SQL vs. NoSQL: Um Estudo Comparativo de Desempenho em Operações CRUD com PostgreSQL e MongoDB

Guilherme Augusto Fernandes

Universidade Federal de Uberlândia
Faculdade de Computação
Bacharelado em Sistemas de Informação

Monte Carmelo - MG
2026

Guilherme Augusto Fernandes

SQL vs. NoSQL: Um Estudo Comparativo de Desempenho em Operações CRUD com PostgreSQL e MongoDB

Trabalho de Conclusão de Curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Área de concentração: Sistemas de Informação

Orientador: Professora Fabíola Souza Fernandes Pereira

Monte Carmelo - MG
2026

Dedico este trabalho à minha esposa, Beatriz, pelo amor, pela compreensão e por estar ao meu lado em cada desafio e conquista desta caminhada. Dedico também, aos meus pais , Oswaldo e Delmira, esta conquista com profunda gratidão. Obrigado por todo o amor, pelos ensinamentos, pelo exemplo de caráter, honestidade e perseverança. Nada disso seria possível sem o apoio, os conselhos e a confiança que sempre depositaram em mim. Cada etapa vencida até aqui carrega um pouco da história, dos esforços e dos sonhos de vocês.

Agradecimentos

Primeiramente a Deus, por me dar força, saúde e perseverança para superar os desafios e chegar até esta etapa. À minha esposa Beatriz, pelo amor, companheirismo e apoio incondicional durante toda esta jornada. Aos meus pais, pelo incentivo, pelos valores transmitidos e por nunca medirem esforços para que eu chegasse até aqui. À minha orientadora, Prof.ª Fabíola Souza Fernandes Pereira, pela dedicação, paciência e pelos ensinamentos fundamentais para a realização deste trabalho. À Universidade Federal de Uberlândia, pelo ensino de qualidade e pela estrutura oferecida ao longo do curso. A todos que, de alguma forma, contribuíram para a conclusão deste trabalho, meu sincero obrigado.

"Os dados são um recurso precioso e durarão mais que os próprios sistemas."
(Tim Berners-Lee)

Resumo

O crescimento exponencial no volume de dados gerados por aplicações modernas tem impulsionado a adoção de diferentes paradigmas de gerenciamento de banco de dados, tornando relevante a compreensão das diferenças de desempenho entre sistemas relacionais e não relacionais. Este trabalho apresenta uma análise comparativa de desempenho entre o PostgreSQL, representando o paradigma relacional (SQL), e o MongoDB, representando o paradigma orientado a documentos (NoSQL), por meio da execução das operações fundamentais de criação, leitura, atualização e exclusão de dados (CRUD). Os experimentos foram conduzidos sobre dois conjuntos de dados distintos: um *dataset* de e-commerce com aproximadamente 600.000 registros e um *dataset* de imóveis do Airbnb com aproximadamente 450.000 registros, sendo este último composto por uma relação 1:N entre imóveis e avaliações. Os testes foram implementados por meio de uma aplicação em Java, e cada operação foi executada cinco vezes para o cálculo de média e desvio padrão. Os resultados indicam que o MongoDB apresentou desempenho superior nas operações de inserção, atualização em larga escala e exclusão, além de maior eficiência em consultas sobre dados semiestruturados. O PostgreSQL, por sua vez, demonstrou vantagem em operações de leitura analítica sobre conjuntos de dados tabulares de grande volume, evidenciando melhor desempenho do otimizador de consultas em cenários de varredura completa sobre dados estruturados. Conclui-se que a escolha entre os paradigmas deve considerar a natureza dos dados e os padrões de acesso da aplicação: dados semiestruturados e cargas intensas de escrita tendem a favorecer soluções NoSQL, enquanto dados tabulares com consultas analíticas complexas se beneficiam do modelo relacional.

Palavras-chave: Banco de dados, PostgreSQL, MongoDB, NoSQL, SQL.

Lista de ilustrações

Lista de tabelas

Lista de siglas

API Application Programing Interface - *Interface de Programação de Aplicações*

CRUD Create, Read, Update e Delete - *Criar, Ler, Atualizar e Deletar*

NoSQL Not Only SQL - *Não Apenas SQL*

SQL Structured Query Language - *Linguagem de Consulta Estruturada*

SGBD Sistema Gerenciador de Banco de Dados

ACID Atomicidade, Consistência, Isolamento e Durabilidade

Sumário

Capítulo 1

Introdução

O avanço tecnológico das últimas décadas transformou profundamente a forma como organizações armazenam e gerenciam informações. Com a expansão da internet e o surgimento de aplicações cada vez mais complexas, os sistemas de gerenciamento de banco de dados passaram a ocupar um papel central na infraestrutura de software moderna, sendo determinantes para o desempenho e a confiabilidade das aplicações. Diante da diversidade de soluções disponíveis atualmente, compreender as características e limitações de cada abordagem torna-se essencial para orientar decisões arquiteturais mais assertivas. Neste contexto, este capítulo apresenta a motivação, o problema de pesquisa, os objetivos, os métodos adotados e a organização deste trabalho.

1.1 Motivação

O crescimento exponencial no volume de dados gerados por setores como saúde, finanças e comércio eletrônico tem intensificado a demanda por sistemas de gerenciamento de banco de dados (SGBDs) capazes de sustentar operações de leitura e escrita em escala cada vez maior (KUMAR, 2026). Estimativas indicam que o volume global de dados criados, capturados e consumidos saltou de 147 zettabytes em 2024 para 181 zettabytes em 2025 (KUMAR, 2026), evidenciando que a capacidade de processar essas informações de forma eficiente deixou de ser uma vantagem competitiva para se tornar um requisito estrutural das aplicações modernas (ELMASRI; NAVATHE, 2005). Eles são amplamente utilizados em sistemas onde a integridade dos dados e a consistência transacional são requisitos fundamentais, como sistemas bancários e aplicações de ERP *(Enterprise Resource Planning)*. Por outro lado, os bancos de dados NoSQL, que oferecem esquemas flexíveis e maior capacidade de escalabilidade, são preferidos em ambientes de *Big Data* e computação em nuvem, onde a natureza dos dados é mais dinâmica e não estruturada, como em redes sociais e análise de logs de eventos (LÓSCIO; OLIVEIRA; PONTES, 2011).

Essa divisão, no entanto, não é absoluta: a escolha entre os dois paradigmas frequentemente carece de critérios objetivos de desempenho, sendo conduzida mais por convenção

de mercado ou familiaridade da equipe de desenvolvimento do que por evidências empíricas sobre o comportamento real de cada sistema em operações de manipulação de dados (CRUD). É justamente essa lacuna entre a decisão arquitetural e a evidência de desempenho que motiva a investigação comparativa proposta neste trabalho.

1.2 Problema

Apesar da existência de estudos comparativos entre bancos de dados relacionais e não relacionais, a literatura ainda carece de resultados consistentes sobre o desempenho desses paradigmas em operações fundamentais de manipulação de dados (CRUD). Parte dessa lacuna decorre de limitações metodológicas presentes em trabalhos correlatos: enquanto alguns estudos comparam bancos de dados NoSQL orientados a documentos com variantes específicas de SGBDs relacionais que não incluem o PostgreSQL (LI; MANOHARAN, 2013), outros apresentam caráter predominantemente introdutório, sem aprofundamento experimental (LÓSCIO; OLIVEIRA; PONTES, 2011). Adicionalmente, os poucos trabalhos que comparam diretamente MongoDB contra um banco SQL em operações CRUD, como (GYőRöDI et al., 2015), reportam desempenho superior do MongoDB em todas as operações avaliadas, resultado que carece de replicação sob diferentes volumes e estruturas de dados para verificar sua generalização.

Essa divergência entre os trabalhos correlatos, somada à escassez de estudos que isolem o efeito da estrutura dos dados (tabular versus semiestruturada) e do tipo de relacionamento entre entidades (ausência de relação versus relação 1:N) sobre o desempenho de cada paradigma, configura a lacuna que este trabalho busca investigar. Compreender como PostgreSQL e MongoDB se comportam sob diferentes volumes, estruturas e relacionamentos de dados é relevante tanto para a literatura quanto para a prática de desenvolvimento de software, contribuindo para decisões arquiteturais fundamentadas em evidências empíricas, e não apenas em convenção de mercado.

1.3 Hipóteses

Esta pesquisa parte das seguintes hipóteses, formuladas com base nas características intrínsecas de cada paradigma de banco de dados e que serão verificadas por meio dos experimentos descritos no Capítulo de experimentos.

- **H1 – Desempenho em operações de escrita:** O MongoDB apresenta desempenho superior ao PostgreSQL nas operações de inserção, atualização e exclusão de dados, em razão da ausência de validações de integridade referencial e da flexibilidade de seu esquema orientado a documentos. *Pergunta associada: O MongoDB é mais rápido que o PostgreSQL nas operações de escrita (Create, Update e Delete)?*

- **H2 – Desempenho em leitura sobre dados semiestruturados:** O MongoDB apresenta melhor desempenho em consultas sobre dados semiestruturados e com relacionamentos aninhados, devido ao modelo orientado a documentos, que permite o acesso direto às informações sem a necessidade de operações de junção entre tabelas. *Pergunta associada: O MongoDB é mais eficiente que o PostgreSQL em consultas sobre dados com estrutura hierárquica ou semiestruturada?*
- **H3 – Desempenho em leitura sobre dados tabulares:** O PostgreSQL apresenta melhor desempenho em consultas analíticas sobre grandes volumes de dados tabulares homogêneos, em virtude da maturidade do seu otimizador de consultas e do suporte a operações relacionais complexas. *Pergunta associada: O PostgreSQL é mais eficiente que o MongoDB em consultas analíticas sobre dados estruturados em formato tabular?*

1.4 Objetivos

1.4.1 Objetivo Geral

Investigar e comparar o desempenho dos paradigmas de bancos de dados relacional e orientado a documentos, representados respectivamente pelo PostgreSQL e pelo MongoDB, em operações fundamentais de manipulação de dados, contribuindo para a compreensão das condições em que cada abordagem apresenta vantagens mensuráveis de desempenho.

1.4.2 Objetivos Específicos

- Analisar o comportamento de desempenho do PostgreSQL e do MongoDB nas operações de criação, leitura, atualização e exclusão de dados (CRUDs), sob diferentes volumes e estruturas de dados;
- Investigar a influência da natureza dos dados, estruturados e semiestruturados, no desempenho das operações realizadas em cada paradigma;
- Comparar a variabilidade e a previsibilidade dos tempos de execução entre os dois sistemas, considerando múltiplas execuções de cada operação;
- Investigar como a estrutura de relacionamento entre entidades, considerando ausência de relações e relações do tipo 1:N, influencia o desempenho de cada paradigma nas operações CRUD;
- Analisar a consistência e a estabilidade dos tempos de execução de cada sistema por meio do cálculo de média e desvio padrão, avaliando a previsibilidade de desempenho em cenários de grande volume de dados;

- ❏ Contribuir com evidências empíricas que auxiliem desenvolvedores e arquitetos de software na escolha do sistema de gerenciamento de banco de dados mais adequado para diferentes cenários de aplicação.

1.5 Contribuições

Este trabalho apresenta as seguintes contribuições:

- ❏ Evidências empíricas sobre o desempenho comparativo entre PostgreSQL e MongoDB nas operações CRUD, considerando dois conjuntos de dados com características distintas, fornecendo subsídios concretos para a escolha do paradigma mais adequado em diferentes cenários de aplicação;
- ❏ Análise do impacto da natureza dos dados no desempenho dos sistemas avaliados, demonstrando que dados semiestruturados tendem a favorecer o MongoDB, enquanto dados tabulares homogêneos tendem a favorecer o PostgreSQL em consultas analíticas;
- ❏ Investigação do efeito da estrutura de relacionamento entre entidades no desempenho das operações de manipulação de dados, evidenciando que relações do tipo 1:N influenciam de forma significativa o tempo de execução, especialmente nas operações de atualização e exclusão;
- ❏ Caracterização da previsibilidade e estabilidade de cada sistema por meio da análise do desvio padrão dos tempos de execução, revelando que o PostgreSQL tende a apresentar maior consistência em operações de leitura, enquanto o MongoDB demonstra menor variabilidade em operações de escrita em grande escala.

1.6 Organização da Monografia

Esta monografia está organizada em cinco capítulos. O Capítulo 2 apresenta a fundamentação teórica, abordando os conceitos relacionados a bancos de dados relacionais e não relacionais, bem como aspectos relevantes para a análise de desempenho dessas tecnologias. O Capítulo 3 descreve os materiais e métodos utilizados, incluindo os datasets empregados, o ambiente experimental e a metodologia adotada para a execução dos experimentos. O Capítulo 4 apresenta os experimentos realizados e a análise dos resultados obtidos, comparando o desempenho dos bancos de dados nas operações de inserção, leitura, atualização e exclusão de dados. Por fim, o Capítulo 5 apresenta as conclusões do trabalho, destacando os principais resultados obtidos e as direções para trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Neste capítulo serão abordadas as principais características do banco de dados de modelo relacional SQLs e do modelo não relacional NoSQLs.

2.1 Modelo relacional

> "O modelo relacional foi proposto por Edgar Codd em 1970, como uma nova maneira de representação de dados. Neste seu trabalho Codd mostrou que uma visão relacional dos dados permite a sua descrição em uma maneira natural, sem que sejam necessárias estruturas adicionais para sua representação, provendo uma maior independência dos dados em relação aos programas. Em complementação, apresentou bases para tratar problemas como redundância e consistência. Mais tarde, em outro trabalho, Codd definiu uma álgebra relacional e provou, por meio de sua equivalência com o cálculo relacional, que ela era relacionalmente completa, dando fundamentação teórica ao modelo relacional"(MACARIO; BALDO, 2009, p. 1)

Com o surgimento do modelo relacional na década de 1970, a área de bancos de dados passou a contar com uma abordagem mais estruturada para o armazenamento e manipulação de informações. Em razão de suas características e da consistência de sua proposta, esse modelo tornou-se amplamente adotado, substituindo gradualmente os modelos hierárquico e de rede, predominantes até então. Entre os fatores que impulsionaram sua popularização destacam-se a organização dos dados em relações e a possibilidade de expressar consultas complexas de forma mais intuitiva e eficiente.

De acordo com (ELMASRI; NAVATHE, 2005, p. 20), o modelo relacional se baseia em três conceitos fundamentais: entidade, atributo e relacionamento. Uma entidade pode ser vista como um objeto do cotidiano, cujas características são definidas como atributos. Além disso, essas entidades podem se relacionar entre si, sendo denominado como relacionamento. O modelo relacional organiza os dados em tabelas (entidades), permitindo que se estabeleçam conexões entre diferentes tabelas. Para realizar operações de consulta, inserção, atualização e exclusão de dados, é necessário utilizar uma linguagem apropriada. A manipulação dos dados requer a manutenção da integridade, que é garantida por meio

de regras de normalização e do paradigma ACID (Atomicidade, Consistência, Isolamento e Durabilidade).

2.1.1 Características

O Modelo Relacional (MR) é formado por uma ou mais tabelas que representam os dados e suas relações. As tabelas são compostas por linhas e colunas, onde as linhas correspondem às tuplas (ou registros) e as colunas representam os atributos. Para que a integridade dos dados seja mantida, utilizam-se chaves. A presença das chaves possibilita a identificação única das linhas e o estabelecimento de relacionamentos entre as tabelas. As chaves podem ser primárias ou estrangeiras. As chaves primárias permitem identificar as tuplas (linhas) de forma única e determinar sua ordem, enquanto as chaves estrangeiras tornam os valores de um atributo dependentes de outro atributo existente, geralmente em uma tabela diferente.

