UNIVERSIDADE FEDERAL DE SANTA MARIA
COLÉGIO POLITÉCNICO DA UFSM
CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS PARA INTERNET

Marcelo Costa de Lima

UM ESTUDO SOBRE BANCOS DE DADOS SÉRIES TEMPORAIS

Santa Maria, RS
2023

Marcelo Costa de Lima

UM ESTUDO SOBRE BANCOS DE DADOS SÉRIES TEMPORAIS

Trabalho de Conclusão de Curso apresentado ao Curso Superior de Tecnologia em Sistemas para Internet da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Tecnólogo em Sistemas para Internet**.

Orientador: Prof. Dr. Daniel Lichtnow

Santa Maria, RS
2023

Marcelo Costa de Lima

UM ESTUDO SOBRE BANCOS DE DADOS SÉRIES TEMPORAIS

Trabalho de Conclusão de Curso apresentado ao Curso Superior de Tecnologia em Sistemas para Internet da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Tecnólogo em Sistemas para Internet**.

Aprovado em 13 de dezembro de 2023:

Daniel Lichtnow, Dr. (UFSM)
(Presidente/Orientador)

Juçara Salete Gubiani, Dra. (UFSM)

Rafael Gressler Milbradt, Dr. (UFSM)

Santa Maria, RS
2023

DEDICATÓRIA

Aos meus pais, por tudo que fizeram por mim e a toda minha família

AGRADECIMENTOS

Primeiramente agradeço a Deus pela oportunidade de viver, agradeço a todos meu familiares e amigos pelo apoio e confiança.

Agradeço ao professor Daniel Lichtnow pela orientação e ensinamentos durante o desenvolvimento deste trabalho.

Agradeço também a todos os professores do curso de Sistemas para Internet por todo o conhecimento e sabedoria compartilhados durante o andamento do curso.

RESUMO

UM ESTUDO SOBRE BANCOS DE DADOS SÉRIES TEMPORAIS

AUTOR: Marcelo Costa de Lima
ORIENTADOR: Prof. Dr. Daniel Lichtnow

Com o aumento da popularidade de plataformas IoT (*Internet of Things*), houve também um aumento na utilização de sensores, consequentemente aumentando também a quantidade de dados produzidos por estes sensores, por possuir uma grande quantidade de dados gerados a cada quantia determinada de tempo (a cada segundo, por exemplo), bancos de dados relacionais talvez não sejam a melhor escolha para armazenar estes dados de série temporais, sendo necessário a recorrer a soluções NoSQL. Mesmo recorrendo a estas soluções de bancos de dados NoSQL, se viu necessário a criação de bancos de dados específicos para armazenamento destes dados de séries temporais. Neste trabalho, é apresentado um estudo sobre bancos de dados séries temporais. Explicando os conceitos básicos de séries temporais, as principais características e desafios dos bancos de dados séries temporais, e as principais aplicações e benefícios desse tipo de sistema, utilizando alguns dos bancos de dados séries temporais mais populares e utilizados na atualidade, como InfluxDB e TimeScaleDB. Por fim, é proposto algumas direções para pesquisas futuras nessa área, que ainda apresenta muitas oportunidades e desafios para o desenvolvimento de soluções inovadoras e eficientes.

Palavras-chave: NoSQL. Banco de Dados Relacional. Bancos de Dados Séries Temporais. InfluxDB. TimeScaleDB.

ABSTRACT

A STUDY ON TIME SERIES DATABASES

AUTHOR: Marcelo Costa de Lima
ADVISOR: Prof. Dr. Daniel Lichtnow

With the increasing popularity of IoT (Internet of Things) platforms, there has also been an increase in the utilization of sensors, consequently leading to a higher volume of data produced by these sensors. Due to the large amount of data generated within specific time intervals (e.g., each second), relational databases may not be the best choice for storing this time series data, necessitating the adoption of NoSQL solutions. Even with these NoSQL database solutions, it has become necessary to create specific databases for storing this time series data. This work presents a study on time series databases. It explains the basic concepts of time series, the main characteristics and challenges of time series databases, and the primary applications and benefits of this type of system. It also explores some of the most popular and widely used time series databases today, such as InfluxDB and TimeScaleDB. Finally, the work proposes directions for future research in this area, which still presents many opportunities and challenges for the development of innovative and efficient solutions.

Keywords: NoSQL. Relational Database. Time Series Databases. InfluxDB. TimeScaleDB.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

TSDB	Time-Series database
IoT	Internet of Things
NoSQL	Not Only SQL
SGDB	Sistema de Gerenciamento de Banco de Dados
SQL	Structured Query Language
ACID	Atomicidade, Consistência, Isolamento e Durabilidade
XML	Extensible Markup Language (Linguagem de Marcação Extensível)
YAML	YAML Ain't Markup Language (YAML Não é uma Linguagem de Marcação)
JSON	JavaScript Object Notation (Notação de Objetos JavaScript)
BSON	Binary JSON (JSON Binário)
CAP	Consistency, Availability, Partition Tolerance
InfluxQL	Influx Query Language
UI	User Interface

SUMÁRIO

1 INTRODUÇÃO

Historicamente, os dados de série temporal (*time-series data*) são associados principalmente a aplicações em finanças, como bancos por exemplo. Com o aumento da popularidade de plataformas de Internet das Coisas (IoT Internet of Things), a importância do uso dos dados de séries temporais também cresceu, se popularizando em diversas áreas, como mercados financeiros, previsão e monitoramento do tempo, monitoramento de sensores, e outros. (KULKARNI; BOOZ; TOTH, 2022)

Esse novo grande volume de dados temporais, principalmente em aplicações que necessitam de alto desempenho, foi crucial para o surgimento de bancos de dados de séries temporais (TSDB Time-Series Database), uma vez que este tipo de banco de dados se tornou um padrão para armazenamento de dados temporais, por ser otimizado e desenvolvido com esta finalidade. (KULKARNI; BOOZ; TOTH, 2022)

Mas isto não é uma regra, uma vez que bancos de dados relacionais podem ser usados para armazenamento de dados temporais, na realidade, existem aplicações que utilizam ambos os tipos de banco de dados em conjunto para armazenamento de informações. Alguns bancos relacionais possuem inclusive extensões para melhor tratar e utilizar dados de série temporal, como é o caso do TimeScaleDB, sendo este uma extensão do PostgreSQL para dados de séries temporais, trazendo ao banco novas capacidade de análise e armazenamento de dados de série temporais.

Cada um desses tipos de bancos de dados, relacionais e TSDB, possuem características que melhor se encaixam para tratar diferentes tipos de problemas que possam ocorrer na persistência de dados, sendo necessário um estudo prévio para ver qual melhor se encaixa para o contexto da aplicação.

Os TSDB, podem ser bem flexíveis quanto a implementação, podendo ser tanto banco de dados relacionais ou podem ser bancos de dados NoSQL (*Not Only SQL*, ou em português, "Não apenas SQL"). NoSQL são sistemas de gerenciamento de banco de dados que não utilizam o modelo relacional, permitindo maior flexibilidade e escalabilidade. Eles são úteis para armazenar grandes quantidades de dados não estruturados, como imagens e informações que não ficam associadas entre si, As principais características de bancos de dados NoSQL são: escalabilidade horizontal: onde é disponibilizado um número de máquinas para realizar o armazenamento e processamento dos dados. Esquemas flexíveis: pode ou não possuir esquemas pré-definidos sobre sua estrutura de dados. (CRUZ, 2015)

Neste trabalho, é apresentado um estudo sobre TSDBs, considerando suas definições, características e as uso. O trabalho também explora alguns questionamentos acerca do porquê utilizar um TSDB e não um banco relacional para persistência de dados de série temporal.

1.1 OBJETIVOS

1.1.1 Objetivo geral

O objetivo deste trabalho é investigar as principais motivações para o uso de *Time Series Databases* (Banco de dados de série temporal), em quais situações eles são considerados melhores e adequados para a aplicação e como eles são utilizados para aprimorar o funcionamento das aplicações e busca de dados, fazendo ainda comparações com o modelo relacional.

1.1.2 Objetivos específicos

Como objetivos específicos para o desenvolvimento deste trabalho, podem ser listados os seguintes:

- Apresentar os principais conceitos relacionados a banco de dados de série temporal;
- Expor as principais vantagens para a utilização destes bancos;
- Testar alguns dos principais bancos de dados de série temporal, comparando com bancos de dados relacionais com e sem extensões;

2 REFERENCIAL TEÓRICO

Como o estudo sobre bancos de dados representa uma área bastante abrangente, este capítulo apresenta os conceitos básicos sobre esse assunto. O presente capítulo aborda tópicos relacionados com os bancos de dados relacionais, suas características e como as novas demandas de dados impulsionaram o surgimento de tecnologias alternativas para o gerenciamento de dados: os sistemas NoSQL.

2.1 SISTEMA DE GERENCIAMENTO DE BANCO DE DADOS

Sistemas de Gerenciamento de Bancos 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, executando consultas, inserções, atualizações e exclusões de dados. Os SGBDs desempenham um papel fundamental em várias aplicações, desde sistemas de gerenciamento de estoque até redes sociais e sistemas bancários. Existem diferentes tipos de SGBDs, como SGBDs relacionais e SGBDs não relacionais. Os SGBDs relacionais, baseados no modelo relacional, são os mais tradicionais e amplamente utilizados, exemplos populares incluem o Oracle, MySQL e PostgreSQL (ELMASRI; NAVATHE, 2010). Já os SGBDs não relacionais (NoSQL) são projetados para lidar com tipos de dados não estruturados ou semiestruturados, os mais populares incluem MongoDB, Cassandra e Redis.

2.1.1 Banco de dados Relacionais

Na década de 70, foi proposta a tecnologia relacional de banco de dados, os quais se mantiveram como solução comercial para armazenamento e gerenciamento de dados convencionais, dados que possuem uma estrutura fixa, bem definida e com tipos de dados simples, como por exemplo dados gerados e manipulados por sistemas de controle de estoque e folha de pagamento. (LÓSCIO; OLIVEIRA; PONTES, 2011)

Nesse modelo, os dados são armazenados em estruturas tabulares compostas por linhas e colunas. As tabelas representam entidades e os relacionamentos entre elas são estabelecidos por meio de chaves primárias e chaves estrangeiras. Quando um banco de dados relacional é utilizado, a estrutura dos dados é definida por meio de um esquema, que especifica as tabelas, os atributos, os tipos de dados e as restrições aplicáveis. Essa definição é armazenada no catálogo do banco de dados, que contém informações sobre a estrutura do esquema. (ELMASRI; NAVATHE, 2010)

Para acessar e manipular os dados, os usuários utilizam a linguagem SQL (*Structured Query Language*). O SQL permite executar operações como inserção, atualização, exclusão e consulta de dados. As consultas SQL permitem extrair informações específicas dos dados armazenados nas tabelas do banco de dados. (SILBERSCHATZ; KORTH; SUDARSHAN, 2019)

2.1.1.1 Transações ACID

Os bancos de dados relacionais, normalmente trabalham com o tipo de transação ACID, de acordo com Ramakrishnan e Gehrke (2000) o acrônimo ACID referencia as quatro propriedades da transação, sendo elas: Atomicidade, Consistência, Isolamento e Durabilidade, conforme detalhado a seguir:

- Atomicidade: as operações de uma transação devem ser executadas corretamente, ou seja, caso houver falha em ao menos uma operação da transação, a transação inteira deve ser cancelada.
- Consistência: garante que uma transação leve o banco de dados de um estado válido para outro estado válido. Isso significa que se uma transação violar a consistência dos dados, todas as alterações realizadas por ela serão desfeitas.
- Isolamento: cada transação deve ser executada como se fosse a única transação em execução no sistema. Isso garante que a execução simultânea de várias transações não interfira no resultado final ou na consistência dos dados.
- Durabilidade: garante que, uma vez que uma transação tenha sido confirmada ou finalizada, suas alterações sejam permanentes e persistam mesmo em caso de falha do sistema. Isso é alcançado por meio da gravação das alterações em um meio de armazenamento persistente, como disco, para garantir que as alterações não sejam perdidas.

As transações ACID, em resumo, buscam preserver a integridade e consistência dos dados e suas relações em um banco de dados, atráves de um conjunto de operações internas.

2.1.2 Banco de Dados NoSQL

Com o passar do tempo, com o surgimento de dados cada vez mais complexos e que necessitam de uma maior manipulação, foram criadas as primeiras propostas de bancos de dados NoSQL (NotOnly SQL ou, em português, Não apenas SQL) para aprimorar o desempenho e a manipulação dados não estruturados ou semi-estruturados provenientes do surgimento de dados complexos (LÓSCIO; OLIVEIRA; PONTES, 2011).

Os sistemas NoSQL possibilitam a criação de estruturas de dados mais dinâmicas e flexíveis, comparado ao modelo relacional. Por este motivo eles ganharam grande espaço sendo utilizados por empresas que possuem uma quantidade maior de dados complexos, como Facebook e Google, atendo as demandas de escabilidade, tratamento de dados e disponibilidade destas empresas (LÓSCIO; OLIVEIRA; PONTES, 2011).

Os bancos de dados NoSQL geralmente oferecem uma abordagem de esquema flexível, permitindo que os dados sejam armazenados sem a necessidade de um esquema rígido predefinido. Isso proporciona maior agilidade no desenvolvimento de aplicativos, pois permite que a estrutura dos dados seja alterada de forma mais fácil e rápida (SADALAGE; FOWLER, 2012).

Os sistemas de bancos de dados NoSQL, são projetados para serem altamente disponíveis e tolerantes a falhas. Eles geralmente adotam técnicas como replicação de dados, particionamento e balanceamento de carga para garantir que os dados estejam disponíveis mesmo em caso de falhas de hardware ou rede.

Uma das principais características dos bancos de dados NoSQL é que eles não seguem o modelo relacional clássico baseado em tabelas. Em vez disso, eles adotam diversos modelos de dados, como chave-valor, documentos, colunas largas e grafos. Cada modelo tem suas próprias vantagens e é adequado para diferentes tipos de aplicações (SADALAGE; FOWLER, 2012).

Outra característica importante é a escalabilidade horizontal, que permite que o banco de dados seja distribuído em vários servidores ou nós, permitindo um processamento paralelo e um aumento da capacidade de armazenamento e desempenho. Isso é especialmente útil em ambientes com grandes volumes de dados e altas demandas de leitura e gravação, como por exemplo aplicações de redes sociais (LÓSCIO; OLIVEIRA; PONTES, 2011).

2.1.2.1 Teorema CAP

O Teorema CAP é um teorema criado pelo Dr. Eric Brewer (e por isso também chamado de Teorema de Brewer), e é empregado para descrever o comportamento de um sistema de bancos de dados distribuídos. Basicamente, respalda a tese de que não se pode obter os três requerimentos básicos simultaneamente (SADALAGE; FOWLER, 2012). Esses três requerimentos são:

- Consistência (Consistency): significa que todos os clientes veem os mesmos dados ao mesmo tempo, independentemente do nó em que se conectam. Para garantir isso, sempre que os dados são gravados em um nó, eles devem ser instantaneamente encaminhados ou replicados para todos os outros nós do sistema antes que a gravação seja considerada "bem-sucedida".

- Disponibilidade (Availability): significa que qualquer cliente que fizer uma solicitação de dados obterá uma resposta, mesmo que um ou mais nós estejam desativados ou com fa-

lhas. Em outras palavras, todos os nós em funcionamento no sistema distribuído retornam uma resposta válida para qualquer solicitação, sem exceção.

- Tolerância a particionamento (Partition Tolerance): "Partição" é uma quebra de comunicações dentro de um sistema distribuído, uma conexão perdida ou temporariamente lenta entre dois nós. "Tolerância a particionamento" significa que o *cluster* deve continuar a funcionar mesmo se ocorrer uma ou mais falhas de comunicação entre os nós no sistema.

Em outras palavras, é quase impossível para um banco de dados NoSQL ser completamente consistente, sempre disponível e tolerante a partições ao mesmo tempo. Os bancos de dados NoSQL geralmente optam por enfatizar duas das propriedades enquanto abrem mão parcialmente da terceira, dependendo das necessidades do aplicativo e do cenário de uso. Alguns sistemas NoSQL focam na disponibilidade e tolerância a partições, oferecendo eventual consistência. Outros priorizam a consistência e a tolerância a partições, aceitando uma possível indisponibilidade temporária.

2.1.3 Tipos de Bancos de Dados NoSQL

Os sistemas NoSQL podem ser classificados, de acordo com o modelo de dados, em: Chave-valor, Documentos, Grafos e Colunas. Cada um desses tipos será detalhado a seguir, juntamente com um exemplo de uma tecnologia mais relevante no mercado.

2.1.3.1 Modelo em Chave-Valor

O modelo Chave-valor é o que possui a representação mais simples dentre os sistemas de bancos de dados NoSQL. Em resumo a representação desse tipo de banco de dados é composta por um conjunto de chaves que são associadas a um único valor.(LÓSCIO; OLIVEIRA; PONTES, 2011)

Como citado, cada valor é associado a uma chave única, permitindo a recuperação eficiente dos dados por meio da chave correspondente. Esses bancos de dados são simples e eficientes em termos de desempenho, sendo adequados para casos de uso que requerem operações rápidas de leitura e gravação. O funcionamento de um banco de dados chave-valor é relativamente direto, os dados são organizados em uma estrutura de armazenamento, geralmente otimizada para acesso rápido por meio das chaves. Quando um dado é inserido no banco de dados, ele é associado a uma chave única. Posteriormente, quando ocorre uma operação de recuperação, o banco de dados localiza rapidamente o valor correspondente à chave fornecida (SADALAGE; FOWLER, 2012).

