilizá-la, é necessário baixar o plugin e instalá-lo manualmente antes de ativá-lo no banco de dados.

A tabela foi criada com a seguinte estrutura de colunas:

- **`event_timestamp`**: Registro do momento do evento, do tipo `TIMESTAMP`, definido como `NOT NULL`.
- **`temperature`**: Valor da temperatura registrada, armazenado como `FLOAT(4)`, definido como `NOT NULL`.
- **`sensor_name`**: Nome do sensor, armazenado como `VARCHAR(10)`, definido como `NOT NULL`.

Diferentemente das outras engines testadas, o ColumnStore não permite a criação de chaves primárias, pois sua estrutura de armazenamento colunar não trabalha com índices tradicionais. Em vez disso, a engine utiliza um esquema interno de segmentação de dados para otimizar a recuperação de informações. Assim como não oferece suporte para índices, pois as consultas são otimizadas por meio do armazenamento colunar e técnicas de compressão.

3.1.4 Banco de Dados InfluxDB

A quinta configuração testada utilizou o InfluxDB, um banco de dados especializado em séries temporais, projetado para armazenar e processar grandes volumes de dados. Diferente dos bancos relacionais, como MariaDB, que utilizam tabelas estruturadas e índices manuais, o InfluxDB adota um modelo baseado em buckets, measurements, fields e tags, otimizando a indexação e recuperação de dados temporais.

- **`measurement`**: Nome da série de dados, equivalente a uma tabela relacional. Para este estudo, os dados foram armazenados na `sensor_data`.
- **`timestamp`**: Registro do momento do evento, utilizado automaticamente como indexador principal.
- **`fields`**:
  - **`temperature`**: Valor da temperatura registrada, armazenado como um campo (`FLOAT`).
- **`tags`**:
  - **`sensor_name`**: Identificação do sensor responsável pela medição (`STRING`).

Diferentemente das engines testadas no MariaDB, o InfluxDB não exige a definição manual de índices ou chaves primárias. A indexação é feita automaticamente com base na combinação de timestamp e tags, permitindo consultas otimizadas por tempo e metadados.

Além disso, enquanto os bancos relacionais exigem particionamento manual para melhorar a performance, o InfluxDB gerencia automaticamente a retenção dos dados.

3.2 Metodologia de Teste

Para o estudo, foram configurados cinco ambientes de teste independentes, com os bancos de dados sendo executados em contêineres Docker. Um script em Python será

responsável por gerar e inserir os dados em cada tabela, garantindo que todos os modelos recebam exatamente o mesmo conjunto de dados de séries temporais. Durante esse processo, serão registrados o tempo de inserção de cada conjunto, bem como o consumo de armazenamento do banco, memória Random Access Memory (RAM) e Swap Area for Memory Management (SWAP). Posteriormente, serão realizados testes de consulta (query) para comparar os bancos, utilizando consultas proporcionais que retornem os mesmos resultados, tanto no MariaDB como no InfluxDB. Para cada consulta, serão registrados o tempo de execução, o consumo de RAM e SWAP, garantindo que todos os testes utilizem a mesma base de dados e o mesmo computador, assegurando a padronização da configuração de hardware.

3.2.1 Plano de Experimento

Antes de iniciar o experimento, serão realizados testes preliminares para garantir que nenhum fator externo, como limitações de processamento do computador ou problemas no script Python, interfira nos resultados. Medidas adicionais, como desligar conexões Wi-Fi e Bluetooth, serão adotadas para minimizar possíveis interferências. Durante a execução dos testes, o computador será dedicado exclusivamente ao processamento do experimento, mantendo apenas o terminal necessário aberto para rodar o script, sem qualquer outro uso até a conclusão dos testes.

Para evitar influências na velocidade de escrita, as linhas a serem inseridas são carregadas previamente na RAM, eliminando o tempo de abertura e leitura do arquivo Comma-Separated Values (CSV) durante a inserção. Além disso, foram configurados lotes de 100 mil linhas para cada operação. Nesse processo, os dados são carregados na memória RAM, uma conexão é aberta com o banco de dados, e o tempo de inserção é cronometrado desde o início até a conclusão. Assim que a conexão é estabelecida, o tempo de início é registrado e, ao final da inserção, o tempo total é medido, assim tendo o tempo que a inserção levou. Além do tempo de inserção, também são monitorados o consumo de RAM e SWAP.

3.2.1.1 Configuração do Ambiente e Testes

Os experimentos serão realizados localmente, em um ambiente de teste controlado, onde os bancos de dados rodam em contêineres Docker na mesma máquina que executa o script Python responsável por rodar os experimentos de inserções e consultas.

Antes de iniciar o experimento, os bancos de dados são iniciados do zero, garantindo que não haja nenhuma informação pré-existente. As tabelas são criadas em cada banco de acordo com as especificações previamente definidas, e, em seguida, inicia-se o processo de inserção dos dados, com o monitoramento dos parâmetros.

As políticas de retenção de dados foram desativadas, garantindo que os dados permaneçam íntegros em todos os bancos, sem um tempo de vida predefinido. Além disso, para otimização dos SGBD, foram consideradas apenas a paginação e a criação de índices, sem levar em conta o cache interno das consultas realizadas.

Para avaliar os diferentes sistemas, foram implementadas tabelas com estruturas ajustadas às capacidades e características de cada engine. As tabelas foram criadas com base nos seguintes critérios:

- **MariaDB com InnoDB básico:** A engine InnoDB é a configuração padrão no MariaDB e foi escolhida para representar um banco de dados relacional tradicional, sem otimizações específicas. Esse modelo busca avaliar o desempenho em cenários básicos de inserção e consulta de dados, servindo como uma linha de base para as comparações.

- **MariaDB com InnoDB otimizado:** Utilizando a mesma engine InnoDB, este modelo foi configurado com otimizações específicas, como particionamento de dados por ano e criação de índices para acelerar consultas em intervalos temporais específicos. A intenção é comparar diretamente com o InnoDB básico, analisando o impacto dessas otimizações. Além das otimizações relacionadas à indexação e paginação, não foram realizadas outras melhorias no SGBD. O foco da análise permaneceu exclusivamente nesses dois aspectos, sem ajustes adicionais na configuração do mecanismo de armazenamento, cache ou estruturação dos dados."

- **MariaDB com MyRocks otimizado:** A engine MyRocks foi projetada para cenários com alta taxa de escrita e otimizada para operações intensivas de inserção. Neste estudo, utilizou-se particionamento por ano e criação de índices específicos, permitindo analisar seu desempenho em comparação com outras engines e avaliar seu comportamento em cenários que demandam grande volume de dados. Assim como o InooDB não foram realizadas outras melhorias no SGBD além das otimizações relacionadas à indexação e paginação

- **MariaDB com ColumnStore:** A engine ColumnStore é projetada para processamento de grandes volumes de dados, especialmente séries temporais. Sua arquitetura é mais fechada e predefinida, o que significa que não suporta a criação de índices ou particionamento manualmente.

