KARYME REIS

Estudo comparativo entre banco de dados relacionais e imutáveis para um serviço de projeções de retorno de investimentos

São Paulo
2024

KARYME REIS

Estudo comparativo entre banco de dados relacionais e imutáveis para um serviço de projeções de retorno de investimentos

Versão Original

Monografia apresentada ao PECE – Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo como parte dos requisitos para a conclusão do curso de MBA em Engenharia de Software.

Área de Concentração: Engenharia de Software

Orientador: Prof. Me. Bruno Sofiato

São Paulo
2024

FICHA CATALOGRÁFICA

[Colocar na versão final do trabalho. Obter em:

https://www.poli.usp.br/bibliotecas/servicos/catalogacao-na-publicacao]

Nome: REIS, Karyme

Título: Estudo comparativo entre banco de dados relacionais e imutáveis para um serviço de projeções de retorno de investimentos

Monografia apresentada ao PECE – Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo como parte dos requisitos para a conclusão do curso de MBA em Engenharia de Software.

Aprovado em: / /

Banca Examinadora

Prof(a). Dr(a). _____

Instituição: _____

Julgamento: _____

Prof(a). Dr(a). _____

Instituição: _____

Julgamento: _____

Prof(a). Dr(a). _____

Instituição: _____

Julgamento: _____

DEDICATÓRIA

Dedico este trabalho à tecnologia, que nos inspira a transformar o impossível em realidade, e à minha amada Luana que, com sua paciência e seu carinho, acompanha e inspira a finalização de mais este trabalho.

AGRADECIMENTOS

À Universidade de São Paulo – USP, que representa um pilar de excelência acadêmica e inspiração, proporcionando o ambiente ideal para a realização deste trabalho e minha formação contínua.

Ao professor orientador Bruno Sofiato, cuja dedicação e interesse genuíno em estimular que o melhor fosse feito foram fundamentais para a construção deste trabalho.

Aos meus pais, que com amor sempre incentivaram minha busca por conhecimento, proporcionando a base sólida que sustenta minhas conquistas.

E à Luana, minha companheira de jornada, cuja presença constante, compreensão e motivação iluminam cada passo, transformando desafios em realizações.

A todos, minha sincera gratidão.

RESUMO

REIS, K. **Estudo comparativo entre banco de dados relacionais e imutáveis para um serviço de projeções de retorno de investimentos**. 2024. Monografia (MBA em Engenharia de Software). Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo, São Paulo, 2024.

Este trabalho abordou o crescente desafio de selecionar modelos de bancos de dados adequados para aplicações críticas no setor financeiro, especificamente para projeções de retorno de investimentos. Motivado pela necessidade de soluções que combinem desempenho, auditabilidade e escalabilidade, foi desenvolvido um protótipo que comparou bancos de dados relacionais representado pelo PostgreSQL e imutáveis representado pelo Datomic. Utilizando APIs construídas em Java com Spring Boot, os dados de ativos fictícios foram atualizados diariamente e consultados em diferentes pontos no tempo para avaliar características como tempo de resposta, consistência de resultados e suporte a auditabilidade, alinhadas ao modelo de qualidade ISO 25010. Os resultados indicaram que os bancos de dados imutáveis são mais eficientes em consultas temporais e auditabilidade, enquanto os relacionais apresentam vantagens no processamento de atualizações em massa. Este estudo se propôs a contribuir para a compreensão dos benefícios e das limitações de cada abordagem, sugerindo caminhos para futuros estudos mais rigorosos e abrangentes no contexto de sistemas financeiros.

Palavras-chave: banco de dados relacional, banco de dados imutável, PostgreSQL, Datomic, projeção de investimentos.

ABSTRACT

REIS, K. **Comparative Study Between Relational and Immutable Databases for an Investment Return Projection Service**. 2024. Monograph (MBA in Software Engineering). Continuing Education Program in Engineering of the Polytechnic School of the University of São Paulo, São Paulo, 2024.

This study addressed the growing challenge of selecting suitable database models for critical applications in the financial sector, specifically for investment return projections. Driven by the need for solutions that combine performance, auditability, and scalability, a prototype was developed to compare relational databases, represented by PostgreSQL, and immutable databases, represented by Datomic. Using APIs built in Java using Spring Boot, data from fictitious assets were updated daily and queried at different points in time to evaluate characteristics such as response time, result consistency, and auditability support, aligned with the ISO 25010 quality model. The results indicated that immutable databases are more efficient for temporal queries and auditability, while relational databases have advantages in bulk update processing. This study aimed to contribute to the understanding of the benefits and limitations of each approach, suggesting avenues for more rigorous and comprehensive future studies in the context of financial systems..

Keywords: relational database, immutable database, PostgreSQL, Datomic, investment projection

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivações

De acordo com o Corporate Finance Institute (CFI Team, 2020), o setor financeiro enfrenta desafios significativos com o crescimento exponencial na geração de dados e com a complexidade crescente dos sistemas de informação. A digitalização e o uso de tecnologias como *big data* e *machine learning* aumentaram a demanda por processamento eficiente de grandes volumes de dados, estruturados e não estruturados, fundamentais para a tomada de decisões, gerenciamento de riscos — essenciais no mercado financeiro.

A integração eficiente desses dados representa um grande obstáculo para instituições financeiras, especialmente aquelas que lidam com sistemas legados e fragmentados. Fusões e aquisições frequentemente criam silos de dados, dificultando a gestão integrada e elevando os custos. A precisão, integridade e auditabilidade dos dados são essenciais para garantir a escalabilidade, segurança cibernética e conformidade com regulamentações rigorosas.

Desde os anos 1970, bancos de dados relacionais, como PostgreSQL e MySQL, têm dominado o armazenamento e gerenciamento de dados, oferecendo uma estrutura robusta e confiável para operações transacionais. Segundo Elmasri e Navathe (2015), esses sistemas atendem a requisitos de confiabilidade e precisão no armazenamento e manipulação de dados. Contudo, com o crescimento contínuo do volume de dados e a demanda por maior transparência e auditabilidade, surgiram novos paradigmas que complementam ou desafiam o modelo relacional tradicional.

Bancos de dados imutáveis, como o Datomic, introduziram uma nova abordagem ao permitir que cada modificação nos dados seja preservada como um novo estado, sem sobrescrever dados antigos. Estudos como os de Kleppmann (2017) destacam as vantagens dos sistemas imutáveis em contextos que exigem controle de versão e transparência. No setor financeiro, a imutabilidade se mostrou vantajosa para atender às exigências de auditoria, permitindo uma trilha completa para rastrear alterações essenciais para regulamentações. Essas inovações levantam questões sobre qual modelo de banco de dados oferece maior eficiência e confiabilidade em um setor dinâmico como o financeiro.

1.2 Objetivo

A partir da problemática apresentada, este trabalho teve como objetivo comparar o desempenho e a adequação de dois modelos de banco de dados: o relacional e o imutável. Para executar a comparação entre os dois modelos, foi desenvolvido um protótipo de um serviço de cálculo de projeções de investimentos em Java[1]. A partir deste protótipo, foram realizadas medições com base na norma ISO/IEC 25010, que define um modelo de qualidade para sistemas e produtos de software.