Nesse modelo, os dados são mantidos em uma única estrutura de dados, facilitando assim escalabilidade horizontal. A tabela 1 exemplifica como são organizados os dados em um sistema Chave-Valor

chave	**valor**
name	Peter
age	32

Tabela 1 – Exemplo de como são organizados dados em um sistema Chave-valor

Riak KV[1], Ehcache[2], Hazelcast[3] e Redis[4], são exemplos de sistemas NoSql que utilizam o modelo de chave-valor. Na data de hoje, de acordo com os dados do site DB-Engines[5], o Redis ocupa o primeiro lugar no *ranking* de tecnologias de chave-valor mais utilizadas no momento.

2.1.3.2 Modelo em Documentos

Um banco de dados de documentos é uma estrutura de armazenamento de dados que se assemelha ao armazenamento de chave/valor, mas em que os valores armazenados são documentos. Um documento é uma coleção de campos e valores nomeados, cada um dos quais pode ser um item escalar simples ou elementos compostos, como listas e documentos filhos. Os dados nos campos de um documento podem ser codificados de várias maneiras, incluindo XML, YAML, JSON, BSON ou até mesmo armazenados como texto simples. Nota-se que o termo "documento"neste contexto, não implica uma estrutura de texto livre, mas sim o nome dado a uma coleção de itens de dados relacionados que constituem uma entidade (SHARP et al., 2013).

Nesse modelo, os documentos contêm informações relacionadas, semelhante a um objeto em linguagens de programação ou entidade. Cada documento é armazenado em uma coleção e pode ter uma estrutura diferente dos outros documentos da mesma coleção. Isso proporciona flexibilidade na adição, remoção e modificação de campos nos documentos, sem a necessidade de alterar a estrutura da coleção inteira (SHARP et al., 2013). Com essa liberdade de criar documentos diferentes entre si, é facilitado o armazenamento de dados heterogêneos, como múltiplas variações de características em produtos de e-commerce.

Couchbase[6], Firebase Realtime Database[7] e MongoDB[8] são exemplos de sistemas NoSql

[1]https://riak.com/products/riak-kv/index.html

[2]https://www.ehcache.org/

[3]https://hazelcast.com/

[4]https://redis.io/

[5]DB-Engines é um site que coleta e apresenta informações sobre sistemas de gerenciamento de banco de dados e também monta um ranking com base nestes dados. -https://db-engines.com/en/ranking/key-value+store . Acessado em 28 de maio de 2023

[6]https://www.couchbase.com/

[7]https://firebase.google.com/docs/database?hl=pt-br

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

que utilizam o modelo de documentos. De acordo com o já citado anteriormente DB-Engines, o MongoDB ocupa o primeiro lugar atualmente no *ranking* de tecnologias que utilizam o modelo de documentos.

2.1.3.3 Modelo em Grafos

Este modelo, representa os dados em forma de grafos. Um banco de dados de modelo de grafos armazena dois tipos de informações: nós onde representam entidades, como pessoas, objetos, lugares, eventos, etc., e as arestas, que representam as conexões ou relacionamentos entre essas entidades. Tanto os nós quanto as arestas podem ter propriedades que fornecem informações sobre aquele nó ou aresta (como colunas em uma tabela). Além disso, as arestas podem ter uma direção que indica a natureza do relacionamento. O objetivo de um banco de dados de modelo de grafos é permitir que um aplicativo execute consultas de forma eficiente, percorrendo a rede de nós e arestas, e analise os relacionamentos entre as entidades (SHARP et al., 2013).

Os bancos de dados de modelo em grafo oferecem vantagens específicas em relação aos outros modelos, especialmente ao lidar com relacionamentos complexos e consultas de navegação. Esses bancos de dados são muito eficazes para consultas envolvendo descoberta de padrões e relacionamentos, como recomendações personalizadas, análise de redes sociais, rotas de navegação, detecção, trapaça, etc.

JanusGraph[9], Memgraph[10], AllegroGraph[11] e Neo4j[12] são exemplos de sistemas que utilizam o modelo em grafo. De acordo com o *ranking* do DB-Engines, o Neo4j é o sistema mais utilizado atualmente.

2.1.3.4 Modelo em Colunas

Este modelo, que também é conhecido como modelo colunar, é um modelo que é estruturado em linhas e colunas, se assemelhando ao modelo relacional. Este modelo de banco de dados usa o conceito de famílias de colunas. Em outras palavras, visa agrupar colunas que armazenam o mesmo campo de dados (SHARP et al., 2013).

Em um banco de dados de modelo de colunas, cada coluna é armazenada como uma estrutura separada em disco, contendo os valores de todas as linhas correspondentes para aquela coluna. Isso permite que o banco de dados recupere apenas as colunas necessárias para uma consulta, em vez de recuperar linhas inteiras, tornando as operações de leitura mais eficien-

[9] https://janusgraph.org/
[10] https://memgraph.com/
[11] https://allegrograph.com/
[12] https://neo4j.com/

tes. Essa estrutura é especialmente útil em situações onde há uma necessidade frequente de consultas analíticas (SILVA et al., 2021).

Como exemplos de bancos de dados desse tipo, pode-se citar o HBase, Accumulo[13] e o Cassandra[14]. O Cassandra é o lider do *ranking* dos mais utilizados no site DB-Engines.

2.1.3.5 Série Temporal

Os bancos de dados de série temporal (do inglês *Time-Series Database*) podem ser implementados usando NoSQL e sistemas de gerenciamento de banco de dados relacional, dependendo dos requisitos específicos e casos de uso. Diferente de outros modelos, este não possui uma estrutura fixa, podendo variar desde modelo colunar até o modelo de chave-valor dentro do mesmo sistema (DUNNING et al., 2014).

Muitos bancos de dados NoSQL são adequados para lidar com dados de séries temporais devido ao seu esquema flexível e capacidade de lidar com grandes volumes de gravações. Bancos de dados NoSQL, como Apache Cassandra e InfluxDB[15], são comumente usados para armazenamento de dados de séries temporais, este último liderando o *ranking*[16] de mais utilizado atualmente. Bancos de dados de séries temporais serão melhor detalhado no caítulo a seguir.

[13] https://accumulo.apache.org/
[14] cassandra.apache.org
[15] https://www.influxdata.com/
[16] https://db-engines.com/en/ranking/time+series+dbms

3 TIME SERIES DATABASE

Dados de série temporal ou em inglês *Time-series Data*, referem-se a uma coleção de observações ou medidas feitas em intervalos de tempo sequenciais. Esses dados são coletados ao longo do tempo, geralmente em intervalos regulares, e podem abranger uma ampla variedade de fenômenos, como temperatura, preço de ações, dados climáticos, vendas mensais, entre outros (MORETTIN; CASTRO, 2008).

Dados coletados como uma série temporal são mais úteis do que uma única medição quando se trata do tempo absoluto em que algo ocorreu, da ordem em que eventos específicos aconteceram ou da determinação de taxas de mudança. A base de uma série temporal é a medição repetida de parâmetros ao longo do tempo, juntamente com os momentos em que as medições foram feitas. Conjuntos de dados de séries temporais são tipicamente usados em situações em que as medições, uma vez feitas, não são revisadas ou atualizadas, mas sim, onde a quantidade de medições se acumula, com novos dados adicionados para cada parâmetro medido a cada determinado intervalo de tempo. Embora algumas abordagens para a melhor forma de armazenar, acessar e analisar esse tipo de dados sejam relativamente novas, a ideia de dados de séries temporais é na verdade bastante antiga (DUNNING et al., 2014).

Um Time Series Database (Banco de Dados de Séries Temporais) é um tipo de banco de dados projetado especificamente para armazenar, gerenciar e analisar dados organizados em sequência temporal. Ele é otimizado para lidar com grandes volumes de dados sequenciais, capturados em intervalos regulares ao longo do tempo.

Em resumo, enquanto o banco de dados relacional é melhor para manipular dados baseados em relações entre entidades, o banco de dados do tipo time-series é mais adequado para lidar com dados que mudam ao longo do tempo, por exemplo dados de sensores, logs de servidores, etc.

3.1 CARACTERÍSTICAS E USOS

De acordo com Dunning et al. (2014), algumas das características de um *Time Series Database* são:

- Armazenamento eficiente. Esses bancos de dados são projetados para armazenar e acessar eficientemente grandes volumes de dados sequenciais. Eles utilizam estruturas de índice e compactação especializada para otimizar o armazenamento e a recuperação dos dados.
- Indexação temporal. A indexação temporal é uma característica fundamental de um banco de dados de séries temporais. Ela permite que os dados sejam organizados e consultados com base em carimbos de data e hora, facilitando a análise e a recuperação de

dados específicos.

- Suporte a consultas baseadas no tempo. Esses bancos de dados oferecem recursos para consultas e análises baseadas em intervalos de tempo específicos. Isso permite a execução de consultas como "recuperar os dados dos últimos 30 dias"ou "calcular a média horária dos últimos seis meses".

- Processamento de dados em tempo real. Muitos bancos de dados de séries temporais são capazes de processar dados em tempo real, permitindo a ingestão e análise contínua de fluxos de dados em tempo real. Isso é útil em aplicações que exigem monitoramento em tempo real e tomada de decisões baseada em dados recentes.

Os bancos de dados de séries temporais têm uma ampla variedade de usos em diferentes setores, como finanças, energia, saúde, IoT (Internet das Coisas) e monitoramento de infraestrutura.

3.2 EXEMPLOS DE BANCOS DE DADOS DE SÉRIE TEMPORAIS

Existem várias opções de Bancos de Dados de Série Temporais disponíveis no. Alguns dos exemplos mais populares são:

- InfluxDB[1]: 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 fornece recursos avançados de consulta, armazenamento e agregação de dados, além de suporte a integrações com ferramentas populares de visualização e análise. Com base nos dados do DBEngines, o InfluxDB atualmente ocupa a primeira colocação no *ranking* dos mais utilizados.

- TimescaleDB[2]: O TimescaleDB é uma extensão do PostgreSQL que oferece recursos de armazenamento e consulta otimizados para dados de séries temporais. Ele combina a confiabilidade e a flexibilidade do PostgreSQL com a escalabilidade e o desempenho necessários para processar grandes volumes de dados em tempo real.

- Prometheus[3]: O Prometheus é um sistema de monitoramento e alerta de código aberto, que também funciona como um banco de dados de séries temporais. Ele é amplamente utilizado para coletar métricas de sistemas distribuídos e aplicativos em tempo real, permitindo o monitoramento contínuo e a geração de alertas com base em condições pré-definidas.

[1] https://www.influxdata.com/
[2] https://www.timescale.com/
[3] https://prometheus.io/

- OpenTSDB[4]: O OpenTSDB é um banco de dados de séries temporais distribuído, construído sobre o Hadoop[5] e o HBase[6], ambos da Apache. Ele é projetado para lidar com grandes volumes de dados de séries temporais e oferece recursos avançados de armazenamento, agregação e consulta. É frequentemente utilizado em ambientes de Big Data para análise de dados em tempo real.

Esses são apenas alguns exemplos de bancos de dados de séries temporais disponíveis. Cada um possui suas próprias características, vantagens e casos de uso específicos. A escolha do TSDB adequado dependerá das necessidades da aplicação, dos requisitos de escalabilidade, desempenho e recursos desejados. Serão melhores abordados e detalhados nas próximas subseções, os bancos InfluxDB e o TimescaleDB.

3.2.1 InfluxDB

O InfluxDB é um banco de dados de séries temporais de alto desempenho, projetado especificamente para lidar com dados de séries temporais. Ele foi criado para atender as necessidades de aplicações que lidam com grandes volumes de dados de séries temporais (INFLUXDATA, 2023).

O InfluxDB é uma opção para armazenar e consultar dados de séries temporais, pois oferece velocidade de gravação e consulta, escalabilidade horizontal, suporte a consulta em SQL, integração com diversas ferramentas e bibliotecas, além de recursos avançados de gerenciamento de dados (INFLUXDATA, 2023).

O InfluxDB possui uma arquitetura cliente-servidor e suporta diversas linguagens de programação, como Python[7], Java[8], Go[9], entre outras, por meio de bibliotecas. Além disso, é possível integrá-lo com outras ferramentas populares, como Grafana[10] e Telegraf[11], para visualização e coleta de dados (INFLUXDATA, 2023).

3.2.1.1 Organização do InfluxDB

Os dados no InfluxDB são organizados em bancos de dados e as informações são armazenadas em campos e tags. Os campos são as informações numéricas que mudam ao longo do

[4] http://opentsdb.net/
[5] https://hadoop.apache.org/
[6] https://hbase.apache.org/
[7] https://www.python.org/
[8] https://www.java.com/pt-BR/
[9] https://go.dev/
[10] https://grafana.com/
[11] https://www.influxdata.com/time-series-platform/telegraf/

tempo, como valores de sensores, enquanto os tags são informações que ajudam a identificar e categorizar os campos, como o nome do sensor, localização ou tipo de dispositivo.

O InfluxDB oferece suporte a uma linguagem de consulta chamada InfluxQL, que permite que os usuários pesquisem e agreguem dados de séries temporais de maneira eficiente e rápida. A linguagem suporta funções como GROUP BY e WHERE para filtrar e agrupar dados, e JOIN para combinar dados de múltiplas medições (semelhante aos bancos SQL tradicionais). Também possui recursos avançados de gerenciamento de dados, como políticas de retenção de dados, que permitem que os usuários definam quanto tempo os dados devem ser mantidos e como eles devem ser descartados, por exemplo, manter dados mais atuais e descartar dados antigos (INFLUXDATA, 2023).

3.2.1.2 Estrutura do InfluxDB

De acordo com a Documentação do InfluxDB, a estrutura do InfluxDB é baseada em quatro conceitos principais: measurements, tags, fields e timestamps. Vamos entender cada um deles:

- Fields, representam colunas que são nomeadas e armazenam dados importantes para a representação da série, funciona no modelo chave-valor. É composto por dois "campos", a Field Key que é uma string e representa o nome da coluna, e o Field Value que corresponde ao valor armazenado. Não se pode ter um registro sem pelo menos um Field, trata-se de uma informação básica. No *field* é possível gravar dados de todos os tipos, mas o importante é que ele seja um dado que você exibiria num gráfico. Vale ressaltar que os *fields* não são indexados, portanto a busca por um *field* no banco, implica na leitura de todos os valores, resultando em uma consulta mais lenta.

- Tags, são pares de chave-valor que são usados para indexar e filtrar os pontos de dados. Eles são armazenados como metadados e não ocupam muito espaço, por ser indexado, ele será o alvo das suas consultas no banco, em outras palavras, ele que será usado no "WHERE"na hora de realizar a sua consulta.

- Timestamps, são os valores que indicam o momento em que o ponto de dados foi gerado ou registrado. Eles são armazenados como valores numéricos em nanossegundos desde a época Unix (1 de janeiro de 1970). Por exemplo, pode-se ter um timestamp como 1639747200000000000 que corresponde a 17 de dezembro de 2021 às 12:00:00 UTC.

- Measurements, são equivalentes a tabelas em bancos de dados relacionais. Measurement é um agrupador de Tags e Fields, o nome do Measurement representa o valor dos valores que são armazenados nele, tal qual uma tabela.

Figura 1 – Exemplo da estrutura de uma Measurement.

Name: temperatura

index	time	celcius	cidade	estado	fahrenheit
1	1687637377043902976.0000000000	27.0000000000	SantaMaria	RS	80.6000000000
2	1687637436738262784.0000000000	26.0000000000	PortoAlegre	RS	78.8000000000
3	1687637473296799488.0000000000	20.0000000000	RioGrande	RS	68.0000000000

Fonte: Elaborada pelo autor.

A Figura 1 mostra a estrutura de uma *measurement* que guarda os registros de temperatura em graus *celcius* e *fahrenheits* das cidades do estado do Rio Grande do Sul. Analisando a estrutura, pode-se entender que "temperatura"se refere ao nome da *measurement*, "celcius"e "fahrenheit"são os *field keys*, já os campos "cidade"e "estado"são as *tag keys*. Os valores abaixo de *time* são os timestamps. Os valores diretamente abaixo dos *fields keys*, são os *fields values* e consequentemente os valores abaixo das *tags keys*, são as *tags values*.

3.2.2 TimeScaleDB

O TimeScaleDB é um banco de dados de código aberto projetado para tornar o SQL escalável para dados de séries temporais. Ele é construído a partir do PostgreSQL e empacotado como uma extensão do PostgreSQL, fornecendo particionamento automático no tempo e no espaço (chave de particionamento), bem como suporte completo ao SQL (TIMESCALE, 2023).

3.2.2.1 Organização do TimeScaleDB

O TimeScaleDB utiliza do conceito de hipertabelas, que são tabelas virtuais que agrupam várias tabelas individuais que contêm os dados reais. Essas tabelas individuais são chamadas de *chunks* e são criadas por particionar os dados da hipertabela em uma ou duas dimensões: por um intervalo de tempo e por uma (opcional) "chave de particionamento"como id do dispositivo, localização, id do usuário, etc (TIMESCALE, 2023).

As hipertabelas permitem que os usuários interajam com o TimeScale DB como se fosse um banco de dados PostgreSQL padrão, usando os mesmos comandos SQL para criar, alterar, inserir, selecionar e consultar dados. Além disso, o TimeScale DB oferece funções nativas de SQL chamadas hiperfunções, que facilitam e aceleram o processamento de dados de séries temporais (TIMESCALE, 2023).

O TimeScaleDB é uma solução para armazenar e analisar dados de séries temporais. Podendo escrever milhões de pontos de dados por segundo por nó, escalar horizontalmente para petabytes, comprimir os dados usando algoritmos avançados e desacoplar o armazenamento

da computação. Ele também pode ser usado na nuvem, com o Timescale Cloud, um serviço gerenciado que oferece backups automáticos e atualizações (TIMESCALE, 2023).