Segundo (SCHMID; SWENSON, 1975, p. 219-220)), uma base de dados relacional é descrita em nível de esquema pela definição do modelo de dados. Essa definição especifica o nome de cada relação, todos os nomes dos atributos e a chave primária, além dos domínios subjacentes. Adicionalmente, são fornecidas informações sobre restrições independentes do tempo, como dependências funcionais, declarações de inclusão de conjuntos e restrições de integridade dinâmica. O autor também destaca a importância das regras de normalização, que são aplicadas passo a passo nas tabelas, com o objetivo de evitar a redundância e a mistura de assuntos distintos em uma mesma tabela.

2.1.2 ACID (Atomicidade, Consistência, Isolamento, Durabilidade)

O termo ACID é referente a um conjunto de propriedades que garantem confiabilidade das transações em bancos de dados, especialmente em sistemas de bancos de dados relacionais. Essas propriedades asseguram que as operações de transações, que podem envolver leitura e escrita de dados, ocorram, de forma consistente e segura. De acordo com (ELMASRI; NAVATHE, 2005, p. 403) a Atomicidade garante que uma transação seja tratada como uma unidade indivisível de processamento, ou seja, ela será executada completamente ou não será executada de forma alguma. Se qualquer parte da transação falhar, nenhuma mudança será aplicada ao banco de dados. A Preservação da Consistência assegura que a transação, ao ser executada integralmente, levará o banco de dados de um estado consistente para outro estado também consistente. Isso significa que todas as regras e restrições de integridade do banco de dados serão mantidas. O Isolamento protege as transações para que sejam executadas de forma independente, sem interferência de outras transações concorrentes. Isso garante que cada transação pareça ser executada de forma isolada, mesmo que haja múltiplas transações em andamento. Por fim, a

Durabilidade (ou permanência) assegura que as alterações realizadas por uma transação confirmada sejam persistentes no banco de dados, e essas mudanças não serão perdidas, mesmo em caso de falha do sistema.

2.1.3 Operações

Segundo (ELMASRI; NAVATHE, 2005, p. 100), o Modelo Relacional utiliza a linguagem SQLs (Structured Query Language), que possibilita a manipulação dos dados armazenados em tabelas. As operações básicas são:

- **Insert**: Comando responsável por inserir novos dados nas tabelas do banco de dados. Cada nova linha (ou tupla) é adicionada de acordo com a estrutura definida pela tabela, respeitando a ordem e os tipos de dados dos atributos.
- **Select**: Comando que permite consultar e recuperar dados de uma ou mais tabelas. É amplamente utilizado para realizar consultas específicas, com opções de filtros, ordenação e agregações para obter subconjuntos de dados, de acordo com critérios definidos pelo usuário.
- **Update**: Comando utilizado para modificar os dados existentes em uma tabela. Com ele, é possível alterar valores de uma ou mais colunas de um ou mais registros, respeitando condições especificadas. Ele é importante para manter os dados atualizados e consistentes ao longo do tempo.
- **Delete**: Comando responsável por remover dados existentes em uma tabela. Ao contrário do comando 'Drop', que remove a estrutura da tabela, o 'Delete' apenas exclui os dados, permitindo que a estrutura da tabela continue disponível para uso futuro.

2.1.4 Normalização

De acordo com (EESSAAR, 2016, p. 9) a teoria da normalização de banco de dados fornece uma base teórica para a organização eficiente dos dados, visando reduzir a redundância e evitar anomalias de atualização. Embora essa teoria seja aplicável a qualquer modelo de dados, ela é mais frequentemente associada ao modelo relacional, que é amplamente utilizado. A normalização é um processo que envolve várias etapas, conhecidas como formas normais. Cada forma normal aplica um conjunto de regras para garantir que as tabelas estejam bem estruturadas. A Primeira Forma Normal (1FN), por exemplo, exige que cada campo contenha apenas um valor único e que exista uma chave candidata para evitar duplicações. O processo de normalização não apenas melhora a integridade dos dados, mas também facilita a aplicação de restrições, tornando as relações entre os dados mais claras.

Embora a normalização ajude a eliminar redundâncias, ela não garante um design perfeito. O uso de duplicação controlada pode ser benéfico em sistemas distribuídos, melhorando a disponibilidade e a eficiência das consultas, mas exige um gerenciamento cuidadoso da consistência dos dados.

2.1.5 Limitações

Apesar de garantir a integridade dos dados, o Modelo Relacional possui algumas limitações significativas. Uma das principais é a falta de flexibilidade estrutural. Bancos de dados relacionais utilizam esquemas rígidos, o que significa que qualquer mudança na estrutura dos dados (como a adição de novos atributos) requer modificações no esquema do banco e, frequentemente, interrupções no sistema. Isso pode ser um problema em ambientes onde os requisitos de dados mudam com frequência (RAMAKRISHNAN; GEHRKE, 2002, p. 123).

Outra limitação é a escalabilidade vertical. Bancos de dados relacionais foram projetados para funcionar em um único servidor, o que significa que, à medida que o volume de dados cresce, é necessário adicionar mais recursos a esse servidor (como memória ou processamento). No entanto, essa abordagem tem um limite e não escala de maneira eficiente quando comparada à escalabilidade horizontal, comum em bancos de dados NoSQLs (RAMAKRISHNAN; GEHRKE, 2002, p. 456).

Para superar essas limitações, novas tecnologias foram desenvolvidas, entre as quais se destaca o NoSQLs. Esse termo engloba uma variedade de soluções de armazenamento de dados criadas para atender às necessidades de sistemas em que os bancos de dados tradicionais, baseados no modelo relacional, se mostram ineficazes. As características e benefícios dessas tecnologias serão abordados na próxima seção.

2.2 Modelos não relacionais

O termo NoSQL (abreviação de "Not Only SQL", ou "Não Apenas SQL"em português) foi introduzido por Carlo Strozzi em 1998 (BARBOSA, 2013). Esse modelo foi proposto para atender à crescente demanda de dados e informações das aplicações web, que continuam a aumentar exponencialmente. O NoSQL surgiu com o objetivo de gerenciar grandes volumes de dados semi-estruturados ou não estruturados, que exigem alta disponibilidade e escalabilidade (LÓSCIO; OLIVEIRA; PONTES, 2011, p. 1). Com o avanço da web, o volume de dados gerados pelos usuários cresceu de forma exponencial, aumentando significativamente a demanda por soluções eficientes de armazenamento e processamento. Essa necessidade impulsionou o desenvolvimento de novas tecnologias, que possibilitam uma manipulação mais eficaz dos dados, além de facilitar o compartilhamento de informações no ambiente online. Um exemplo que ilustra o surgimento dessas

novas tecnologias é a rede social X (antiga Twitter). Nessa plataforma, há um fluxo massivo de dados, com milhões de usuários publicando constantemente textos, imagens, vídeos e interagindo por meio de funcionalidades como chats e enquetes. A vasta quantidade de informações geradas em tempo real torna a X um ambiente que exige soluções eficientes para o armazenamento e processamento de grandes volumes de dados, garantindo assim a alta disponibilidade e o bom funcionamento da plataforma.

Em anos mais recentes, diversas empresas de tecnologia têm desenvolvido suas próprias soluções NoSQLs para lidar com a crescente demanda por escalabilidade e desempenho no processamento de grandes volumes de dados. Em 2012, o Facebook lançou o RocksDB, um banco de dados embutido, otimizado para armazenamento rápido em dispositivos SSD, projetado para processar grandes volumes de dados e ser eficiente em leituras e gravações sequenciais (Meta Platforms, Inc., 2012). Da mesma forma, a Amazon aprimorou seu serviço NoSQL com o Amazon DynamoDB, lançado em 2012, oferecendo uma solução completamente gerenciada com escalabilidade automática, armazenamento distribuído e alta disponibilidade para uma ampla gama de aplicações na nuvem (VOGELS, 2012).

Além disso, o Google continuou a inovar, lançando o Firestore em 2017, uma base de dados NoSQL que facilita o desenvolvimento de aplicações escaláveis e em tempo real, integrando-se com serviços na nuvem e permitindo sincronização em múltiplos dispositivos (Google LLC, 2017). A Microsoft, por sua vez, apresentou o Azure Cosmos DB, um banco de dados NoSQL multi-modelo que oferece suporte a diversos tipos de APIs (como MongoDB, Cassandra e Gremlin), permitindo escalabilidade global, latência baixa e garantia de consistência configurável (Microsoft Corporation, 2017).

Essas novas soluções NoSQL foram desenvolvidas para atender às necessidades de plataformas com grande fluxo de dados, como redes sociais e aplicações móveis, oferecendo alta escalabilidade, suporte a replicação, particionamento de dados e alta performance em consultas distribuídas, sem sacrificar a consistência e a disponibilidade.

2.2.1 Características dos bancos de dados NoSQL

Os bancos de dados NoSQLs possuem características distintas em relação aos modelos de banco de dados relacionais, o que os torna apropriados para o armazenamento de grandes quantidades de dados, tanto estruturados quanto não estruturados. A seguir, serão apresentadas algumas dessas características.

2.2.2 Esquema flexível ou ausência de esquema

Segundo (LÓSCIO; OLIVEIRA; PONTES, 2011), os bancos de dados NoSQL também caracterizam-se por permitir uma modelagem mais flexível dos dados, sem a necessidade de um esquema rígido previamente definido. Essa característica possibilita adaptar a estrutura armazenada às necessidades da aplicação, favorecendo a evolução dos dados

ao longo do tempo e o tratamento de informações heterogêneas. Por outro lado, essa flexibilidade reduz os mecanismos de validação estrutural presentes nos bancos de dados relacionais, nos quais a definição prévia do esquema contribui para garantir maior consistência e integridade dos dados.

2.2.3 Escalabilidade horizontal

Com o aumento do volume de dados, a necessidade de escalabilidade e melhorias de desempenho torna-se fundamental. A escalabilidade horizontal é uma abordagem eficaz para lidar com esse desafio, permitindo a adição de mais máquinas ao sistema para armazenamento e processamento de dados. Ao contrário dos bancos de dados relacionais, nos quais a alta concorrência pode gerar contenção associada a mecanismos de bloqueio, muitos bancos de dados NoSQL utilizam arquiteturas e estratégias de distribuição que reduzem esse impacto, favorecendo a escalabilidade horizontal. Essa característica os torna particularmente adequados para gerenciar grandes volumes de dados que crescem exponencialmente, como os gerados pela Web 2.0 (LÓSCIO; OLIVEIRA; PONTES, 2011, p. 5). Para exemplificar, a Figura 1 ilustra o aumento no número de máquinas, uma solução conhecida como escalonamento horizontal. Essa abordagem é amplamente utilizada e disponibilizada em diversos serviços de nuvem, como Amazon Web Services, Google Cloud e Azure.

Figura 1 – Escalonamento horizontal

Fonte: https://inverodigital.com/

2.2.4 Suporte nativo a replicação

Segundo (LÓSCIO; OLIVEIRA; PONTES, 2011, p. 5) uma das formas de proporcionar escalabilidade em bancos de dados é através da replicação nativa, que reduz o tempo necessário para recuperar informações. Existem duas abordagens principais para a replicação: a Master-Slave (Mestre-Escravo) e a Multi-Master. Na arquitetura Master-Slave, todas as operações de escrita são realizadas no nó mestre, que, em seguida, replica essas escritas em N nós escravos, onde N representa o número de escravos. Embora essa abordagem torne as leituras mais rápidas, ela pode criar um gargalo em termos de capacidade

de escrita, o que a torna menos adequada para sistemas que lidam com grandes volumes de dados. Por outro lado, na arquitetura Multi-Master, vários nós podem atuar como mestres, permitindo a distribuição das operações de escrita e, consequentemente, diminuindo o gargalo. No entanto, essa configuração pode levar a conflitos de dados devido à concorrência nas operações de escrita.

2.2.5 API simples para acesso aos dados

O objetivo das soluções NoSQL é fornecer um acesso eficiente aos dados, priorizando alta disponibilidade e escalabilidade. Segundo Lóscio et al. (LÓSCIO; OLIVEIRA; PONTES, 2011), o foco dessas soluções não está em como os dados são armazenados, mas sim em como recuperá-los de maneira rápida e eficaz. Para alcançar esse objetivo, é fundamental que APIs simples sejam desenvolvidas, facilitando o acesso a essas informações e permitindo que qualquer aplicação interaja com os dados do banco de forma ágil e eficiente.

2.3 Modelos de Dados NoSQL

Nesta seção, serão descritos alguns dos principais modelos de dados NoSQL, focando em destacar suas diferenças, qualidades e defeitos. Os modelos que serão apresentados são: orientado a documentos, chave-valor, orientado a colunas e orientados a grafos. De acordo com (DATE, 2003, p. 37-40) um modelo de dados pode ser descrito como uma abstração formal utilizada para organizar e estruturar os dados de um sistema de informações. Ele define tanto a maneira como os dados serão armazenados, quanto as relações e restrições impostas sobre eles, garantindo que possam ser acessados e manipulados de maneira eficiente e consistente. Um bom modelo de dados serve como base para a implementação de bancos de dados, proporcionando uma estrutura lógica que representa entidades, seus atributos e os relacionamentos entre elas. Assim, os modelos de dados são fundamentais para manter a integridade e a coerência do sistema de gerenciamento de banco de dados, oferecendo um padrão para a construção de sistemas complexos.

2.3.1 Orientados a documentos

Os sistemas orientados a documentos são uma categoria de bancos de dados NoSQL que armazenam e gerenciam dados em formatos estruturados, como JSON, XML ou BSON. Esses sistemas oferecem aos desenvolvedores uma estrutura mais flexível e dinâmica em comparação com os bancos de dados relacionais tradicionais. Cada documento pode conter diferentes tipos e quantidades de informações, permitindo uma evolução contínua do esquema conforme os requisitos mudam. Isso resulta em alta escalabilidade, pois novos tipos de dados podem ser adicionados sem grandes alterações na estrutura

existente. Além disso, esses sistemas suportam nativamente operações complexas, como a indexação de campos dentro dos documentos, facilitando consultas rápidas e eficientes. Essa flexibilidade e eficiência tornam os sistemas orientados a documentos uma escolha popular para aplicações que exigem alta velocidade de desenvolvimento e a capacidade de lidar com grandes volumes de dados não estruturados. Entre as aplicações que utilizam o modelo orientado a documentos, destacam-se o MongoDB que é escrito em C++ e utiliza BSON como seu formato de armazenamento interno, com suporte para JSON na interface de usuário e na comunicação de dados, proporcionando flexibilidade e eficiência para aplicações que exigem um gerenciamento dinâmico de dados. Também é possível citar a ELK Stack a qual é composta por ferramentas desenvolvidas em Java, Ruby e JavaScript, e aceita diversos formatos de dados, com foco principal em JSON.

É possível visualizar na Figura 2, um exemplo de documento que representa um registro de um livro em um sistema de registros de livros, no qual os campos foram definidios como: titulo, autor, ano de publicacao, genero e disponibilidade. Desta forma para cada campo valores associados.

Figura 2 – Exemplo de documento

Um caso de uso ideal para bancos de dados orientados a documentos é em aplicações que gerenciam conteúdos dinâmicos, como sistemas de gerenciamento de conteúdo (CMS) ou e-commerce. Nesse cenário, cada produto ou página pode ter diferentes atributos (tamanhos, cores, descrições, etc.), e os dados são frequentemente semiestruturados. Bancos orientados a documentos, como MongoDB, são perfeitos para armazenar esses dados, já que permitem flexibilidade no formato dos documentos (geralmente em JSON), onde cada documento pode ter uma estrutura própria. Isso facilita a modelagem de dados não estruturados ou com esquemas dinâmicos, permitindo maior agilidade no desenvolvimento e alterações frequentes nos dados, sem a necessidade de remodelar todo o banco de dados.