- **InfluxDB:** O InfluxDB é um banco de dados especializado em séries temporais, projetado para lidar com altas taxas de escrita e leitura. Sua estrutura utiliza tags e fields, que otimizam a indexação e compactação, permitindo consultas rápidas e eficiente armazenamento de grandes volumes de dados temporais. Ele é o modelo de referência para comparar com soluções baseadas em bancos relacionais.

3.2.1.2 Preparação dos Dados

Os dados de entrada consistem em séries temporais geradas por um script em Python, desenvolvido para este estudo. Cada registro contém três colunas: a primeira representa um timestamp, a segunda armazena o valor referente à informação associada àquele momento, e a terceira identifica o nome do sensor, sendo que cada registro é obtido a partir de medições simuladas a cada 5 segundos. Esses dados são organizados em arquivos locais no formato CSV, totalizando aproximadamente 3,6 GB e 100 milhões de linhas, garantindo que estejam devidamente preparados para serem carregados e processados nos bancos de dados.

3.2.1.3 Query para comparação

A seguir estão as query realizadas para comparar os bancos de dados:

- **Query 1: Informações de 1 Ano para um Sensor Específico**

  Essa consulta filtra os dados correspondentes a um intervalo de 1 ano para um único sensor, retornando aproximadamente 6 milhões de linhas.

  Consulta em bancos relacionais (SQL):

```
SELECT event_timestamp, temperature, sensor_name
FROM {table_name}
WHERE event_timestamp >= '2023-01-01 00:00:00'
AND event_timestamp < '2024-01-01 00:00:00'
AND sensor_name = 'Sensor A';
```

  Consulta otimizada em bancos relacionais (SQL)

```
SELECT event_timestamp, temperature, sensor_name
FROM {table_name}
WHERE year_number = 2023
AND sensor_name = 'Sensor A';
```

  Consulta no InfluxDB (Flux):

```
from(bucket: "influx_bucket")
|> range(start: 2023-01-01T00:00:00Z, stop: 2024-01-01T00:00:00Z)
|> filter(fn: (r) => r._measurement == "sensor_data")
|> filter(fn: (r) => r._field == "temperature")
```

```
|> filter(fn: (r) => r.sensor_name == "Sensor A")
|> keep(columns: ["_time", "_value", "sensor_name"])
|> yield(name: "complete_data")
```

Nessas consultas, os dados são filtrados para um intervalo de um ano, considerando um sensor específico. Além disso, as colunas retornadas incluem o timestamp, o valor da temperatura e o nome do sensor.

A principal diferença entre a consulta SQL normal e a consulta otimizada está na forma como o filtro temporal é aplicado e na eficiência da paginação. Na consulta tradicional, a cláusula WHERE utiliza um intervalo de datas explícito, em contrapartida, na consulta otimizada, a filtragem ocorre diretamente sobre a paginação.

- **Query 2: Informações de 1 Dia sem Restrições de Sensores**

  Essa consulta filtra os dados de um único dia para todos os sensores, retornando aproximadamente 138 mil linhas.

  Consulta em bancos relacionais (SQL):

```
SELECT event_timestamp, temperature, sensor_name
FROM {table_name}
WHERE event_timestamp >= '2023-01-02 00:00:00'
AND event_timestamp < '2023-01-03 00:00:00';
```

  Consulta no InfluxDB (Flux):

```
from(bucket: "influx_bucket")
|> range(start: 2023-01-02T00:00:00Z, stop: 2023-01-03T00:00:00Z)
|> filter(fn: (r) => r._measurement == "sensor_data")
|> filter(fn: (r) => r._field == "temperature")
|> keep(columns: ["_time", "_value", "sensor_name"])
|> yield(name: "complete_data")
```

  Nessas consultas, os dados são filtrados para um intervalo de 24 horas, abrangendo todos os sensores registrados no banco. As colunas retornadas incluem o timestamp, o valor da temperatura e o nome do sensor.

  No caso dos bancos relacionais, a otimização da foi feita adicionando uma condição que aproveite a paginação dos dados, similar à abordagem utilizada na Query 1, reduzindo a necessidade de varredura completa da tabela e melhorando o desempenho.

- **Query 3: Média Semanal para um Sensor Específico**

  Essa consulta filtra 5 meses de dados de um único sensor, agrupando-os por semana e calculando a média da temperatura em cada intervalo semanal. Como resultado, a consulta retorna 22 linhas.

  Consulta em bancos relacionais (SQL):

```
SELECT
    YEARWEEK(event_timestamp, 1) AS week_interval,
    AVG(temperature) AS avg_temp
FROM sensor_data
WHERE sensor_name = 'Sensor B'
AND event_timestamp >= '2023-01-02 00:00:00'
AND event_timestamp < '2023-06-01 00:00:00'
GROUP BY week_interval
ORDER BY week_interval;
```

  Consulta no InfluxDB (Flux):

```
from(bucket: "influx_bucket")
|> range(start: 2023-01-02T00:00:00Z, stop: 2023-06-01T00:00:00Z)
|> filter(fn: (r) => r._measurement == "sensor_data")
|> filter(fn: (r) => r.sensor_name == "Sensor B")
|> filter(fn: (r) => r._field == "temperature")
|> keep(columns: ["_time", "_value", "sensor_name"])
|> aggregateWindow(every: 1w, fn: mean, createEmpty: false)
```

  Nessas consultas, os dados são filtrados para um período de 5 meses, considerando apenas um sensor específico. Em seguida, os valores são agrupados por semana e a média semanal da temperatura é calculada.

  A consulta otimizada em SQL inclui um filtro adicional para aproveitar a paginação dos dados, similar à abordagem utilizada na query 1 e 2.

  As colunas retornadas incluem o intervalo semanal, a média da temperatura e o nome do sensor.

- **Query 4: Soma Mensal para um Sensor Específico**

  Essa consulta filtra todos os dados de um único sensor ao longo do período registrado, agrupando-os por mês e calculando a soma dos valores de temperatura em cada intervalo mensal. Como resultado, a consulta retorna 25 linhas.

Consulta em bancos relacionais (SQL):

```
SELECT
    DATE_FORMAT(event_timestamp, '%Y-%m') AS month_start,
    SUM(temperature) AS sum_temperature
FROM {table_name}
WHERE sensor_name = 'Sensor A'
GROUP BY month_start
ORDER BY month_start;
```

Consulta no InfluxDB (Flux):

```
from(bucket: "influx_bucket")
|> range(start: 0)
|> filter(fn: (r) => r._measurement == "sensor_data")
|> filter(fn: (r) => r.sensor_name == "Sensor A")
|> filter(fn: (r) => r._field == "temperature")
|> keep(columns: ["_time", "_value", "sensor_name"])
|> aggregateWindow(every: 1mo, fn: sum, createEmpty: false)
```

Nessas consultas, os dados são filtrados para um período completo, considerando apenas um sensor específico. Em seguida, os valores são agrupados por mês e a soma mensal da temperatura é calculada.

As colunas retornadas incluem o mês de referência, a soma da temperatura e o nome do sensor.