3.2.2.2 Estrutura do TimeScaleDB

De acordo com a Documentação do TimeScaleDB, a estrutura do TimeScaleDB é baseada em alguns componentes principais: hipertabela, particionamento, chunks, índices, funções de agregação. Onde:

- Hipertabela, como já citado anteriormente, o TimeScaleDB introduz o conceito de "hipertabela", que é uma tabela especial no PostgreSQL projetada para armazenar dados de séries temporais. Uma hipertabela é criada por meio de uma instrução SQL simples e é composta por uma tabela base e uma tabela de metadados. A tabela base contém os dados reais de séries temporais, enquanto a tabela de metadados controla o particionamento e o dimensionamento dos dados. Cada hipertabela é composta de tabelas filhas chamadas de *chunks* (traduzido literalmente por "blocos"). Cada bloco recebe um intervalo de tempo e contém apenas dados desse intervalo.

- Particionamento, é um aspecto fundamental da estrutura do TimeScaleDB. Ele divide automaticamente os dados de uma hipertabela em partições menores, com base em critérios de tempo, como intervalo fixo ou tamanho máximo.

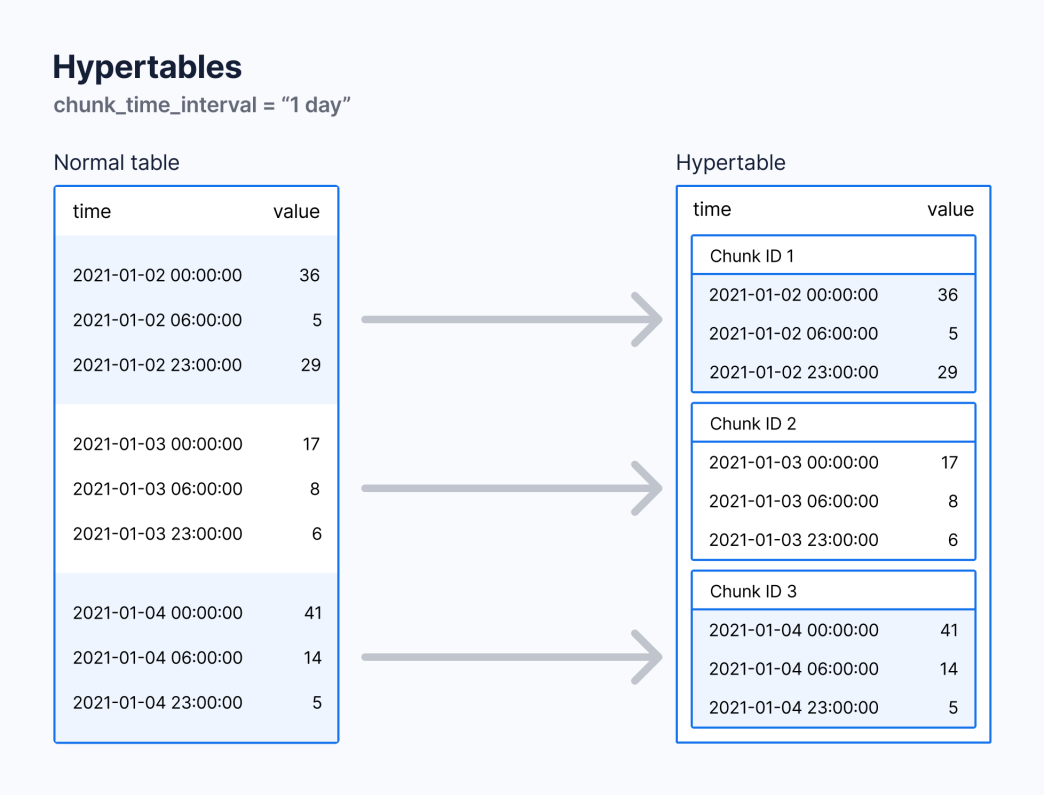
- *Chunks* (em português pedaços), a hipertabela é automaticamente dividida em *chunks*, onde cada *chunk* corresponde a um intervalo de tempo específico em uma região do espaço da partição.

- Índices, o TimeScaleDB suporta índices que melhoram significativamente o desempenho das consultas em dados de séries temporais. Esses índices são projetados para acelerar as consultas que envolvem uma combinação de filtros de tempo e outros atributos, permitindo que as consultas sejam executadas de forma eficiente mesmo em grandes volumes de dados.

- Funções de agregação, o TimeScaleDB oferece uma variedade de funções de agregação especializadas para análise de séries temporais. Isso inclui cálculos comuns, como média, soma, mínimo e máximo, bem como funções mais avançadas, como desvio padrão e regressão linear. Essas funções simplificam a análise de dados de séries temporais, permitindo que os usuários extraiam informações valiosas de forma rápida e eficiente.

A Figura 2 mostra os dados de uma tabela normal em uma hipertabela, onde foi setado o intervalo de tempo do *chunk* para um dia. Como resultado, as medições foram salvas agregadas pelo dia, ou seja, cada bloco (*chunk*) armazena dados do mesmo dia. Dados de dias diferentes são armazenados em blocos diferentes.

Figura 2 – Exemplo de hipertabela.



Fonte: https://docs.timescale.com/use-timescale/latest/hypertables/about-hypertables/

4 USANDO BANCOS DE DADOS DE SÉRIES TEMPORAIS

Neste capítulo, será explorado os recursos e as vantagens dos bancos de dados de séries temporais para armazenar e analisar dados que variam ao longo do tempo. Serão utilizados dois exemplos destes bancos: InfluxDB e TimescaleDB. Para cada um deles, será exemplificado e mostrado como criar um banco de dados, inserir dados de séries temporais e realizar consultas. Os dados que serão utilizados são referentes à temperatura e à umidade de diferentes cidades.

4.1 INFLUXDB

4.1.1 Criando base de dados

Em um modelo relacional de banco de dados, é criado uma "database"(base de dados), onde serão armazenadas as tabelas e seus dados. No InfluxDB, a nomenclatura é um pouco diferente onde a comumente chamada "database"é nomeada de "bucket"(balde). Cada *bucket* possui uma política de retenção, que em outras palavras é um mecanismo que permite controlar o tempo de vida dos dados armazenados nos buckets. Cada bucket tem uma política de retenção associada, que define quanto tempo os dados devem ser mantidos no bucket antes de serem excluídos automaticamente pelo InfluxDB. A política de retenção pode ser configurada na criação do bucket ou alterada posteriormente. A política de retenção é opcional e, se não for especificada, o bucket terá uma retenção infinita, ou seja, os dados nunca serão apagados (INFLUXDATA, 2023).

Figura 3 – Exemplo de código de como criar um bucket.

```
PS C:\Users\Marcelo> influx bucket create --name exemplos-tcc
ID                  Name            Retention       Shard group duration    Organization ID         Schema Type
ad4cb089deb8e097    exemplos-tcc    infinite        168h0m0s                6360801c2758d79b        implicit
PS C:\Users\Marcelo>
```

Fonte: Elaborada pelo autor

A Figura 3 mostra um exemplo de código onde é criado um bucket chamado "exemplos-tcc"através da *tag* --name". Nota-se que no retorno do comando, a retenção está como "infinita", pois no comando de criação não foi passado uma política de retenção.

4.1.2 Inserção de dados

A inserção de dados no InfluxDB possui uma sintaxe semelhante a dos bancos relacionais tradicionais. Como o InfluxDB utiliza *fields e tags*, eles são passados no momento da

inserção de dados. Os fields são os dados que serão armazenados e podem ser de diferentes tipos, como inteiros, floats, strings, booleanos, entre outros. Já as tags são pares chave-valor que são usados para identificar e agrupar pontos relacionados. A sintaxe para inserir um ponto no InfluxDB é a seguinte:

INSERT <measurement-name>,<tag-key>=<tag-value> <field-key>=<field-value>, <timestamp>

Figura 4 – Exemplo de código de como inserir registros.

```
> INSERT temperatura,cidade=SantaMaria,estado=RS celcius=27,fahrenheit=80.6
> INSERT temperatura,cidade=PortoAlegre,estado=RS celcius=26,fahrenheit=78.8
> INSERT temperatura,cidade=RioGrande,estado=RS celcius=20,fahrenheit=68
>
```

Fonte: Elaborada pelo autor

A Figura 4 mostra a inserção de três registros no bucket "exemplos-tcc", onde o nome da *measurement* é "temperatura". Nota-se que podem ser inseridos múltiplos *fields keys e fields values* assim como múltiplas *tags keys e tags values*, bastando apenas separá-los por uma ","(vírgula). Vale salientar que como não foi passado o valor do timestamp, ele é automaticamente inserido com o timestamp da hora da inserção.

4.1.3 Consulta de dados

Para consultar os dados no InfluxDB, é possível usar duas linguagens: Flux e FluxQL. Flux é uma linguagem desenvolvida nativa do InfluxDB que oferece mais flexibilidade e funcionalidades do que FluxQL, que é uma linguagem baseada em SQL. Com Flux, é possível realizar operações como filtrar, agrupar, transformar e agregar os dados de forma mais eficiente e expressiva. Com FluxQL, é possível realizar consultas mais simples e familiares para quem já conhece SQL, mas com algumas limitações e diferenças de sintaxe.

Figura 5 – Exemplo de uma consulta em Flux.

```
1  from(bucket: "exemplos-tcc")
2    |> range(start: -6h, stop:now())
3    |> filter(fn: (r) =>
4      r._measurement == "temperatura" and
5      r._field == "celcius"
6    )
7
```

Fonte: Elaborada pelo autor

A Figura 5 mostra um exemplo de consulta em Flux, que retorna as medições das temperatura em celcius em um intervalo de tempo das últimas 6 horas até o momento atual, juntamente com a cidade e estado e o timestamp (retorno da consulta observado na Figura 6).

O código da Figura 5 pode-se ser explicado da seguinte maneira:

Figura 6 – Retorno da consulta.

table _result	_measurement group string	_field group string	_value no group double	_start group dateTime:RFC3339	_stop group dateTime:RFC3339	_time no group dateTime:RFC3339	cidade group string	estado group string
0	temperatura	celcius	26	2023-06-24T21:12:21.135Z	2023-06-25T03:12:21.135Z	2023-06-25T00:30:03.593Z	PortoAlegre	RS
1	temperatura	celcius	20	2023-06-24T21:12:21.135Z	2023-06-25T03:12:21.135Z	2023-06-25T00:30:34.645Z	RioGrande	RS
2	temperatura	celcius	27	2023-06-24T21:12:21.135Z	2023-06-25T03:12:21.135Z	2023-06-25T00:26:28.016Z	SantaMaria	RS

Fonte: Elaborada pelo autor

- A linha inicial indica a fonte de dados na qual a consulta será realizada. Neste caso, o banco de dados é chamado de "exemplos-tcc".
- A linha 2 especifica o intervalo de tempo para os dados a serem recuperados. Ela usa as funções "range"e "now"para definir o início e o fim do intervalo. Neste caso, o intervalo é de 6 horas atrás até o momento atual.
- A linha 3 aplica um filtro aos dados recuperados. Ela usa a função "filter"para especificar uma condição que os registros devem atender. Neste caso, a condição é que o campo "_measurement"deve ser igual a "temperatura"e o campo "_field"deve ser igual a "celcius". Isso significa que apenas os registros que representam medições de temperatura em graus Celsius serão recuperados.

Figura 7 – Exemplo da consulta em FluxQL.

```
> use "exemplos-tcc"
> SELECT celcius,cidade,estado,time FROM "temperatura" WHERE time>now() -6h AND time<now()
>
```

Fonte: Elaborada pelo autor

Figura 8 – Retorno da consulta em FluxQL.

Name: temperatura

index	time	celcius	cidade	estado
1	1687652788016201728.0000000000	27.0000000000	SantaMaria	RS
2	1687653003593847552.0000000000	26.0000000000	PortoAlegre	RS
3	1687653034645543424.0000000000	20.0000000000	RioGrande	RS

5 Columns, 3 Rows, Page 1/1
Table 1/1, Statement 1/1

Fonte: Elaborada pelo autor

A Figura 7 mostra a mesma consulta da Figura 5 reescrita no formato FluxQL, formato este semelhante ao SQL. A nível de comparação, a Figura 8 mostra o retorno obtido pelo FluxQL.

Na Figura 9 é feito uma consulta utilizando um *field key* como condição no "WHERE", embora não seja o recomendado, pois ao usar uma cláusula WHERE no InfluxDB, geralmente

Figura 9 – Exemplo de consulta utilizando Where.

```
select "celcius" from temperatura where celcius > 15
```

Fonte: Elaborada pelo autor

Figura 10 – Retorno da consulta da Figura 9.

index	time	celcius
1	1687652788016201728.0000000000	27.0000000000
2	1687653003593847552.0000000000	26.0000000000
3	1687653034645543424.0000000000	20.0000000000

3 Columns, 3 Rows, Page 1/1
Table 1/1, Statement 1/1

Fonte: Elaborada pelo autor

é especificado condições com base em "tag keys"para filtrar pontos de dados com base em metadados. Os "field keys"não são comumente utilizados na cláusula WHERE, pois eles não são projetados para esse propósito. Ao contrário dos "field keys", as "tag keys"não são indexadas e compactadas como os "field keys". Isso significa que o uso de "tag keys"em consultas é mais eficiente em termos de desempenho do que o uso de "field keys"quando se trata de filtrar e agrupar dados. Sendo recomendado utilizar "field keys"em outras cláusulas, como SELECT e GROUP BY, para realizar operações e agregações nos valores dos campos. A Figura 10 mostra o retorno da consulta realizada na Figura 9, como pode ser notado, a consulta é feita normalmente utilizando o field key como parâmetro.

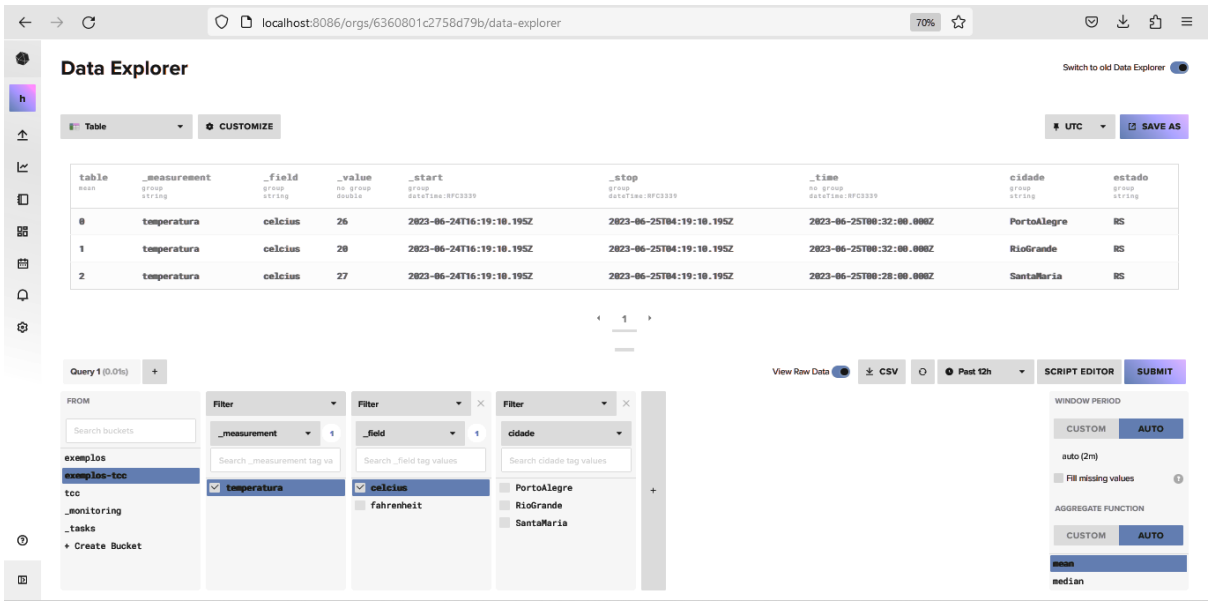
4.1.4 Gerando gráficos com InfluxDB

O InfluxDB possui também uma UI (*User Interface*), que pode ser utilizada desde para a criação dos *buckets* até realizar consultas. Dentre as mais variadas funcionalidades, pode-se destacar a geração de gráficos das consultas realizadas, vale ressaltar que ao utilizar a UI para fazer as consultas, estas podem ser feitas sem digitar qualquer linha de código, apenas selecionando quais campos é desejado buscar.

A Figura 11 mostra a tela da interface visual onde podem ser feitas consultas, nota-se que não é necessário inserir código algum, seja ele Flux ou FluxQL, basta apenas selecionar os campos, e é obtido o resultado da consulta.

Uma das formas de visualizar esses dados é através de gráficos, onde podem mostrar diferentes tipos de informações, como tendências, padrões, anomalias, correlações e distribui-

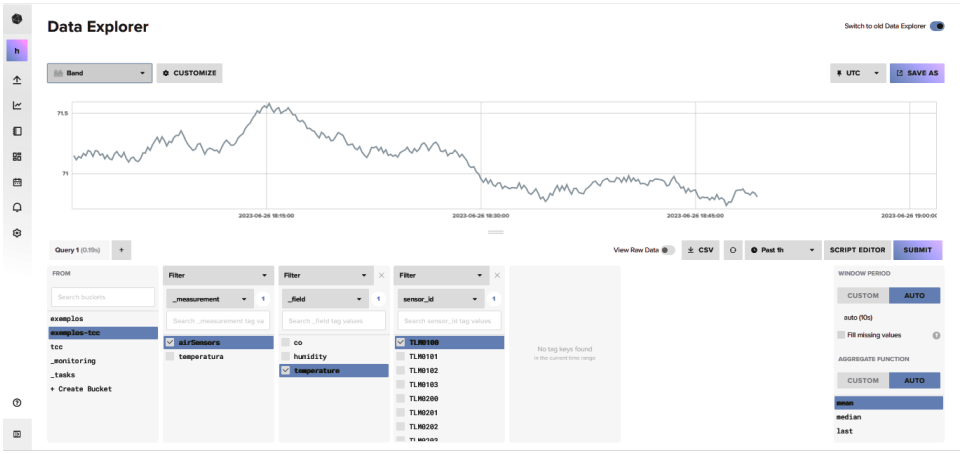
Figura 11 – Tela onde é pode-se realizar consultas na interface visual do InfluxDB.