2.3.2 Chave-valor

De acordo com (LÓSCIO; OLIVEIRA; PONTES, 2011) este pode ser considerado o modelo mais simples, por conta de permitir a visualização do banco de dados como uma

grande tabela hash. Assim sendo, o banco de dados consiste em um conjunto de chaves as quais estão ligadas a apenas um valor, como é possível visualizar na Figura 3. Como o modelo de armazenamento chave-valor não é muito exigente, os bancos de dados que utilizam esse modelo podem ter alto desempenho em diversas situações, porém costumam não ser eficaz em situações que haja a necessidade de fazer consultas e agregações mais complicadas. O armazenamento do tipo chave-valor é projetado para ter escalabilidade horizontal e alta velocidade, porém em contextos em que manter os indíces é pouco necessário ou não é necessário. Porém, o modelo chave-valor também possuí desvantagens, pois ele não é útil em situações em que é necessário realizar consultas complexas sobre os dados, o modelo acaba por não ter muita habilidade com índices. Afinal, ele só realiza operações básicas de leitura e escrita (REDMOND; WILSON; CARTER, 2012). Alguns sistemas de gerenciamento de banco de dados (SGBDs) que utilizam o modelo de armazenamento chave-valor incluem, por exemplo, Voldemort, Redis e Riak.

Um exemplo clássico de aplicação que utiliza o modelo de chave-valor é um carrinho de compras em uma loja online. Cada usuário que acessa a loja pode adicionar itens ao seu carrinho de compras. Nesse caso, a chave pode ser o identificador único do usuário (por exemplo, um ID de sessão), e o valor seria a lista de itens que o usuário adicionou ao carrinho. Isso permite que cada usuário tenha um carrinho de compras independente, sem interferir nos carrinhos de outros usuários.

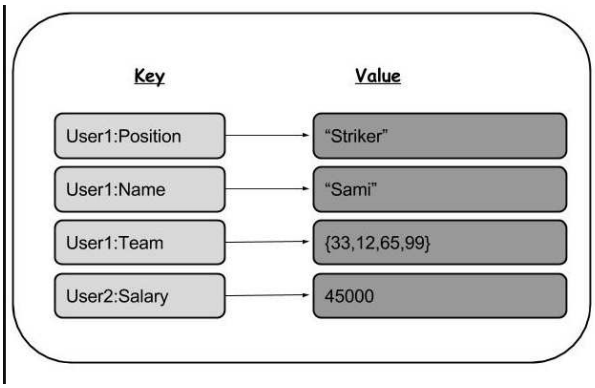

Figura 3 – Exemplo de chave-valor

2.3.3 Orientado a Coluna

O modelo orientado a coluna possuí esse nome, pois os dados de uma determinada coluna são armazenados juntos. Por outro lado, um modelo orientado a linha, assim como um banco de dados relacional, guarda as informações relacionadas a uma tupla juntas. A princípio essa diferença pode parecer não ser relevante, porém o impacto dessa diferença é considerável. Nos bancos que são orientados a coluna, a adição de colunas tem um baixo custo computacional e é feita linha por linha. As linhas podem ter um conjunto diferente de colunas, como é possível visualizar na Figura 4, ou até mesmo nenhuma, permitindo

assim que as tabelas possam permanecer vazias, sem gerar custos no armazenamento dos valores nulos (REDMOND; WILSON; CARTER, 2012). Esse modelo apresenta algumas vantagens, como o fato de ter sido desenvolvido para oferecer escalabilidade horizontal. Dessa forma, torna-se adequado para situações envolvendo problemas de *Big Data*, utilizando grupos com vários nós. Além disso, geralmente oferece suporte para recursos de compressão, que ajudam a reduzir o tamanho dos dados armazenados, e controle de versão. Apesar disso, esse modelo também apresenta desvantagens, como a necessidade de saber, antes de projetar o banco, a forma como os dados serão utilizados, e não apenas como serão estruturados. Se os padrões de uso não puderem ser definidos antes do início do projeto, o modelo pode acabar não sendo a melhor escolha. Alguns sistemas de gerenciamento de banco de dados (SGBDs) que utilizam o modelo de armazenamento orientado a coluna são: Apache Cassandra, HBase e o Amazon SimpleDB.

Um caso de uso típico para bancos de dados orientados a colunas é em aplicações de análise de grandes volumes de dados (Big Data), como em sistemas de Business Intelligence (BI). Nesse cenário, a empresa precisa processar e consultar grandes conjuntos de dados de forma eficiente para gerar relatórios e análises em tempo real. Bancos orientados a colunas são ideais para consultas analíticas que envolvem agregações de dados em grandes tabelas, já que armazenam os dados por coluna, permitindo a leitura seletiva de colunas específicas e otimizando operações de leitura e processamento em larga escala, ao contrário dos bancos de dados orientados a linhas, que processam todos os dados da linha.

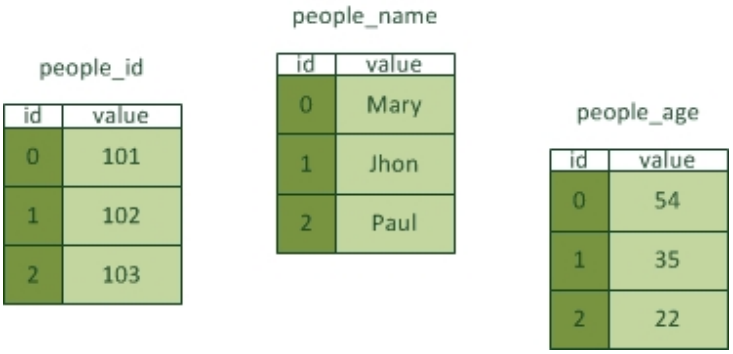

Figura 4 – Sistema orientado a coluna

Fonte:
https://isaiasbarroso.wordpress.com/2012/06/20/banco-de-dados-orientado-a-colunas/

2.3.4 Orientado a Grafos

O modelo orientado a grafos diverge dos demais modelos de banco de dados em diversos aspectos, sendo composto por três elementos básicos: os nós, que representam as entidades armazenadas; os relacionamentos, que correspondem às conexões entre os nós; e as propriedades, que representam os atributos associados aos nós e aos relacionamentos

(Neo4j, 2025). Segundo a documentação do Neo4j (Neo4j, 2025), essa estrutura favorece a representação de dados altamente conectados e possibilita a realização eficiente de consultas baseadas nos relacionamentos existentes entre as entidades. A Figura 5 apresenta um exemplo dessa estrutura.

Uma das principais vantagens desse modelo está na capacidade de representar e percorrer relacionamentos complexos de forma eficiente, permitindo a execução de consultas que envolvem múltiplas conexões entre os dados. Dessa forma, bancos de dados orientados a grafos são amplamente utilizados em aplicações como redes sociais, sistemas de recomendação, detecção de fraudes e mecanismos de análise de redes. Como desvantagem, a utilização desse modelo pode não ser a alternativa mais adequada para aplicações cujos dados apresentam poucas relações entre si, uma vez que seus benefícios são mais evidentes em cenários com elevado grau de conectividade entre as entidades.

Alguns exemplos de bancos de dados que utilizam o modelo orientado a grafos são o Amazon Neptune, o ArangoDB e o Neo4j. Um caso de uso clássico para bancos de dados orientados a grafos é em sistemas de recomendação em redes sociais. Nesse cenário, a rede social precisa recomendar novos amigos, grupos ou conteúdos aos usuários com base nas conexões já existentes, interações passadas (como curtidas, comentários) e interesses compartilhados. O banco de grafos é ideal para modelar esses relacionamentos complexos, pois permite navegar eficientemente pelas conexões entre usuários e seus interesses, otimizando o processo de recomendação de maneira muito mais eficiente do que bancos relacionais, que exigiriam joins complexos para realizar as mesmas consultas.

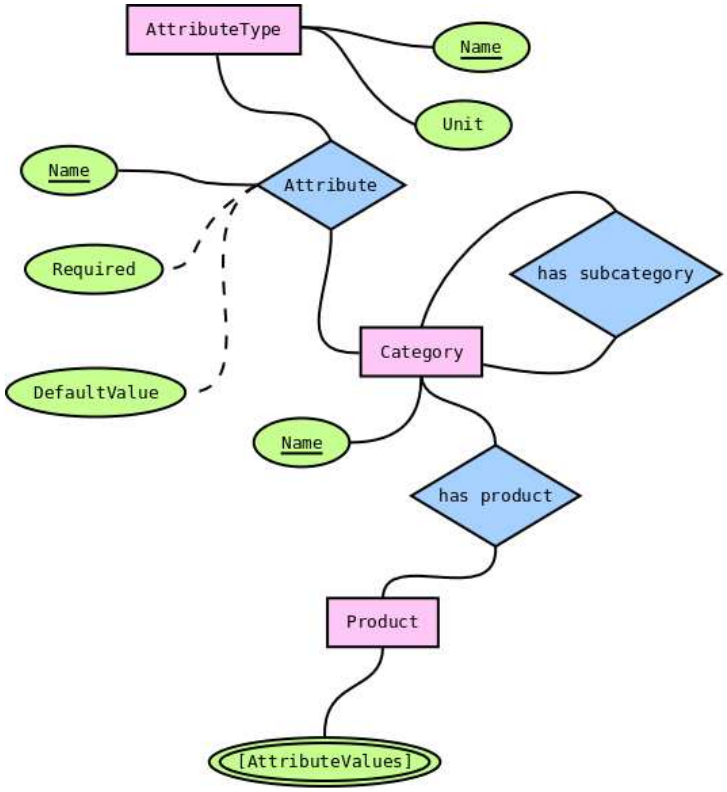

Figura 5 – Sistema orientado a grafos

Fonte:
https://mvjufs.blogspot.com/2014/12/banco-de-dados-nosql-orientado-grafos.html

2.4 Exemplos práticos de sistemas de gerenciamento de banco de dados NoSQL

Nesta Seção serão apresentados alguns exemplos de sistemas de gerenciamento de bancos de dados NoSQL e casos de uso reais, bem como os benefícios de seu uso.

2.4.1 Chave-Valor: Redis

Caso de uso: Redis é amplamente utilizado pela Twitter como um cache para melhorar o desempenho de suas aplicações. O Twitter armazena dados de sessões e resultados de consultas frequentes no Redis, acelerando o tempo de resposta para os milhões de usuários ativos da plataforma.

Benefício: O uso de Redis reduziu drasticamente o tempo de resposta em operações de leitura, permitindo que o Twitter fornecesse uma experiência de usuário mais rápida e estável, mesmo durante picos de tráfego.

2.4.2 Orientado a Colunas: Apache Cassandra

Caso de uso: A Netflix utiliza o Cassandra para armazenar dados de visualização de milhões de usuários em todo o mundo. O Cassandra é usado para processar e analisar grandes volumes de dados em tempo real, fornecendo insights que alimentam os sistemas de recomendação da plataforma.

Benefício: Cassandra oferece alta escalabilidade e capacidade de gerenciar grandes quantidades de dados distribuídos geograficamente, o que é essencial para a Netflix manter sua operação global.

2.4.3 Orientado a Documentos: MongoDB

Caso de uso: A Forbes utiliza o MongoDB como parte de seu sistema de gerenciamento de conteúdo para armazenar e servir artigos, blogs e conteúdo dinâmico para milhões de leitores. A flexibilidade no armazenamento de documentos em formato JSON foi fundamental para a Forbes escalar seu sistema rapidamente.

Benefício: A flexibilidade do MongoDB permite que a Forbes adapte rapidamente seu sistema às mudanças no conteúdo, melhorando o tempo de lançamento de novas funcionalidades e a experiência de seus leitores.

2.4.4 Orientado a Grafos: Neo4j

Caso de uso:O eBay utiliza Neo4j para detecção de fraudes em tempo real, analisando as conexões entre transações, vendedores e compradores. O banco de dados de grafos ajuda a identificar padrões de comportamento fraudulento mais rapidamente do que seria possível com bancos de dados relacionais.

Benefício: O uso de Neo4j permitiu ao eBay detectar fraudes de maneira mais eficiente, melhorando a segurança e reduzindo as perdas financeiras na plataforma.

2.5 Trabalhos relacionados

Existem diversos aspectos que devem ser analisados ao mensurar o desempenho de um Banco de Dados (BD), especialmente ao comparar as abordagens entre sistemas SQL e NoSQL. O rápido avanço da web 2.0, aliado ao aumento exponencial de dados gerados, trouxe novos desafios para o gerenciamento eficiente dessas informações. Nesse contexto, os métodos tradicionais de bancos de dados relacionais, que outrora eram a solução dominante, começaram a se mostrar limitados diante de demandas crescentes de escalabilidade, flexibilidade e volume de dados.

Um trabalho relevante nesse contexto é o de (BRITO, 2013), que realizou uma análise comparativa conceitual entre SGBDs relacionais e bancos de dados NoSQL, com foco nos

critérios de escalabilidade, consistência e disponibilidade. O autor conclui que o NoSQL apresenta vantagens significativas em escalabilidade horizontal, por ter sido projetado nativamente para distribuição de dados entre múltiplos servidores, enquanto o modelo relacional se destaca em consistência transacional por meio das propriedades ACID. Esse estudo contribui para a compreensão das diferenças arquiteturais entre os paradigmas, porém se limita a uma análise conceitual, sem mensuração empírica de desempenho.

Avançando para estudos com experimentação empírica, (LI; MANOHARAN, 2013) conduziram uma das primeiras comparações sistemáticas de desempenho entre bancos SQL e NoSQL em operações de leitura, escrita e exclusão. Os autores demonstraram que nem todo banco NoSQL apresenta desempenho superior ao SQL em todas as operações, evidenciando que a vantagem de cada paradigma depende fortemente do tipo de operação e da estrutura dos dados analisados. Esse resultado antecipa o padrão observado no presente trabalho, no qual o MongoDB demonstrou superioridade em operações de escrita, enquanto o PostgreSQL apresentou vantagem em consultas analíticas sobre dados tabulares.

No âmbito específico das comparações entre bancos orientados a documentos e bancos relacionais, (GYőRöDI et al., 2015) compararam o desempenho do MongoDB e do MySQL em operações de inserção, seleção e agregação, por meio de uma aplicação web. Os autores concluíram que o MongoDB apresentou velocidade de inserção significativamente superior, sendo até dez vezes mais rápido em cenários de grande volume, enquanto o MySQL mostrou-se mais eficiente em consultas complexas que envolvem junções entre tabelas. Esses resultados são consistentes com os obtidos no presente trabalho, reforçando que a vantagem de cada paradigma está condicionada ao tipo de operação e à estrutura dos dados.

Mais recentemente, (CARVALHO; Sá; BERNARDINO, 2023) avaliaram o desempenho de três bancos de dados NoSQL orientados a documentos, sendo eles Couchbase, CouchDB e MongoDB, utilizando o *Yahoo! Cloud Serving Benchmark* (YCSB), com conjuntos de dados de até 10.000.000 de registros. Os resultados indicaram que o MongoDB apresentou o melhor tempo de execução na maioria dos cenários, com exceção das operações de varredura completa, reforçando que o desempenho do banco orientado a documentos varia conforme o tipo de carga de trabalho. Esse comportamento é consistente com os resultados do presente trabalho, em que o MongoDB demonstrou eficiência superior em operações de escrita e em leituras sobre dados semiestruturados, mas desempenho inferior ao PostgreSQL em varreduras analíticas sobre dados tabulares homogêneos.