- **Query 5: Média em Intervalos de 15 Minutos para um Sensor Específico**

  Essa consulta filtra todos os dados de um único sensor ao longo do período registrado, agrupando-os em intervalos de 15 minutos e calculando a média dos valores de temperatura para cada intervalo. Como resultado, a consulta retorna aproximadamente 70 mil linhas.

  Consulta em bancos relacionais (SQL):

```
SELECT FROM_UNIXTIME(FLOOR(UNIX_TIMESTAMP(event_timestamp) /
   (15 * 60)) * (15 * 60)) AS interval_15min,
    AVG(temperature) AS avg_temp
FROM {table_name}
WHERE sensor_name = 'Sensor A'
```

```
GROUP BY interval_15min
ORDER BY interval_15min;
```

Consulta no InfluxDB (Flux):

```
from(bucket: "influx_bucket")
|> range(start: 0)
|> filter(fn: (r) => r._measurement == "sensor_data")
|> filter(fn: (r) => r.sensor_name == "Sensor A")
|> filter(fn: (r) => r._field == "temperature")
|> keep(columns: ["_time", "_value", "sensor_name"])
|> aggregateWindow(every: 15m, fn: mean, createEmpty: false)
```

Nessas consultas, os dados são filtrados para um único sensor e agrupados em intervalos de 15 minutos. A média da temperatura é calculada para cada intervalo de tempo.

- **Query 6: Valor Máximo e Mínimo em um Período de 10 Dias**

  Essa consulta filtra os dados de um intervalo de 10 dias sem aplicar restrições a sensores. Como resultado, retorna os valores máximo e mínimo da temperatura registrados nesse período, gerando apenas 2 valores.

  Consulta em bancos relacionais (SQL):

```
SELECT
    MAX(temperature) AS max_temp,
    MIN(temperature) AS min_temp
FROM {table_name}
WHERE event_timestamp >= '2023-01-01 00:00:00'
AND event_timestamp < '2023-01-10 00:00:00';
```

  Consulta no InfluxDB (Flux):

```
from(bucket: "influx_bucket")
|> range(start: 2023-01-01T00:00:00Z, stop: 2023-01-10T00:00:00Z)
|> filter(fn: (r) => r._measurement == "sensor_data")
|> filter(fn: (r) => r._field == "temperature")
|> reduce(
    identity: {max: float(v: "-inf"), min: float(v: "inf")},
```

```
        fn: (r, accumulator) => ({
            max: if r._value > accumulator.max then
                        r._value else accumulator.max,
            min: if r._value < accumulator.min then
                        r._value else accumulator.min
        })
    )
```

Nessas consultas, os dados são filtrados para um intervalo de 10 dias, considerando todas as medições disponíveis no banco. O maior e o menor valor de temperatura registrados nesse período são extraídos.

A consulta otimizada em SQL inclui um filtro adicional para aproveitar a paginação dos dados, similar à abordagem utilizada na query anteriores.

- **Query 7: Contagem total de Linhas**

  Essa consulta retorna a quantidade total de registros na tabela, sem a aplicação de filtros adicionais. O resultado consiste em apenas um valor, representando o total de linhas armazenadas no banco de dados.

  Consulta em bancos relacionais (SQL):

```
SELECT COUNT(*) AS total_rows
FROM {table_name};
```

  Consulta no InfluxDB (Flux):

```
from(bucket: "influx_bucket")
|> range(start: 0)
|> filter(fn: (r) => r._measurement == "sensor_data")
|> count(column: "_value")
|> yield(name: "row_count")
```

  Nessas consultas, a contagem total de registros é obtida sem restrições de tempo ou filtros adicionais.

Nas colunas do InfluxDB, foram aplicados filtros indexados para otimizar a execução das consultas. Caso a consulta fosse realizada sem esses filtros, o tempo de resposta seria significativamente maior, pois o InfluxDB precisaria percorrer um volume muito

maior de séries temporais antes de aplicar os critérios de seleção. A indexação permite que apenas os dados relevantes sejam acessados diretamente.

3.2.1.4 Execução do Experimento

O experimento consistirá em três etapas principais:

- Inserção de dados: serão realizadas em lote por meio de um script Python, que registrará séries temporais em cada banco de dados. Durante o processo, o tempo de execução será monitorado para cada banco, e as métricas de desempenho serão coletadas e registradas para análise.
- Recuperação de dados: Em seguida, consultas serão realizadas para recuperar subconjuntos de dados com base em diferentes intervalos de tempo e cálculos estáticos dentro da query. O tempo de resposta para cada consulta será registrado e comparado, assim como o uso de RAM e SWAP.
- Uso de espaço de armazenamento: Após as inserções, será verificado o espaço de armazenamento utilizado por cada banco.

3.2.1.5 Medição, Coleta de Dados e Ferramentas

As medições serão realizadas múltiplas vezes para garantir os resultados e será utilizado as seguintes ferramentas:

- Python: Utilizado para gerar os dados, gerenciar a inserção e execução das consultas, além de coletar os tempos de resposta, monitoramento do uso de recursos do sistema e geração de gráficos. Foram empregadas as bibliotecas `pandas`, `psutil`, `matplotlib`, `subprocess`, `pymysql`, `datetime` e `influxdb_client`.
- MariaDB e InfluxDB: Os dois sistemas de banco de dados a serem comparados.
- Docker: Para ativar cada uma das configurações dos bancos.

Os testes foram conduzidos com as seguintes especificações da máquina:

- Processador: 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz (4 núcleos, 8 threads, até 4.7 GHz)
- Memória RAM: 16 GB
- Armazenamento: SSD NVMe
- Sistema Operacional: Ubuntu 24.04

Os dados coletados serão analisados, verificando as vantagens e desvantagens de cada configuração de banco de dados para o cenário de séries temporais. A análise incluirá:

- Gráficos comparando o tempo de inserção e recuperação e armazenamento dos dados.
- Além dos gráficos, serão calculadas e analisadas várias métricas estatísticas, como média, desvio padrão, e taxa de transferência dos dados.

4 RESULTADOS OBTIDOS

Os testes realizados consistiram em aproximadamente 100 inserções e 50 rodadas de query para cada configuração, com o objetivo de calcular as médias e os desvios padrão das métricas.

4.1 Análise do Tempo de Inserção, Armazenamento e Memória

A Tabela 1 apresenta as métricas relacionadas ao tempo médio de inserção, o desvio padrão e o espaço de armazenamento utilizado por cada banco de dados. Já a Tabela 2 detalha o consumo de memória RAM e SWAP.

Tabela 1 – Métricas de Inserção e Armazenamento dos Bancos de Dados.

Banco e Engine	Tempo Médio (s)	Desvio padrão (s)	Armazenamento (GB)
InnoDB	4.67	1.23	4.06
InnoDB Otimizado	14.14	3.81	14.98
MyRocks	36.43	7.83	5.71
ColumnStore	24.72	5.33	0.42
InfluxDB	15.29	0.21	0.68

Fonte: Próprio autor.