De acordo com a ISO/IEC 25010:2011, a análise de desempenho incluiu a eficiência do processamento de consultas temporais, medida em termos de tempo de resposta durante a execução de projeções para diferentes pontos no tempo. A auditabilidade foi avaliada com base na capacidade do sistema de manter registros completos e rastreáveis de modificações nos dados ao longo do tempo, permitindo reconstruções históricas precisas. A pesquisa busca, assim, identificar qual modelo de banco de dados — representado pelo PostgreSQL (relacional) e pelo Datomic (imutável) — oferece maior eficiência e melhor suporte à preservação do histórico de dados, aspectos críticos para um serviço de cálculo de projeção de investimentos em cenários de conformidade regulatória e auditoria.

1.3 Justificativas

A relevância deste estudo reside no crescente interesse de *fintechs* e empresas de investimento em soluções tecnológicas que ofereçam segurança, auditabilidade e desempenho, especialmente em ambientes altamente regulamentados. O setor financeiro exige ferramentas que permitem maior transparência tornando a escolha do modelo de banco de dados um fator estratégico para o sucesso das operações.

Essa questão é especialmente relevante no contexto deste trabalho, pois o protótipo — que tem por objetivo calcular projeções de retorno para carteiras de investimentos, atenderia qualquer banco ou *fintech* dados os devidos ajustes de contexto. E considerando o fato de que estes cálculos de projeção devem ser feitos

[1] O repositório que contém o código-fonte completo pode ser acessado em https://github.com/karymereis1/return-calculator (REIS, 2024).

com base em dados atuais ou em qualquer ponto no tempo, a escolha entre um banco de dados relacional, como o PostgreSQL, e um banco de dados imutável, como o Datomic, afeta diretamente a eficiência, a precisão e a rastreabilidade dos cálculos. Com base nos estudos de Kleppmann (2017), que destacam a aplicação de bancos de dados imutáveis para controle de versão e de auditoria, este estudo visa fornecer uma contribuição para empresas que têm necessidades semelhantes na escolha de soluções tecnológicas.

1.4 Método de Pesquisa

O método de pesquisa adotado foi o Design Science Research (DSR), uma metodologia utilizada em engenharia de software para a criação de artefatos tecnológicos que resolvem problemas complexos. Segundo Peffers *et al.* (2007), o DSR oferece uma abordagem estruturada para o desenvolvimento e para a avaliação de artefatos voltados para solucionar desafios específicos e práticos.

Este método é especialmente adequado para áreas que demandam precisão e auditabilidade, como o setor financeiro, contexto deste trabalho. A criação do protótipo de um sistema de projeção de retorno de investimentos foi baseada nesta metodologia, pois ele facilitará a execução dos dois modelos de bancos em cenários “idênticos” de consulta a dados históricos para projeções de investimentos, permitindo uma comparação direta entre eles em termos de desempenho e capacidade de auditabilidade.

1.5 Estrutura do Trabalho

O trabalho foi organizado em cinco capítulos que se complementam para apresentar de forma progressiva a pesquisa, desde as motivações até as conclusões finais:

a) capítulo 1 - Introdução: apresenta as motivações, o objetivo, as justificativas, o método de pesquisa e a estrutura do trabalho, contextualizando o problema e delineando o propósito da pesquisa;

b) capítulo 2 e 3 - Fundamentação Teórica: realiza uma revisão sobre bancos de dados relacionais e imutáveis, explorando suas estruturas, características, vantagens e desvantagens. Conceitos como normalização, transações ACID,

escalabilidade e imutabilidade são discutidos com base em autores da área, como Elmasri e Navathe (2015), Hickey (2012) e Kleppmann (2017), contextualizando esses conceitos no setor financeiro;

c) capítulo 4 - Desenvolvimento: descreve o processo de implementação do protótipo de sistema de projeção de retorno de investimentos, utilizando o PostgreSQL e o Datomic. São detalhadas as etapas de aplicação da metodologia DSR, incluindo as escolhas de arquitetura, configuração dos bancos de dados e execução do protótipo para coletar resultados;

d) capítulo 5 - Análise de Resultados: apresenta e discute os resultados obtidos a partir da aplicação do protótipo em cenários práticos. As métricas de desempenho foram analisadas com foco no tempo de resposta, auditabilidade e capacidade de gerenciar dados históricos;

e) capítulo 6 - Considerações Finais: realizou uma síntese das principais conclusões do estudo, sugerindo melhorias e possibilidades de aplicação futura para bancos de dados imutáveis no setor financeiro. Foram discutidas as limitações do trabalho e propostas para pesquisas futuras na área.

2 BANCOS DE DADOS RELACIONAIS

Os bancos de dados relacionais emergiram como uma solução inovadora para a crescente demanda por métodos eficientes de organização e manipulação de grandes volumes de dados. Em 1970, Edgar F. Codd propôs que os dados fossem organizados em relações – estruturas tabulares onde cada linha (ou tupla) representasse uma instância de uma entidade, e cada coluna, um atributo específico dessa entidade. Além de introduzir esse modelo relacional, Codd desenvolveu o conceito de álgebra relacional, um conjunto de operações matemáticas que permitia a manipulação precisa e eficiente das relações, fundamentando as operações de consulta e manipulação dos dados. Esse modelo relacional foi a base sobre a qual linguagens como SQL (*Structured Query Language*) foram construídas, facilitando a inserção, atualização, exclusão e recuperação de dados de forma consistente e padronizada (Elmasri & Navathe, 2015).

A organização dos dados em tabelas inter-relacionadas trouxe diversos benefícios, sendo a normalização um dos mais importantes. Com base na obra *Fundamentals of Database Systems* de Elmasri e Navathe (2015), a normalização é descrita como um processo essencial para reduzir a redundância e assegurar a integridade dos dados em bancos de dados relacionais. Esta técnica organiza dados em tabelas inter-relacionadas, decompostas em estruturas menores, para evitar anomalias de atualização e manter a integridade referencial[2]. Esta estruturação facilita a rastreabilidade e análise de dados, especialmente útil em setores como o financeiro, onde operações complexas demandam consistência e precisão.

A adoção da SQL como interface padrão entre sistemas de dados tornou os bancos relacionais adaptáveis a diversas aplicações. Conforme destacado por Elmasri (2015), a padronização da SQL permitiu que sistemas, como PostgreSQL e MySQL, seguissem o modelo relacional de maneira semelhante, facilitando uma curva de aprendizado mais simples e aumentando a portabilidade entre plataformas.

---

[2] Refere-se a uma regra essencial nos bancos de dados relacionais que assegura a consistência nas relações entre tabelas. Essa regra exige que uma chave estrangeira em uma tabela sempre aponte para um registro válido em outra tabela, evitando a existência de dados órfãos ou inconsistentes no sistema e garantindo a integridade dos dados armazenados (Elmasri & Navathe, 2015).

Esses aspectos consolidaram os bancos de dados relacionais como uma escolha preferencial em setores que exigem precisão e consistência, onde a gestão rigorosa de grandes volumes de dados é essencial.

