Universidade de Rio Verde
Faculdade de Engenharia de Software
Curso: Ciência de Dados e Machine Learning
Disciplina: Banco de Dados Relacional e Não Relacional

Trabalho Prático e Final da Disciplina

Professor: *Dr. Willian Garcias de Assunção - williang@unirv.edu.br*
Acadêmicos: *Emanuel Alves, Felipe Peretti, Thiago Siqueira*

Sumário

1 Introdução e Objetivo

Este trabalho tem como objetivo a aplicação integrada dos conceitos de bancos de dados relacionais e não relacionais estudados na disciplina. O foco é a implementação de uma solução baseada no padrão de **Persistência Poliglota**, onde diferentes tecnologias de armazenamento de dados são selecionadas para atender a requisitos específicos de um sistema. A atividade consiste em projetar e prototipar a camada de persistência de dados para um sistema de negócio definido.

2 Arquitetura

A arquitetura final da camada de persistência foi definida com base na responsabilidade dos dados. O Postgres SQL foi utilizado como banco de dados relacional, com o propósito de armazenar dados criticos para a aplicação, ou seja, informações que exigem alta consistência, integridade transacional e durabilidade, o diagrama conceitual pode ser visto na Figura 1. Neste modelo relacional, foram definidas quatro entidades: cliente, transações financeiras, pedidos e item do pedido, onde a entidade cliente armazena dados básicos sobre o cliente tendo como chave primária o id cliente, a entidade pedidos armazena informações importantes referentes a pedidos realizados por um cliente, desta forma foi necessário utilizar o id cliente como chave estrangeira para realizar o relacionamento. A entidade itens do pedido é um relacionamento entre a entidade pedidos e produtos onde um pedido pode ter vários produto e um produto pode

estar em vários pedidos. Foi adicionado também uma entidade de transações para armazenar dados sobre o pagamento do pedido, para realizar este relacionamento foi adicionado uma chave estrangeira com id do pedido.

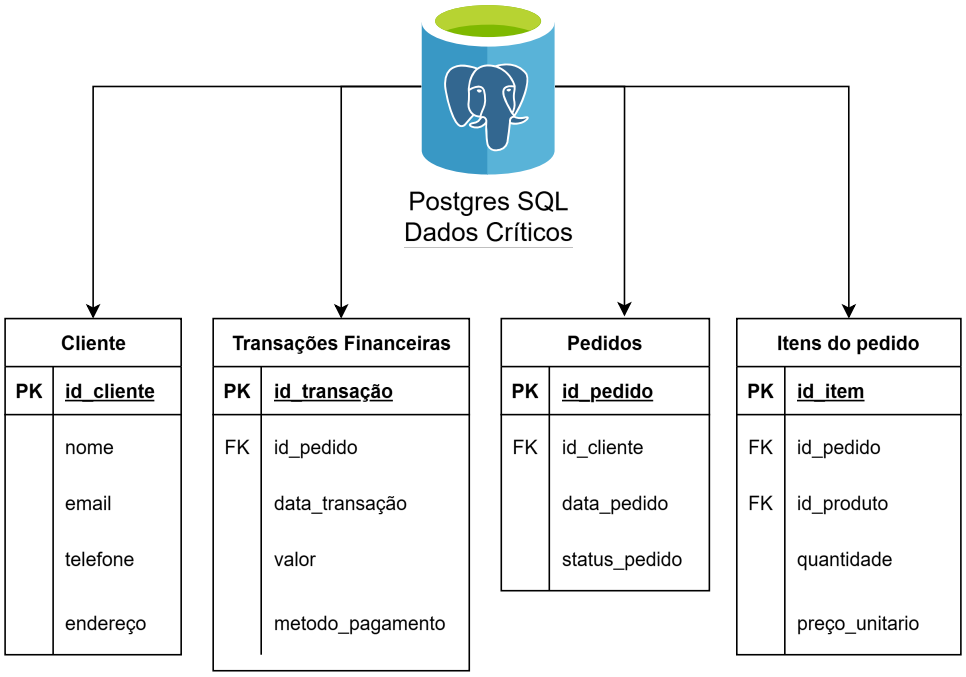

Figura 1: Diagrama conceitual modelo de dados relacional

O MongoDB foi utilizado nesta aplicação como banco de dados não relacional baseado em documentos, com o objetivo de beneficiar a flexibilidade de um esquema dinâmico, como por exemplo, a entidade produto que pode ter um esquema dinâmico, e não seria apropriado para um banco relacional. O diagrama conceitual das entidades da base de dados baseado em documentos pode ser visualizado na Figura 2. Foi definido uma coleção de produtos, onde foram mantidos alguns atributos que definem um produto, fornecendo a flexibilidade de manter o esquema dinâmico, os documentos sao referenciados pelo atributo id produto, também foi mantido uma coleção de perfil de usuário onde o documento é referenciado pelo id do cliente.