Os resultados apresentados na Tabela 1 mostram que o InnoDB, em sua configuração básica, obteve o menor tempo médio de inserção. Já o InnoDB Otimizado, configurado com índices e paginação, levou mais que o dobro do tempo de inserção em comparação ao InnoDB básico. Além disso, a otimização resultou em um consumo de armazenamento significativamente maior, o maior entre os bancos testados, isso por conta dos índices e paginação.

O ColumnStore, apesar de ser projetado para consultas analíticas, demonstrou desempenho inferior no contexto de inserções, com um tempo médio elevado e o segundo maior desvio padrão no tempo entre os bancos testados, indicando maior instabilidade nos tempos de execução, porém destacou-se pela eficiência na compactação dos dados, utilizando apenas 0,42 GB.

Por outro lado, o InfluxDB, especializado em séries temporais, apresentou um desempenho próximo ao ColumnStore em termos de compactação, utilizando 0,68 GB para armazenar a mesma quantidade de informações. Seu tempo médio de inserção foi de 15,29 segundos, um valor intermediário em relação aos bancos baseados no MariaDB.

O MyRocks, projetado para otimização de operações de escrita, apresentou um desempenho abaixo do esperado, com um tempo médio de inserção de 36 segundos, sendo

o pior resultado nas inserções e exibindo o maior desvio padrão. Esse comportamento pode ser explicado pela estrutura de armazenamento baseada em LSM-Tree, utilizada pelo MyRocks, em contraste com a abordagem de B-Tree do InnoDB.

Segundo Tirmazi (2025), embora as LSM-Tree sejam eficientes para grandes volumes de inserções em lote e escritas sequenciais, seu modelo de armazenamento depende de operações periódicas de merge e compactação, que podem se tornar gargalos de desempenho. Assim, apesar das vantagens do MyRocks em certas cargas de trabalho, sua arquitetura pode penalizar cenários de inserção em lote quando o custo das operações de compactação se torna significativo.

Tabela 2 – Métricas de Memória dos Bancos de Dados.

Banco e Engine	RAM (GB)	Desvio padrão (GB)	SWAP (GB)
InnoDB	2.60	0.03	0
InnoDB Otimizado	3.02	0.02	0
MyRocks	3.83	0.16	0
ColumnStore	4.02	0.03	0
InfluxDB	5.21	0.39	0

Fonte: Próprio autor.

Os resultados da Tabela 2 evidenciam diferenças significativas no consumo de memória RAM entre os bancos de dados testados. O InnoDB, em sua configuração básica, apresentou o menor consumo médio de RAM (2,60 GB), com uma variação mínima (0,03 GB), sugerindo um uso eficiente da memória. O InnoDB Otimizado, embora tenha melhorado a indexação e paginação, teve um leve aumento no consumo médio de RAM (3,02 GB), mantendo a estabilidade da variação.

O MyRocks, conhecido por sua eficiência em gravações otimizadas, utilizou 3,83 GB de RAM, apresentando uma variação maior (0,16 GB), o que pode indicar flutuações no uso de memória devido à sua compactação e manuseio de dados. Já o ColumnStore, teve um consumo médio um pouco mais elevado (4,02 GB), mas com uma variação estável. Por outro lado, o InfluxDB destacou-se pelo maior consumo médio de RAM (5,21 GB), com a maior variação entre os bancos testados (0,39 GB).

Nenhum dos bancos testados utilizou SWAP, o que indica que todos operaram dentro da capacidade de RAM disponível, sem necessidade de recorrer a memória virtual no disco, o que poderia impactar negativamente o desempenho.

Os resultados apresentados na Tabela 3 permitem calcular a taxa de inserção de linhas por segundo para cada banco de dados testado. O InnoDB, em sua configuração padrão, obteve a melhor performance, inserindo 214.132 linhas em 1 segundo, consolidando-se como a solução mais eficiente para operações de escrita massiva.

O InnoDB Otimizado, apesar das melhorias estruturais, registrou uma taxa signi-

Tabela 3 – Métricas de Inserção dos Dados.

Banco e Engine	Linhas Inseridas	Tempo Médio (s)
InnoDB	214132	1
InnoDB Otimizado	70720	1
MyRocks	27450	1
ColumnStore	40453	1
InfluxDB	65402	1

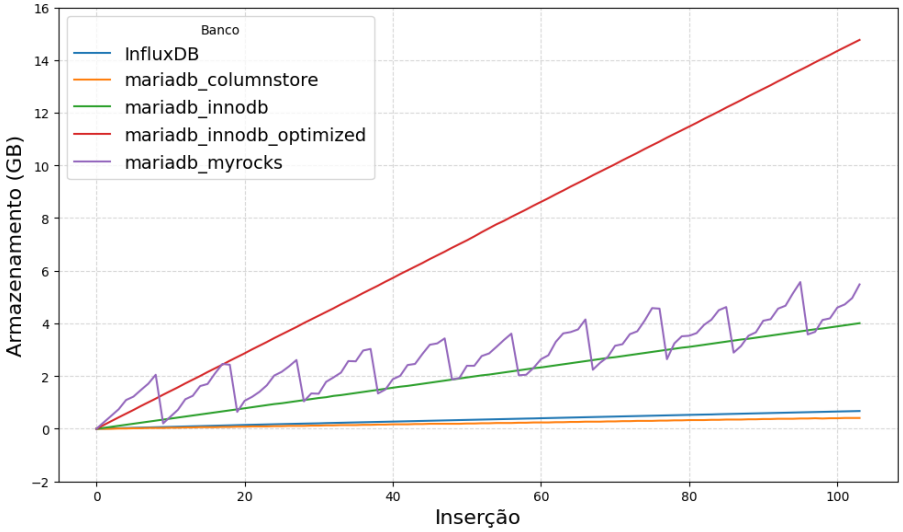
Fonte: Próprio autor.

ficativamente menor, com 70.720 linhas por segundo, devido ao custo adicional de manutenção de índices e paginação, que impactam a velocidade de inserção.

O InfluxDB, especializado em séries temporais, manteve um desempenho intermediário, com 65.402 linhas por segundo, reforçando sua capacidade de lidar com grandes volumes de dados. O ColumnStore, apresentou uma taxa de 40.453 linhas por segundo, um desempenho inferior ao InfluxDB.

Já o MyRocks, conhecido por sua compactação avançada e escrita otimizada, teve a menor taxa de inserção entre os bancos testados, com 27.450 linhas por segundo.

Figura 1 – Armazenamento dos bancos.



Fonte: Próprio autor.

A Figura 1 apresenta uma comparação do uso de armazenamento entre diferentes bancos de dados ao longo do tempo. O MariaDB com ColumnStore e o InfluxDB apresentaram um crescimento quase linear e constante no uso de armazenamento, com consumo significativamente menor em relação aos outros bancos, refletindo sua arquitetura otimizada para grandes volumes de dados. Por outro lado, o MariaDB com MyRocks demonstrou um comportamento cíclico, causado pelos mecanismos de compactação internas assim como pelo particionamento ou pela forma de gestão de índices. O MariaDB com InnoDB Otimizado teve o maior crescimento no uso de armazenamento, sugerindo

que as otimizações aplicadas impactaram a ocupação de espaço, especialmente pela organização de dados e índices. Já o MariaDB com InnoDB padão apresentou um padrão de crescimento mais moderado no uso de armazenamento.