2.1 Propriedades ACID e sua Importância no Setor Financeiro

Conforme descrito por Elmasri e Navathe (2015), os bancos de dados relacionais implementam as propriedades ACID — Atomicidade, Consistência, Isolamento e Durabilidade — para garantir operações de leitura e escrita confiáveis, especialmente em setores críticos, onde falhas podem ter consequências graves. Cada uma dessas propriedades desempenha um papel específico descrito abaixo.

a) atomicidade: essa propriedade assegura que cada transação seja completamente executada ou integralmente revertida, sem deixar o banco de dados em um estado intermediário. Em uma transferência entre contas bancárias, por exemplo, a atomicidade garante que o débito e o crédito ocorram em conjunto ou que ambas as operações sejam canceladas, evitando inconsistências;

b) consistência: a consistência mantém o banco de dados em estados válidos. Cada transação deve seguir regras e restrições do sistema, o que é essencial para a precisão de processos como balanços financeiros e auditorias;

c) isolamento: essa propriedade assegura que transações concorrentes não interfiram umas nas outras, mantendo a integridade dos dados em operações simultâneas. No contexto financeiro, isso significa que alterações em valores de ativos não serão corrompidas devido a atualizações concorrentes;

d) durabilidade: após confirmada, uma transação é permanentemente gravada no sistema, mesmo em casos de falhas como quedas de energia. Isso garante a imutabilidade dos registros, característica fundamental para a confiabilidade de dados financeiros.

2.2 Escalabilidade e Desafios dos Bancos de Dados Relacionais

Apesar da popularidade e robustez dos bancos de dados relacionais, eles não foram projetados para lidar com os desafios impostos por sistemas distribuídos e

pelo volume crescente de dados gerados por aplicações modernas. Como apontado por Stonebraker e Hellerstein (2005), os bancos de dados relacionais são altamente eficazes para operações transacionais simples, mas começam a enfrentar limitações quando é necessário realizar análises históricas ou processar dados em tempo real em larga escala. Uma das maiores dificuldades dos bancos de dados relacionais é a escalabilidade, que geralmente é viabilizada apenas de forma vertical — ou seja, aumentando o poder de processamento de um único servidor (Stonebraker & Hellerstein, 2005). De modo geral, bancos de dados relacionais dificultam a escalabilidade horizontal, necessária para distribuir o processamento em várias máquinas, tornando suas limitações mais evidentes se houver um crescimento considerável do volume de dados ou se houver a necessidade de processamento distribuído.

Algumas soluções, como o Oracle Real Application Clusters (RAC), permitem uma forma de escalabilidade horizontal para aumentar o poder computacional, embora o armazenamento continue centralizado (Oracle Corporation, 2023). Técnicas como *sharding*[3] também são usadas para distribuir dados entre diferentes servidores, mas, como observa Kleppmann (2017), essas abordagens apresentam desafios significativos em termos de complexidade e limitações arquiteturais em bancos de dados relacionais tradicionais.

Outro desafio associado aos bancos de dados relacionais advém do uso de *joins*, uma operação necessária devido à aplicação de normalização, tema já discutido na seção anterior (Elmasri & Navathe, 2015). Como abordado anteriormente (seção 2.1), a normalização organiza dados em múltiplas tabelas relacionadas para reduzir a redundância e assegurar a integridade dos dados. Embora a normalização traga benefícios significativos para a consistência dos dados, ela cria dependência de *joins* para a reconstrução de informações durante as consultas. O uso extensivo desses *joins*, especialmente em bancos de dados fortemente normalizados com muitas tabelas, pode aumentar significativamente a latência de consultas e reduzir o desempenho, tornando as análises históricas e consultas complexas mais lentas e menos eficientes. Como resultado, muitas

[3] *Sharding* é uma técnica de particionamento de dados que distribui registros entre múltiplos servidores, ou *shards*, para melhorar a escalabilidade e o desempenho de bancos de dados. Cada *shard* armazena uma parte dos dados, permitindo consultas paralelas e aumentando a capacidade de processamento *(KLEPPMANN, 2017).*

empresas começaram a considerar alternativas, como bancos de dados *NoSQL*[4] e bancos de dados imutáveis, que oferecem maior flexibilidade e desempenho em ambientes distribuídos.

2.3 O Modelo Relacional versus Abordagens Alternativas

O surgimento de novas arquiteturas de banco de dados trouxe à tona discussões sobre as limitações do modelo relacional em certas aplicações modernas. Enquanto bancos de dados relacionais oferecem garantias robustas de consistência e de integridade, eles frequentemente enfrentam desafios para escalar horizontalmente, como discutido na seção passada. Kleppmann (2017) observa que, nesse contexto, bancos de dados NoSQL emergiram como alternativas que priorizam a escalabilidade e a capacidade de lidar com grandes volumes de dados. Embora ofereçam flexibilidade e desempenho em sistemas distribuídos, esses bancos acabam tendo que sacrificar algumas das garantias transacionais que caracterizam os sistemas relacionais tradicionais.

Existem diferentes tipos de bancos de dados NoSQL, cada um adequado a cargas de trabalho específicas e a formas variadas de organização de dados. Bancos de dados que armazenam em pares chave-valor, como o Redis (Kleppmann, 2017), permitem consultas rápidas e escaláveis, especialmente em aplicações que exigem armazenamento e recuperação sem relações complexas entre os dados. Já os bancos baseados em documentos, como o MongoDB (Chodorow, 2013), utilizam formatos flexíveis como JSON, tornando-os ideais para aplicações que demandam um esquema menos rígido e adaptação frequente de dados. Existem também os orientados a colunas, como o Cassandra (Lakshman; Malik, 2010), que são otimizados para grandes volumes de dados e processamento analítico, apresentando excelente desempenho em operações de leitura em massa. Esses tipos de bancos de dados NoSQL oferecem flexibilidade e desempenho em ambientes distribuídos, sendo adequados para aplicações que exigem escalabilidade horizontal e tolerância a falhas.

---

[4] *NoSQL* é um modelo de banco de dados projetado para lidar com grandes volumes de dados não estruturados ou semi-estruturados, com flexibilidade em relação ao esquema e suporte a escalabilidade horizontal. Ele é bastante utilizado em aplicações como *big data* e computação distribuída. Exemplos incluem bancos orientados a documentos, chave-valor, grafos e colunas. (CHODOROW, 2013)

Apesar das vantagens dos bancos de dados NoSQL em termos de flexibilidade e escalabilidade, o modelo relacional continua amplamente utilizado principalmente devido à sua maturidade, ferramentas consolidadas e capacidade de lidar com operações transacionais críticas que exigem consistência e integridade. Em muitos casos, a combinação de bancos de dados relacionais com soluções NoSQL pode fornecer o melhor dos dois mundos: a robustez e consistência das transações ACID com a flexibilidade e escalabilidade dos sistemas modernos.

3 BANCOS DE DADOS IMUTÁVEIS

Conforme discutido anteriormente, a necessidade de auditoria detalhada e rastreabilidade em alguns setores como o financeiro, por exemplo, evidencia limitações nos bancos de dados relacionais tradicionais, especialmente em aplicações que exigem um histórico inalterável de dados. Stonebraker e Hellerstein (2005) discutem as limitações dos bancos de dados relacionais nestes contextos, observando que o modelo enfrenta dificuldades para manter uma consistência histórica em ambientes de alta demanda transacional. Nesse cenário, os bancos de dados imutáveis emergem como uma opção, permitindo que cada estado dos dados seja preservado em uma estrutura que registra todas as alterações como novos eventos, sem sobrescrever dados anteriores (Hickey 2012; Kleppmann, 2017). Esse modelo oferece uma base sólida para o controle de versões e uma “verdade histórica” imutável, proporcionando rastreabilidade confiável e auditabilidade integral.