Esses estudos, em conjunto, evidenciam que a escolha entre bancos de dados SQL e NoSQL está diretamente relacionada às características dos dados e aos padrões de acesso da aplicação. Enquanto os bancos relacionais oferecem estrutura consolidada para consultas analíticas complexas e consistência transacional, os bancos NoSQL se destacam em escalabilidade horizontal e cargas intensas de escrita. O presente trabalho contribui

com evidências empíricas que complementam essa discussão, comparando especificamente PostgreSQL e MongoDB em operações CRUD sobre dois conjuntos de dados com características distintas, um tabular e outro semiestruturado com relações entre entidades.

Capítulo 3

Metodologia

O desempenho de um sistema, no contexto deste trabalho, é operacionalizado como o tempo de execução necessário para concluir uma operação de manipulação de dados (inserção, consulta, atualização ou exclusão), medido a partir da camada de aplicação Java e expresso por meio da média e do desvio padrão de cinco execuções sucessivas de cada operação, conforme detalhado adiante neste capítulo.

Esse tempo é influenciado por múltiplos fatores, entre os quais se destacam o hardware utilizado, a linguagem de programação, o algoritmo empregado na implementação de cada operação e o sistema de gerenciamento de banco de dados (SGBD) adotado. O algoritmo impacta o desempenho ao determinar a forma como os dados são processados e enviados ao banco: a estratégia de inserção em lote (*batch insert*), por exemplo, reduz o número de requisições ao SGBD em comparação a inserções individuais, influenciando diretamente o tempo total de execução. Já o SGBD impacta o desempenho por meio das características de seu modelo de dados e mecanismo de armazenamento: bancos de dados relacionais, como o PostgreSQL, organizam os dados em tabelas normalizadas, de modo que operações envolvendo relações entre elas exigem operações de junção (*joins*) para combinar registros, o que pode demandar maior esforço computacional. Bancos de dados orientados a documentos, como o MongoDB, armazenam informações relacionadas dentro de um mesmo documento, o que reduz a necessidade de junções, mas introduz outros fatores relevantes de desempenho, como o tamanho médio dos documentos e a forma como os índices são utilizados nas consultas.

Neste trabalho, foram criados dois cenários controlados utilizando *datasets* distintos. O primeiro é um *dataset* de e-commerce, composto por 21 colunas e aproximadamente 600.000 registros obtido na plataforma kaggle (ZUSMANI, 2017). A partir desse conjunto de dados, foram realizadas modelagens para dois sistemas de gerenciamento de banco de dados: PostgreSQL, como representante do modelo relacional, e MongoDB, como representante do modelo NoSQL. No PostgreSQL, o *dataset* foi estruturado em uma única tabela, na qual todos os registros foram armazenados sem a existência de relações entre outras tabelas. Já no MongoDB, os mesmos dados serão organizados em documentos

dentro de uma coleção, onde cada registro armazenará seus atributos em formato BSON.

O segundo cenário utiliza um *dataset* de imóveis do Airbnb para a cidade de Seattle, disponível na plataforma Kaggle (SWSW1717, 2021), composto por dois arquivos CSV: o primeiro contendo os registros dos imóveis disponíveis para locação e o segundo reunindo os reviews dos usuários que alugaram esses imóveis. Esse conjunto de dados apresenta uma relação de 1:N, em que um imóvel pode possuir diversos reviews. No PostgreSQL, essa estrutura foi modelada como uma relação 1:N, enquanto no MongoDB foi criada uma coleção na qual cada documento representa um imóvel e contém um array de reviews associado. Esse *dataset* possui aproximadamente 450.000 registros.

Foram realizados testes de inserção, consulta, atualização e exclusão de dados em ambos os bancos de dados. Cada um desses procedimentos foi repetido cinco vezes, registrando-se o tempo de execução de cada repetição para o cálculo da média e do desvio padrão. Essa repetição tem como objetivo reduzir a influência de fatores externos e transitórios sobre a medição, como o tempo de inicialização (*warm-up*) da máquina virtual Java, o estado do cache do sistema operacional e do banco de dados, e a eventual execução de processos em segundo plano, permitindo obter uma estimativa mais estável do tempo de execução de cada operação. O objetivo desses testes é comparar o desempenho de cada operação em PostgreSQL e MongoDB, buscando identificar qual abordagem oferece maior eficiência para o tipo de dado e volume propostos. A escolha desses modelos se baseia em sua adequação às características do *dataset* e do caso de uso, permitindo uma análise prática e relevante sobre as diferenças de desempenho entre SQL e NoSQL.

3.1 Ambiente de testes

O ambiente de testes utilizado para comparar os tempos de execução das instruções SQL com os comandos dos bancos NoSQL foi composto por um computador configurado conforme as especificações apresentadas na Tabela 1. Já a Tabela 2 apresenta as especificações dos softwares utilizados.

Especificações	**Descrição**
Processador	AMD Ryzen 5 5600, 3.5GHz (4.4GHz Turbo), 6-Cores 12-Threads
Memória RAM	32GB DDR4 3200 MT/s
SSD	512GB NVME M.2
Sistema operacional	Windows 11 Pro 64 bits

Tabela 1 – Especificações do ambiente de teste.

Ambos os sistemas gerenciadores de banco de dados foram utilizados em configuração padrão de instalação (*standalone*), operando em um único nó, sem a aplicação de estratégias de distribuição, como replicação, particionamento (*sharding*) ou balanceamento de

Software	**Versão**
PostgreSQL	17.5
MongoDB	8.0.9
Java	21

Tabela 2 – Softwares utilizados e suas respectivas versões

carga entre múltiplos nós. Da mesma forma, não foram adotadas estratégias de paralelismo customizadas na execução das operações, além do agrupamento de registros em lotes de 500 unidades (*batch insert*) utilizado nas operações de inserção, descrito na Seção 3. Essa escolha teve como objetivo isolar o efeito do modelo de dados, relacional ou orientado a documentos, sobre o desempenho medido, evitando que fatores adicionais relacionados a infraestrutura distribuída ou processamento paralelo interferissem na comparação.

Todos os experimentos, para ambos os sistemas, foram executados na mesma máquina, cujas especificações são apresentadas na Tabela 2. Essa decisão garante que as condições de hardware fossem idênticas entre os testes de PostgreSQL e MongoDB, eliminando variações decorrentes de diferenças de infraestrutura. Por outro lado, essa escolha também constitui uma limitação do estudo: os resultados obtidos refletem o comportamento relativo dos dois sistemas especificamente nesse ambiente computacional, podendo não se generalizar diretamente para configurações distintas de hardware, ambientes distribuídos ou infraestruturas em nuvem.

Além disso, o tempo de execução, embora seja uma métrica relevante para avaliar desempenho, está sujeito à influência de fatores externos ao modelo de dados propriamente dito. Entre eles, destacam-se o escalonamento de processos, o chaveamento de contexto e o gerenciamento de memória realizados pelo sistema operacional, a criação e sincronização de *threads* pela aplicação Java, e estratégias internas dos próprios SGBDs, como gerenciamento de *buffers* e *cache*, política de escrita em disco (*write through* ou *write back*) e mecanismos de concorrência. Dessa forma, as diferenças de tempo observadas entre PostgreSQL e MongoDB nem sempre refletem exclusivamente as características do modelo relacional ou orientado a documentos, podendo também ser parcialmente influenciadas por esses fatores. A repetição de cada operação cinco vezes, com cálculo de média e desvio padrão, descrita na seção seguinte, busca atenuar, ainda que não eliminar completamente, esse tipo de variação.

3.2 Cenários de comparação

Os cenários utilizados neste estudo foram elaborados a partir de duas bases de dados: uma de e-commerce e outra de alocação de imóveis do Airbnb na cidade de Seattle, ambas disponíveis publicamente na plataforma Kaggle [1], que disponibiliza diversos conjuntos de

[1] <https://www.kaggle.com>

dados de diferentes tipos e finalidades.

O *dataset* de e-commerce contém informações relacionadas a transações de compra, incluindo itens adquiridos, datas das transações, métodos de pagamento, percentuais de desconto, entre outros atributos relevantes.

Por sua vez, o *dataset* do Airbnb apresenta duas tabelas distintas: a primeira reúne dados sobre imóveis disponíveis para locação, como valor do aluguel, localização (rua), tipo de quarto e data da última avaliação; já a segunda tabela contém os reviews dos usuários, com informações como data da avaliação, imóvel alugado, comentário e nome do avaliador.

Esses dois cenários possibilitam uma análise prática de como diferentes sistemas de gerenciamento de banco de dados lidam com operações essenciais em contextos de grande volume de transações. Além disso, permitem observar o comportamento dos modelos de dados em situações contrastantes: um *dataset* tabular sem relacionamentos explícitos entre entidades, em que todos os registros estão contidos em uma única estrutura, e outro com relações entre entidades, representando a associação entre imóveis e seus respectivos reviews.

3.3 Arquitetura do sistema

Os dados que alimentaram os bancos de dados foram baseados em duas bases de dados. Para realizar as operações e medir o tempo de execução de cada uma, foi desenvolvida uma aplicação dedicada aos testes[2]. Essa aplicação foi implementada em Java, linguagem escolhida por sua versatilidade e ampla compatibilidade com os dois modelos de banco de dados utilizados: PostgreSQL e MongoDB. A IDE empregada para o desenvolvimento foi o IntelliJ IDEA, por oferecer um ambiente robusto, com recursos avançados de depuração e integração facilitada com múltiplos bancos de dados, o que contribui para uma melhor gestão e execução dos experimentos.

O tempo de execução medido para cada operação inclui a camada de aplicação desenvolvida em Java, uma vez que os testes não foram realizados diretamente nos bancos de dados, mas sim por meio da aplicação, a fim de simular cenários reais de uso. Essa abordagem garante que os tempos obtidos reflitam de forma mais fiel o desempenho da aplicação ao interagir com os bancos de dados, oferecendo uma visão mais próxima do comportamento em ambiente real.

O processo de carga dos dados seguiu o fluxo apresentado na Figura 6, no qual a aplicação Java realiza a leitura das informações contidas nos arquivos CSV e, em seguida, insere esses dados no banco de dados correspondente (PostgreSQL ou MongoDB).

[2] O código-fonte completo da aplicação está disponível em: <https://github.com/guigutox/TCC>

Figura 6 – Diagrama de inserção

Já para as operações de busca, o fluxo ocorre conforme ilustrado na Figura 7: a aplicação Java envia ao banco de dados uma requisição solicitando a execução da query e, na sequência, recebe o resultado retornado.

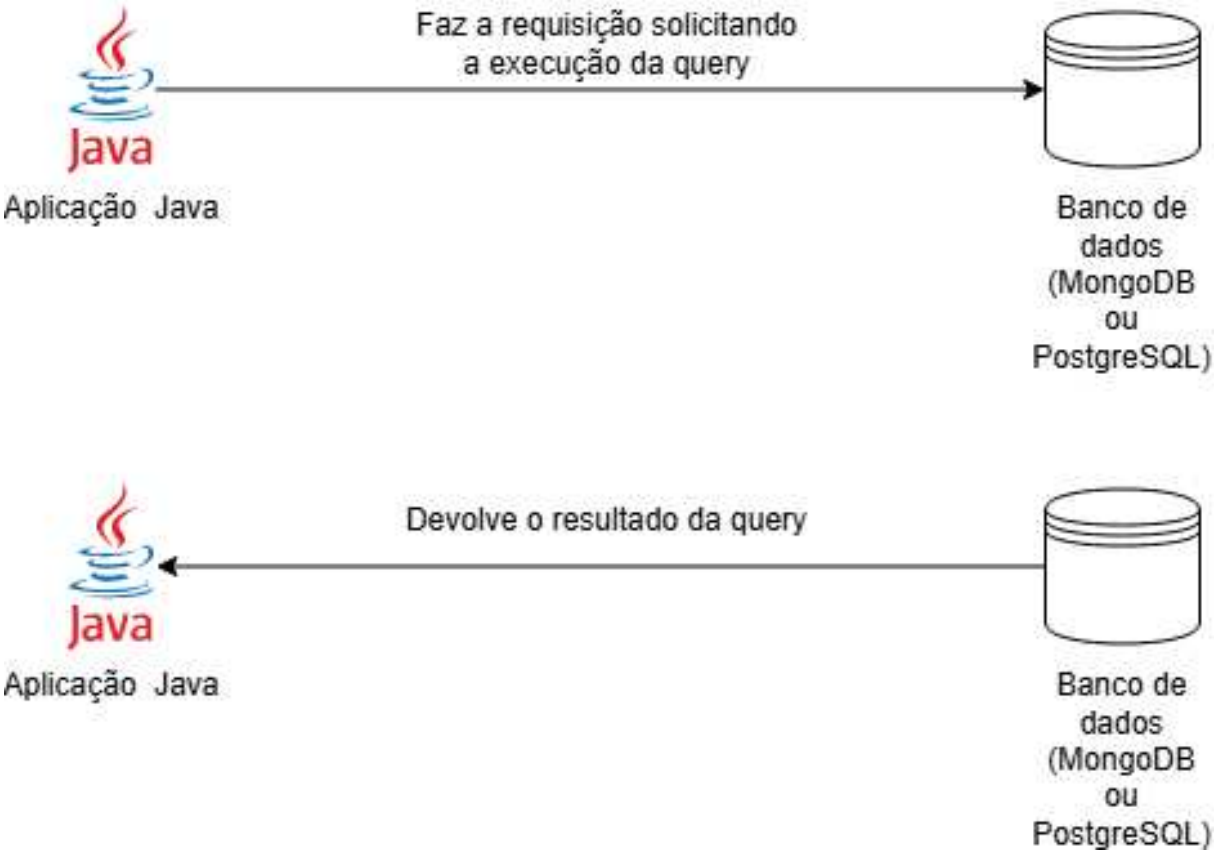

Figura 7 – Diagrama de busca

3.4 Modelagem

Uma vez obtidas as bases de dados, é necessário realizar a modelagem de acordo com o paradigma e bancos de dados que serão utilizados. A seguir , são detalhadas as duas modelagens exploradas nesse trabalho: relacional e orientada a documentos.

3.4.1 Modelo relacional

Nas Figuras 8 e 9, é possível visualizar a representação dos dados em formato de diagrama de modelo lógico. Esses diagramas ilustram como as informações estão organizadas para cada conjunto de dados. Na Figura 8, referente ao conjunto de dados do e-commerce do Paquistão, cada produto possui um identificador próprio (*ID*) que o diferencia de forma única, atuando como chave primária, além de outros atributos. Cada atributo possui um tipo de dado específico, que define o formato das informações armazenadas.

Já na Figura 9, correspondente ao conjunto de dados do Airbnb, observa-se uma relação entre as entidades *listings* e *reviews*. A entidade *listings* possui o *ID* como chave primária, identificando cada registro de forma única, enquanto a entidade *reviews* possui seu próprio *ID* e também o *ID* do *listing* ao qual está vinculada. Essa estrutura caracteriza uma relação do tipo 1:N entre *listings* e *reviews*, em que um *listing* pode ter vários *reviews* e cada *review* pertence a apenas um *listing*. Esse tipo de relacionamento é amplamente utilizado em bancos de dados relacionais, como o SQL.