No geral, o ColumnStore se mostrou mais eficiente em termos de compactação de dados e utilização de espaço, logo atrás segue o InfluxDB, que ficou bem próximo a engine do MariaDB, enquanto o InnoDB otimizado demonstrou a maior ocupação de armazenamento com quase 35 vezes o espaço do menor banco, o que pode influenciar na escolha de soluções para aplicações que lidam com grandes volumes de dados.

4.2 Análise do Tempo de consulta e Memória

Foram realizadas 50 rodadas de cada consulta para calcular as médias e os desvios padrão do tempo de execução, além do uso médio de memória RAM e SWAP por cada banco de dados. A seguir, discutimos os resultados obtidos para cada query.

Tabela 4 – Métricas de consulta: 1 dia, 138 mil linhas.

Banco e Engine	Tempo Médio (s)	Desvio padrão (s)	Max RAM (GB)	Desvio padrão (GB)
InnoDB	37.22	0.09	6.42	0.22
InnoDB Otimizado	0.82	0.04	6.61	0.33
MyRocks	0.86	0.04	6.64	0.35
ColumnStore	0.87	0.04	6.74	0.13
InfluxDB	0.51	0.45	6.88	0.06

Fonte: Próprio autor.

A Tabela 4 apresenta as métricas de tempo de execução e consumo de memória para a execução de consultas em um intervalo de 1 dia, retornando um total de 138.240 linhas. Os resultados mostram que o InfluxDB obteve o menor tempo médio de execução, apesar de uma maior variação no tempo de resposta. O ColumnStore voltado para dados ou as engines otimizadas obteveram tempos médios próximos, com menor variação.

O InnoDB padrão, por outro lado, teve um desempenho significativamente inferior, demonstrando limitações na execução da consulta para esse volume de dados com filtro de 1 dia.

No que diz respeito ao consumo de memória foi relativamente próxima para todas as engines. Além disso, em todos os testes realizados, o uso de swap foi zero, indicando que a carga de trabalho foi inteiramente suportada pela memória RAM disponível.

A Tabela 5 apresenta as métricas de execução de consultas considerando um período de 1 ano, retornando um total de 6.307.200 linhas. Os resultados mostram que o InfluxDB obteve o menor tempo médio de execução, seguido pelo MyRocks, InooDB Otimizado e ColumnStore, que apresentou um tempo médio maior, enquanto o InnoDB padrão teve um tempo de execução significativamente superior.

Tabela 5 – Métricas de consulta: 1 ano, 6 milhões de linhas.

Banco e Engine	Tempo Médio (s)	Desvio padrão (s)	Max RAM (GB)	Desvio padrão (GB)
InnoDB	50.63	0.38	7.67	0.22
InnoDB Otimizado	27.07	0.61	7.64	0.32
MyRocks	26.82	0.74	7.64	0.35
ColumnStore	29.62	0.16	7.76	0.10
InfluxDB	24.47	0.66	10.05	0.05

Fonte: Próprio autor.

Em relação ao consumo de memória, o InfluxDB apresentou o maior uso de RAM, o que pode ser atribuído à sua estrutura para consultas em séries temporais. Já as engines do MariaDB foram relativamente próximas. Além disso, em todos os testes realizados, o uso de swap foi zero, indicando que a carga de trabalho foi inteiramente suportada pela memória RAM disponível.

Tabela 6 – Métricas de consulta: média do agrupamento, 22 linhas.

Banco e Engine	Tempo Médio (s)	Desvio padrão (s)	Max RAM (GB)	Desvio padrão (GB)
InnoDB	34.86	0.05	6.39	0.22
InnoDB Otimizado	1.94	0.03	6.55	0.33
MyRocks	2.03	0.10	6.58	0.33
ColumnStore	0.49	0.02	6.70	0.13
InfluxDB	0.12	0.01	6.79	0.06

Fonte: Próprio autor.

A Tabela 6 apresenta as métricas de execução para consultas agregadas utilizando a média agrupada em intervalos de semanas no período de 5 meses, retornando um total de 22 linhas. Os resultados demonstram uma diferença significativa no tempo de execução entre as diferentes engines avaliadas.

O InfluxDB se destacou com o menor tempo médio, reforçando sua eficiência para cálculos agregados em séries temporais. O ColumnStore apresentou um desempenho próximo porém levando quase 4 vezes o tempo do InfluxDB, sendo a melhor alternativa entre as engines relacionais.

Já as engines MyRocks e InnoDB Otimizado tiveram tempos médios próximos, indicando um desempenho razoável, mas ainda inferior às soluções especializadas. O InnoDB padrão, por sua vez, apresentou o pior resultado.

No que diz respeito ao consumo de memória foi relativamente próxima para todas as engines. Além disso, em todos os testes realizados, o uso de SWAP foi zero, indicando que a carga de trabalho foi inteiramente suportada pela memória RAM disponível.

Esses resultados reforçam a superioridade do InfluxDB para operações de agregação temporal, enquanto o ColumnStore se destaca entre as engines relacionais.

Tabela 7 – Métricas de consulta: soma do agrupamento, 25 linhas.

Banco e Engine	Tempo Médio (s)	Desvio padrão (s)	Max RAM (GB)	Desvio padrão (GB)
InnoDB	38.68	0.08	6.39	0.32
InnoDB Otimizado	39.25	0.11	6.37	0.36
MyRocks	32.32	1.04	6.42	0.32
ColumnStore	3.60	0.16	6.76	0.07
InfluxDB	2.05	0.06	6.76	0.05

Fonte: Próprio autor.

A Tabela 7 apresenta as métricas de execução para consultas que realizam a soma dos dados agrupados mensalmente, retornando um total de 25 linhas. Os resultados evidenciam que a engine InfluxDB foi a mais eficiente, seguida pelo ColumnStore, ambos apresentando uma variação de tempo relativamente baixa, o que indica estabilidade no processamento dessas consultas.

Por outro lado, as engines relacionais tradicionais apresentaram desempenho muito inferior, destacando suas limitações para agregações e somas sobre grandes volumes de dados históricos. Notavelmente, a versão otimizada do InnoDB, que utilizava índices e paginação, teve um desempenho pior do que o InnoDB padrão. Isso sugere que, para esse tipo de consulta que busca o histórico completo do banco, as otimizações implementadas não apenas não trouxeram ganhos, como também impactaram negativamente a performance, possivelmente devido à sobrecargas adicionais geradas pelas estratégias de indexação e paginação.

No que diz respeito ao consumo de memória foi relativamente próxima para todas as engines. Além disso, em todos os testes realizados, o uso de SWAP foi zero, indicando que a carga de trabalho foi inteiramente suportada pela memória RAM disponível.

Tabela 8 – Métricas de consulta: média de agrupamento, 70 mil linhas.