Fonte: Elaborada pelo autor

ções. Para os seguintes exemplos de gráficos, foi utilizado uma base dados exemplo do GitHub[1] do InfluxDB, por conter uma grande quantidade de dados já inseridos, bastando apenas fazer o *download* do arquivo CSV e importar ele no InfluxDB.

Figura 12 – Exemplo de gráfico gerado.



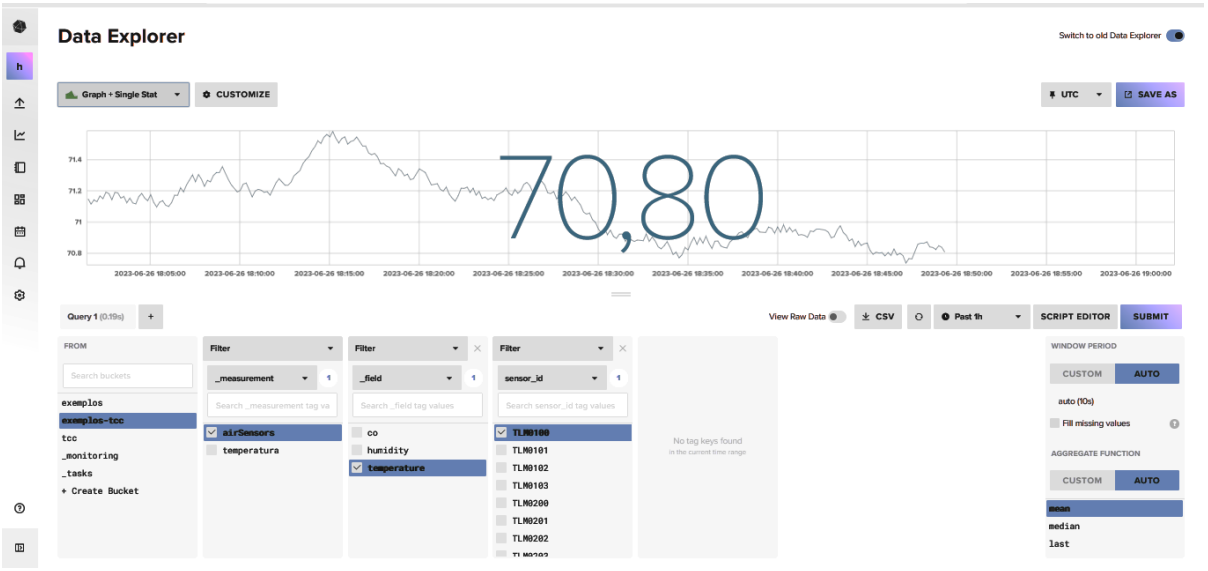
Fonte: Elaborada pelo autor

Para contextualizar, a base de dados possui dados de sensores que mensuram tanto a temperatura (em graus Fahrenheit) quanto a umidade e o nível de CO, possuindo dados de oito sensores diferentes, cada um identificado por um identificador (algo como o modelo de sensor usado).

A Figura 12 mostra um exemplo de gráfico gerado com leituras de temperatura de um sensor com identificação igual a "TLM0100", na última 1 hora.

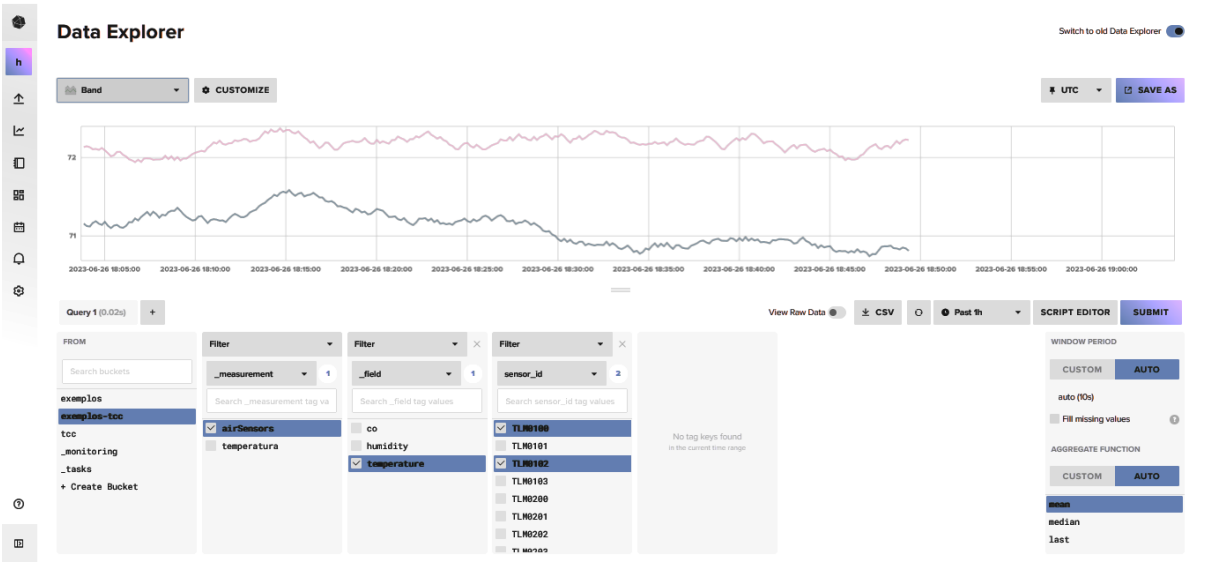
[1] https://github.com/influxdata/influxdb2-sample-data/tree/master/air-sensor-data

Figura 13 – Exemplo de gráfico com média gerado.



Fonte: Elaborada pelo autor

Figura 14 – Exemplo de gráfico com dados de dois sensores.



Fonte: Elaborada pelo autor

Já a Figura 13, mostra o mesmo gráfico da figura 12, mas agora com a média de entre todas as temperaturas lidas.

Na Figura 14 é mostrado os gráficos de dois sensores diferentes, com identificação iguais a "TLM0100"e "TLM0102", representados pelas azul e rosa, respectivamente.

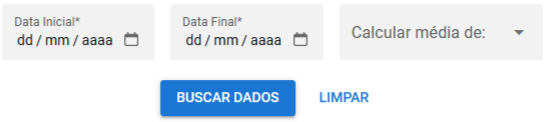
A utilização desta funcionalidade de geração de gráficos, permite que seja feita uma análise visual rapidamente e facilmente dos dados, como por exemplo, na Figura 14 é possível notar que o sensor "TLM0102"(na cor rosa) mensurou temperaturas mais elevadas em relação ao sensor "TLM0100"(na cor azul). Esta rápida análise pôde ser feita apenas olhando o gráfico gerado, não sendo necessário uma análise deveras maçante dos valores das tabelas, conside-

rando que seja utilizado a visualização padrão dos dados, em formato de tabela.

4.1.5 Aplicação em um projeto Web

Este capítulo visa proporcionar uma análise sobre a concepção, arquitetura e implementação de um projeto que utiliza a API do InfluxDB. O ambiente de desenvolvimento escolhido para a construção deste projeto é Vue.js [2] (Vue 3), um framework JavaScript progressivo, e aprimorado com o também framework Quasar [3], proporcionando uma abordagem eficiente e moderna para o desenvolvimento de aplicações Web. A seguir, serão explorados os aspectos chave do processo de desenvolvimento, destacando as principais características e funcionalidades incorporadas, com o intuito de oferecer uma compreensão abrangente do sistema desenvolvido.

Figura 15 – Tela da aplicação Web desenvolvida



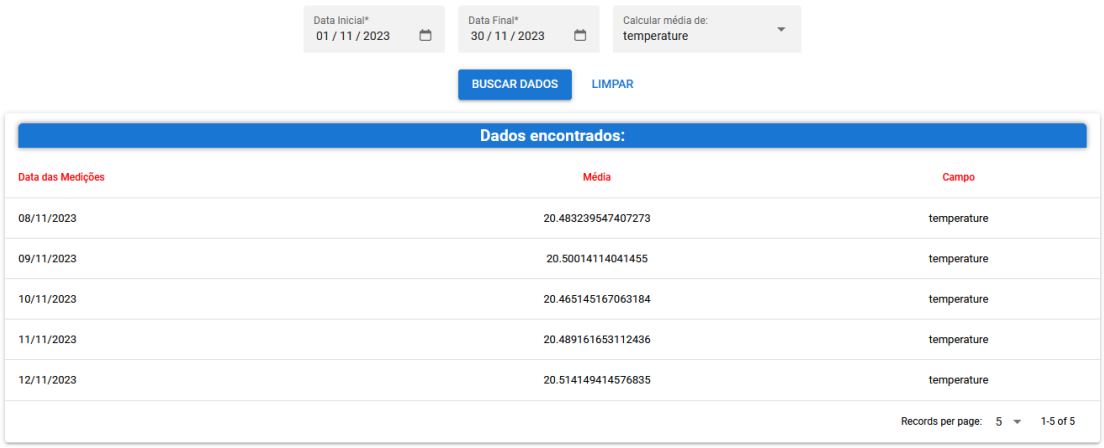
Fonte: Elaborada pelo autor

A aplicação conta com uma única tela que permite aos usuários selecionar duas datas significativas - uma data inicial e uma final. Além disso, é possível escolher entre medições de temperatura ou umidade. Ao clicar no botão correspondente, o sistema realiza uma requisição à API do InfluxDB, utilizando a linguagem de consulta Flux. A API responde à solicitação com os dados relevantes, e o sistema, por sua vez, apresenta uma tabela exibindo a média da temperatura ou umidade entre as datas especificadas.

Esta abordagem de realizar uma chamada direta a API do InfluxDB, não apenas simplifica o acesso às informações desejadas, mas também simplifica o desenvolvimento da aplicação.

[2] https://vuejs.org/
[3] https://quasar.dev/

Figura 16 – Tela da aplicação Web com o resultado da média de temperaturas



Fonte: Elaborada pelo autor

Ao integrar diretamente a API do InfluxDB e utilizar a linguagem Flux o sistema assegura a precisão e a eficiência na obtenção das médias dos dados.

Observando a Figura 17 é possível notar que foi feita uma função para realizar a chamada direta a API do InfluxDB, não sendo necessário nenhum tipo de conexão utilizando usuário e senha. Para isto ser possível, o InfluxDB disponibiliza a geração de um *token* de acesso, minimizando os riscos de se realizar requisições com utilizando senhas como parâmetros.

Um outro ponto a ser observado, é que a *query* de consulta possui sintaxe igual a utilizada diretamente no InfluxDB, não sendo necessário qualquer tipo de modificação na sintaxe para funcionar. No contexto do sistema desenvolvido, os únicos parâmetros da *query* que são dinâmicos, são a data inicial e final, e também o tipo de medição (temperatura ou umidade).

Após realizar a chamada, é necessário uma pequena organização dos dados oriundos do InfluxDB, isto é necessário para melhorar a representação na tabela. O formato dos dados retornados pelo InfluxDB não possui uma formatação adequada, é como se a API do InfluxDB retornasse um arquivo de texto com todos os dados, como mostra a Figura 18.

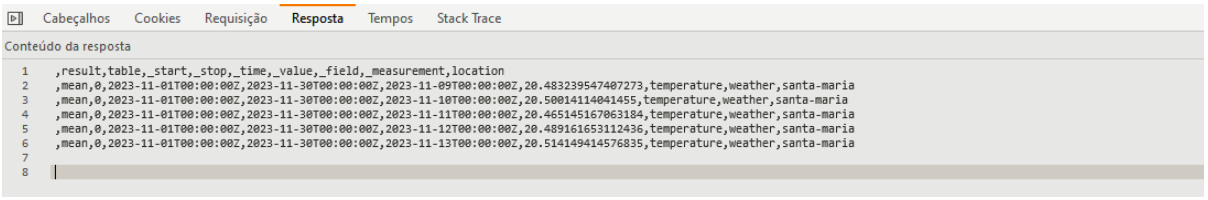
Para organizar estes dados, foi necessário o uso de uma expressão regular, também conhecida como ”*regex*” ou ”*regexp*”, é uma sequência de caracteres que define um padrão de busca. Esses padrões são usados para realizar operações de busca, correspondência e manipulação de strings em textos. As expressões regulares são amplamente utilizadas em programação, edição de texto e processamento de dados, permitindo realizar tarefas como validação de formatos, extração de informações específicas e substituição de padrões em strings (ROBBINS, 2001). A Figura 19 mostra a expressão regular utilizada para extrair a data e o valor da média da medição selecionada.

Figura 17 – Trecho do código da aplicação onde realiza a chamada a API do InfluxDB

```
const dados = ref<DadosMedicao[]>([])

async function onSubmit () {
  try {
    const url = 'http://localhost:8086'
    const query = `from(bucket: "temperatura")
      |> range(start: ${dataInicial.value}, stop: ${dataFinal.value})
      |> filter(fn: (r) => r["_measurement"] == "weather")
      |> filter(fn: (r) => r["_field"] == "${parametro.value}")
      |> filter(fn: (r) => r["location"] == "santa-maria")
      |> aggregateWindow(every: 24h, fn: mean, createEmpty: false)
      |> yield(name: "mean")`

    await axios.post(`${url}/api/v2/query?orgID=6360801c2758d79b`, query, {
      headers: {
        Authorization: 'Token Zbs-O-VxxoIVS9yHBR70eWSgJPgTnALzsteZBzF2GcF6cuqkLLbTs7Cl5Psej_j-
U2PM9scsUOFHfNvjwkyYZA==',
        'Content-Type': 'application/vnd.flux'
      }
    }).then(resp => {
      if (resp.status === 200) {
        achouDados.value = true
        let matches
        while ((matches = regexData.exec(resp.data)) !== null) {
          const data = matches[1].split('T')[0]
          const valor = matches[2]

          const medicao: DadosMedicao = {
            date: data,
            value: valor,
            type: parametro.value
          }

          dados.value.push(medicao)

          console.log(dados.value)
        }
      } else {
        achouDados.value = false
      }
    })
  } catch (error) {
    console.error(error)
  }
}
```

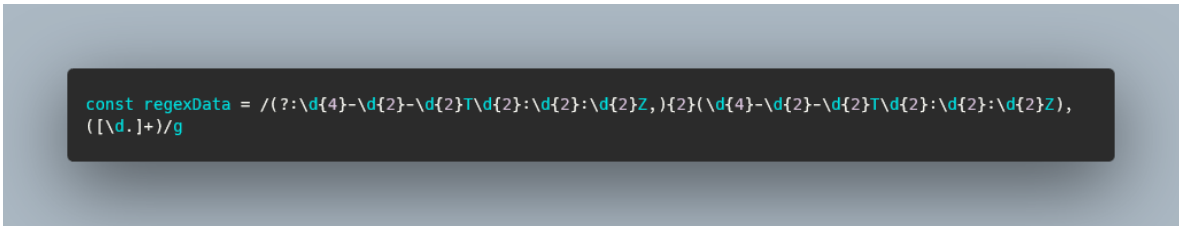
Fonte: Elaborada pelo autor

Figura 18 – Retorno da consulta realizada a API do InfluxDB



Fonte: Elaborada pelo autor

Figura 19 – Expressão regular utilizada para extrair data e a média da medição



Fonte: Elaborada pelo autor

4.2 TIMESCALEDB

4.2.1 Criando base de Dados

Para criar uma base de dados no TimeScaleDB, o processo difere um pouco em relação ao InfluxDB. Enquanto no InfluxDB chamamos a base de dados de "bucket", no TimeScaleDB a organização dos dados é feita através de "hipertabelas".

Para criar uma hipertabela no TimeScaleDB, é necessário definir os atributos específicos de séries temporais, como o nome da tabela, a coluna de tempo e as colunas adicionais que armazenam os dados. Além disso, é possível configurar a política de retenção durante a criação da hipertabela, definindo a duração em unidades de tempo para a retenção dos dados (TIMESCALE, 2023). Essa abordagem do TimeScaleDB oferece maior flexibilidade e recursos avançados para trabalhar com dados de séries temporais, permitindo um gerenciamento eficiente e otimizado em comparação ao InfluxDB.

Figura 20 – Código de como criar uma hipertabela no TimeScale.

```
CREATE TABLE temperatures (
  time        TIMESTAMPTZ       NOT NULL,
  cidade      TEXT              NOT NULL,
  estado      TEXT              NOT NULL,
  temperature DOUBLE PRECISION  NOT NULL
);

SELECT create_hypertable('temperatures', 'time', migrate_data => true);
go

SELECT add_retention_policy('temperatures', interval '30 days');
go
```

Fonte: Elaborada pelo autor

A Figura 20 mostra um código exemplo de como criar uma tabela e a converter para hipertabela no TimeScale, o código pode ser entendido a seguir:

- Linha 1 até 6, cria a tabela chamada "temperatures"com quatro colunas: "time"do tipo TIMESTAMPTZ (timestamp com informações de fuso horário), "cidade"do tipo TEXT (para armazenar o nome da cidade), "estado"do tipo TEXT (para armazenar o nome do

estado) e "temperature"do tipo DOUBLE PRECISION (para armazenar os valores de temperatura). O NOT NULL indica que nenhum dos campos pode ser nulo.