Figura 8 – Modelo relacional do *dataset* Paquistão

3.4.2 Modelo Orientado a documentos

O paradigma NoSQL orientado a documentos oferece uma abordagem flexível de armazenamento de dados, uma vez que é livre de esquema. Isso significa que não é necessário definir previamente uma estrutura rígida para os dados, e que essa estrutura pode ser modificada dinamicamente em tempo de execução, conforme as necessidades da aplicação. De acordo com (MongoDB Inc., 2024), o MongoDB armazena dados dentro de documentos em um formato semelhante ao *JSON* (utilizando *BSON*, uma versão binária do *JSON*), estruturando-os em pares de chave/valor. Essa abordagem permite maior flexibilidade na organização e recuperação das informações.

Nas Figuras 10 e 11, é possível observar a representação dos dados no MongoDB. A Figura 10 apresenta a estrutura utilizada para o conjunto de dados do e-commerce do Paquistão, enquanto a Figura 11 mostra a organização dos dados referente ao conjunto do Airbnb.

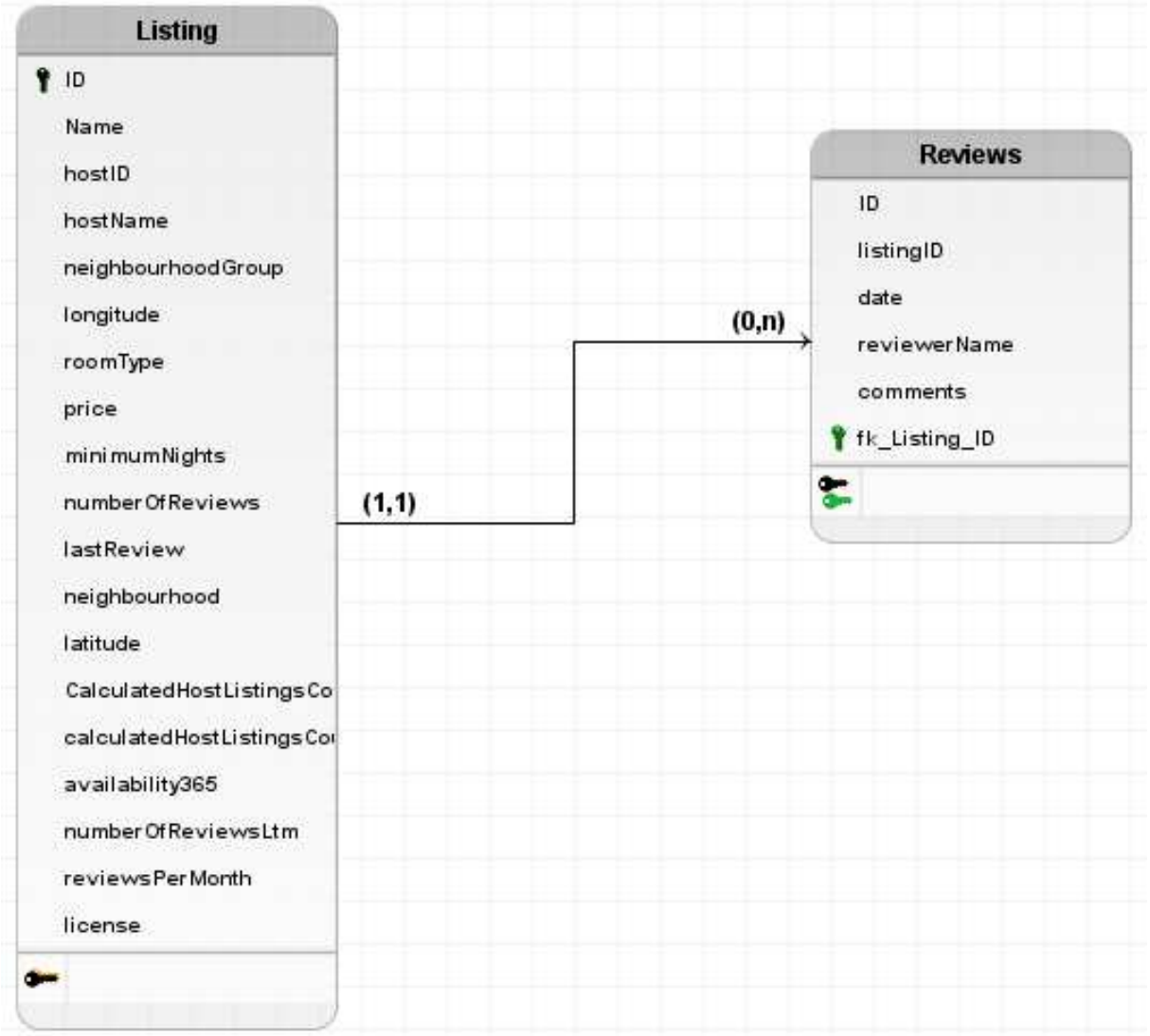

Figura 9 – Modelo relacional do *dataset* Airbnb

3.5 Operações

A seguir, as operações utilizadas para inserir dados tanto no mongo quanto no postgreSQL, bem como suas respectivas particularidades.

3.5.1 Insert - Implementação

A inserção de dados foi implementada tanto no MongoDB quanto no PostgreSQL utilizando uma estratégia de inserção em lote (batch insert), com o objetivo de reduzir a quantidade de requisições ao banco de dados e melhorar o desempenho das operações de escrita. Em ambos os casos, os dados são importados a partir de arquivos CSV contendo informações de *datasets* utilizados nos experimentos.

No MongoDB, a inserção foi implementada por meio da classe *MongoImporter*, res-

```
{
  "_id": {
    "$oid": "690658d57a6617269bfa3f00"
  },
  "item_id": 211133,
  "status": "canceled",
  "created_at": null,
  "sku": "kcc_Buy 2 Frey Air Freshener & Get 1 Kasual Body Spray Free",
  "price": 250,
  "qty_ordered": 1,
  "grand_total": 240,
  "increment_id": {
    "$numberLong": "100147444"
  },
  "category_name_1": "Beauty & Grooming",
  "sales_commission_code": null,
  "discount_amount": 0,
  "payment_method": "cod",
  "Working Date": "07/01/2016",
  "BI Status": "Gross",
  "MV": "240",
  "Year": "2016",
  "Month": "7",
  "Customer Since": "01/07/2016"
}
```


Figura 10 – Modelo NoSQL do *dataset* Paquistão

```
{
  "_id": {
    "$oid": "690647ce936a4649bf368a8d"
  },
  "id": "1008658967343673865",
  "name": "Sweet Suite w/Arcade Box, Gym, Dog Park & Parking!",
  "hostId": "161841146",
  "hostName": "Patrick And Ellen",
  "neighbourhoodGroup": "Rainier Valley",
  "neighbourhood": "Columbia City",
  "latitude": 47.56271345281905,
  "longitude": -122.27633851635596,
  "roomType": "Entire home/apt",
  "price": 215,
  "minimumNights": 2,
  "numberOfReviews": 7,
  "lastReview": "2024-05-31",
  "reviewsPerMonth": 0.97,
  "calculatedHostListingsCount": 1,
  "availability365": 16,
  "numberOfReviewsLtm": 7,
  "license": "STR-OPLI-19-000963",
  "reviews": [
    {
      "id": "1169022429358833123",
      "date": "2024-05-31",
      "reviewerId": "7374745",
      "reviewerName": "Shayna",
      "comments": "Atualizado para análise"
    }
  ]
}
```


Figura 11 – Modelo NoSQL do *dataset* Airbnb

ponsável por realizar a importação em lote de documentos. A estratégia adotada utiliza lotes de 500 documentos por operação, permitindo maior eficiência na escrita ao banco. Durante o processo de importação, o sistema realiza a conversão automática dos tipos de dados, garantindo que valores numéricos, datas e demais campos sejam armazenados corretamente no formato adequado do MongoDB. Além disso, foi implementado um tratamento para valores ausentes, prevenindo falhas na inserção e assegurando a integridade dos dados persistidos. Para os diferentes *datasets* utilizados nos experimentos, a

mesma lógica de inserção foi reaproveitada, variando apenas os campos mapeados para cada estrutura de documento.

No PostgreSQL, a inserção foi implementada por meio da classe *ListingRepository-Postgres*, responsável pela importação em lote dos dados utilizando *PreparedStatements*. Assim como no MongoDB, foram utilizados lotes de 500 registros por operação, reduzindo o número de interações com o banco de dados e melhorando o desempenho do processo de escrita. Durante a importação, o sistema realiza a conversão automática de diferentes tipos de dados, incluindo valores numéricos (como preços e quantidade de noites), coordenadas geográficas (latitude e longitude), datas (como última avaliação) e outros atributos presentes nos *datasets*. Para campos que podem possuir valores ausentes, especialmente datas, foi utilizado o método *setNull()*, evitando falhas durante a inserção e garantindo a consistência dos dados.

Para as diferentes entidades armazenadas no PostgreSQL, como *listings* e *reviews*, a mesma lógica de processamento em lote foi aplicada, variando apenas os campos mapeados e respeitando o relacionamento de chave estrangeira entre as tabelas. A estrutura do código foi organizada de forma modular, com métodos específicos para cada entidade, permitindo medições independentes de desempenho e facilitando futuras manutenções.

3.5.2 Read - Implementação

No caso do MongoDB, as consultas foram implementadas utilizando o *MongoDB Aggregation Framework*, permitindo a execução de operações analíticas diretamente sobre os documentos armazenados na coleção. Para isso, foi desenvolvida uma classe responsável por registrar e executar as consultas, possibilitando a execução sequencial das operações e a medição do tempo necessário para cada uma delas.

As consultas são definidas como funções que recebem a coleção do banco de dados e executam *pipelines* de agregação compostos por diferentes estágios, como *$match*, *$group*, *$sort*, *$project*, *$limit* e *$addFields*. Esses estágios permitem realizar filtragens, agrupamentos, cálculos agregados e transformações nos documentos durante o processamento da consulta. Em diversos casos, também são utilizadas expressões de conversão de tipos, como o operador *$convert*, permitindo transformar valores originalmente armazenados como texto em tipos numéricos apropriados para operações matemáticas.

Ao todo, foram implementadas 20 consultas analíticas sobre a base de dados de e-commerce. Essas consultas incluem análises como contagem de pedidos por status ou método de pagamento, identificação dos clientes com maior faturamento, produtos mais vendidos, categorias com maior receita, cálculo de ticket médio, análise de descontos, faturamento por período e métricas relacionadas ao comportamento de compra dos clientes. Algumas consultas também realizam transformações adicionais nos dados, como a normalização de campos com nomes distintos que representam o mesmo atributo e a conversão

de datas a partir de *strings* para possibilitar cálculos temporais, como a identificação do dia da semana em que os pedidos foram realizados.

No caso do PostgreSQL, as consultas foram implementadas utilizando a linguagem SQL padrão, executadas utilizando a API JDBC para comunicação com o banco de dados. Para isso, foi criada uma classe responsável por executar as consultas e medir o tempo de execução de cada operação, permitindo a comparação de desempenho entre os sistemas. As consultas são enviadas ao banco utilizando objetos do tipo *Statement*, e os resultados são processados por meio de um *ResultSet*. Durante a execução, o sistema registra o tempo total necessário para o processamento de cada consulta.

As mesmas consultas citadas anteriormente foram realizadas para PostgreSQL utilizando recursos da linguagem SQL, envolvendo operações como contagem de registros, agrupamento (*GROUP BY*), ordenação (*ORDER BY*), filtragem de dados (*WHERE*), aplicação de funções de agregação como *SUM*, *AVG* e *COUNT*, além de limitações de resultados com *LIMIT*. As consultas também exploram cálculos condicionais utilizando expressões *CASE*, bem como funções de manipulação de datas, como a extração do dia da semana a partir de campos do tipo data. Essas operações foram projetadas para representar cenários típicos de análise de dados em sistemas de comércio eletrônico, permitindo avaliar o desempenho do banco de dados relacional na execução de consultas analíticas sobre grandes volumes de dados.

Agregação

Conta o total de pedidos por status, mostrando quantos pedidos existem em cada situação.

Figura 12 – Consulta de agregação no MongoDB para contagem de pedidos por status, *dataset* Paquistão

Figura 13 – Consulta de agregação no PostgreSQL para contagem de pedidos por status, *dataset* Paquistão

Filtros condicionais

Agrupa pedidos por ano e mês para mostrar o volume mensal ao longo do tempo.

Figura 14 – Consulta com filtros e agrupamento temporal no MongoDB, *dataset* Paquistão

Figura 15 – Consulta com filtros e agrupamento temporal no PostgreSQL, *dataset* Paquistão

Ordenação

Define as cinco categorias com maior faturamento.

Figura 16 – Consulta com ordenação de categorias por faturamento no MongoDB, *dataset* Paquistão

Figura 17 – Consulta com ordenação de categorias por faturamento no PostgreSQL, *dataset* Paquistão

Operações numéricas

Calcula quanto a empresa faturou com pedidos pagos em dinheiro na entrega (*cash on delivery*) e quanto faturou com pedidos pagos por outros métodos.

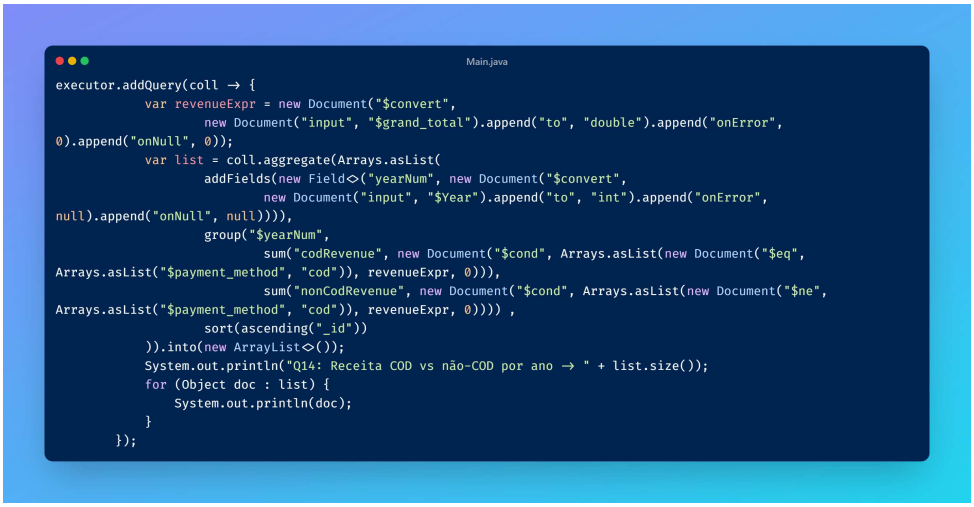

Figura 18 – Consulta com operações numéricas para cálculo de faturamento por método de pagamento no MongoDB, *dataset* Paquistão

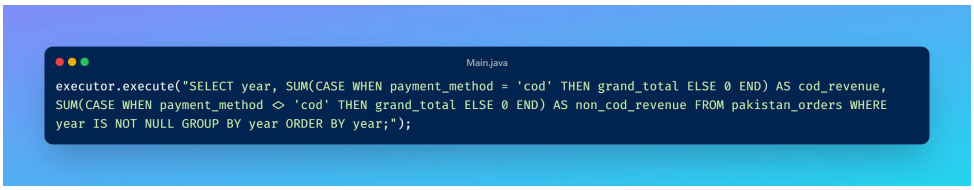

Figura 19 – Consulta com operações numéricas para cálculo de faturamento por método de pagamento no PostgreSQL, *dataset* Paquistão

3.5.3 Update - Implementação

No caso das operações de atualização no PostgreSQL, foi implementado um método responsável por executar comandos SQL do tipo *UPDATE*. Esse método estabelece uma conexão com o banco de dados por meio da API JDBC, cria um objeto *Statement* e executa a instrução SQL recebida como parâmetro. Durante a execução, o tempo necessário para completar a operação é medido utilizando a diferença entre os instantes inicial e final da execução, permitindo registrar o desempenho da atualização no banco de dados.