Banco e Engine	Tempo Médio (s)	Desvio padrão (s)	Max RAM (GB)	Desvio padrão (GB)
InnoDB	40.12	0.06	6.39	0.32
InnoDB Otimizado	40.92	0.10	6.39	0.36
MyRocks	35.23	0.74	6.40	0.31
ColumnStore	5.67	0.13	6.76	0.07
InfluxDB	2.20	0.09	6.85	0.04

Fonte: Próprio autor.

A Tabela 8 apresenta as métricas de execução para consultas que calculam a média dos dados agregados em intervalos de 15 minutos, retornando 70.177 linhas.

Os resultados mostram que o InfluxDB teve o menor tempo médio de execução, o ColumnStore apresentou um desempenho próximo, consolidando-se como uma boa opção entre os bancos relacionais testados.

As engines InnoDB e MyRocks tiveram tempos de execução significativamente mais altos. O InnoDB otimizado, apesar de apresentar melhorias, ainda obteve um tempo médio elevado, o pior entre os bancos testados. O alto tempo médio dessas engines reforça a dificuldade dos bancos relacionais tradicionais em lidar com agregações temporais em intervalos de tempo.

No que diz respeito ao consumo de memória foi relativamente próxima para todas as engines. Além disso, em todos os testes realizados, o uso de SWAP foi zero, indicando que a carga de trabalho foi inteiramente suportada pela memória RAM disponível.

Tabela 9 – Métricas de consulta: máximo e mínimo.

Banco e Engine	Tempo Médio (s)	Desvio padrão (s)	Max RAM (GB)	Desvio padrão (GB)
InnoDB	35.87	0.09	6.39	0.33
InnoDB Otimizado	2.05	0.13	6.37	0.36
MyRocks	3.80	0.20	6.41	0.31
ColumnStore	0.06	0.00	6.76	0.07
InfluxDB	5.31	0.05	6.56	0.04

Fonte: Próprio autor.

A Tabela 9 apresenta as métricas de execução para consultas que calculam os valores máximos e mínimos em um intervalo de 10 dias, retornando 2 valores.

Os resultados mostram que o ColumnStore teve um desempenho excepcional, com o menor tempo, demonstrando extrema eficiência para operações de agregação baseadas em valores máximos e mínimos. O InnoDB Otimizado e o MyRocks também apresentaram bons resultados.

Já o InfluxDB, que normalmente se destaca em operações temporais, obteve um tempo médio de 5,31s, ficando atrás do ColumnStore e das versões otimizadas do InnoDB e MyRocks. O InnoDB padrão, por outro lado, apresentou um tempo médio significativamente superior.

No que diz respeito ao consumo de memória foi relativamente próxima para todas as engines. Além disso, em todos os testes realizados, o uso de SWAP foi zero, indicando que a carga de trabalho foi inteiramente suportada pela memória RAM disponível.

Tabela 10 – Métricas de consulta: conta linhas.

Banco e Engine	Tempo Médio (s)	Desvio padrão (s)	Max RAM (GB)	Desvio padrão (GB)
InnoDB	27.22	0.07	6.40	0.33
InnoDB Otimizado	34.83	1.54	6.87	0.37
MyRocks	33.06	1.17	6.39	0.31
ColumnStore	0.37	0.01	6.75	0.07
InfluxDB	0.80	0.04	6.54	0.04

Fonte: Próprio autor.

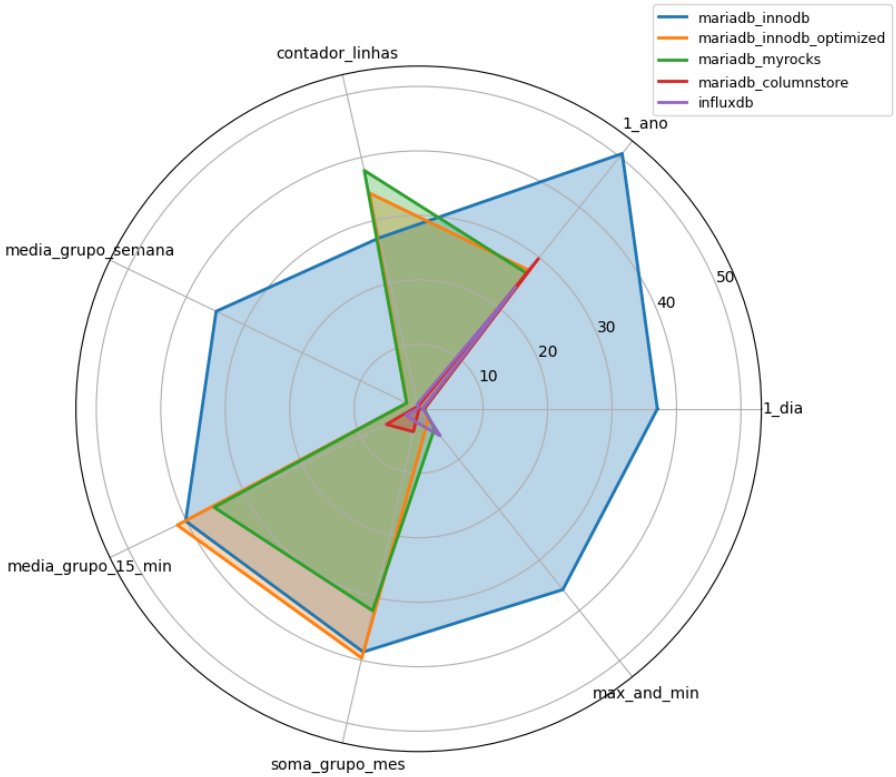
A Tabela 10 apresenta as métricas de execução para consultas que realizam a contagem do número total de linhas na base de dados, retornando 1 valor.

Os resultados indicam que o ColumnStore obteve o melhor desempenho, seguido pelo InfluxDB. Esses tempos reforçam a eficiência dessas engines em operações de contagem, especialmente em bancos projetados para manipulação eficiente de grandes volumes de dados.

Já as engines relacionais tradicionais apresentaram tempos significativamente maiores. O MyRocks e o InnoDB otimizado tiveram tempos mais elevados, respectivamente, sugerindo que a contagem de registros em suas estruturas internas é mais custosa. O InnoDB padrão obteve um tempo relativamente menor, mas ainda assim muito superior ao das soluções otimizadas.

No que diz respeito ao consumo de memória foi relativamente próxima para todas as engines. Além disso, em todos os testes realizados, o uso de SWAP foi zero, indicando que a carga de trabalho foi inteiramente suportada pela memória RAM disponível.

Figura 2 – Comparação tempo médio de consulta.



Fonte: Próprio autor.