3.1 Estrutura e Funcionamento dos Bancos de Dados Imutáveis

Nos bancos de dados imutáveis cada alteração é registrada em uma estrutura de *append-only log* (registro apenas para adição), criando uma linha do tempo de estados acessível para consultas em qualquer momento passado (Kleppmann, 2017). Essa arquitetura está alinhada ao conceito de *event sourcing*, descrito por Fowler (2005) como uma prática onde cada evento registra uma mudança específica no sistema, permitindo a reconstrução do estado atual a partir da sequência de eventos. Hickey (2012), criador do Datomic, reforça essa abordagem ao definir os dados imutáveis como um "log imutável", em que cada estado é registrado como um carimbo de tempo, garantindo rastreabilidade total e proteção contra manipulações retroativas.

Além do Datomic, outras plataformas como o *EventStore*[5] e o *Blockchain*[6] , aplicam princípios de imutabilidade para assegurar a integridade dos dados históricos. O EventStore, por meio do *event sourcing*, permite a reconstrução precisa do estado do sistema em qualquer ponto no tempo, sendo valioso em sistemas com alta exigência de rastreabilidade (Fowler, 2005). O Blockchain, por sua vez, opera em uma rede distribuída, ideal para contextos onde a segurança e a validação descentralizada dos dados são críticas, especialmente em transações financeiras (Narayanan *et al.*, 2016).

3.2 Vantagens e Desvantagens da Imutabilidade

A imutabilidade oferece vantagens significativas para sistemas críticos, especialmente em setores regulados conforme viemos falando. Uma das principais vantagens é a garantia de auditabilidade: ao armazenar permanentemente cada modificação dos dados como um novo estado, é possível rastrear e analisar todas as transações realizadas. Esse atributo é essencial para atender a requisitos de conformidade regulatória, pois permite demonstrar que uma transação ocorreu de forma precisa e no tempo correto, facilitando auditorias e prevenindo fraudes (Hickey, 2012).

Outro benefício é a simplificação da construção de sistemas distribuídos. Como destaca Kleppmann (2017), ao adotar a imutabilidade, eliminam-se as necessidades de complexos mecanismos de controle de concorrência, como bloqueios e transações distribuídas. Dessa forma, a ordem dos eventos e o estado dos dados são mantidos independentemente do número de nós, permitindo que o sistema escale de maneira previsível. Isso resulta em sistemas distribuídos mais resilientes e menos propensos a inconsistências, uma característica valiosa em ambientes de alta carga e complexidade.

---

[5] É uma prática de modelagem de sistemas onde cada mudança de estado é registrada como um evento imutável em uma sequência cronológica. Essa abordagem é amplamente utilizada para garantir rastreabilidade e auditabilidade em sistemas distribuídos, como explicado por Fowler (2015).

[6] Tecnologia de armazenamento de dados descentralizada e imutável, estruturada como uma cadeia de blocos, onde cada bloco contém um registro de transações e um ponteiro para o bloco anterior. Operando em uma rede distribuída de nós, o *Blockchain* valida e autentica dados de forma coletiva, proporcionando segurança e resistência a fraudes. Isso o torna especialmente útil em transações financeiras e outros cenários que exigem confiança entre múltiplas partes (NARAYANAN *et al.*, 2016).

Entretanto, a imutabilidade apresenta desafios que precisam ser considerados. O aumento no uso de armazenamento é uma das desvantagens mais significativas, pois cada alteração nos dados gera uma nova versão. Em sistemas de grande porte, como os financeiros, onde ocorrem milhares de transações diárias, a infraestrutura precisa ser robusta o suficiente para suportar o volume crescente de dados históricos (Kleppmann, 2017). Além disso, a consulta a versões múltiplas pode tornar as operações de leitura mais complexas e demoradas, o que pode ser um desafio em aplicações que exigem respostas em tempo real ou análises de dados intensivas. Essa complexidade adicional exige planejamento cuidadoso na implementação de consultas e no gerenciamento de dados para manter o desempenho adequado em aplicações críticas.

Outro ponto que pode ser um desafio para esse tipo de armazenamento está relacionado à conformidade com a Lei Geral de Proteção de Dados (LGPD) e outras regulamentações de proteção de dados pessoais. A LGPD, assim como o GDPR europeu, exige que os dados pessoais possam ser corrigidos, excluídos ou anonimizados a pedido dos titulares, introduzindo o "direito ao esquecimento" e impondo o desafio de exclusão definitiva dos dados históricos (BRASIL, 2018). Em bancos de dados imutáveis, onde todas as alterações são preservadas permanentemente como novos eventos, atender a essas exigências pode ser complexo, pois a estrutura foi projetada para manter um registro histórico completo, o que dificulta a remoção ou anonimização total dos dados.

Para superar essas limitações e estar em conformidade com a LGPD, implementações adicionais, como criptografia e técnicas de anonimização irreversível, podem ser necessárias, embora isso envolva custo e complexidade significativos (SOUZA *et al.*, 2020). Essa característica representa um desafio para bancos de dados imutáveis, especialmente em aplicações que armazenam dados pessoais sensíveis, e destaca a importância de considerar cuidadosamente o uso de imutabilidade em sistemas que devem atender a regulamentações de privacidade.

3.3 Aplicações no Setor Financeiro

A imutabilidade dos dados se mostra particularmente valiosa em aplicações financeiras, onde a integridade e a rastreabilidade das informações são requisitos

fundamentais. Em plataformas de investimento, por exemplo, a imutabilidade permite que decisões críticas sejam embasadas em dados históricos confiáveis, uma vez que todos os estados anteriores dos dados são preservados e podem ser consultados de maneira auditável (Stonebraker; Hellerstein, 2005). Esse nível de preservação histórica torna as projeções e análises mais precisas e seguras, facilitando a rastreabilidade necessária para as auditorias.

Outro uso essencial dos bancos de dados imutáveis no setor financeiro está na conformidade regulatória. Instituições financeiras precisam cumprir requisitos rigorosos de auditoria e de regulamentação, como as diretrizes do Comitê de Supervisão Bancária da Basileia, que exigem manutenção de registros completos de todas as transações[7]. Os bancos de dados imutáveis respondem a essa necessidade ao armazenar cada transação de forma permanente e rastreável, simplificando o acesso ao histórico completo das operações para fins de auditoria e controle (Kleppmann, 2017). Como destaca Kleppmann, “a imutabilidade permite construir sistemas em que a confiabilidade e a auditabilidade são inerentes à estrutura dos dados”, garantindo que o registro das operações reflita com precisão todos os estados.

No contexto de negociações de alta frequência, onde múltiplos participantes realizam operações simultâneas, a imutabilidade proporciona uma camada adicional de segurança ao assegurar que as operações e registros sejam mantidos em sequência cronológica e sem possibilidade de reversão, minimizando o risco de inconsistências no histórico das transações.

3.4 Comparação com o Modelo Relacional

A comparação entre bancos de dados imutáveis e relacionais destaca propósitos e estruturas diferentes. Bancos de dados relacionais foram projetados para oferecer flexibilidade e desempenho em ambientes transacionais, permitindo atualizações e exclusões frequentes. Isso os torna ideais para sistemas que

---

[7] O Comitê de Supervisão Bancária da Basileia estabelece normas internacionais para a gestão e auditoria de riscos em instituições financeiras, recomendando práticas de agregação e relatórios de dados de risco que garantam transparência e integridade das informações. Para mais detalhes, ver: BANK FOR INTERNATIONAL SETTLEMENTS. *Basel Committee on Banking Supervision: Principles for Effective Risk Data Aggregation and Risk Reporting*. Basel: BIS, 2011. Disponível em: https://www.bis.org/publ/bcbs239.htm. Acesso em: 27 out. 2024.

requerem alto desempenho em consultas e operações de leitura e escrita. Em contraste, os bancos de dados imutáveis são projetados para preservar todas as versões dos dados, conforme explicado nos capítulos anteriores, especialmente valiosa em contextos regulados por garantir transparência.