Foram implementados três cenários de atualização sobre a tabela *pakistan_orders*. O primeiro consiste no incremento do valor do campo *price* em todas as tuplas da tabela. O segundo realiza a duplicação do valor presente no campo *discount_amount*, aplicando a atualização apenas aos registros cujo desconto seja maior que zero. Por fim, o terceiro cenário altera o valor do campo *status* para "canceled" em todos os pedidos cuja quantidade solicitada (*qty_ordered*) seja igual a um. Essas operações foram definidas de forma a representar atualizações em massa sobre grandes volumes de dados, permitindo observar o comportamento do sistema gerenciador de banco de dados relacional durante esse tipo de modificação.

No MongoDB, as operações de atualização foram implementadas utilizando os métodos de modificação de documentos disponibilizados pela API do driver Java, especialmente a

operação *updateMany*. De forma semelhante ao procedimento adotado no PostgreSQL, o tempo de execução de cada operação é medido a partir da diferença entre o instante inicial e final da execução do método.

Assim como no modelo relacional, foram definidos três cenários de atualização sobre a coleção *orders*. O primeiro consiste no incremento do valor do campo *price* em dez unidades para os documentos cujo campo esteja armazenado como um valor numérico. O segundo realiza a duplicação do valor presente no campo *discount_amount* para os documentos que possuem desconto maior que zero, utilizando o operador de multiplicação. Por fim, o terceiro cenário altera o valor do campo *status* para "canceled" em todos os documentos cujo valor de *qty_ordered* seja igual a um. Essas operações demonstram diferentes formas de modificação de documentos no MongoDB, permitindo avaliar o desempenho do banco de dados orientado a documentos em operações de atualização em larga escala.

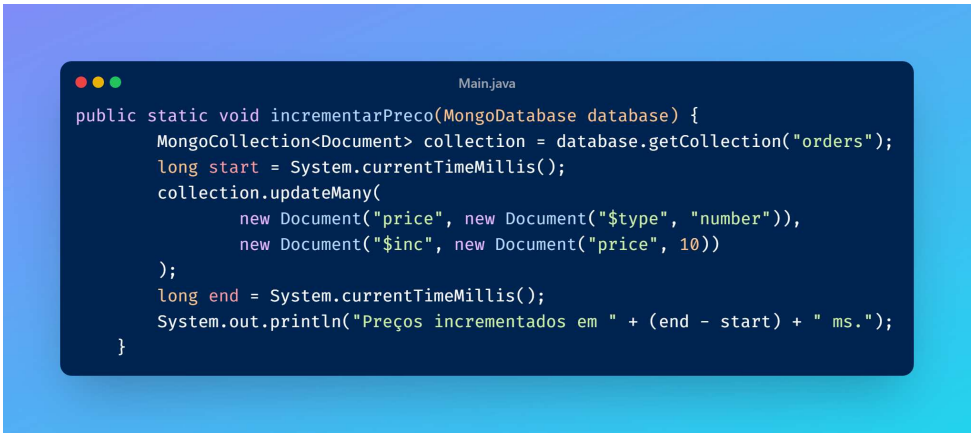

Figura 20 – Operação de atualização que incrementa o valor do campo *price* em dez unidades no MongoDB, *dataset* Paquistão

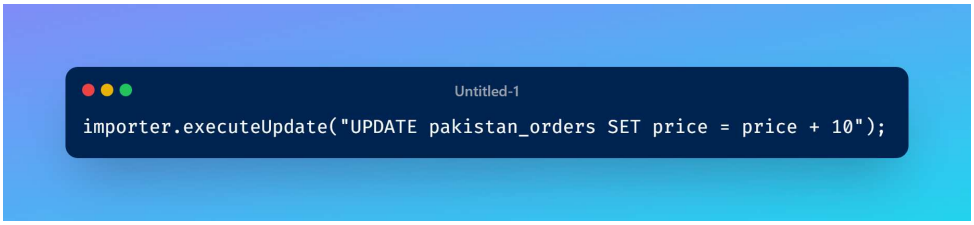

Figura 21 – Operação de atualização que incrementa o valor da coluna *price* em dez unidades no PostgreSQL, *dataset* Paquistão

3.5.4 Delete - Implementação

Para a avaliação das operações de remoção de dados, foi implementado um cenário de exclusão total dos registros armazenados na base de dados. No PostgreSQL, essa operação

é realizada por meio do comando *DELETE FROM pakistan_orders*, que remove todos os registros presentes na tabela.

No MongoDB, o procedimento equivalente é realizado sobre a coleção *orders* utilizando o método *deleteMany* com um filtro vazio, o que resulta na remoção de todos os documentos armazenados na coleção. Assim como no modelo relacional, o tempo de execução da operação é medido pela aplicação, permitindo registrar o desempenho do banco de dados durante a exclusão de grandes volumes de dados.

Esse cenário foi definido com o objetivo de avaliar o comportamento dos sistemas de gerenciamento de banco de dados em operações de remoção em larga escala, possibilitando a comparação do tempo necessário para eliminar completamente os registros armazenados em cada modelo de banco de dados.

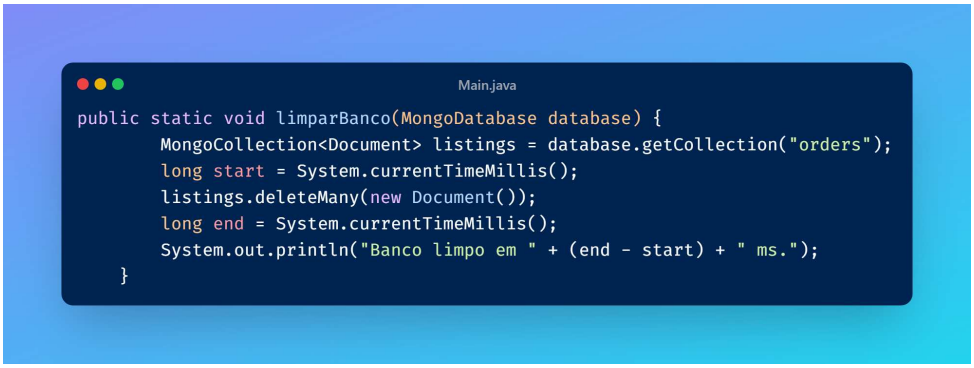

Figura 22 – Operação de exclusão de todos os documentos da coleção orders no MongoDB, *dataset* Paquistão

Figura 23 – Operação de exclusão de todos os registros da tabela pakistan orders no PostgreSQL, *dataset* Paquistão

Capítulo 4

Experimentos

Este trabalho tem como objetivo realizar uma comparação de desempenho entre o PostgreSQL e o MongoDB, representando, respectivamente, os paradigmas de bancos de dados relacionais (SQL) e orientados a documentos (NoSQL). Conforme definido no Capítulo 3, a métrica adotada para mensurar o desempenho é o tempo de execução de cada operação, medido a partir da camada de aplicação Java e expresso em milissegundos. Como métrica complementar, foi utilizado o desvio padrão dos tempos obtidos entre as repetições, indicando a variabilidade (e, por consequência, a previsibilidade) de cada sistema na execução de uma mesma operação. Não foram avaliadas outras métricas de desempenho, como consumo de CPU, uso de memória ou *throughput*, uma vez que o foco deste trabalho está no tempo de resposta percebido pela aplicação ao interagir com cada banco de dados, cenário mais próximo da experiência real de uso.

O foco dos testes foi a execução das quatro operações fundamentais do modelo *Create*, *Read*, *Update* e *Delete*.

Para cada operação, foram consideradas duas variáveis independentes: o sistema de gerenciamento de banco de dados (PostgreSQL ou MongoDB) e o conjunto de dados utilizado (Paquistão ou Airbnb), o que permite observar o efeito da estrutura dos dados (tabular, no caso do Paquistão, e semiestruturada com relação 1:N, no caso do Airbnb) sobre o desempenho de cada operação. Cada operação foi executada cinco vezes, com o propósito de calcular a média e o desvio padrão dos tempos obtidos, reduzindo o impacto de eventuais variações e garantindo maior estabilidade nos resultados. Dessa forma, buscou-se evitar que uma única execução atípica influenciasse significativamente a análise e as inferências realizadas a partir dos dados.

O primeiro conjunto de testes refere-se à operação *Create* (inserção de dados), apresentada a seguir.

4.1 C – Create

Nesta etapa, foi avaliado o desempenho das operações de inserção em ambos os sistemas gerenciadores de banco de dados. Os testes consistiram em inserir os registros das duas bases utilizadas, *Airbnb* e *Pakistan E-commerce Data*, em suas respectivas estruturas no PostgreSQL e no MongoDB. Cada inserção foi repetida cinco vezes para cada sistema, e a Figura 24 apresenta a média dos tempos de execução obtidos para cada conjunto de dados.

4.1.1 Análise das operações de inserção

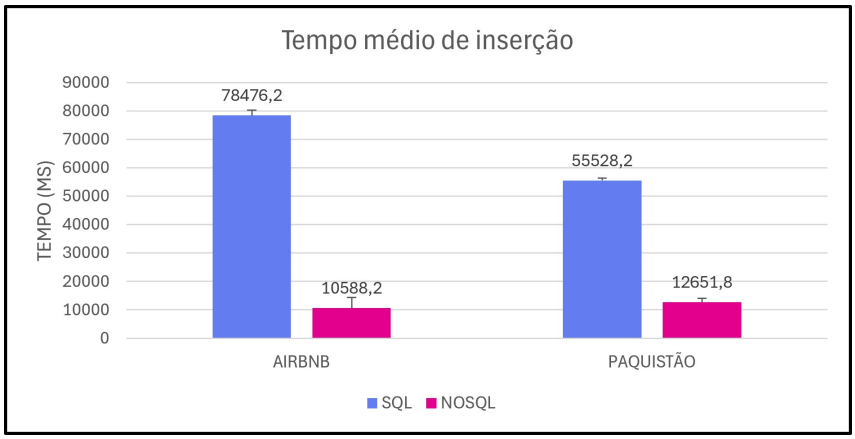

Figura 24 – Tempo médio das operações de inserção com barras de erro representando a variabilidade entre as execuções

A Figura 24 apresenta o tempo médio das operações de inserção nos bancos de dados avaliados, bem como a variabilidade observada entre as execuções do experimento, representada pelas barras de erro.

Os resultados indicam que o MongoDB apresentou desempenho significativamente superior ao PostgreSQL nas operações de inserção, com tempos médios aproximadamente quatro a sete vezes menores. Esse comportamento pode estar relacionado ao modelo orientado a documentos adotado pelo MongoDB, que reduz a necessidade de operações associadas à manutenção de relacionamentos entre entidades durante a inserção dos dados. Em contrapartida, o PostgreSQL, por seguir o modelo relacional e garantir as propriedades ACID, realiza verificações adicionais durante as inserções, como controle transacional, manutenção de índices e verificações de integridade, o que aumenta o tempo total da operação (EESSAAR, 2016).

A diferença de desempenho foi mais acentuada na base *Airbnb*, que possui maior volume de dados e diversos campos textuais de tamanho variável, como os comentários da entidade *reviews*. Isso ocorre porque, no modelo relacional, cada inserção segue passando pelas mesmas verificações de integridade e pelo mesmo controle transacional indepen-

dentemente do tamanho do campo, de modo que campos textuais maiores aumentam o volume de dados a validar e registrar a cada operação. Já no MongoDB, o documento inteiro é serializado e gravado de uma só vez, sem etapas adicionais de verificação entre entidades, o que faz com que o crescimento no tamanho dos campos textuais tenha impacto proporcionalmente menor no tempo de inserção.

Quanto à variabilidade entre as execuções, é necessário fazer uma ressalva metodológica. Todos os experimentos, para ambos os sistemas, foram executados na mesma máquina, o que garante que diferenças de hardware não expliquem eventuais diferenças de dispersão entre PostgreSQL e MongoDB. Ainda assim, como o MongoDB é de quatro a sete vezes mais rápido que o PostgreSQL nessa operação, essa diferença de escala deve ser levada em conta ao interpretar o desvio padrão de cada sistema, já que operações mais rápidas tendem naturalmente a apresentar desvios absolutos menores.

De modo geral, os resultados sugerem que, sob as condições testadas nesta máquina, o MongoDB ofereceu maior desempenho nas operações de inserção, enquanto o PostgreSQL apresentou menor dispersão absoluta nos tempos de execução entre as repetições.

4.2 R – Read

Nesta etapa, foram avaliadas as operações de leitura (*Read*) nos dois sistemas gerenciadores de banco de dados, PostgreSQL e MongoDB, sobre as bases *Airbnb* e *Paquistão E-commerce*. As consultas foram classificadas em quatro categorias principais: agregação/agrupamento, filtros/condicionais, ordenação/ranking e estatísticas/operações numéricas. Cada consulta foi executada cinco vezes, e a análise considera tanto os tempos médios quanto a variabilidade observada entre as execuções.

4.2.1 Análise das operações de consulta

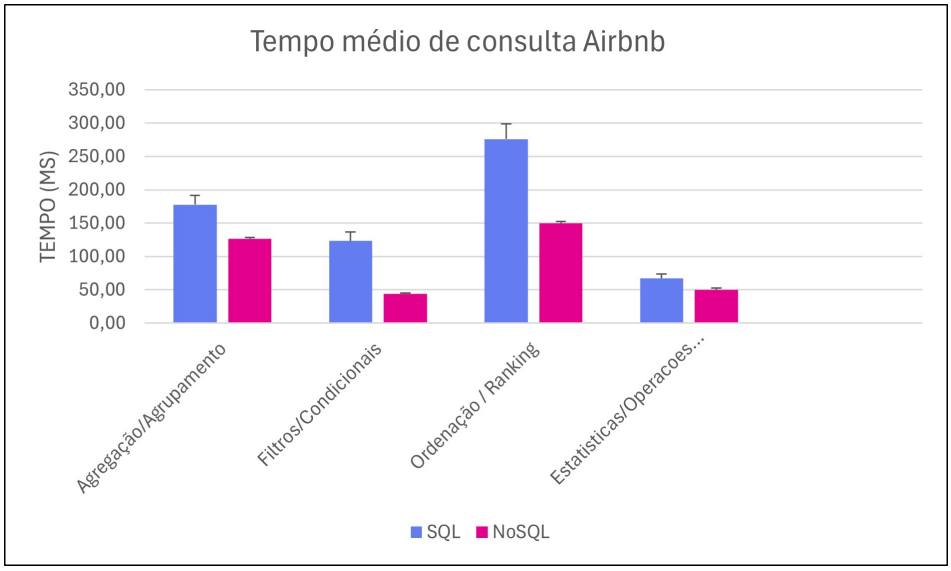

Figura 25 – Tempos médios das queries de leitura na base Airbnb, com barras de erro representando o desvio padrão

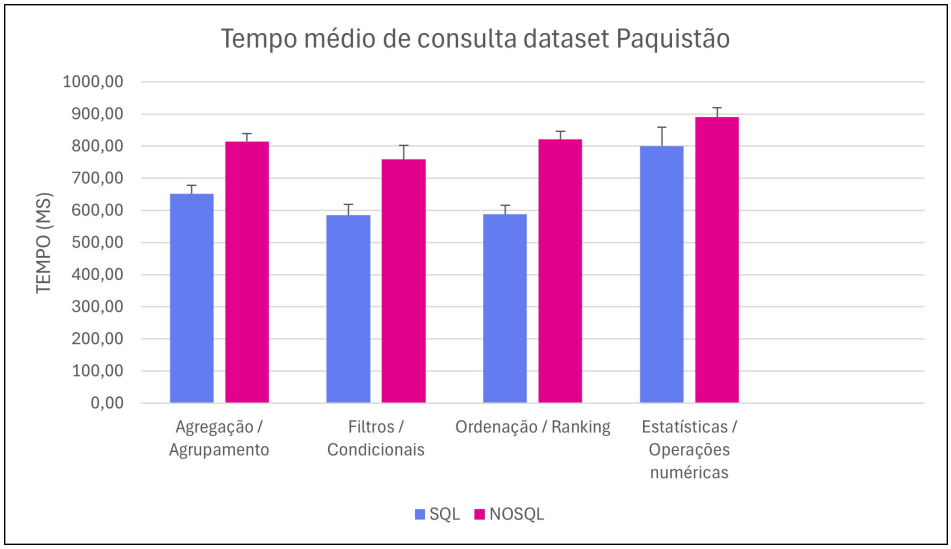

Figura 26 – Tempos médios das queries de leitura na base Paquistão, com barras de erro representando o desvio padrão

As Figuras 25 e 26 apresentam os tempos médios de execução das consultas de leitura, juntamente com a variabilidade observada entre as execuções, representada pelas barras de erro correspondentes ao desvio padrão.