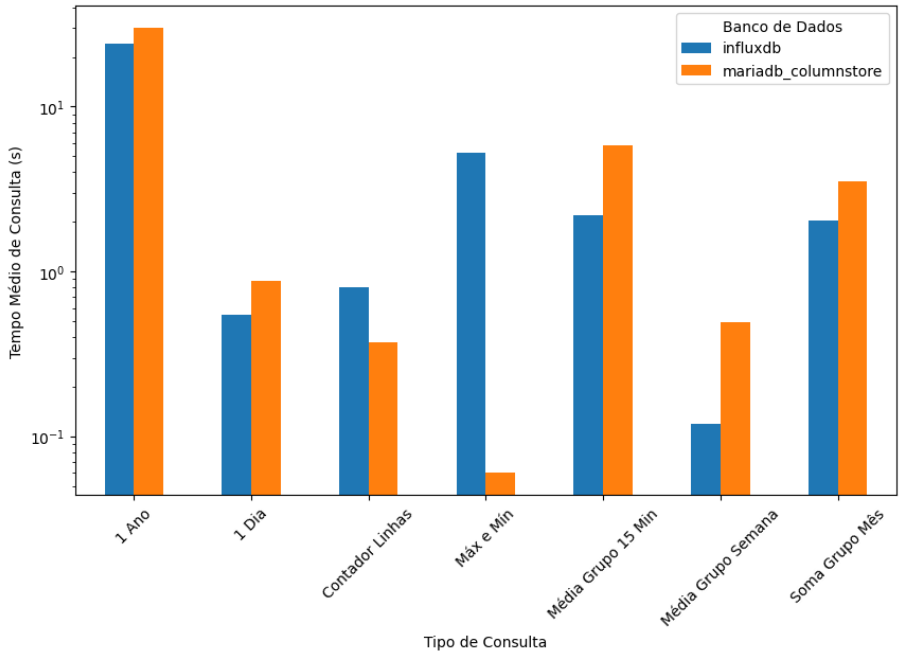
A Figura 2 apresenta a comparação dos tempos médios de consultas. Os resultados indicam que, de maneira inesperada, as versões otimizadas do InnoDB e MyRocks em alguns casos, mesmo contando com índices e paginação, apresentaram um desempenho inferior ao InnoDB padrão nas consultas que recuperam todo o histórico de um sensor.

Esse comportamento pode ser explicado pela sobrecarga gerada pelo uso excessivo de índices, que, embora otimize buscas filtradas, pode tornar a recuperação completa de registros mais lenta devido ao maior número de acessos fragmentados ao armazenamento.

Além disso, a implementação de paginação introduz uma camada extra de gerenciamento de memória que, para consultas que precisam percorrer toda a base de dados, pode resultar em um tempo de execução maior do que uma simples leitura sequencial realizada pelo InnoDB padrão.

Dessa forma, a análise dos tempos médios de execução apresentados na Figura 2 reforça que, embora otimizações como índices e paginação sejam vantajosas em cenários específicos, sua aplicação em consultas que exigem a recuperação completa do histórico de um sensor pode introduzir penalidades significativas.

Figura 3 – Gráfico de barra entre InfluxDB e MariaDB ColumnStore



Fonte: Próprio autor.

A Figura 3 apresenta uma versão filtrada da análise de desempenho das consultas, destacando as duas soluções com os melhores tempos médios de execução: InfluxDB e MariaDB ColumnStore, ambas demonstrando eficiência superior às engines relacionais tradicionais. Vale ressaltar que a escala do eixo Y está em logaritmo, permitindo uma melhor visualização das diferenças de desempenho, especialmente quando há variações significativas entre os tempos de resposta.

O InfluxDB, projetado para armazenamento e consulta de séries temporais, obteve tempos de resposta consistentemente baixos na maioria das categorias analisadas. Sua arquitetura otimizada permitiu maior eficiência, especialmente em consultas que envolvem

filtragem rápida e agregação de dados em intervalos temporais específicos, sendo o mais rápido em 5 das 7 categorias de consulta.

Já o MariaDB ColumnStore, embora não tenha superado o InfluxDB na maioria dos cenários, apresentou desempenho competitivo, especialmente em consultas que envolvem grandes volumes de dados históricos. Sua estrutura orientada a colunas mostrou-se eficiente para cargas de trabalho analíticas, superando o InfluxDB em duas das consultas realizadas.

Esses resultados evidenciam a eficiência de ambas as soluções para diferentes cenários de análise de séries temporais, reforçando que a escolha entre elas deve considerar o tipo de consulta e o volume de dados envolvidos.

5 CONCLUSÃO

Este estudo apresentou uma comparação entre os bancos de dados MariaDB e InfluxDB no contexto do armazenamento e manipulação de séries temporais, considerando métricas como tempo de inserção, tempo de recuperação, uso de armazenamento e consumo de memória. Os experimentos conduzidos demonstraram que a escolha do banco de dados impacta diretamente o desempenho e a eficiência no gerenciamento de grandes volumes de dados temporais.

Os resultados evidenciaram que, enquanto o MariaDB, especialmente com a engine InnoDB, apresentou tempos de inserção mais rápidos em sua configuração padrão, ele teve dificuldades em consultas que exigem agregações temporais ou recuperação de grandes períodos. Já a engine ColumnStore, voltada para cargas analíticas, e o InfluxDB, especializado em séries temporais, apresentaram um desempenho superior em consultas e armazenamento otimizado, reduzindo significativamente o espaço utilizado sem comprometer a velocidade de recuperação dos dados.

Um ponto relevante observado foi que a engine ColumnStore, apesar de fazer parte de um banco relacional, apresentou um desempenho muito próximo, e em alguns casos até superior, ao do InfluxDB, que é um banco especializado em séries temporais. Essa constatação reforça a necessidade de considerar não apenas a performance bruta, mas também fatores como custo de manutenção e facilidade de adoção. Como a ColumnStore utiliza SQL, uma linguagem amplamente difundida no mercado e já dominada por grande parte dos desenvolvedores, sua implementação e manutenção podem ser mais acessíveis. Em contrapartida, o InfluxDB possui uma estrutura diferente, baseada em consultas específicas para séries temporais, exigindo uma curva de aprendizado maior para administradores e desenvolvedores que não estão familiarizados com esse modelo.

Dessa forma, para aplicações que necessitam lidar com grandes volumes de séries temporais, a escolha de um banco de dados especializado, como o InfluxDB, ou de uma engine otimizada para análise de dados, como o ColumnStore, mostra-se uma alternativa significativamente mais eficiente em relação a configuração padrão do banco relacional, mesmo quando otimizado com índices e particionamento. O ganho de desempenho observado reforça a importância de avaliar a natureza dos dados e o perfil de consultas antes de definir a solução de armazenamento, garantindo que o banco de dados escolhido seja adequado ao volume e à complexidade das operações que serão realizadas.

5.1 Trabalhos futuros

Como continuidade deste estudo, sugerem-se os seguintes caminhos para trabalhos futuros:

- **Avaliação da versão 3 do InfluxDB:** No decorrer do desenvolvimento deste trabalho, o InfluxDB lançou a nova versão V.3, que promete ser mais rápida e eficiente. No entanto, essa versão ainda está em fase de testes. Assim, um estudo futuro poderia ampliar a análise de desempenho, considerando novos cenários e incorporando essa atualização do InfluxDB.
- **Impacto de Workloads em Larga Escala:** Avaliar o comportamento dos bancos de dados quando submetidos a volumes de dados ainda maiores, simulando cenários mais próximos de aplicações reais na indústria e na IoT.
- **Análise de Custo-Benefício:** Comparar não apenas o desempenho técnico, mas também os custos de infraestrutura e operação associados ao uso de cada banco de dados em diferentes arquiteturas.
- **Integração com Modelos de Machine Learning:** Explorar como cada banco se comporta na integração com modelos preditivos, avaliando o impacto do tempo de consulta em pipelines de IA e aprendizado de máquina.
- **Testes em Ambientes de Banco de Dados Híbridos:** Investigar a viabilidade de um modelo híbrido que combine bancos relacionais e bancos de séries temporais para otimizar desempenho e armazenamento. Essa abordagem poderia avaliar como o MariaDB pode ser utilizado para armazenar metadados e relações complexas, enquanto o InfluxDB gerencia a ingestão e recuperação eficiente de séries temporais. Estudos futuros poderiam explorar estratégias de sincronização, replicação de dados entre diferentes motores.