Stonebraker e Hellerstein (2005) observam que, embora bancos relacionais sejam otimizados para consultas em tempo real, eles podem enfrentar desafios em cenários onde a integridade e a preservação do histórico completo são indispensáveis. Assim, enquanto bancos de dados imutáveis são ideais para rastreabilidade e conformidade regulatória, bancos de dados relacionais continuam sendo preferidos em sistemas que exigem alta eficiência em transações rápidas e operações contínuas de atualização.

3.5 Considerações do Capítulo

Este capítulo discutiu as principais características dos bancos de dados relacionais e imutáveis, bem como suas vantagens e desvantagens em aplicações financeiras. A flexibilidade e a alta performance dos bancos de dados relacionais os tornam amplamente utilizados em sistemas críticos. Por outro lado, bancos de dados imutáveis oferecem uma solução robusta para contextos que demandam auditabilidade e preservação histórica, como ocorre no setor financeiro. A combinação dessas duas abordagens pode atender tanto às necessidades de escalabilidade e desempenho quanto à conformidade regulatória e integridade dos dados, compondo uma estratégia balanceada para atender aos requisitos de sistemas financeiros modernos.

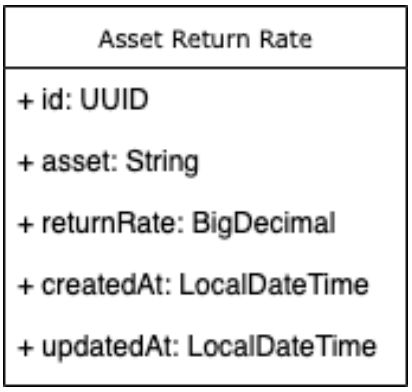
4 PROTÓTIPO

4.1 Arquitetura Geral e Funcionalidades

O protótipo desenvolvido neste estudo visou avaliar o desempenho e a adequação dos bancos de dados relacional e imutável em um sistema de cálculo de projeções de investimentos. Para isso, foi construído um micro serviço em Java utilizando o *framework* Spring Boot[8], configurado para operar com duas instâncias locais de banco de dados: o PostgreSQL e o Datomic. Com o objetivo de promover a transparência e permitir a replicação dos resultados, o código-fonte foi disponibilizado em um repositório público no GitHub[9].

O diagrama apresentado na Figura 1 descreve o modelo de dados principal utilizado, implementado em ambos os bancos de dados, com atributos como identificação, nome, taxa de retorno e datas de criação e atualização.

Figura 1 – Diagrama de Classes do Modelo de Dados Persistido



Fonte: Elaborado pela autora (2024).

No entanto, no PostgreSQL, para implementar o controle temporal foi utilizado o *Hibernate Envers*. Esse *framework* gera tabelas auxiliares automaticamente, fornecendo mecanismos para rastreamento e acesso a estados históricos dos dados, o que possibilita a recuperação de versões anteriores das entidades armazenadas. A Figura 2 ilustra o relacionamento entre a tabela principal e as

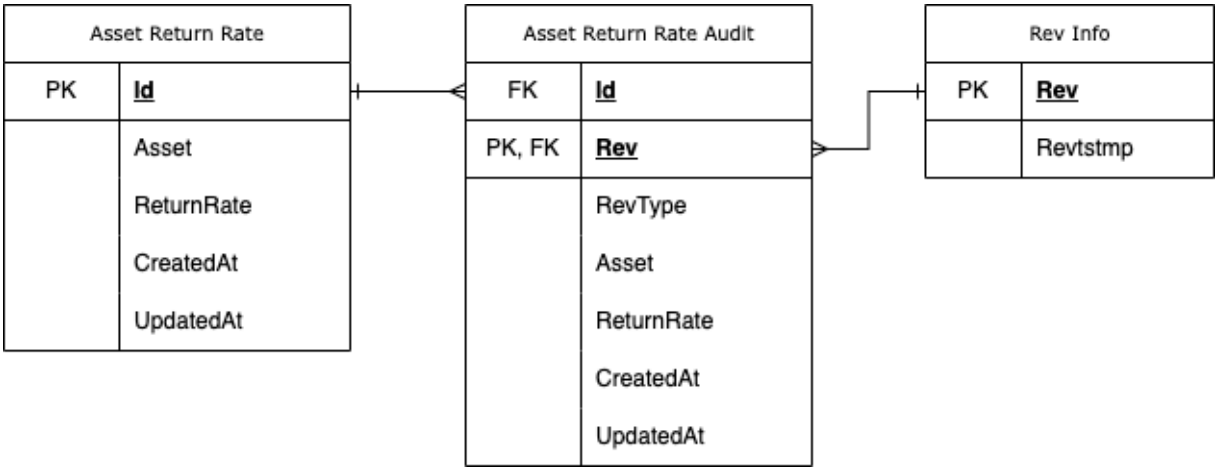
---

[8] Segundo Rajput (2018), o Spring Boot é um framework de código aberto para Java que facilita o desenvolvimento de aplicações autônomas e escaláveis, oferecendo configuração simplificada e inicialização automática. Para mais detalhes consulte a documentação no site https://docs.spring.io/spring-boot/index.html

[9] Disponível em https://github.com/karymereis1/return-calculator.

tabelas auxiliares geradas pelo *Hibernate Envers*. Esse recurso permite consultas temporais similares ao método *asOf*[10] do Datomic, que oferece acesso a estados passados dos dados por meio de um registro imutável de alterações.

Figura 2 – Diagrama Entidade Relacionamento do *Hibernate Envers*



Fonte: Elaborado pela autora (2024).

Essas funcionalidades foram essenciais para garantir que as consultas temporais realizadas nos dois bancos de dados fossem comparáveis para que fossem possíveis as análises realizadas ao longo deste trabalho. O micro serviço contou com três APIs[11] principais:

a) API de criação e atualização dos dados: serviu para realizar tanto a criação, quanto a atualização diária das taxas de retorno dos ativos fictícios, permitindo o armazenamento de dados históricos em ambos os bancos;

b) API de consulta dos dados: foi utilizada para validar a consistência dos dados para fins comparativos nos dois bancos, podendo ser utilizada com e sem o parâmetro de data;

---

[10] O método *asOf* do Datomic é uma funcionalidade que permite consultar o estado de uma base de dados em um momento específico no passado, garantindo acesso a um histórico imutável dos dados. Disponível em: https://docs.datomic.com/client-tutorial/history.html#as-of-query. Acesso em: 05 dez. 2024.

[11] Uma API (Application Programming Interface) é um conjunto de definições e protocolos que permitem a comunicação entre diferentes sistemas de software, facilitando a integração e o intercâmbio de dados. Fonte: FIELDING, R. T. Architectural Styles and the Design of Network-based Software Architectures. University of California, Irvine, 2000.

c) API de cálculo de projeções: permitiu calcular a projeção de retorno para uma carteira de investimento com base nas alocações informadas. A API também possibilitou o cálculo para uma data específica, utilizando a taxa de retorno registrada naquela data, ou, se ausente, usa a taxa atual.