Agregação/Agrupamento

Na base Airbnb, o MongoDB apresentou desempenho superior ao PostgreSQL em operações de agregação, com tempo médio de 126,32 ms, enquanto o PostgreSQL apresentou média de 177,64 ms. Isso representa uma redução de aproximadamente 51 ms, ou cerca de 29% no tempo de execução a favor do MongoDB (Figura 25). Esse resultado reflete

a eficiência do modelo orientado a documentos para operações de agregação sobre dados semiestruturados.

Já na base Paquistão, o PostgreSQL apresentou melhor desempenho, com tempo médio de 651,29 ms, enquanto o MongoDB registrou média de 814,43 ms. Nesse caso, o PostgreSQL executou as consultas cerca de 20% mais rapidamente, com diferença aproximada de 163 ms (Figura 26). Esse comportamento sugere que o otimizador de consultas do PostgreSQL apresenta maior eficiência em varreduras completas sobre dados estruturados, uma vez que nenhum índice customizado foi criado em nenhum dos dois sistemas durante os experimentos.

Filtros/Condicionais

Nas consultas de filtragem, o MongoDB demonstrou desempenho significativamente superior na base Airbnb, com tempo médio de 43,63 ms, enquanto o PostgreSQL apresentou média de 123,27 ms. Isso representa uma redução de aproximadamente 80 ms, ou cerca de 65% no tempo médio de execução (Figura 25). Esse resultado pode estar associado à eficiência do MongoDB na busca dentro de documentos individuais, favorecida pelo modelo de armazenamento orientado a documentos

Na base Paquistão, o PostgreSQL apresentou melhor desempenho, com tempo médio de 584,75 ms, enquanto o MongoDB registrou média de 759,00 ms. Nesse caso, o PostgreSQL foi aproximadamente 23% mais rápido, com diferença de cerca de 174 ms (Figura 26). Esse padrão é consistente com o comportamento observado na categoria anterior, reforçando a vantagem do otimizador de consultas do PostgreSQL em varreduras sobre dados tabulares estruturados.

Ordenação/Ranking

Nas operações de ordenação, o MongoDB apresentou vantagem na base Airbnb, com tempo médio de 149,44 ms, enquanto o PostgreSQL registrou média de 275,98 ms. Isso representa redução de aproximadamente 126 ms, ou cerca de 46% no tempo médio de execução (Figura 25). Esse resultado pode estar relacionado ao acesso direto aos documentos, reduzindo a necessidade de operações adicionais como junções entre tabelas.

Por outro lado, na base Paquistão, o PostgreSQL demonstrou melhor desempenho, com tempo médio de 588,16 ms, enquanto o MongoDB apresentou média de 820,68 ms. Nesse cenário, o PostgreSQL executou as consultas cerca de 28% mais rapidamente, com diferença aproximada de 232 ms (Figura 26). Esse resultado, observado de forma consistente nas categorias de consulta na base Paquistão, sugere que o PostgreSQL apresenta vantagem em varreduras completas sobre grandes volumes de dados homogêneos, possivelmente devido à maturidade do seu executor de consultas para esse tipo de operação.

Estatísticas/Operações numéricas

Nas consultas estatísticas, o MongoDB apresentou leve vantagem na base Airbnb, com tempo médio de 49,85 ms, enquanto o PostgreSQL apresentou média de 67,18 ms. Essa diferença representa redução aproximada de 17 ms, ou cerca de 26% no tempo de execução (Figura 25).A menor diferença absoluta entre os sistemas nessa categoria, de apenas 17 ms, sugere que em operações estatísticas simples sobre dados semiestruturados os dois sistemas tendem a apresentar desempenho próximo.

Na base Paquistão, o PostgreSQL mostrou-se mais eficiente, registrando média de 800,15 ms, enquanto o MongoDB apresentou tempo médio de 890,40 ms. Nesse caso, o PostgreSQL apresentou desempenho cerca de 10% superior, com diferença aproximada de 90 ms (Figura 26). Vale destacar que essa foi a menor diferença observada entre os dois sistemas na base Paquistão, sugerindo que em operações estatísticas e numéricas simples os dois sistemas tendem a convergir em desempenho. Ainda assim, o padrão geral se mantém, com o PostgreSQL apresentando vantagem em varreduras completas sobre grandes volumes de dados homogêneos, possivelmente devido à maturidade do seu executor de consultas para esse tipo de operação

De forma geral, os resultados indicam que o desempenho das consultas de leitura depende fortemente das características do conjunto de dados analisado. O MongoDB apresentou melhor desempenho em consultas sobre dados semiestruturados e com acesso direto a documentos, como observado na base Airbnb, onde a estrutura aninhada favorece o modelo orientado a documentos. Em contrapartida, o PostgreSQL demonstrou maior eficiência em varreduras completas sobre grandes volumes de dados homogêneos, como observado na base Paquistão, resultado consistente com a maturidade do seu executor de consultas para esse tipo de operação.

4.3 U - Update

Nesta etapa foram analisadas as operações de atualização (*Update*) realizadas nos bancos PostgreSQL e MongoDB sobre as bases Airbnb e Paquistão. Foram executados três tipos de atualizações: *bulk update*, atualização com filtro direto em listagens (*listings*) e atualização com filtro em avaliações (*reviews*). Cada operação foi repetida cinco vezes, sendo considerados os tempos médios de execução e a variabilidade observada entre as execuções.

4.3.1 Análise das operações de atualização

As Figuras 27 e 28 apresentam os tempos médios das operações de atualização realizadas nos dois sistemas gerenciadores de banco de dados, juntamente com a variabilidade entre as execuções representada pelas barras de erro correspondentes ao desvio padrão.

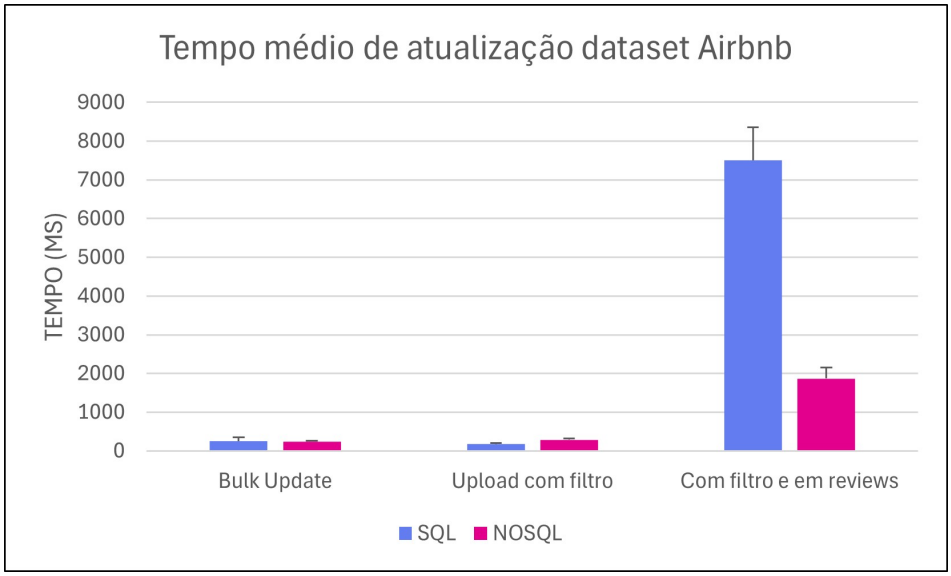

Figura 27 – Tempos médios das operações de atualização na base Airbnb, com barras de erro representando o desvio padrão

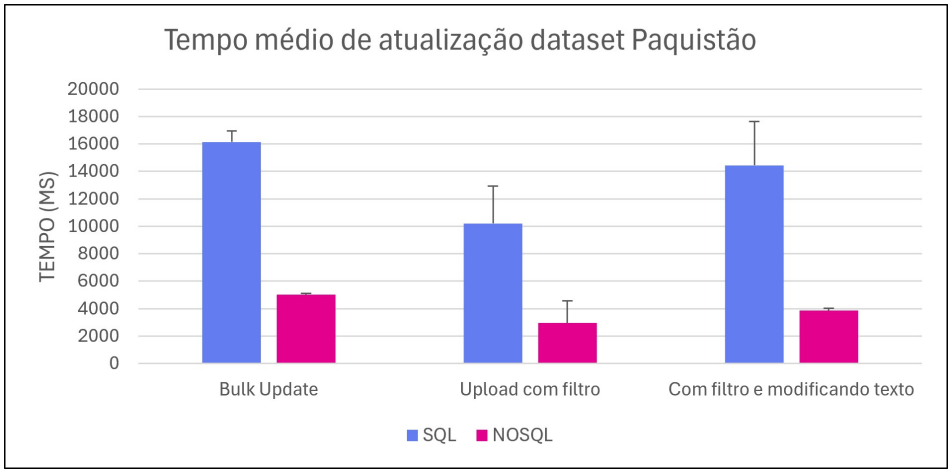

Figura 28 – Tempos médios das operações de atualização na base Paquistão, com barras de erro representando o desvio padrão

Base Airbnb

Na base Airbnb, os resultados indicam desempenhos relativamente próximos entre os bancos em algumas operações. No *bulk update*, o MongoDB apresentou tempo médio de 242,8 ms, enquanto o PostgreSQL registrou 253 ms, resultando em pequena vantagem para o banco orientado a documentos (Figura 27).

Na atualização com filtro em *listings*, entretanto, o PostgreSQL apresentou melhor desempenho, com tempo médio de 172,8 ms, enquanto o MongoDB registrou 284,2 ms. Uma possível explicação para esse comportamento está relacionada à forma como os dados foram modelados e processados pelo PostgreSQL nesse cenário, favorecendo operações de atualização realizadas sobre conjuntos específicos de registros.

A maior diferença observada nessa base ocorreu na atualização com filtro em *reviews*. O PostgreSQL apresentou tempo médio de 7502,6 ms, enquanto o MongoDB registrou 1869,6 ms, representando diferença superior a 5600 ms. Esse resultado sugere que o modelo orientado a documentos lida de forma mais eficiente com estruturas de dados aninhadas, nas quais múltiplas informações relacionadas podem ser atualizadas dentro de um único documento.

Base Paquistão

Na base Paquistão, o MongoDB apresentou desempenho significativamente superior em todas as operações analisadas (Figura 28). No *bulk update*, o PostgreSQL registrou tempo médio de 16140 ms, enquanto o MongoDB apresentou 4999,8 ms.

Na atualização com filtro, os tempos médios foram de 10200,2 ms para o PostgreSQL e 2936 ms para o MongoDB. Já na atualização com filtro em avaliações, o PostgreSQL apresentou média de 14430,4 ms, enquanto o MongoDB registrou 3857,4 ms.

Essa diferença pode estar relacionada ao fato de que a base Paquistão consiste em uma única tabela grande e sem relações complexas. Nesse cenário, o PostgreSQL, por manter controle transacional e mecanismos de bloqueio típicos de bancos relacionais, pode apresentar maior sobrecarga em atualizações de grande volume. O MongoDB, por sua vez, tende a realizar atualizações de forma mais direta sobre os documentos armazenados.

De forma geral, os resultados indicam que o MongoDB apresenta melhor desempenho em operações de atualização que envolvem grandes volumes de dados ou estruturas semiestruturadas. O PostgreSQL, por outro lado, manteve desempenho competitivo em operações mais simples, especialmente quando realizadas com filtros diretos sobre conjuntos menores de registros.

4.4 D – Delete

Nesta etapa, foram analisadas as operações de exclusão (*Delete*) nos bancos PostgreSQL e MongoDB, utilizando as bases Airbnb e Paquistão. Cada operação foi repetida cinco vezes, sendo considerados os tempos médios de execução e a variabilidade observada entre as execuções.

4.4.1 Análise das operações de exclusão

A Figura 29 apresenta os tempos médios de execução das operações de exclusão realizadas nos dois sistemas gerenciadores de banco de dados, bem como a variabilidade observada entre as execuções, representada pelas barras de erro correspondentes ao desvio padrão.

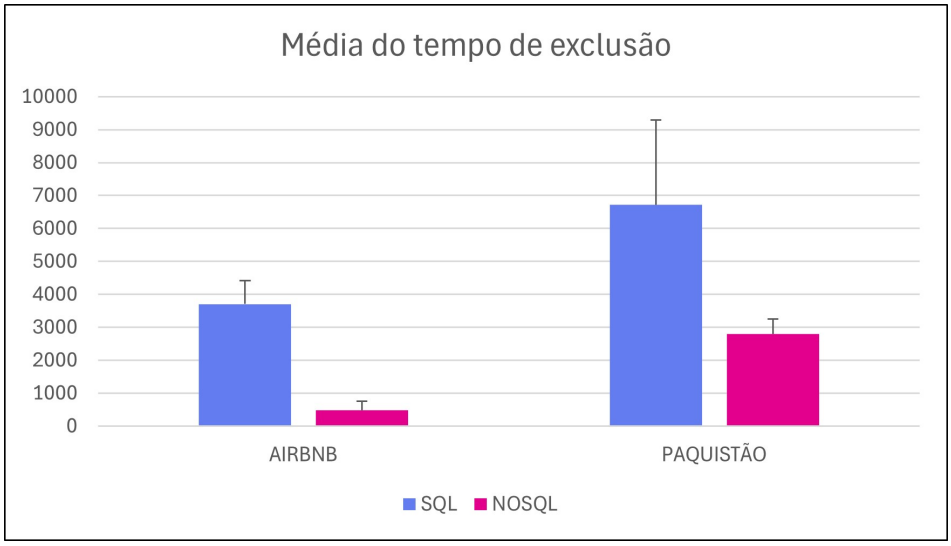

Figura 29 – Tempos médios das operações de exclusão nas bases Airbnb e Paquistão, com barras de erro representando o desvio padrão

Na base Airbnb, o MongoDB apresentou tempo médio de 483,2 ms, significativamente menor que o PostgreSQL, que registrou 3701,8 ms. Isso representa uma diferença superior a 3200 ms entre os sistemas (Figura 29), indicando maior eficiência do banco orientado a documentos nessa operação.