Essas futuras investigações poderão aprofundar ainda mais o entendimento sobre a melhor escolha de banco de dados para aplicações que envolvem grandes volumes de séries temporais, possibilitando otimizações mais refinadas e estratégias eficazes de armazenamento e processamento.

REFERÊNCIAS

CARR, C. N. et al. Modelagem de dados para a internet das coisas (iot): desafios e soluções ao modelar dados gerados por dispositivos conectados. *observatório de la economía latinoamericana*, v. 21, n. 12, p. 23858–23874, dez. 2023. Disponível em: https://ojs.observatoriolatinoamericano.com/ojs/index.php/olel/article/view/1421. 21, 24, 25

DLUGOKENSKI, R. *Bancos de dados para monitoramento de desempenho de grandes redes*. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul, 2016. 25, 26

ELGRABLY, I. S. Uma análise experimental entre sistemas gerenciadores de banco de dados open source. *Academia.edu*, 2015. 25, 26, 27

ERDELT, J. J. P. K. Dbms-benchmarker: Benchmark and evaluate dbms in python. *Miscellaneous*, v. 7, n. 79, p. 4628–4628, 2022. 28

GARG, D. et al. Torquedb: Distributed querying of time-series data from edge-local storage. In: MALAWSKI, M.; RZADCA, K. (Ed.). *Euro-Par 2020: Parallel Processing*. Cham: Springer International Publishing, 2020. p. 281–295. ISBN 978-3-030-57675-2. 30

GIMENO-SALES, F. J. et al. Pv monitoring system for a water pumping scheme with a lithium-ion battery using free open-source software and iot technologies. *Sustainability*, MDPI, v. 12, n. 24, p. 10651, 2020. Disponível em: https://www.mdpi.com/2071-1050/12/24/10651. 30

GRZESIK, P.; MROZEK, D. Comparative analysis of time series databases in the context of edge computing for low power sensor networks. In: KRZHIZHANOVSKAYA, V. V. et al. (Ed.). *Computational Science – ICCS 2020*. Cham: Springer International Publishing, 2020. p. 371–383. ISBN 978-3-030-50426-7. 31

INFLUXDATA. *InfluxDB Storage Engine Documentation*. [S.l.], 2024. Acessado em: 12 ago. 2024. Disponível em: https://docs.influxdata.com/influxdb/v1/concepts/storage_engine. 31

LEE, B.-H.; AN, M.; LEE, S.-W. A case for space compaction of b-tree nodes on flash storage. *IEEE Access*, v. 11, p. 38149–38156, 2023. 29

LIMA, M. C. de. *Um estudo sobre bancos de dados séries temporais*. Dissertação (Mestrado) — Universidade Federal de Santa Maria, 2023. 24, 30, 31

MARIADB. *MariaDB Storage Engines Documentation*. [S.l.], 2024. Acessado em: 12 ago. 2024. Disponível em: https://mariadb.com/docs/server/storage-engines/. 27

MARIADB. *Foreign Keys*. 2025. Acesso em: 20 fev. 2025. Disponível em: https://mariadb.com/kb/en/foreign-keys/. 33

NAQVI, S. N. Z.; YFANTIDOU, S.; ZIMÁNYI, E. Time series databases and influxdb. *Studienarbeit, Université Libre de Bruxelles*, v. 12, p. 1–44, 2017. 29

Oliveira Junior, J. C. de; SCHIMIGUEL, J. Implementando uma plataforma big data para visualizaçao de dados gerados por dispositivo iot. *Revista de Ubiquidade*, v. 2, n. 2, p. 85–111, 2019. 21, 23, 24

OLIVEIRA, M. d. S. et al. Banco de dados no-sql x banco de dados sql: estudo de desempenho em grandes massas. *South American Development Society Journal*, v. 4, n. 11, p. 298, ago. 2018. Disponível em: https://www.sadsj.org/index.php/revista/article/view/162. 26

SILBERSCHATZ, A. *Database System Concepts*. 1. ed. Rio de Janeiro: Elsevier, 2016. 20-25 p. Traduzido para o português. ISBN 9788535245356. 26

SOUZA, E. C. de; OLIVEIRA, M. R. de. comparativo entre os bancos de dados mysql e mongodb: quando o mongodb é indicado para o desenvolvimento de uma aplicação. *fatectq*, interfacetecnologica, 2019. Disponível em: https://revista.fatectq.edu.br/interfacetecnologica/article/view/664/411. 27, 29

TAVARES, J. K. *Avaliação de desempenho de bancos de dados para armazenamento de séries temporais*. Dissertação (Mestrado) — Universidade Federal de Campina Grande, 2021. 27

TIRMAZI, H. *LSM Trees in Adversarial Environments*. 2025. Cryptology ePrint Archive, Paper 2025/218. Disponível em: https://eprint.iacr.org/2025/218. 50

VASILE, M.-E.; AVOLIO, G.; SOLOVIEV, I. Evaluating influxdb and clickhouse database technologies for improvements of the atlas operational monitoring data archiving. *Journal of Physics: Conference Series*, IOP Publishing, v. 1525, n. 1, p. 012027, apr 2020. Disponível em: https://dx.doi.org/10.1088/1742-6596/1525/1/012027. 29

VENCESLAU, F. A. M. *Sistema para suporte à implantação e monitoramento de aplicações de campus inteligente baseado em protocolos de IoT*. Dissertação (Mestrado) — Universidade Federal da Paraíba, 2021. 23, 30

WEILAND, E. *Ambiente de recomendação de índices para bancos de dados MySQL*. Dissertação (Mestrado) — Universidade de Santa Cruz do Sul, 2016. 29

Apêndices

APÊNDICE A – REPOSITÓRIO DO CÓDIGO-FONTE

Este apêndice apresenta o link para o repositório GitHub onde está disponível o código desenvolvido para este trabalho. O repositório contém os *scripts* utilizados para a implementação, testes e análise comparativa entre os bancos de dados MariaDB e InfluxDB, bem como as consultas utilizadas para a avaliação do desempenho.

O código pode ser acessado no seguinte link:

https://github.com/VirgilioFilipi/TCC_Database_comparison_of_databases

O repositório inclui instruções sobre como executar os *scripts*, detalhes sobre a estrutura do projeto e informações adicionais para reproduzir os experimentos realizados.