4.2 Estrutura do Experimento

Para avaliar a implementação do protótipo e garantir que ele atende aos requisitos estabelecidos, a execução do experimento informal foi realizada manualmente utilizando o Postman[12]. Essa ferramenta foi escolhida por sua flexibilidade e facilidade de uso, servindo para verificar o retorno das APIs em diferentes cenários de consulta.

4.2.1 Medições Realizadas

As execuções realizadas com o Postman tiveram como objetivo aferir características do protótipo em relação a cada banco de dados, alinhadas aos requisitos da ISO/IEC 25010:2011 sobre modelos de qualidade de sistemas e software (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011). As medições focaram em três características de qualidade:

a) comportamento temporal (*Time Behavior*): avaliar o tempo necessário para que as APIs retornem os resultados das projeções de investimentos e dos processos de atualização de dados. Essa métrica, parte da característica de Eficiência de Desempenho (*Performance Efficiency*), foi aferida medindo tempos médios, mínimos e máximos de resposta das APIs em diferentes cenários de consulta;

b) adequação funcional (*Functional Suitability*): garantir que as APIs retornem os mesmos resultados para ambos os bancos de dados tanto ao realizar cálculos de projeção, quanto ao consultar os dados brutos em datas

[12] O Postman é uma ferramenta de desenvolvimento utilizada para testar, monitorar e documentar APIs. Ele permite que desenvolvedores realizem requisições HTTP, analisem os retornos das APIs e automatizem testes de integração. É amplamente utilizado devido à sua interface intuitiva e capacidade de gerenciar cenários complexos de teste. Fonte: POSTMAN. *API Platform for Building and Using APIs*. Disponível em: https://www.postman.com/. Acesso em: 27 out. 2024.

específicas. Essa avaliação garante que os dados retornados são corretos em ambos cenários;

c) auditabilidade (*Accountability*): analisar a capacidade dos bancos de dados de fornecer um histórico completo e confiável de alterações nos dados, de forma a atender a requisitos de rastreabilidade e conformidade regulatória. Essa característica foi avaliada verificando a consistência e a precisão dos dados históricos retornados por ambos os bancos e a facilidade de reconstruir estados passados utilizando o *asOf* no Datomic e o *Hibernate Envers* no PostgreSQL.

4.2.2 Ferramentas Utilizadas

As ferramentas e o ambiente configurados para a realização dos testes foram escolhidos para oferecer uma análise manual do comportamento do sistema e dos bancos de dados. No entanto, é importante observar que este experimento foi conduzido de forma exploratória, sem os rigores necessários para assegurar validade estatística. Essa limitação indica a necessidade de trabalhos futuros que implementem experimentos mais formais e metodologias estatísticas adequadas.

a) Postman: versão 11.18.4, utilizado para enviar requisições às APIs de forma manual, registrar os tempos de resposta e inspecionar os dados retornados. A ferramenta foi útil para verificar a funcionalidade das APIs e identificar diferenças iniciais no desempenho dos bancos de dados. No entanto, sua aplicação não contempla cenários automatizados ou de alta carga;

b) bancos de dados:

- *PostgreSQL:* versão 14.13 (compilado via *Homebrew*), configurado localmente para armazenar os dados históricos das taxas de retorno dos ativos;
- *Datomic:* versão Free Edition, também configurado localmente, com funcionalidades nativas de controle temporal utilizando *asOf*. Ambas as instâncias foram inicializadas com os mesmos dados para garantir comparabilidade;

c) *frameworks* utilizados:

- *Spring Boot:* versão 3.3.3, usada para desenvolver o micro serviço, fornecendo integração simplificada com o Hibernate e APIs RESTful;
- *PostgreSQL Driver:* versão 42.7.2, garantindo compatibilidade com o banco de dados relacional PostgreSQL;
- *Hibernate Envers:* versão 6.5.2, utilizado para rastreamento de estados históricos no PostgreSQL;
- *Datomic Peer:* versão 1.0.7075, configurado para suportar o modelo imutável de dados;
- *Jackson Databind:* versão 2.15.0, responsável pela serialização e desserialização de objetos JSON;

d) máquina de execução:

- *Sistema Operacional:* macOS 14.7.1 (23H222);
- *Processador:* 2,6 GHz Intel Core i7 6-Core;
- *Memória RAM:* 16 GB 2667 MHz DDR4;
- *Ambiente de Execução:* Java 17, compatível com Spring Boot e Hibernate Envers;

e) planilha de comparação de resultados[13]: utilizada para registrar manualmente os tempos de resposta e os resultados das projeções retornados por ambas as APIs. Essa abordagem, embora limitada, foi suficiente para identificar discrepâncias significativas de desempenho e consistência.

Essa abordagem, embora informal, forneceu uma visão geral do comportamento do sistema e dos bancos de dados no contexto específico do protótipo, mas conforme dito inicialmente, foi limitada em termos de rigor metodológico. Trabalhos futuros poderiam se concentrar na execução de experimentos mais formais, incorporando medições automatizadas, validação estatística dos resultados e cenários de carga mais realistas. Tais aprimoramentos ajudariam a consolidar os achados e torná-los aplicáveis a contextos reais do setor financeiro.

[13] Disponível em: https://l1nk.dev/6u1W6

4.2.3 Procedimentos

O experimento foi conduzido seguindo os passos abaixo, com o objetivo de aferir as características descritas na seção anterior, garantindo consistência e comparabilidade entre os dois bancos de dados.

4.2.3.1 Configuração Inicial

a) criação de dados: mil registros de ativos fictícios foram criados com informações de identificação, nome e taxa de retorno esperado, sincronizados em ambos os bancos de dados (PostgreSQL e Datomic). Essas informações garantem que ambos os bancos contenham dados equivalentes;

b) atualizações diárias: os registros foram atualizados diariamente durante dois meses, com taxas de retorno fictícias geradas dentro de intervalos plausíveis, simulando condições reais de variação de mercado;

c) definição de pontos temporais: datas específicas (como 01 de outubro e 15 de outubro) foram escolhidas como referência para realizar consultas históricas e verificar a capacidade dos bancos de retornar estados passados dos dados.

4.2.3.2 Execuções

As APIs de projeção de retornos para carteiras de investimento e de consulta de dados brutos foram acionadas para realizar consultas em dois cenários: projeções para a data atual, sem o parâmetro de data, e projeções para pontos históricos específicos, utilizando o parâmetro de data. Os dados retornados por ambas as APIs foram verificados e comparados entre os dois bancos, permitindo conferir a consistência dos resultados e validar a adequação funcional de cada solução.

4.2.3.3 Medição do Tempo de Resposta

O Postman foi utilizado para medir manualmente o tempo de resposta das APIs em diferentes cenários. Conforme explicado na seção anterior a esta, foram realizadas consultas para a data atual e para pontos históricos específicos. Os tempos de resposta mínimo, médio e máximo foram registrados em uma planilha,

possibilitando a identificação de diferenças de desempenho entre PostgreSQL e Datomic.

4.2.3.4 Avaliação de Auditabilidade

a) consultas temporais: foram realizadas consultas específicas para datas passadas utilizando o método *asOf* no Datomic e o *Hibernate Envers* no PostgreSQL;

b) validação de estados históricos: os dados retornados foram comparados para garantir que cada banco de dados fosse capaz de recuperar estados históricos precisos e consistentes, avaliando a rastreabilidade das alterações.