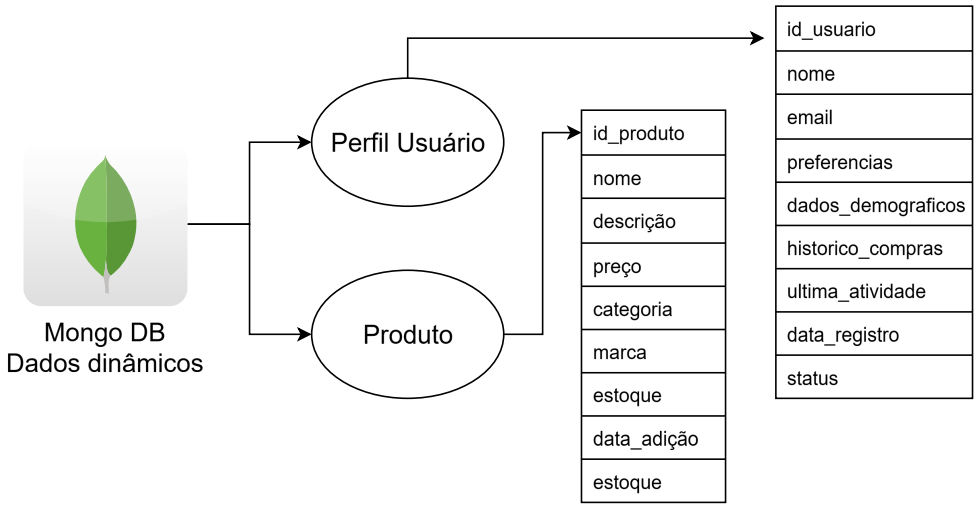

Figura 2: Diagrama conceitual modelo de documentos

Para este projeto foi utilizado o Redis, um banco de dados não relacional do tipo chave-valor para armazenar dados temporários. O Redis foi selecionado para armazenar sessões de usuários, produtos acessados recentemente e outros dados que exigem uma disponibilidade muito alta e por outro lado não dependem de uma persistência de longo prazo, além de diminuírem o custo operacional para as demais bases de dados. Foram definidas algumas chaves importantes para este projeto, que foram importantes e essenciais para o funcionamento esperado.

User session armazena uma chave hash que é um tipo de chave que pode armazenar múltiplos valores, entre estes valores temos informações de login e sessão do usuário, sendo que a chave é a própria sessão do usuário, bastando apenas essa chave para recuperar os valores relacionados a sessão deste usuário. Outra chave definida foi o *user cart* que armazena um hash com múltiplos valores estes valores são id de itens que o usuário adicionou em seu carrinho. Na chave *user preferences* armazenamos valores referente a preferencias do usuário com escopo de sessão, por exemplo, idioma do sistema, moeda corrente, tema e notificações.

Outra chave adicionada foi a de últimos produtos visualizados, que armazena os últimos produtos visualizados pelo usuário naquela sessão. O modelo conceitual pode ser visto na Figura 3.

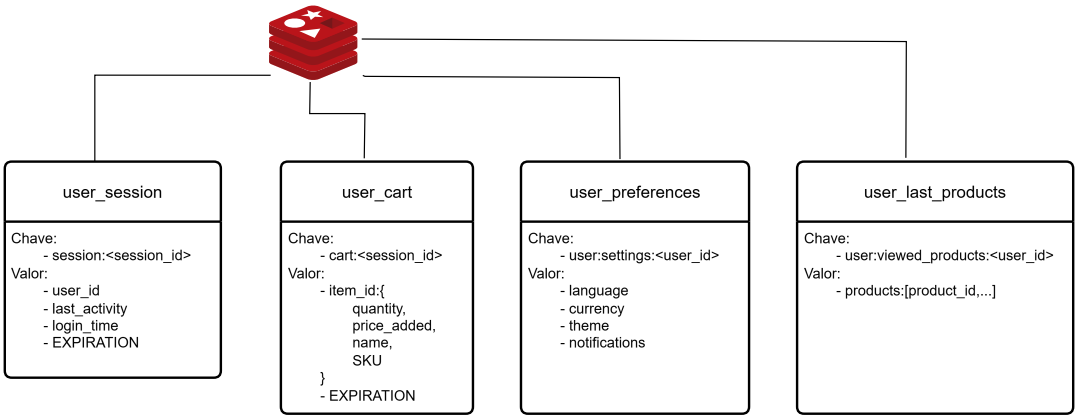

Figura 3: Diagrama conceitual modelo chave-valor

O ClickHouse, um banco de dados colunar, foi utilizado nesta arquitetura para persistir eventos do usuário como cliques em determinados produtos, termos de busca e interações do usuário com o sistema. Estes dados foram utilizados de forma analítica para gerar insights operacionais sobre o comportamento do usuário. Um banco de dados colunar foi selecionado para este caso pois o volume de ingestão de dados é muito alto além de exigir muitas consultas agregadas com alto volume de dados, tarefa esta que não seria performático e com alto custo computacional em outros modelos de bases de dados. Para este modelo colunar foram definidas as entidades de eventos e consulta analática, apresentados na Figura 4.