- Linha 8, essa linha cria uma hipertabela no TimeScaleDB. A função "create_hypertable"é usada para converter a tabela "temperatures"em uma hipertabela, permitindo que o TimeScaleDB otimize o armazenamento e a consulta dos dados de séries temporais. O primeiro argumento é o nome da tabela a ser convertida ("temperatures"), o segundo argumento é o nome da coluna de tempo ("time") que será usado para particionar e organizar os dados temporalmente. O "migrate_data => true"indica que os dados existentes na tabela devem ser migrados para a hipertabela.
- Linha 11, adiciona uma política de retenção à hipertabela. A função "add retention policy"é usada para definir por quanto tempo os dados devem ser mantidos na hipertabela antes de serem automaticamente excluídos. O primeiro argumento é o nome da hipertabela a ser configurada ("temperatures"), e o segundo argumento é o intervalo de tempo para a retenção dos dados (neste caso, 30 dias).

4.2.2 Inserção de dados

No TimeScaleDB, a inserção de dados segue uma abordagem semelhante em termos de sintaxe, mas com algumas diferenças em relação ao InfluxDB. No TimeScaleDB, os dados são inseridos usando a linguagem SQL padrão.

Figura 21 – Código exemplo de inserção de dados.

```
INSERT INTO temperatures (time, cidade, estado, temperature) VALUES
    ('2023-07-01 12:00:00', 'santaMaria', 'RS', 23.5),
    ('2023-07-01 12:15:00', 'portoAlegre', 'RS', 24.8),
    ('2023-07-01 12:30:00', 'saoPaulo', 'SP', 17.0),
    ('2023-07-01 12:45:00', 'guarulhos', 'SP', 18.4);
go
```

Fonte: Elaborada pelo autor

A Figura 21 mostra um exemplo onde são inseridos quatro valores na hipertabela, onde "2023-07-01 12:00:00"é o valor para a coluna "time", "santaMaria"é o valor para a coluna "cidade", "RS"é o valor para a coluna "estado"e "23.5"é o valor para a coluna "temperature", e o mesmo vale para as linhas seguintes no código.

É importante destacar que, no TimeScaleDB, os campos não são explicitamente divididos em "fields"e "tags"como no InfluxDB. No TimeScaleDB, todos os campos são tratados como colunas regulares da tabela, e você pode definir seus tipos de dados conforme necessário.

4.2.3 Consulta de dados

No TimeScaleDB, as consultas aos dados são realizadas utilizando a linguagem SQL padrão. Ao contrário do InfluxDB, não há uma linguagem específica como o Flux ou FluxQL no TimeScaleDB. No TimeScaleDB, pode-se aproveitar toda as funcionalidades da linguagem SQL para consultar e manipular os dados de séries temporais.

Com a linguagem SQL, é possível realizar operações como filtrar dados usando cláusulas WHERE, agrupar dados usando cláusulas GROUP BY, transformar dados usando cláusulas JOIN e realizar agregações usando funções como COUNT, SUM, AVG, entre outras.

Figura 22 – Código exemplo de consulta no TimeScaleDB.

```
SELECT time, cidade, estado, temperature
FROM temperatures
WHERE estado = 'RS'
ORDER BY time DESC
```

Fonte: Elaborada pelo autor

Figura 23 – Retorno da consulta da Figura 22.

	time	cidade	estado	temperature
1	2023-07-01 12:15:00.000 -0300	portoAlegre	RS	24,8
2	2023-07-01 12:00:00.000 -0300	santaMaria	RS	23,5

Fonte: Elaborada pelo autor

A Figura 22 mostra um código de exemplo para uma consulta, onde é selecionado as colunas "time", "cidade", "estado"e "temperature"da tabela "temperatures". Após é filtrado os resultados para que apenas os registros onde o estado é 'RS' sejam retornados. Em seguida, é ordenado os resultados pelo campo "time"em ordem decrescente. Já a Figura 23 mostra o que é retornado com este código, com base nos dados inseridos anteriormente.

4.2.4 Gerando gráficos no TimeScaleDB

O TimeScaleDB não possui uma User Interface (UI) nativa como o InfluxDB. No entanto, existem várias ferramentas de terceiros que podem ser utilizadas para interagir com o TimeScaleDB, oferecendo funcionalidades semelhantes à UI mencionada. Uma dessas ferramentas populares é o Grafana, uma ferramenta popular para visualização de dados e criação de dashboards. O Grafana oferece uma interface gráfica intuitiva que permite criar painéis interativos e personalizados para visualizar dados do TimeScaleDB.

Com o Grafana, é possível criar consultas aos dados do TimeScaleDB sem precisar digitar código SQL manualmente. Através da interface gráfica do Grafana, pode-se selecionar os

campos desejados, aplicar filtros, definir intervalos de tempo e configurar opções de visualização. Além disso, o Grafana permite a geração de gráficos e a criação de painéis com widgets, como gráficos de linha, gráficos de barras, tabelas e medidores, para representar os dados do TimeScaleDB de forma clara e informativa. Portanto, ao utilizar o Grafana com o TimeScaleDB, é possível aproveitar uma interface gráfica amigável para criar consultas, visualizar dados e gerar gráficos sem a necessidade de escrever código manualmente, assim como no InfluxDB. Essa combinação oferece uma experiência visual e interativa para explorar e apresentar dados de séries temporais armazenados no TimeScaleDB.

Para os exemplos a seguir, será criado uma nova tabela de dados, com novas medições.

Figura 24 – Criando novas tabelas e convertendo uma para hiper tabela.

```
CREATE TABLE sensors(
  id SERIAL PRIMARY KEY,
  type VARCHAR(50)
);
go

INSERT INTO sensors (type) VALUES
('Sensor a'),
('Sensor b'),
('Sensor c'),
('Sensor d');
go

CREATE TABLE sensor_data (
  time TIMESTAMPTZ NOT NULL,
  sensor_id INTEGER,
  temperature DOUBLE PRECISION,
  FOREIGN KEY (sensor_id) REFERENCES sensors (id)
);
go

SELECT create_hypertable('sensor_data', 'time');
go
```

Fonte: Elaborada pelo autor

A Figura 24 mostra o código utilizado para criar duas novas tabelas, uma tabela chamada "sensors"e outra nomeada "sensor_data". Na tabela "sensors"foram inseridos dados de quatro sensores: Sensores A,B,C e D. Após foi criada a tabela "sensor_data", que será responsável por armazenar os dados de temperatura mensurados, ficticiamente, pelos sensores. Após a criação dessa tabela, assim como já feito na primeira subseção "Criando base de Dados", a tabela é transformada em uma hiper tabela.

A Figura 25 mostra um código que insere dados fictícios na tabela "sensor_data". As seis primeiras linhas de código, cria uma tabela temporária chamada "simulated_data"usando a cláusula WITH. Nessa tabela temporária, são gerados dados simulados para o sensor. É gerada uma série de valores de tempo, começando a partir de 24 horas atrás até o tempo atual, com um intervalo de 5 minutos. O "sensor_id"é definido como 1 em todos os registros (neste caso o

Figura 25 – Inserindo dados na nova tabela.

```
WITH simulated_data
AS
(SELECT
  1 as sensor_id, generate_series(now() - interval '24 hour', now(), interval '5 minute') AS time,
  random()*(40-10)+10 AS temperature
)
INSERT INTO sensor_data (time, sensor_id, temperature)
  SELECT time, sensor_id, temperature FROM simulated_data;
go
```

Fonte: Elaborada pelo autor

id 1 é do Sensor A). Os valores para "temperature"são gerados aleatoriamente usando a função "random", com valores entre 10 e 40. Nas linhas restantes do código, os dados simulados da tabela temporária "simulated_data"são inseridos na tabela permanente chamada "sensor_data". A inserção é feita selecionando os valores de "time", "sensor_id"e "temperature"da tabela temporária e inserindo-os nas colunas correspondentes da tabela "sensor_data". O mesmo código será executado para os outros sensores, trocando apenas o "sensor_id", colocando o dos sensores B, C e D.

Figura 26 – Consulta dos dados no Grafana.

```
1  SELECT
2    time_bucket('5 minutes', time) AS period,
3    AVG(temperature) AS avg_temp
4  FROM sensor_data
5  GROUP BY period
6  ORDER BY period;
7
```

Fonte: Elaborada pelo autor

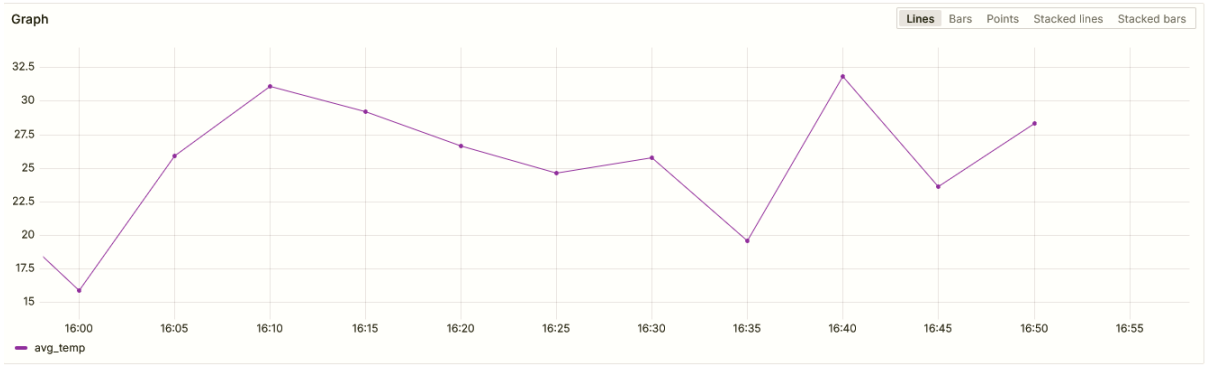
A Figura 26 mostra um código para uma consulta no Grafana, que pode ser entendida assim:

- Das linhas 1 até a 4, esta parte da consulta seleciona duas colunas: "period"e "avg_temp". A função "time_bucket"é usada para agrupar os registros por intervalos de tempo específicos. No caso, o intervalo de tempo é de 5 minutos. A coluna "period"será preenchida com os intervalos de tempo resultantes, enquanto a coluna "avg_temp"conterá a média das temperaturas dentro de cada intervalo. A tabela "sensor_data"é onde serão obtidos os dados.

- Linhas 5 e 6, agrupa os registros com base na coluna "period". Isso significa que os registros serão agrupados de acordo com os intervalos de tempo definidos pela função "time_bucket", e também ordena os resultados em ordem crescente com base na coluna "period", ou seja, os registros serão ordenados de acordo com os intervalos de tempo.

A Figura 27 mostra o gráfico de linha gerado no Grafana pela consulta da Figura 26. Os dados em forma de tabela podem ser visualizados na Figura 28.

Figura 27 – Gráfico gerado pela consulta no Grafana.



Fonte: Elaborada pelo autor

Figura 28 – Retorno no modo tabela gerado pela consulta no Grafana.

period	avg_temp
2023-07-02 16:40:00	13.4
2023-07-02 16:45:00	21.1
2023-07-02 16:50:00	27.0
2023-07-02 16:55:00	31.6
2023-07-02 17:00:00	30.9
2023-07-02 17:05:00	18.9
2023-07-02 17:10:00	27.7
2023-07-02 17:15:00	26.4
2023-07-02 17:20:00	29.0
2023-07-02 17:25:00	23.9
2023-07-02 17:30:00	26.9
2023-07-02 17:35:00	25.6
2023-07-02 17:40:00	17.6
2023-07-02 17:45:00	26.3

Fonte: Elaborada pelo autor

5 TESTE COMPARATIVO COM OS BANCOS

Este capítulo descreve como foram realizados os testes em dois dos sistemas de gerenciamento de bancos de dados apresentados, InfluxDB e PostgreSQL, utilizando a ferramenta de teste de desempenho Apache JMeter[1]. O objetivo principal dos testes foi avaliar o desempenho de inserção e leitura de dados em ambos os bancos de dados, com ênfase no tempo de resposta, através de diferentes volumes de dados.

Quanto a metodologia de teste:

- Para os testes de inserção de dados, foram utilizados cenários simulando diferentes cargas de trabalho, variando o número de requisições por segundo e o tamanho dos conjuntos de dados. Para todos os dados foram utilizados dados fictícios, através do JMeter;
- Os testes de leitura de dados foram realizados com consultas que faziam o acesso a dados históricos e agregados.

De acordo com Apache (2023), o Apache JMeter é uma ferramenta de código aberto que realiza testes de carga e de estresse em recursos estáticos ou dinâmicos oferecidos por sistemas computacionais. Ele foi originalmente projetado para testar aplicações web, mas seu uso se expandiu para outras funções de teste, como SOAP, JDBC, LDAP, JMS, Mail e comandos nativos ou scripts.

O JMeter permite simular cenários de testes mais reais, injetando atrasos e ”paradas” variáveis nos testes para imitar o comportamento dos usuários reais, por exemplo, com os atrasos e paradas, simula um usuário fazendo requisições com um intervalo entre elas, dando um maior realismo ao teste. Ele também suporta o cache de navegador oferecendo um servidor de proxy que intercepta as requisições feitas por um navegador web e criando automaticamente requisições HTTP, no entanto, se for necessário limpar o cache após cada requisição, é possível utilizar o HTTP Cache Manager do JMeter.

O JMeter oferece vários recursos para auxiliar o projeto de teste, como: requisições (elementos visualmente de fácil configuração, podendo escolher entre diferentes tipos de requisições pré definidas como requisições HTTP, LDAP e FTP), temporizador (que são os elementos responsáveis por ”atrasar” ou até mesmo parar as requisições por um determinado tempo setado pelo usuário), extrator de expressão regular (utilizado em testes onde existe a necessidade de extrair um texto ou até mesmo um arquivo de uma página Web), asserções (ou *assertions*, em inglês) são usadas para verificar se as respostas de um servidor atendem a determinados critérios. Elas são usadas para validar se o servidor está respondendo corretamente às solicitações feitas pelo JMeter. As asserções podem ser adicionadas a um teste para verificar se a resposta do servidor contém determinados valores, se a resposta está dentro de um determinado intervalo de tempo, se a resposta contém um determinado padrão, entre outras verificações. O

[1] https://jmeter.apache.org/

JMeter oferece também o recurso de variáveis, podendo ser cadastradas diversas variáveis com valores distintos, neste trabalho por exemplo, as variáveis foram utilizadas para manipular as datas de inserções dos registros, não sendo necessário esperar 24 horas para inserir dados com o *timestamp* de um dia diferente (APACHE, 2023).

Algumas funcionalidades do Apache JMeter de acordo com Halili (2008):

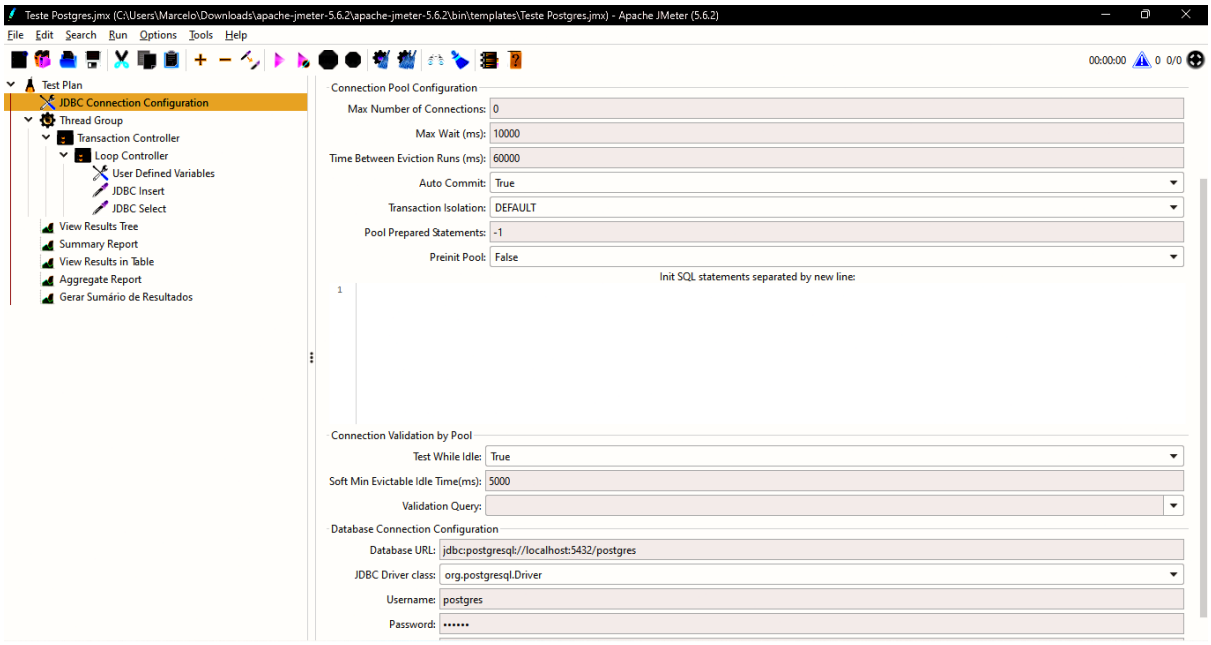
- Teste de carga. Permite simular cargas de tráfego em aplicativos web para avaliar seu desempenho sob diferentes condições de uso;
- Teste de estresse. Permite submeter aplicativos a cargas extremas para verificar como eles se comportam sob pressão máxima;
- Teste de desempenho. Ajuda a medir e analisar o desempenho de aplicativos, incluindo tempos de resposta, taxa de transferência, uso de recursos e muito mais.
- Teste funcional; Além do teste de desempenho, o JMeter também pode ser usado para testar a funcionalidade de um aplicativo;
- Suporte a diversos protocolos, oferece suporte a uma ampla variedade de protocolos, incluindo HTTP, HTTPS, FTP, JDBC, LDAP, SOAP, REST, e mais.
- Análise de resultados, possui a funcionalidade de geração de relatórios detalhados e gráficos que ajudam a interpretar os resultados dos testes.

O foco do presente trabalho está em testes com diferentes modelos bancos de dados de forma a comparar seu desempenho. Assim, será dada ênfase aos recursos voltados para bancos de dados. Estes recursos são demonstrados na Figura 29.