5 ANÁLISE DE RESULTADOS

A análise dos resultados foi conduzida com base nas medições realizadas, conforme descrito no capítulo anterior, abrangendo três principais dimensões de qualidade do protótipo: comportamento temporal, adequação funcional e auditabilidade. Esses aspectos foram avaliados para os dois bancos de dados utilizados — PostgreSQL e Datomic — em diferentes cenários de consulta e atualização.

5.1 Comportamento Temporal

O comportamento temporal, associado ao tempo de resposta das APIs, foi analisado em dois contextos principais: atualização diária dos dados e consultas para realizar o cálculo de projeção de investimentos.

5.1.1 Atualização Diária

Os tempos médios observados para atualizar mil ativos fictícios foram os seguintes:

a) PostgreSQL: ≈ 5,55 segundos;

b) Datomic: ≈ 14,17 segundos.

O PostgreSQL apresentou tempos significativamente menores, o que reflete sua abordagem de atualização direta dos registros existentes. Já o Datomic, devido à natureza imutável de sua arquitetura, registra cada atualização como um novo evento, o que aumenta a sobrecarga computacional e o tempo necessário para processar essas operações.

5.1.2 Consultas de Projeção de Investimentos

Os tempos de resposta para realizar o cálculo de projeção sem parâmetro de data (expectativa de retorno atual dos ativos) e com data específica (expectativa de retorno histórica dos ativos) revelaram diferenças significativas entre o desempenho do PostgreSQL e do Datomic.

5.1.2.1 Sem Parâmetro de Data (projeção atual)

a) PostgreSQL: ≈ 15 milissegundos na primeira chamada e uma média de ≈ 8,56 milissegundos para chamadas subsequentes;

b) Datomic: ≈ 39 milissegundos na primeira chamada e uma média de ≈ 24,56 milissegundos para chamadas subsequentes.

Nessa categoria, o PostgreSQL apresentou um desempenho significativamente superior ao Datomic. Mesmo na primeira execução, o PostgreSQL foi quase três vezes mais rápido do que o Datomic. Após o *caching*[14], a vantagem do PostgreSQL tornou-se ainda mais evidente, com tempos subsequentes reduzidos para menos da metade do tempo do Datomic.

5.1.2.2 Com Parâmetro de Data (projeção histórica)

a) PostgreSQL: ≈ 37 milissegundos na primeira chamada e uma média de ≈ 28 milissegundos para chamadas subsequentes;

b) Datomic: ≈ 26 milissegundos na primeira chamada e uma média de ≈ 24,67 milissegundos para chamadas subsequentes.

Ao contrário do cenário de projeção atual apresentado na seção 5.1.2.1, o Datomic demonstrou superioridade em projeções históricas. Na primeira execução, o Datomic foi cerca de 30% mais rápido que o PostgreSQL. Esse padrão foi mantido nas chamadas subsequentes, com o Datomic apresentando tempos mais consistentes.

5.1.2.3 Análise Geral

Os resultados do experimento demonstraram diferenças evidentes no desempenho dos bancos de dados PostgreSQL e Datomic, refletindo suas arquiteturas e características específicas. Nas projeções para a data atual, o PostgreSQL mostrou-se claramente superior, com tempos de resposta

[14] *Caching* é uma técnica para armazenar temporariamente dados frequentemente acessados em uma memória de acesso rápido, reduzindo o tempo de resposta do sistema e otimizando o desempenho. Essa abordagem é amplamente utilizada em sistemas distribuídos e aplicações web para minimizar a latência e o uso de recursos de hardware. (CHAPPELL, D, *Enterprise Service Bus*. O'Reilly Media, 2004)

significativamente menores, especialmente em chamadas subsequentes, beneficiando-se de mecanismos de *caching*. Mesmo na primeira execução, o PostgreSQL foi consideravelmente mais rápido do que o Datomic. No entanto, em consultas históricas, o Datomic se destacou, apresentando tempos de resposta mais rápidos e consistentes, mesmo na ausência de *caching*. Essa vantagem é atribuída à sua arquitetura imutável e suporte nativo a consultas temporais, enquanto o PostgreSQL depende de tabelas auxiliares geradas pelo *Hibernate Envers*, que introduzem maior complexidade ao processamento histórico.

A Tabela 1 abaixo apresenta um registro detalhado de todos os tempos de execução para os diferentes cenários analisados.

Tabela 1 – Tempos de resposta para as execuções da API de cálculo de projeção

	Tempo de Resposta* (sem parâmetro de data)		***Tempo de Resposta* (com parâmetro de data)***	
Data	**PostgreSQL**	**Datomic**	**PostgreSQL**	**Datomic**
2024-12-05**	15	39	37	26
2024-12-05	9	24	28	24
2024-12-05	9	24	28	24
2024-12-05	8	24	28	25
2024-12-05	9	25	29	25
2024-12-05	9	24	28	24
2024-12-05	8	24	28	25
2024-12-05	8	25	27	24
2024-12-05	8	25	28	25
2024-12-05	9	26	28	26
Média Parcial	**8,56**	**24,56**	**28,00**	**24,67**

**tempo em milisegundos*

***primeira execução sem caching*

Fonte: Elaborado pela autora (2024).

Esses dados reforçam a observação de que o PostgreSQL pode ser uma boa escolha para aplicações que priorizam operações correntes e alta performance em consultas em tempo real, enquanto o Datomic é mais adequado para sistemas que demandam forte rastreabilidade e acesso eficiente a estados históricos.

5.2 Adequação Funcional

A adequação funcional foi avaliada verificando se as APIs retornaram os mesmos resultados para ambos os bancos ao calcular projeções de investimentos em datas específicas. Em todos os testes realizados, os resultados foram consistentes entre os dois bancos de dados, confirmando que o protótipo atende aos requisitos funcionais definidos. Essa consistência evidencia a capacidade de ambos os bancos de representar e acessar corretamente os dados necessários para os cálculos, mesmo em cenários que demandam consultas temporais.

5.3 Auditabilidade

A auditabilidade foi analisada com base na capacidade dos bancos de fornecerem um histórico completo e confiável de alterações nos dados. Os seguintes aspectos foram observados:

a) consistência dos dados históricos: ambos os bancos retornaram informações precisas e consistentes ao acessar estados passados, demonstrando conformidade com os requisitos de rastreabilidade;

b) facilidade de reconstrução de estados:

   - No PostgreSQL, o *Hibernate Envers* foi utilizado para rastrear e acessar alterações, permitindo reconstruir versões anteriores de entidades armazenadas;

   - No Datomic, o método *asOf* possibilitou acessar diretamente os estados passados, oferecendo maior simplicidade e eficiência nesse processo.

Embora ambos os bancos atendam aos requisitos de auditabilidade, o Datomic demonstrou maior praticidade para acessar estados históricos, o que pode ser uma vantagem em cenários que exigem alta frequência de consultas temporais.