Na base Paquistão, o MongoDB também apresentou melhor desempenho, com tempo médio de 2791,2 ms, enquanto o PostgreSQL registrou 6719,8 ms. A diferença de aproximadamente 3900 ms reforça o comportamento observado na base anterior.

Além das diferenças nos tempos médios, observa-se também variação entre as execuções. O MongoDB apresentou menor dispersão nos resultados, com desvios padrão de 268,78 ms na base Airbnb e 463,74 ms na base Paquistão. Já o PostgreSQL apresentou maior variabilidade, com desvios de 708,22 ms e 2564,39 ms, respectivamente.

Esses resultados indicam que, em operações de exclusão, o MongoDB se beneficia do modelo orientado a documentos, permitindo a remoção direta de registros sem a sobrecarga associada à verificação de integridade referencial e aos mecanismos de bloqueio característicos de bancos relacionais.

Por outro lado, o PostgreSQL, embora ofereça forte garantia de integridade e propriedades transacionais ACID, tende a apresentar maior custo computacional em operações de exclusão em larga escala. Esse comportamento pode resultar em tempos de execução mais elevados e maior variabilidade, especialmente em cenários envolvendo grandes volumes de dados.

4.5 Avaliação dos Resultados

A avaliação dos experimentos realizados evidencia diferenças claras no desempenho entre os bancos de dados PostgreSQL e MongoDB, refletindo as características intrínsecas de cada modelo de armazenamento e as estruturas das bases utilizadas.

Create (Inserção de Dados)

Nos testes de inserção, o MongoDB apresentou desempenho significativamente superior ao PostgreSQL, com tempos médios cerca de quatro a sete vezes menores, especialmente na base Airbnb, mais semiestruturada. Esse comportamento está alinhado com o modelo orientado a documentos, que dispensa validações de integridade referencial e operações de junção no momento da escrita, reduzindo a sobrecarga de I/O.

Apesar da maior velocidade do MongoDB, seu desvio padrão foi mais elevado, indicando maior variabilidade entre execuções, possivelmente devido a políticas internas de escrita assíncrona e gerenciamento de cache. Por outro lado, o PostgreSQL apresentou tempos médios mais altos, porém com menor variabilidade, refletindo maior consistência em operações de inserção, o que pode ser importante em aplicações que demandam latência previsível.

Read (Leitura de Dados)

Nas operações de leitura, os resultados foram dependentes da base de dados e do tipo de consulta:

- Na base Airbnb, o MongoDB se destacou em agregações, filtros e ordenações, refletindo eficiência na leitura de dados semiestruturados e acesso direto a documentos.
- Na base Paquistão, o PostgreSQL apresentou vantagem em consultas agregadas, filtros e operações numéricas, evidenciando melhor desempenho em operações realizadas sobre dados tabulares de grande volume.
- Em categorias mais simples, como distribuição/buckets, ambos os SGBDs apresentaram baixo desvio padrão, indicando alta previsibilidade nessas operações.

O desvio padrão geral indica que o MongoDB apresentou maior consistência em consultas simples e médias rápidas na base Airbnb, enquanto o PostgreSQL manteve maior consistência em operações complexas na base Paquistão.

Update (Atualização de Dados)

Nos testes de atualização, o MongoDB apresentou tempos médios menores e menor variabilidade em grande parte dos casos, especialmente nas operações envolvendo dados

aninhados ou filtragem complexa, como em avaliações da base Airbnb. Exceção ocorreu na atualização com filtro direto sobre listings, onde o PostgreSQL apresentou melhor desempenho (172,8 ms contra 284,2 ms do MongoDB), sugerindo que filtros diretos sobre conjuntos menores de registros podem favorecer a estrutura relacional.

O PostgreSQL apresentou tempos mais elevados, principalmente na base Paquistão, devido à necessidade de bloqueios em nível de linha e controle transacional característicos do modelo relacional, fatores que aumentam a sobrecarga em atualizações de grande volume mesmo em tabelas sem relações complexas. O MongoDB, por sua vez, realiza atualizações de forma mais direta sobre os documentos, sem a mesma rigidez transacional, o que contribui para menores tempos de execução nesses cenários

Essa diferença reforça a ideia de que bancos orientados a documentos são mais eficientes em operações que não exigem relacionamentos complexos ou integridade referencial estrita, enquanto bancos relacionais oferecem estabilidade e consistência, porém com maior custo de desempenho em cargas de escrita intensiva (CARVALHO; Sá; BERNARDINO, 2023).

Delete (Exclusão de Dados)

Nas operações de exclusão, o MongoDB demonstrou clara vantagem, com tempos médios significativamente menores e desvio padrão mais baixo, evidenciando maior estabilidade nas remoções.

O PostgreSQL apresentou tempos mais altos e maior variabilidade, resultado da necessidade de manter integridade referencial e aplicar bloqueios transacionais. Esses fatores aumentam o custo de remoções em grandes volumes de dados, especialmente em tabelas tabulares não particionadas.

Síntese geral

Com base nos resultados obtidos nos experimentos, observa-se que:

- Sob as condições testadas, o MongoDB tendeu a se sobressair em operações de escrita e em leituras sobre dados semiestruturados (base Airbnb), apresentando tempos médios menores e boa consistência em operações simples. Nas leituras sobre dados tabulares homogêneos (base Paquistão), o PostgreSQL demonstrou vantagem em todas as categorias de consulta analisadas.

- O PostgreSQL demonstrou vantagens em operações sobre grandes volumes de dados tabulares homogêneos, além de oferecer maior consistência transacional por meio das propriedades ACID.

- A variabilidade dos tempos (desvio padrão) tendeu a ser menor no MongoDB em operações de escrita, enquanto o PostgreSQL apresentou maior variabilidade em

operações de escrita em massa, possivelmente devido aos mecanismos de controle transacional característicos do modelo relacional.

- Os resultados sugerem que o desempenho está fortemente ligado à natureza dos dados analisados: dados semiestruturados tenderam a favorecer o MongoDB, enquanto dados tabulares homogêneos tenderam a favorecer o PostgreSQL, sob os cenários experimentais adotados.

Capítulo 5

Conclusão

O presente trabalho teve como objetivo principal comparar o desempenho entre bancos de dados SQL e NoSQL, analisando como cada tipo de banco se comporta sob diferentes operações de leitura, escrita, atualização e exclusão de dados. Os experimentos realizados permitiram avaliar o tempo médio de execução e a variabilidade das operações em duas bases de dados distintas, contemplando tanto cenários com grande volume de dados quanto estruturas mais semiestruturadas.

Os resultados obtidos revelaram comportamentos distintos entre os dois sistemas em função das características dos dados e do tipo de operação realizada. A análise detalhada das operações de Create, Read, Update e Delete possibilitou verificar como características intrínsecas de cada modelo influenciam o desempenho: sob as condições testadas, o MongoDB tendeu a apresentar operações de escrita mais rápidas e maior eficiência em leituras sobre dados semiestruturados, enquanto o PostgreSQL demonstrou vantagem em consultas de leitura sobre grandes volumes de dados tabulares homogêneos, além de consistência e previsibilidade transacional garantidas pelas propriedades ACID, apesar de maior custo de processamento em cargas intensivas de escrita e exclusão.

Com base nesses resultados, as hipóteses formuladas foram verificadas: H1 foi confirmada, uma vez que o MongoDB apresentou desempenho superior ao PostgreSQL nas operações de inserção, atualização e exclusão de dados; H2 foi confirmada, dado que o MongoDB demonstrou maior eficiência em consultas sobre dados semiestruturados, como observado na base Airbnb; H3 foi confirmada, pois o PostgreSQL apresentou vantagem consistente em consultas analíticas sobre dados tabulares homogêneos, como evidenciado na base Paquistão.

Dessa forma, conclui-se que os objetivos específicos foram efetivamente alcançados. Os tempos de execução foram mensurados e comparados em ambos os paradigmas, as diferenças de desempenho entre operações de leitura e escrita foram identificadas, e as vantagens e limitações de cada abordagem foram caracterizadas considerando diferentes tipos de dados e volumes, fornecendo subsídios para a escolha do banco de dados mais adequado a cada cenário de aplicação

5.1 Trabalhos Futuros

Como continuidade deste trabalho, alguns caminhos podem ser explorados para ampliar a análise realizada. Uma possibilidade consiste na execução dos experimentos em ambientes de computação em nuvem, permitindo observar o comportamento dos bancos de dados em cenários mais próximos de aplicações reais, nos quais fatores como latência de rede, escalabilidade e alocação dinâmica de recursos podem influenciar o desempenho. Também pode ser interessante incluir outros sistemas de gerenciamento de banco de dados na comparação, tanto relacionais quanto NoSQL baseados em diferentes modelos de dados. Além disso, análises envolvendo estratégias de indexação, consultas analíticas mais complexas e cenários com múltiplos usuários concorrentes podem contribuir para uma compreensão mais aprofundada das diferenças entre essas tecnologias. Por fim, a investigação do comportamento dos bancos de dados em arquiteturas distribuídas, considerando aspectos como replicação, particionamento de dados e tolerância a falhas, pode fornecer uma visão mais ampla sobre o desempenho e a escalabilidade desses sistemas.

5.2 Contribuições em Produção Bibliográfica

- FERNANDES, Guilherme. Estudo comparativo de desempenho entre PostgreSQL e MongoDB em operações CRUD. In: Workshop de Trabalhos de Conclusão de Curso (WTDCC), 2025. Anais do WTDCC.

Referências

BARBOSA, M. J. d. O. **Análise Comparativa de Banco de Dados Relacionais e NoSQL em um Ambiente de Computação nas Nuvens**. 2013. Acesso em: 1 out. 2024. Disponível em: <http://fbuni.edu.br/sites/default/files/tcc_-_2013_1_-_maria_josiane_de_oliveira.pdf>. Citado na página 19.

BRITO, R. W. **Bancos de Dados NoSQL x SGBDs Relacionais: Análise Comparativa**. Fortaleza, CE, Brasil, 2013. Acesso em: 1 out. 2024. Disponível em: <https://googlegroups.com/group/nosqlbr/attach/a55077acbc539ceb/Bancos%20de%20Dados%20NoSQL%20x%20SGBDs%20Relacionais%20-%20An%C3%A1lise%20Comparativa.pdf?part=0.1>. Citado na página 28.

CARVALHO, I.; Sá, F.; BERNARDINO, J. Performance evaluation of nosql document databases: Couchbase, couchdb, and mongodb. **Algorithms**, v. 16, n. 2, 2023. ISSN 1999-4893. Disponível em: <https://www.mdpi.com/1999-4893/16/2/78>. Citado 2 vezes nas páginas 29 e 57.

DATE, C. J. **An Introduction to Database Systems**. 8. ed. Boston: Addison-Wesley, 2003. 850 p. Citado na página 22.

EESSAAR, E. The database normalization theory and the theory of normalized systems: Finding a common ground. **Baltic Journal of Modern Computing**, v. 4, n. 1, 2016. Citado 2 vezes nas páginas 18 e 48.

ELMASRI, R.; NAVATHE, S. B. **Sistemas de Banco de Dados**. 4. ed. [S.l.]: Addison Wesley, 2005. Acesso em: 2 out. 2024. Citado 4 vezes nas páginas 12, 16, 17 e 18.

Google LLC. **Cloud Firestore: NoSQL Document Database**. 2017. Disponível em: <https://firebase.google.com/docs/firestore>. Acesso em: 1 jun. 2026. Citado na página 20.

GYőRöDI, C. et al. A comparative study: MongoDB vs. MySQL. In: **2015 13th International Conference on Engineering of Modern Electric Systems (EMES)**. Oradea, Romania: IEEE, 2015. p. 1–6. Citado 2 vezes nas páginas 13 e 29.

KUMAR, N. **Big Data Statistics 2026 (Volume, Growth & Market Size)**. 2026. Acesso em: 11 ago. 2026. Disponível em: <https://www.demandsage.com/big-data-statistics/>. Citado na página 12.

LI, Y.; MANOHARAN, S. A performance comparison of sql and nosql databases. In: **IEEE Pacific RIM Conference on Communications, Computers, and Signal Processing - Proceedings**. [S.l.]: IEEE, 2013. p. 15–19. Citado 2 vezes nas páginas 13 e 29.

LÓSCIO, B. F.; OLIVEIRA, G. R.; PONTES, J. C. d. S. Nosql no desenvolvimento de aplicações web colaborativas. In: **Anais da VIII SBSC**. Paraty, RJ: [s.n.], 2011. Acesso em: 2 out. 2024. Disponível em: <https://www.researchgate.net/profile/Bernadette-Loscio/publication/268201466_NoSQL_no_desenvolvimento_de_aplicacoes_Web_colaborativas/links/576aa72008aef2a864d1ef8c/NoSQL-no-desenvolvimento-de-aplicacoes-Web-colaborativas.pdf>. Citado 7 vezes nas páginas 12, 13, 19, 20, 21, 22 e 23.

MACARIO, C. G. do N.; BALDO, S. M. **Resumo do Modelo Relacional**. 2009. Acesso em: 1 out. 2024. Disponível em: <https://www.ic.unicamp.br/~geovane/mo410-091/Ch03-RM-Resumo.pdf>. Citado na página 16.

Meta Platforms, Inc. **RocksDB: A Persistent Key-Value Store for Fast Storage**. 2012. Disponível em: <https://rocksdb.org>. Acesso em: 1 jun. 2026. Citado na página 20.

Microsoft Corporation. **Azure Cosmos DB: Globally Distributed, Multi-Model Database Service**. 2017. Disponível em: <https://learn.microsoft.com/en-us/azure/cosmos-db/>. Acesso em: 1 jun. 2026. Citado na página 20.

MongoDB Inc. **MongoDB Documentation**. 2024. <https://www.mongodb.com/docs/>. Acesso em: 10 nov. 2025. Citado na página 37.

Neo4j. **Graph Databases for RDBMS Developers**. 2025. <https://neo4j.com/books/definitive-guide-graph-databases-rdbms-developer/>. Acesso em: 10 jun. 2026. Citado na página 26.

RAMAKRISHNAN, R.; GEHRKE, J. **Database Management Systems**. 3. ed. New York: McGraw-Hill, 2002. 123, 456 p. Citado na página 19.

REDMOND, E.; WILSON, J.; CARTER, J. **Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement**. [S.l.]: Pragmatic Bookshelf, 2012. 360 p. (Oreilly and Associate Series). Citado 2 vezes nas páginas 24 e 25.

SCHMID, H. A.; SWENSON, J. R. On the semantics of the relational data model. In: **Proceedings of the 1975 ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: Association for Computing Machinery, 1975. (SIGMOD '75), p. 211–223. ISBN 9781450373289. Disponível em: <https://doi.org/10.1145/500080.500110>. Citado na página 17.

SWSW1717. **Seatle Airbnb Open Data — SQL Project**. 2021. Kaggle. Acesso em: 25 fev. 2026. Disponível em: <https://www.kaggle.com/datasets/swsw1717/seatle-airbnb-open-data-sql-project>. Citado na página 32.

VOGELS, W. **Amazon DynamoDB: A Fast and Scalable NoSQL Database Service Designed for Internet Scale Applications**. 2012. Disponível em:

<https://www.allthingsdistributed.com/2012/01/amazon-dynamodb.html>. Acesso em: 1 jun. 2026. Citado na página 20.

ZUSMANI. **Pakistan's Largest E-Commerce Dataset**. 2017. Kaggle. Acesso em: 25 fev. 2026. Disponível em: <https://www.kaggle.com/datasets/zusmani/pakistans-largest-ecommerce-dataset>. Citado na página 31.