A Figura 29 mostra a tela do JMeter demonstrando a conexão com o banco de dados PostgreSQL. Na Figura 29 à esquerda, podem ser notados outros elementos que foram adicionados ao plano de teste, sendo alguns deles:

- Thread Group, neste elemento pode ser configurado o número de *threads* (usuários) que serão utilizados pelo programa ao realizar um teste de estresse por exemplo;
- Transaction Controller, é um elemento utilizado para controlar a transação, neste caso só há uma transação portanto apenas um elemento, esse componente é necessário para contabilizar o tempo total gasto pela transação;
- Loop Controller, como o nome sugere, é um laço de repetição. Nele pode ser inserido um valor para a quantia de vezes que as operações serão executadas, funciona exatamente como um laço de repetição em uma linguagem de programação;
- User Defined Variables, no JMeter, existe a possibilidade de criar variáveis para serem utilizadas nos testes, inserindo seu nome e um valor para a mesma, por exemplo, pode-se criar uma variável com um identificador em que deseja realizar um *update*, então na *query*, na condição ”WHERE”, pode-se chamar a variável criada.

Figura 29 – Tela do JMeter mostrando a conexão ao PostgreSQL.



Fonte: Elaborada pelo autor

- Requests (JDBC Insert e JDBC Select), é o lugar onde são inseridos os comandos (*queries*), é necessário selecionar o tipo de comando a ser excutado, podendo ser *select, update, commit, rollback, delete, insert* etc. Basta utilizar como um SGDB comum, apenas sinalizar o tipo da *query* utilizada, tendo por exemplo, caso seja um comando para deletar um registro deve-se selecionar ”delete”.

5.1 METODOLOGIA DE TESTES UTILIZADA

Os Sistemas de Gerenciamento de Bancos de Dados (SGBDs) a serem testados e as ferramentas necessárias para o trabalho foram instalados e devidamente configurados através de um *container* Docker[2]. No caso foram instalados os SGDBs PostgreSQL, TimeScaleDB e Influx.

A conectividade com o PostgreSQL é estabelecida através do uso do Java Database Connectivity (JDBC), que de acordo com (PATEL; MOSS, 1997), consiste em um conjunto de classes e interfaces em linguagem Java destinadas a enviar instruções SQL para qualquer banco de dados. Isso possibilita a emissão de requisições a partir do JMeter para os SGBDs e a obtenção de dados por meio dessa conexão.

Já a conexão com o InfluxDB, foi feita através da API disponibilizada pelo mesmo para inserção, leitura e configuração dos *buckets*.

[2]https://www.docker.com/

Os testes foram divididos em duas categorias distintas: carga e estresse. No teste de carga, foi analisado o comportamento de cada SGBD ao aumentar a quantidade de registros no banco de dados em cada operação, sendo registrados os tempos de execução de cada operação. O objetivo foi avaliar a eficácia de cada SGBD em relação a uma operação específica com um determinado número de registros.

Por outro lado, o teste de estresse envolveu submeter o banco de dados a um elevado volume de requisições com um grande número de usuários ou *threads* simultâneos. O intuito foi observar se o banco era capaz de suportar essa carga e se mantinha eficiente nesse processo.

Essas métricas foram utilizadas para mensurar o desempenho dos SGBDs, considerando o tempo necessário para concluir as operações, sua capacidade de execução e identificar em qual ponto o banco de dados não conseguiu atender totalmente às solicitações devido ao estresse aplicado, resultando em erros.

Para a realização dos testes, foi criada uma base de dados semelhante em cada um dos SGBDs. Isso significa que as mesmas informações foram inseridas em cada um deles, garantindo que os resultados das solicitações fossem comparáveis, alterando apenas a sintaxe para atender ao formato requisitado e utilizado pelo InfluxDB.

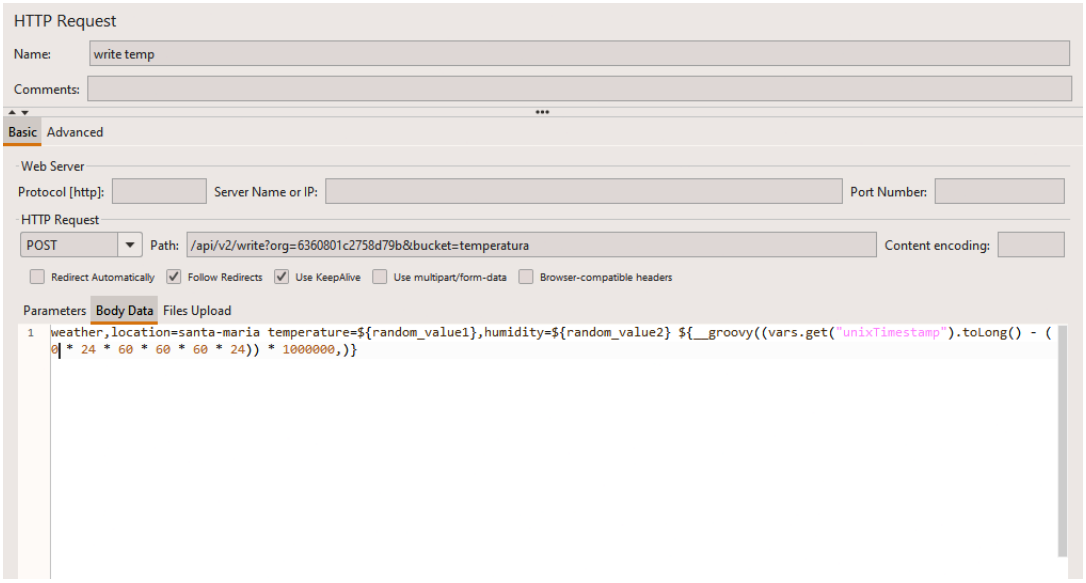
Figura 30 – Tela do JMeter mostrando a query de insert do PostgreSQL.

Fonte: Elaborada pelo autor

A figura 30 mostra a textitquery utilizada para inserir dados no PostgreSQL, a variável "dataHora"inserida é para manipular a data de inserção de dados, podendo simular a inserção de dados de dias diferentes estando no mesmo dia.

Já a figura 31, mostra a *query* utilizada para inserir dados no InfluxDB, nota-se que é diferente o método de inserção, sendo feito através de uma requisição a API do InfluxDB e não através de uma conexão JDBC. A expressão com os números ao final do comando, é pelo mesmo motivo do PostgreSQL, para manipular a data de inserção de dados, por o InfluxDB utilizar o *timestamp* no formato Unix em nanossegundos, não possível criar uma variável semelhante ao PostgreSQL, sendo necessário a utilização desta expressão matemática.

Figura 31 – Tela do JMeter mostrando a query de insert do InfluxDB.



Fonte: Elaborada pelo autor

5.2 RESULTADOS

Este capítulo tem como objetivo apresentar os resultados dos testes realizados em dois sistemas gerenciadores de banco de dados (SGDBs): o InfluxDB e o PostgreSQL. Os testes consistiram em avaliar o desempenho dos SGDBs em cenários de carga e de estresse, variando o número de *threads* (usuários) e o número de inserções na base de dados. Os valores utilizados para esses parâmetros foram baseados em trabalhos semelhantes. O objetivo dos testes foi comparar o tempo de resposta, a taxa de transferência e o consumo de recursos dos SGDBs, a fim de identificar qual deles se adapta melhor às necessidades da aplicação.

5.2.1 Testes de carga

O primeiro teste de carga realizado, foi com 1000 (um mil) registros. Foram inseridos mil registros através do JMeter em cada SGDB e analisado seus desempenhos.

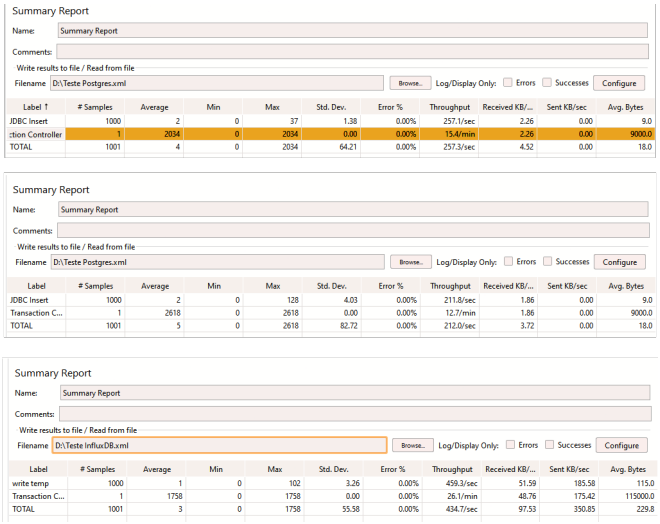
Ao realizar a inserção de 1000 registros, foram obtidos os resultados mostrados na Tabela 2.

PostgreSQL	TimeScaleDB	InfluxDB
2034ms ou 2,034 segundos	2618 ms ou 2,618 segundos	1758 ms ou 1,758 segundos

Tabela 2 – Valores obtidos no teste de 1000 inserções

O segundo teste de carga realizado, foi com 10000 (dez mil) registros. Foram inseridos dez mil registros através do JMeter em cada SGDB e analisado seus desempenhos.

Figura 32 – Tela do JMeter mostrando os tempos de insert no PostgreSQL, TimeScale e InfluxDB respectivamente.



Fonte: Elaborada pelo autor

Ao realizar a inserção de 10000 registros, foram obtidos os seguintes resultados que podem ser visualizados na Tabela 3.

PostgreSQL	TimeScaleDB	InfluxDB
20894ms (20,894 seg)	23527ms (23,527 seg)	14333ms (14,333 seg)

Tabela 3 – Valores obtidos no teste de 10000 inserções

Ao realizar a inserção de 100000 registros, foram obtidos mostrados na Tabela 4.

PostgreSQL	TimeScaleDB	InfluxDB
199115ms ou 3min e 32seg	224386 ms ou 3min e 44seg	124691 ms ou 2min e 5seg

Tabela 4 – Valores obtidos no teste de 100000 inserções

Ao realizar a inserção de 1.000.000 registros, foram obtidos os seguintes resultados, mostrados na Tabela 5.

PostgreSQL	TimeScaleDB	InfluxDB
2042514ms (34min e 04seg)	2127949ms (35min e 28seg)	996855 ms (16min e 37seg)

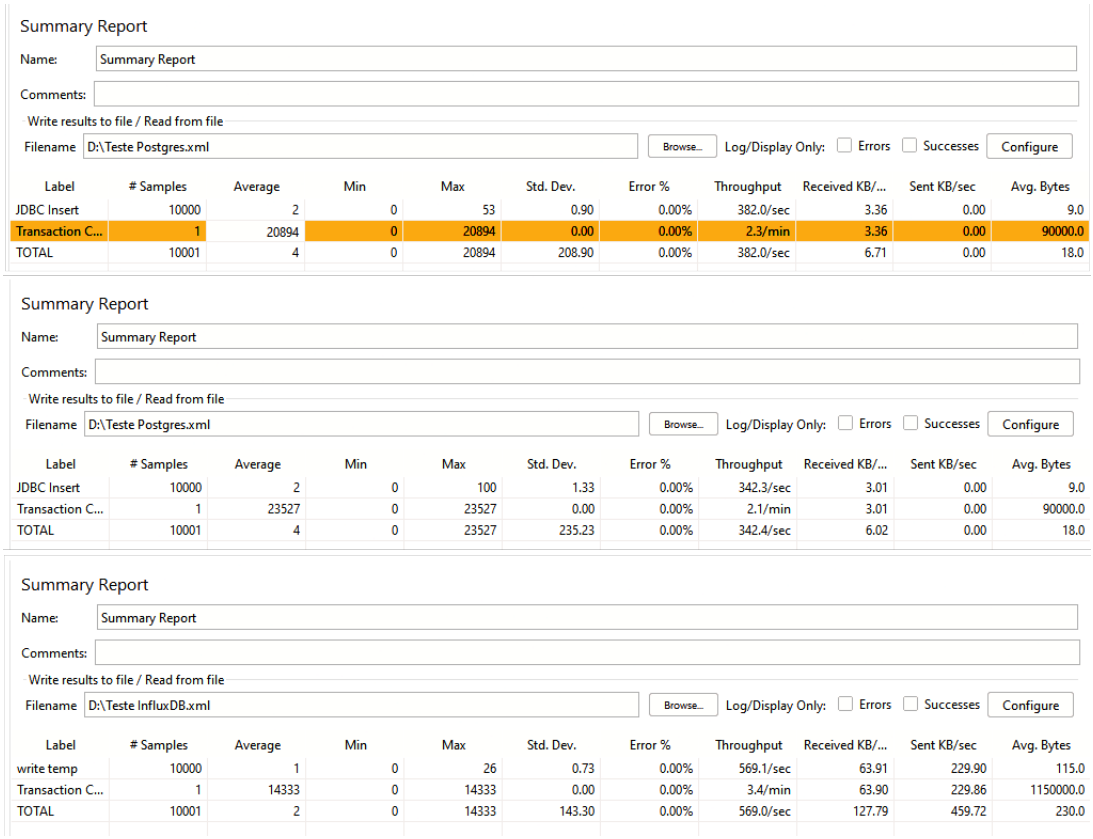
Tabela 5 – Valores obtidos no teste de um milhão de inserções

Ao realizar a inserção de 5.000.000 de registros, foram obtidos os resultados visualizados na Tabela 6.

PostgreSQL	TimeScaleDB	InfluxDB
10256886 (2hr:50min:57seg)	10273940ms (2hr:51min:14seg)	5932970ms (1hr:38min:53seg)

Tabela 6 – Valores obtidos no teste de cinco milhões de inserções

Figura 33 – Tela do JMeter mostrando o tempo de insert de 10000 registros no PostgreSQL, TimeScale e InfluxDB.



Fonte: Elaborada pelo autor

5.2.2 Resultados obtidos

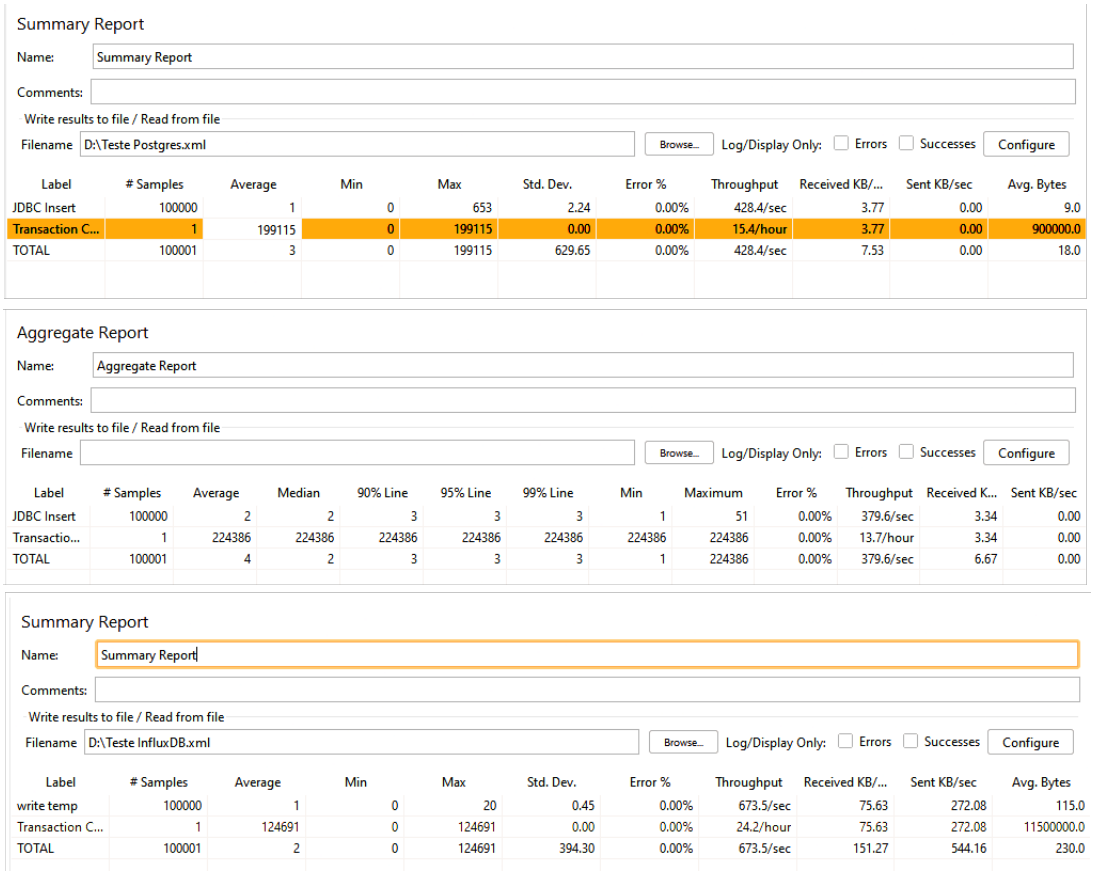
Analisando os resultados dos testes, foram obtidos os resultados mostrados na tabela 7.

Inserções	PostgreSQL	TimeScaleDB	InfluxDB
1.000	2,034 segundos	2,618 segundos	1,758 segundos
10.000	20,894 segundos	23,527 segundos	14,333 segundos
100.000	3m:32s	3m:44s	2m:5s
1 milhão	34m:04s	35m:28s	16m:37
5 milhões	2h:50m:57s	2h:51m:14s	1h:38m:53s

Tabela 7 – Valores obtidos ao finalizar os testes

Os Gráficos 1 e 2, mostram os resultados obtidos nos testes de carga, ou seja, mostra o tempo gasto por cada SGDB em cada volume de dados testados. Ao observar o Gráfico 1, é possível ver que o InfluxDB superou o PostgreSQL e o TimeScaleDB no quesito tempo. Em comparação ao PostgreSQL (por ser o banco relacional utilizado para métrica), é possível notar que a curva de velocidade em relação ao PostgreSQL aumenta conforme o volume de dados inseridos, apenas nos 5 milhões que houve uma pequena queda de *perfomance*.