5.4 Observações Complementares

Durante o experimento, foram observados alguns aspectos informais que, embora não façam parte das medições principais, podem complementar a análise:

a) uso de armazenamento local: o PostgreSQL ocupou aproximadamente 18MB de espaço no disco local, com a maior parte sendo utilizada pelas tabelas de auditoria geradas pelo Hibernate Envers. Já o Datomic registrou um uso de aproximadamente 39MB, refletindo o armazenamento imutável que preserva todas as versões dos dados como eventos independentes;

b) quantidade de registros: durante as verificações realizadas, observou-se que ambos os bancos registraram os mesmos 1.000 ativos fictícios, refletindo as atualizações diárias realizadas ao longo de dois meses, com algumas interrupções ocasionais durante os finais de semana. Foi possível contar um total de aproximadamente 50 mil registros na tabela de auditoria do PostgreSQL, representando todas as transações realizadas ao longo do período. Já no Datomic, o total de transações foi estimado manualmente utilizando consultas ao log de transações.

Essas verificações reforçam que os dados foram devidamente sincronizados entre os bancos, garantindo condições adequadas para os testes de desempenho e funcionalidade. No entanto, como não foram empregadas ferramentas formais para validação estatística, esses números foram tratados como informações complementares e não foram incluídos diretamente como parte das medições principais do trabalho.

5.5 Reflexões Finais

Os resultados obtidos reforçam que a escolha entre bancos de dados relacionais e imutáveis deve ser guiada pelas necessidades específicas da aplicação:

a) PostgreSQL demonstrou superioridade em atualizações de dados e consultas imediatas, sendo uma solução robusta para cenários de alta demanda transacional;

b) Datomic, por sua vez, destacou-se em consultas históricas e auditorias, com maior facilidade para rastrear e acessar estados passados.

Essa análise oferece uma visão detalhada do desempenho e da adequação dos dois modelos de banco de dados, contribuindo para decisões informadas sobre

tecnologias de armazenamento em sistemas financeiros. Trabalhos futuros podem explorar medições adicionais, incluindo uso de armazenamento em contextos mais robustos e análise de impacto em ambientes distribuídos.

6 CONSIDERAÇÕES FINAIS

6.1 Conclusões

Este trabalho avaliou o desempenho e a adequação de dois paradigmas distintos de bancos de dados, o relacional (representado pelo PostgreSQL) e o imutável (representado pelo Datomic), aplicados a um sistema de cálculo de projeções de investimentos. Utilizando métricas alinhadas à ISO 25010, foi possível explorar características como comportamento temporal, adequação funcional e auditabilidade.

Os resultados demonstraram que o PostgreSQL é mais eficiente em operações de escrita e consultas de dados em tempo real, características que justificam sua ampla adoção em sistemas transacionais. Por outro lado, o Datomic destacou-se em consultas históricas, com uma arquitetura que facilita o rastreio temporal e preserva a integridade de dados passados, características importantes para auditorias e conformidade regulatória.

Ambos os bancos apresentaram consistência funcional ao retornar resultados equivalentes nas consultas realizadas. A adaptação do PostgreSQL, com o uso do Hibernate Envers, permitiu replicar funcionalidades de imutabilidade, ainda que com maior esforço de configuração. Assim, a escolha entre os paradigmas deve considerar as prioridades específicas de cada sistema, seja desempenho em tempo real ou rastreabilidade.

6.2 Contribuições do Trabalho

Este estudo buscou oferecer contribuições para a compreensão e o uso de bancos de dados em sistemas críticos. A implementação de um protótipo funcional e o uso de tecnologias de mercado, como *Spring Boot*, *Hibernate Envers*, PostgreSQL e Datomic, permitem uma análise prática e aplicável a cenários reais.

Os resultados fornecem uma base para a escolha de tecnologias, considerando aspectos como adequação funcional e auditabilidade. Além disso, o estudo reforça a necessidade de análise criteriosa dos *trade-offs* envolvidos em

cada paradigma de banco de dados, considerando os requisitos específicos do sistema.

Outra contribuição está na aplicação das métricas da ISO 25010, que trouxe uma abordagem sistemática para avaliar as características de qualidade do sistema. Isso pode inspirar futuros estudos a adotarem padrões similares para comparações mais robustas e fundamentadas.

6.3 Trabalhos Futuros

Diversas oportunidades de aprofundamento emergem deste trabalho. Uma das limitações foi o caráter manual dos testes realizados. Trabalhos futuros poderiam utilizar ferramentas mais avançadas de medição e análise estatística para aumentar a confiabilidade dos resultados, especialmente no que diz respeito ao desempenho e à escalabilidade.

Outro ponto de investigação é a análise de custos operacionais dos dois paradigmas, incluindo não apenas o desempenho técnico, mas também os custos associados à manutenção, infraestrutura e licenciamento. Expandir o estudo para incluir outros tipos de bancos de dados, como NoSQL orientados a documentos ou colunas, poderia fornecer insights adicionais sobre sua aplicabilidade em sistemas financeiros.

Por fim, considerando as exigências legais impostas por regulamentações como a LGPD, estudos futuros podem explorar como bancos de dados, tanto relacionais quanto imutáveis, podem lidar com a anonimização, exclusão e portabilidade de dados. Esse tema, identificado como um desafio no contexto atual, é especialmente relevante para organizações que precisam equilibrar conformidade regulatória com eficiência tecnológica.

REFERÊNCIAS BIBLIOGRÁFICAS

CHODOROW, K. **MongoDB: The Definitive Guide**. 2. ed. Beijing: O'Reilly Media, 2013.

CFI TEAM. **Big Data in Finance - Definition, Uses, Challenges**. Corporate Finance Institute, 2020. Disponível em: https://corporatefinanceinstitute.com/resources/data-science/big-data-in-finance/. Acesso em: 20 out. 2024.

ELMASRI, R.; NAVATHE, S. B. **Fundamentals of Database Systems**. 7. ed. Pearson, 2015.

FOWLER, M. **Event Sourcing**. martinfowler.com, 2005. Disponível em: https://martinfowler.com/eaaDev/EventSourcing.html. Acesso em: 27 out. 2024.

HICKEY, R. **Datomic Documentation**. Datomic, 2012. Disponível em: https://docs.datomic.com/datomic-overview.html#about. Acesso em: 13 out. 2024.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ISO/IEC 25010:2011. **Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models**. Geneva: ISO, 2011.

KLEPPMANN, M. **Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems**. Sebastopol: O'Reilly Media, 2017.

LAKSHMAN, A.; MALIK, P. **Cassandra - a decentralized structured storage system**. In: ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware. New York: ACM, 2010. p. 1-10.

NARAYANAN, A.; BONNEAU, J.; FELTEN, E.; MILLER, A.; GOLDFEDER, S. **Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction**. Princeton: Princeton University Press, 2016.

ORACLE CORPORATION. **Oracle Real Application Clusters (RAC) Documentation**. Disponível em: https://docs.oracle.com/en/database/oracle/oracle-database/19/racdq/. Acesso em: 20 out. 2024.

PEFFERS, K. et al. **A Design Science Research Methodology for Information Systems Research**. Journal of Management Information Systems, v. 24, n. 3, p. 45-77, 2007.

RAJPUT, D. **Mastering Spring Boot 2.0: Build modern, cloud-native, and distributed systems using Spring Boot**. Birmingham: Packt Publishing, 2018.

REIS, K. ***Return Calculator*: sistema para projeções de retorno de investimentos utilizando bancos de dados relacional e imutável**. 2024. Disponível em: https://github.com/karymereis1/return-calculator. Acesso em: 05 dez. 2024.

STONEBRAKER, M.; HELLERSTEIN, J. M. **Readings in Database Systems**. 4. ed. The MIT Press, 2005