Figura 34 – Tela do JMeter mostrando o tempo de insert de 100000 registros no PostgreSQL, TimeScale e InfluxDB.



Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename: D:\Teste Postgres.xml — Browse... — Log/Display Only: Errors Successes — Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
JDBC Insert	100000	1	0	653	2.24	0.00%	428.4/sec	3.77	0.00	9.0
Transaction C...	1	199115	0	199115	0.00	0.00%	15.4/hour	3.77	0.00	900000.0
TOTAL	100001	3	0	199115	629.65	0.00%	428.4/sec	7.53	0.00	18.0

Aggregate Report

Name: Aggregate Report

Comments:

Write results to file / Read from file

Filename: — Browse... — Log/Display Only: Errors Successes — Configure

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received K...	Sent KB/sec
JDBC Insert	100000	2	2	3	3	3	1	51	0.00%	379.6/sec	3.34	0.00
Transactio...	1	224386	224386	224386	224386	224386	224386	224386	0.00%	13.7/hour	3.34	0.00
TOTAL	100001	4	2	3	3	3	1	224386	0.00%	379.6/sec	6.67	0.00

Summary Report

Name: Summary Report

Comments:

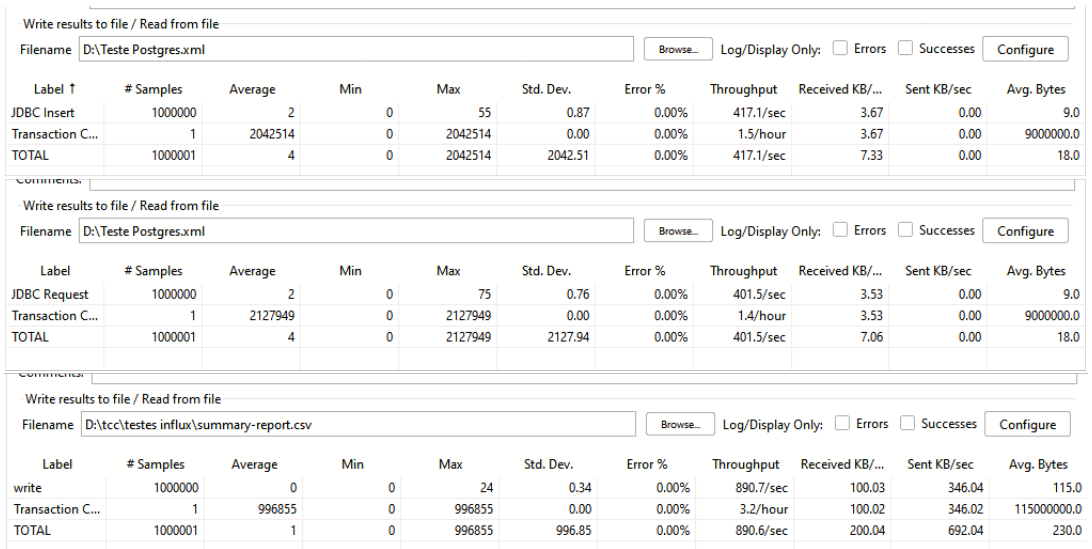
Write results to file / Read from file

Filename: D:\Teste InfluxDB.xml — Browse... — Log/Display Only: Errors Successes — Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
write temp	100000	1	0	20	0.45	0.00%	673.5/sec	75.63	272.08	115.0
Transaction C...	1	124691	0	124691	0.00	0.00%	24.2/hour	75.63	272.08	11500000.0
TOTAL	100001	2	0	124691	394.30	0.00%	673.5/sec	151.27	544.16	230.0

Fonte: Elaborada pelo autor

Figura 35 – Tela do JMeter mostrando o tempo de insert de 1 milhão de registros no PostgreSQL, TimeScale e InfluxDB.



Write results to file / Read from file

Filename: D:\Teste Postgres.xml — Browse... — Log/Display Only: Errors Successes — Configure

Label ↑	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
JDBC Insert	1000000	2	0	55	0.87	0.00%	417.1/sec	3.67	0.00	9.0
Transaction C...	1	2042514	0	2042514	0.00	0.00%	1.5/hour	3.67	0.00	9000000.0
TOTAL	1000001	4	0	2042514	2042.51	0.00%	417.1/sec	7.33	0.00	18.0

Write results to file / Read from file

Filename: D:\Teste Postgres.xml — Browse... — Log/Display Only: Errors Successes — Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
JDBC Request	1000000	2	0	75	0.76	0.00%	401.5/sec	3.53	0.00	9.0
Transaction C...	1	2127949	0	2127949	0.00	0.00%	1.4/hour	3.53	0.00	9000000.0
TOTAL	1000001	4	0	2127949	2127.94	0.00%	401.5/sec	7.06	0.00	18.0

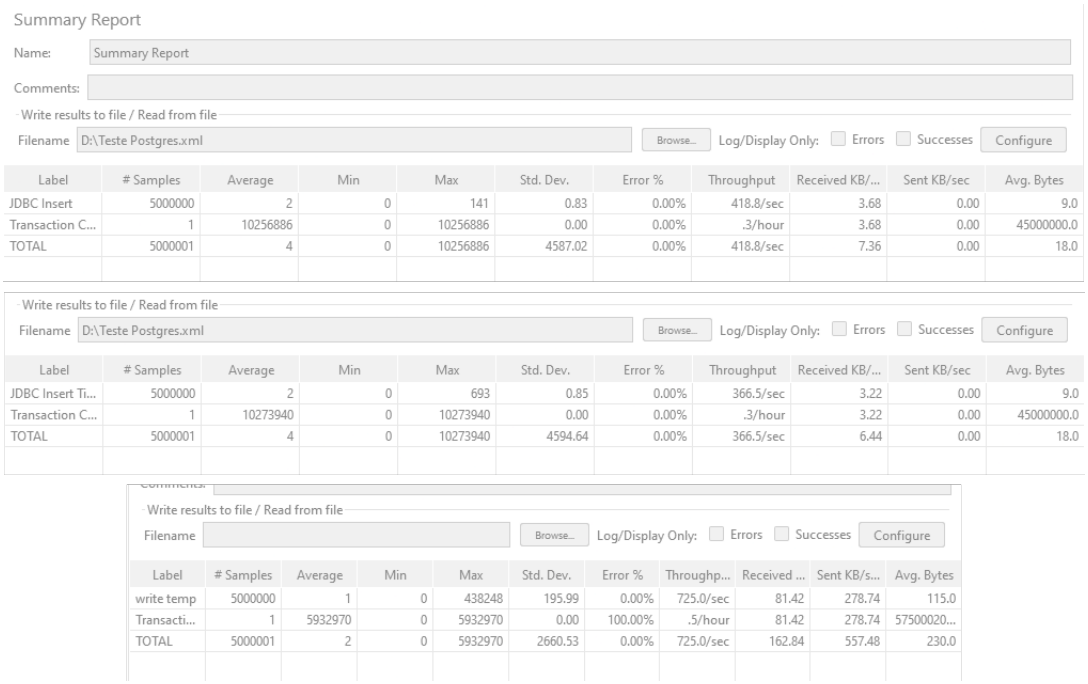
Write results to file / Read from file

Filename: D:\tcc\testes influx\summary-report.csv — Browse... — Log/Display Only: Errors Successes — Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
write	1000000	0	0	24	0.34	0.00%	890.7/sec	100.03	346.04	115.0
Transaction C...	1	996855	0	996855	0.00	0.00%	3.2/hour	100.02	346.02	115000000.0
TOTAL	1000001	1	0	996855	996.85	0.00%	890.6/sec	200.04	692.04	230.0

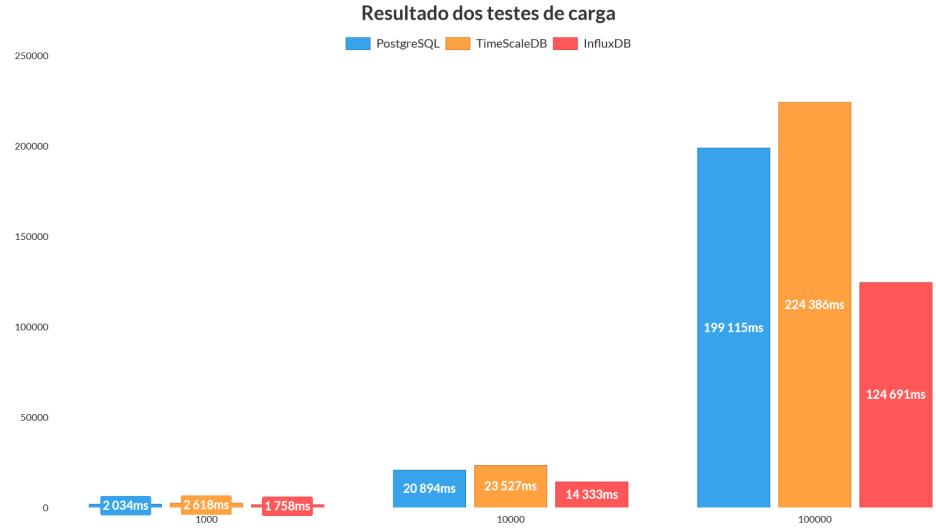
Fonte: Elaborada pelo autor

Figura 36 – Tela do JMeter mostrando o tempo de insert de 5 milhões de registros no PostgreSQL, TimeScale e InfluxDB.



Fonte: Elaborada pelo autor

Gráfico 1 – Representação gráfica dos resultados obtidos nos testes

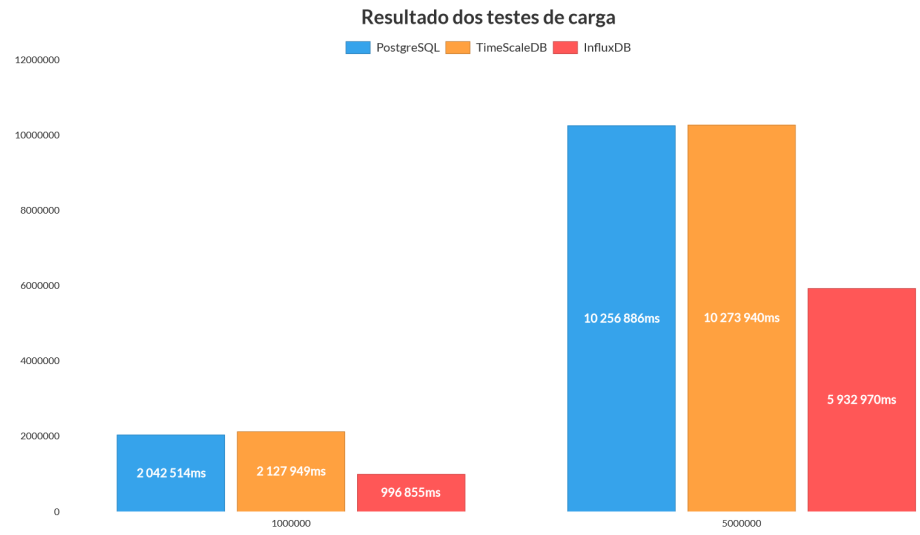


Fonte: Próprio autor.

5.3 TESTES DE LEITURA

Para os testes de leitura foram realizados dois experimentos, cada um com uma quantidade diferente de dados inseridos nos bancos. No primeiro experimento, foram inseridos 100000 dados ao longo de 5 dias em cada banco, e depois foi feita uma consulta para retornar esses dados agrupados por dia. No segundo experimento, foram inseridos 1000000 (um milhão) de dados em cada dia, e a mesma consulta foi repetida. Os resultados dos testes mostraram as

Gráfico 2 – Representação gráfica dos resultados obtidos nos testes



Fonte: Próprio autor.

diferenças de tempo de resposta pelos bancos de dados na leitura dos dados.

A *query* utilizada para esta consulta no PostgreSQL e TimeScaleDB é apresentada na Figura 37.

Figura 37 – Consulta utilizada no PostgreSQL e TimeScaleDB.

```
SELECT
    TO_CHAR(time, 'DD/MM/YYYY') AS data_leitura,
    AVG(t.temperature) as media_temperature,
    AVG(t.umidade) as media_umidade,
    t.cidade,
    t.estado,
    count(*) as linhas
FROM temperatura t
GROUP BY data_leitura, t.cidade, t.estado;
```

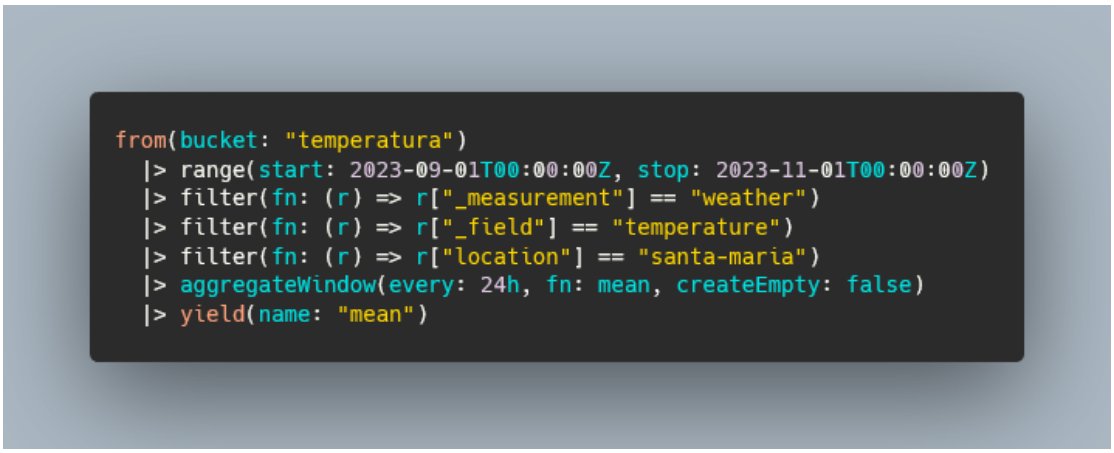
Fonte: Elaborada pelo autor

Já no InfluxDB, a *query* foi feita na linguagem Flux (linguagem nativa do InfluxDB), resultando na consulta apresentada na Figura 38

Ao realizar a consulta nos SGDBs, com agrupamento dos dados em 5 dias distintos (100.000 registros por dia totalizando 500.000 registros) foi obtido o resultado mostrado na Tabela 8.

Sobre a Figura 40, vale ressaltar que as respostas são diferentes entre si pois para cada teste são gerados valores randômicos, ou seja, os valores dos campos ”temperatura” e ”umi-

Figura 38 – Consulta utilizada no InfluxDB.



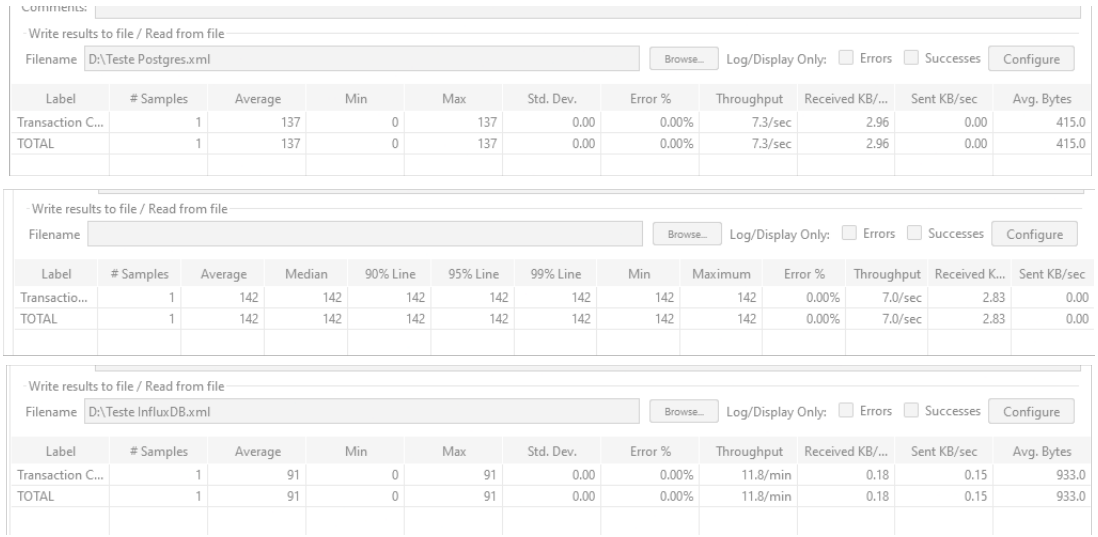
```
from(bucket: "temperatura")
  |> range(start: 2023-09-01T00:00:00Z, stop: 2023-11-01T00:00:00Z)
  |> filter(fn: (r) => r["_measurement"] == "weather")
  |> filter(fn: (r) => r["_field"] == "temperature")
  |> filter(fn: (r) => r["location"] == "santa-maria")
  |> aggregateWindow(every: 24h, fn: mean, createEmpty: false)
  |> yield(name: "mean")
```

Fonte: Elaborada pelo autor

PostgreSQL	TimeScaleDB	InfluxDB
137ms	142ms	91ms

Tabela 8 – Valores obtidos no teste de leitura com 100 mil registros diários

Figura 39 – Tela do JMeter mostrando o tempo de execução da query de leitura de dados no PostgreSQL, TimeScaleDB e InfluxDB.



Fonte: Elaborada pelo autor

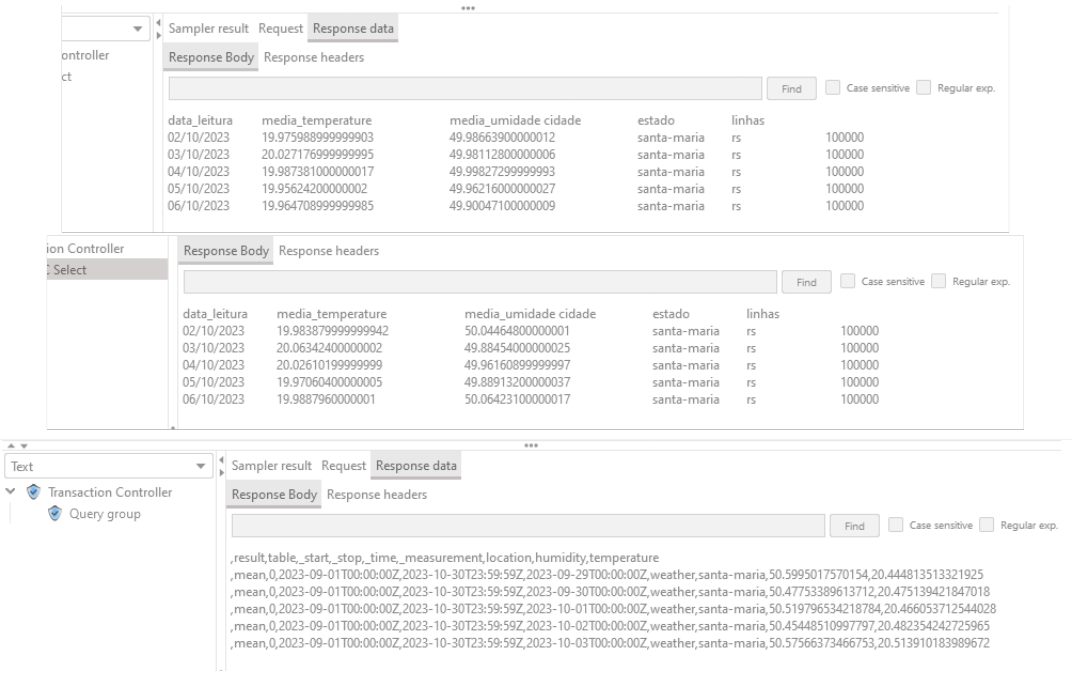
dade" mudam de um teste para o outro.

Ao realizar a consulta nos SGDBs, com agrupamento dos dados em 5 dias distintos (1.000.000 registros por dia totalizando 5.000.000 registros) foi obtido o resultado mostrado na Tabela 9.

PostgreSQL	TimeScaleDB	InfluxDB
1291ms	1952ms	142ms

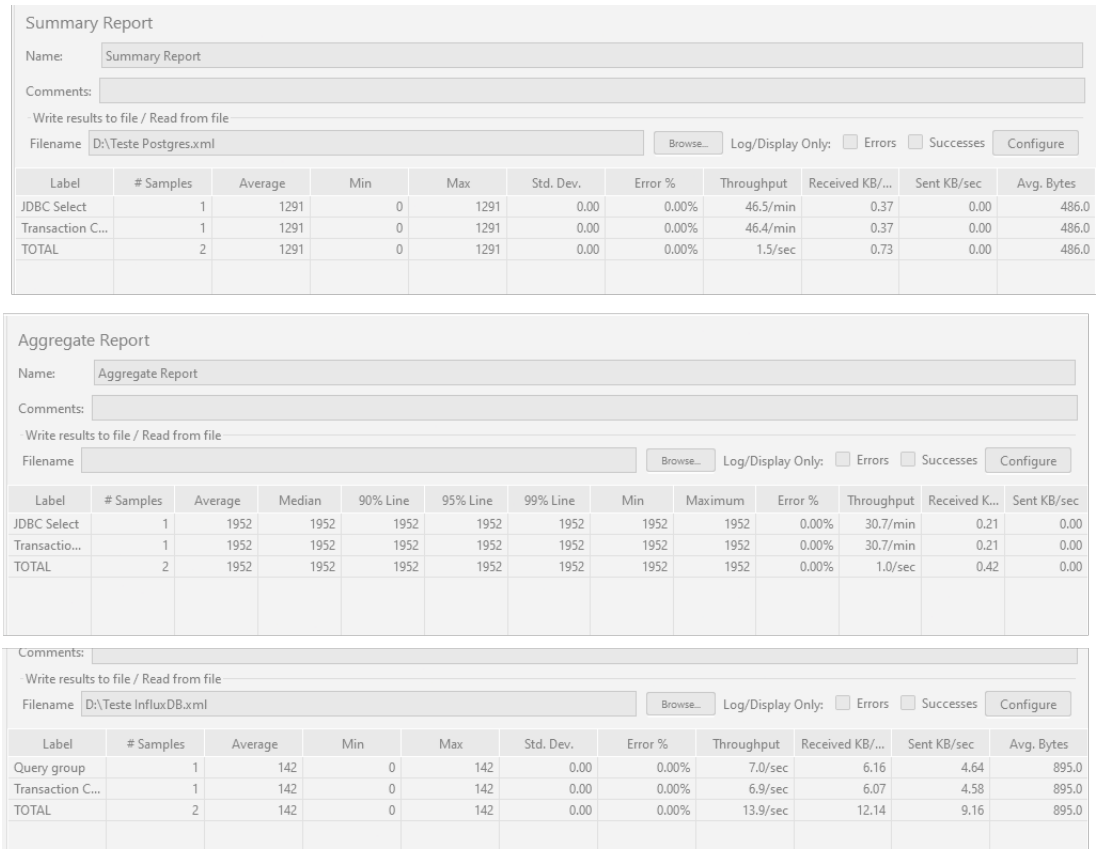
Tabela 9 – Valores obtidos no teste de leitura com 1 milhão de registros diários

Figura 40 – Tela do JMeter mostrando a resposta da query de leitura no PostgreSQL, TimeScaleDB e InfluxDB.



Fonte: Elaborada pelo autor

Figura 41 – Tela do JMeter mostrando o tempo de execução da query de leitura com 1 milhão de registros diários no PostgreSQL, TimeScaleDB e InfluxDB.



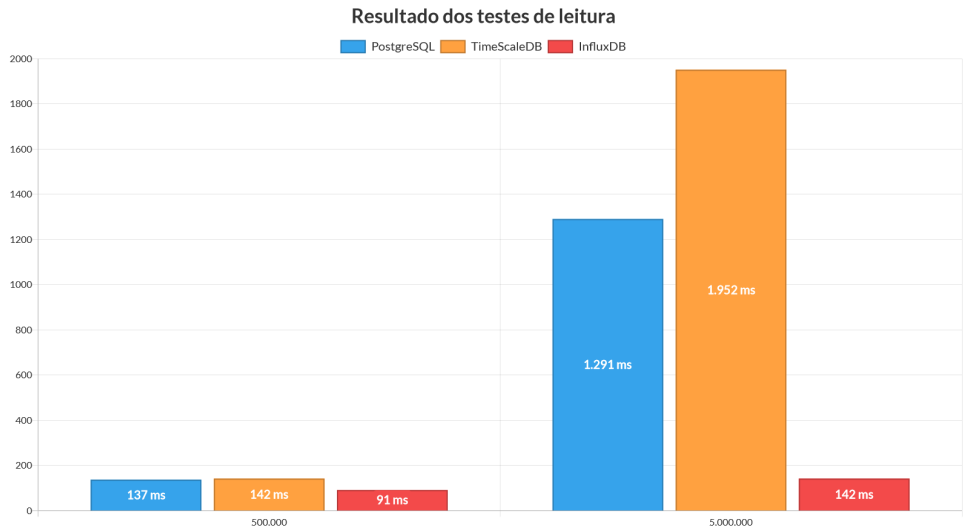
Fonte: Elaborada pelo autor

5.3.1 Resultados obtidos nos testes de leitura

Os dados do resultado obtido nos testes de leitura, são mostrados no Gráfico 3. Analisando os resultados obtidos nos testes de leitura, o InfluxDB se mostrou superior no quesito tempo de busca em relação ao PostgreSQL e TimeScaleDB. Por ser um banco de dados voltado para a análise de dados, na própria *query* de consulta, através dos filtros, a busca é mais rápida. Facilitando assim o cálculo da média.

Em ambos testes de consultas, com 100 mil registros diários e com 1 milhão de registros diários, o InfluxDB se mostrou superior no tempo de busca e cálculo da média das colunas ”temperatura” e ”umidade”.

Gráfico 3 – Representação gráfica dos resultados obtidos nos testes de leitura



Fonte: Próprio autor.

5.4 DISCUSSÃO DE RESULTADOS

Com a análise dos resultados obtidos com os testes nos bancos PostgreSQL, InfluxDB e TimeScaleDB, foi possível analisar o desempenho e o funcionamento dos SGDBs.

Foi possível observar que o InfluxDB é uma opção altamente eficiente para armazenar e consultar dados de séries temporais em tempo real. Sua arquitetura otimizada para esse tipo de dado e sua gama de recursos tornam-no adequado para sistemas que utilizam dados temporais ou dados que necessitem de gráficos gerados rapidamente.

Sobre o TimeScaleDB foi observado a facilidade de escrever consultas e inserções, tendo em vista que se trata de uma extensão para um banco relacional que usar instruções SQL. Porém nos testes realizados, não mostrou ser superior ao InfluxDB e nem ao PostgreSQL em relação ao tempo de execução de consultas e inserções de dados, resultado semelhante obtido por Shah,

Jat e Sashidhar (2022). Nos resultados obtidos por Shah, Jat e Sashidhar (2022), onde entre bancos de dados NoSQL, o InfluxDB se destacou pela alta taxa de inserção e consulta, como visto e analisado neste trabalho.

Outro resultado obtido, foi que o InfluxDB se mostrou superior em consultas agregadas por um intervalo de tempo. Neste quesito, o resultado obtido por (MUSA et al., 2019), onde foram feitos testes no PostgreSQL e no InfluxDB, foi o similar. Indicando que para consultas agregadas, é melhor utilizar o InfluxDB ao invés do PostgreSQL.

5.4.1 Trabalhos Relacionados

Foi constatado a existência de diversos trabalhos que possuem como ênfase medir, comparar e analisar o desempenho entre SGDBs.

FILHO (2015) realizou um teste comparativo entre o MongoDB e o PostgreSQL, usando o JMeter como ferramenta de avaliação. O teste consistiu em importar uma grande quantidade de dados e executar consultas de correspondência exata, retornando os dados conforme o critério definido. O número de usuários foi aumentado e o tempo de início entre eles foi variado, observando o desempenho dos bancos. Ao final, o MongoDB mostrou resultados melhores em comparação ao PostgreSQL.

Já Pires, Nascimento e Salgado (2006) realizaram um estudo comparativo do desempenho de bancos de dados *open source* (bancos de dados de código livre), usando os SGBDs MySQL e PostgreSQL, com o auxílio do *benchmark* OSDB[3], que também é open source. O objetivo do estudo foi avaliar as métricas produzidas pelo OSDB e sugerir melhorias nas funcionalidades dos SGBDs relacionadas ao desempenho. Os resultados mostraram que o MySQL teve um desempenho superior na maioria dos testes. O PostgreSQL só se destacou no módulo de carga e estrutura, principalmente na criação de índices.

Também foi realizado por Shah, Jat e Sashidhar (2022) um trabalho de comparação entre bancos de dados. Foram utilizados para comparação os bancos: TimescaleDB, Druid[4], InfluxDB e Cassandra. Foram obtidos algumas conclusões, como: Devido ao seu mecanismo de armazenamento exclusivo, o InfluxDB possui uma alta taxa de inserção e consulta; o processamento do Druid enfrenta problemas com concorrência excessiva; o TimescaleDB utiliza um formato de armazenamento por linha emprestado do PostgreSQL, resultando em baixa velocidade de consulta; Por fim, o Cassandra é um banco de dados NoSQL que utiliza formato de coluna para armazenar dados e oferece melhor desempenho de leitura e gravação do que o TimescaleDB.

As pesquisas mencionadas visam fazer uma comparação entre SGBDs, usando o software específico para isso. Neste trabalho, o objetivo é semelhante, onde o software a ser usado

[3]https://osdb.sourceforge.net/
[4]https://druid.apache.org/

nos testes é o JMeter. O que difere este trabalho dos exemplos acima citados, são os SGBDs escolhidos para o comparativo, possuindo a presença de um banco de dados de série temporal, um banco de dados relacional e também o mesmo banco de dados com uma extensão para dados de série temporal.

6 CONSIDERAÇÕES FINAIS

Este trabalho explorou a aplicação de bancos de dados de séries temporais, mais especificamente os bancos InfluxDB e TimeScaleDB. A pesquisa abordou os desafios enfrentados ao lidar com grandes volumes de dados temporais e avaliou a capacidade de cada banco de dados em atender às necessidades específicas que surgem ao utilizar dados temporais.

Durante o estudo, foram realizados testes para analisar o desempenho e o funcionamento dos dois SGDBs, onde foi possível observar que o InfluxDB é uma opção altamente eficiente para armazenar e consultar dados de séries temporais e dados agregados por um determinado intervalo de tempo.

Com isto, é possível dizer que a maior vantagem de usar um banco de dados de série temporal no lugar de um relacional, é na velocidade de execução de consultas agregadas. Os testes de leituras realizados na seção 5.3, demonstram a diferença de tempo de execução entre o banco relacional e o InfluxDB. Uma outra vantagem é a de possibilitar uma rápida análise de dados, onde atráves dos resultados de uma consulta, é possível gerar um gráfico sobre os dados obtidos, possibilitando uma análise mais visual dos resultados.

Para dados de série temporal, utilizar um banco próprio para este tipo de dado é o recomendável, mas nada impede que seja feito a inserção e leitura de dados em um banco relacional. Como visto nos resultados obtidos, a maior desvantagem de um banco relacional para um TSDB é o tempo de execução das *querys* de inserção e leitura.

Como sugestão de trabalhos futuros, propõe-se comparar o banco de dados InfluxDB com outros bancos de série temporal, tendo em vista que como visto neste trabalho, em relação ao modelo relacional ele possui um melhor desempenho. Essa comparação pode envolver aspectos como escalabilidade, disponibilidade, consistência, segurança e facilidade de uso. Além disso, pode-se avaliar o impacto do InfluxDB em diferentes cenários de aplicação, como Internet das Coisas, monitoramento de redes, análise de séries financeiras, entre outros. Dessa forma, espera-se contribuir para o avanço do conhecimento sobre as vantagens e desvantagens dos bancos de série temporal em relação aos modelos tradicionais.

REFERÊNCIAS

APACHE. **Sobre**. 2023. Acessado em 09 jun 2023. Disponível em: <https://jmeter.apache.org-/>.

CRUZ, J. M. d. Nosql - esquemas flexíveis para gerenciamento de banco de dados. jun. 2015. Disponível em: <https://ric.cps.sp.gov.br/handle/123456789/411>.

DUNNING, T. et al. **Time Series Databases: New Ways to Store and Access Data**. O'Reilly Media, Incorporated, 2014. ISBN 9781491914724. Disponível em: <https://books.google-.com.br/books?id=xfngrQEACAAJ>.

ELMASRI, R.; NAVATHE, S. B. **Fundamentals of database systems**. [S.l.]: Pearson, 2010. v. 6.

FILHO, M. A. P. M. Sql x nosql: Análise de desempenho do uso do mongodb em relação ao uso do postgresql. **Trabalho de Graduação (Graduação em Ciência da Computação). Universidade Federal de Pernambuco. Recife**, p. 2014–2, 2015.

HALILI, E. H. **Apache JMeter**. [S.l.: s.n.], 2008.

INFLUXDATA. **Documentation**. 2023. Acessado em 09 jun 2023. Disponível em: <https:/-/docs.influxdata.com/>.

KULKARNI, A.; BOOZ, R.; TOTH, A. **What is time-series data, and why do I need A time-series database?** Timescale Blog, 2022. Disponível em: <https://www.timescale.com-/blog/time-series-data/>.

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

MORETTIN, P. A.; CASTRO, T. C. M. d. **Analise de series temporais**. [S.l.]: Edgard Blucher, 2008.

MUSA, E. et al. Comparison of relational and time-series databases for real-time massive datasets. **MIPRO Computers in Technical Systems**, p. 1065–1070, 2019.

PATEL, P.; MOSS, K. **Java database programming with JDBC**. [S.l.]: Coriolis Group, 1997.

PIRES, C. E.; NASCIMENTO, R. O.; SALGADO, A. C. Comparativo de desempenho entre bancos de dados de código aberto. **Escola Regional de Banco de Dados, Anais da ERBD06, Porto Alegre**, 2006.

RAMAKRISHNAN, R.; GEHRKE, J. 18 - transaction management overview. In: ____. **Database Management Systems**. [S.l.]: McGraw-Hill, 2000. cap. 18, p. 524524.

ROBBINS, A. **Effective AWK Programming: Text Processing and Pattern Matching**. [S.l.]: "O'Reilly Media, Inc.", 2001.

SADALAGE, P. J.; FOWLER, M. **NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence**. 1st. ed. [S.l.]: Addison-Wesley Professional, 2012. ISBN 0321826620.

SHAH, B.; JAT, P.; SASHIDHAR, K. Performance study of time series databases. **arXiv preprint arXiv:2208.13982**, 2022.

SHARP, J. et al. **Data Access for Highly-Scalable Solutions: Using SQL, NoSQL, and Polyglot Persistence**. 1st. ed. [S.l.]: Microsoft patterns amp; practices, 2013. ISBN 162114030X.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Database system concepts**. 7. ed. [S.l.]: McGraw-Hill Education, 2019.

SILVA, L. F. C. et al. **Banco de Dados Não Relacional**. 1st. ed. [S.l.]: Sagah, 2021. ISBN 9786556901534.

TIMESCALE. **Timescale Documentation**. 2023. Acessado em 16 jun 2023. Disponível em: <https://docs.timescale.com/>.