Eventos persistem dados relacionados a eventos de interação do usuário com o sistema, utilizado posteriormente para gerar insights e consulta analítica armazenam os logs de analises realizadas, utilizadas para gerar indicadores. Com o modelo definido para o banco de dados colunar o sistema se torna extremamente eficiente para agregar, filtrar e analisar dados de eventos em larga escala, como: *"Quantos cliques tivemos por categoria de produto no último mês?"*, *"Quais os produtos mais visualizados/comprados por clientes de uma determinada região?"* ou construção de funis de conversão.

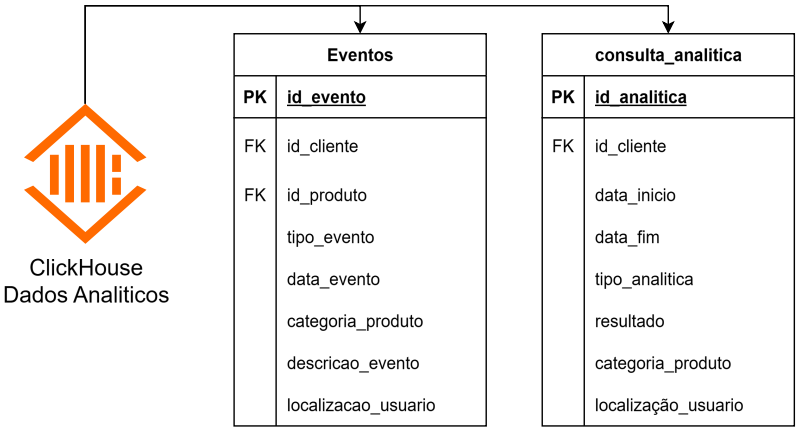

Figura 4: Diagrama conceitual modelo colunar

Para mapear e consultar relacionamentos complexos envolvendo usuários e produtos, foi utilizado um banco de dados baseado em grafos, totalmente otimizado para este tipo de aplicação. Com esta base de dados foi possível armazenar relacionamentos sobre os usuários e os produtos que eles compraram e avaliaram e com base nestes relacionamentos foi possível inferir uma recomendação mais assertiva para o usuário.

O modelo apresentado na Figura 5, apresenta os nos e os relacionamentos do modelo baseado em grafo, onde um cliente compra um produto de determinada marca e de determinada categoria, desta forma podemos deduzir que um outro produto da mesma categoria pai poderia ser relevante para este usuário, criando assim um sistema de recomendação com base nos relacionamentos do usuário.

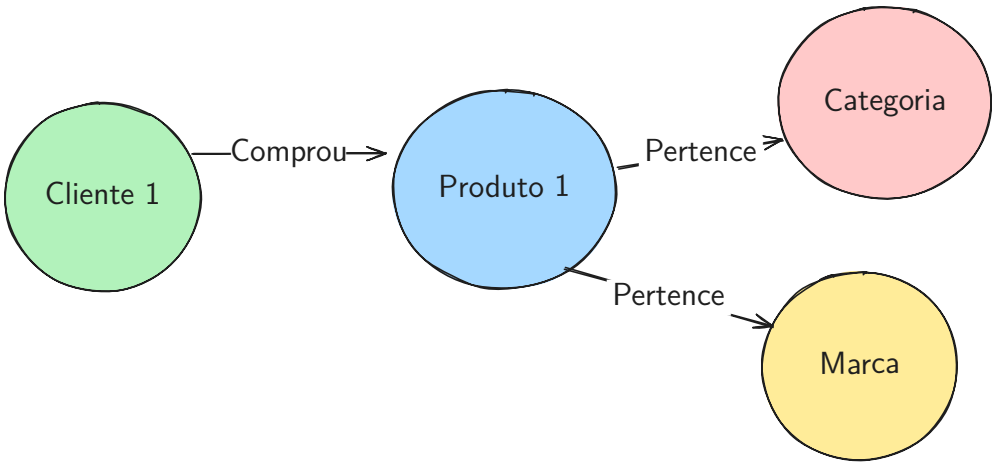

Figura 5: Diagrama conceitual modelo grafo

Na Figura 6, podemos observar a arquitetura global da camada de persistência para este projeto. Esta arquitetura de persistência adota uma abordagem de persistência poliglota, selecionando tecnologias de banco de dados otimizadas para atender a requisitos específicos de dados e cargas de trabalho.

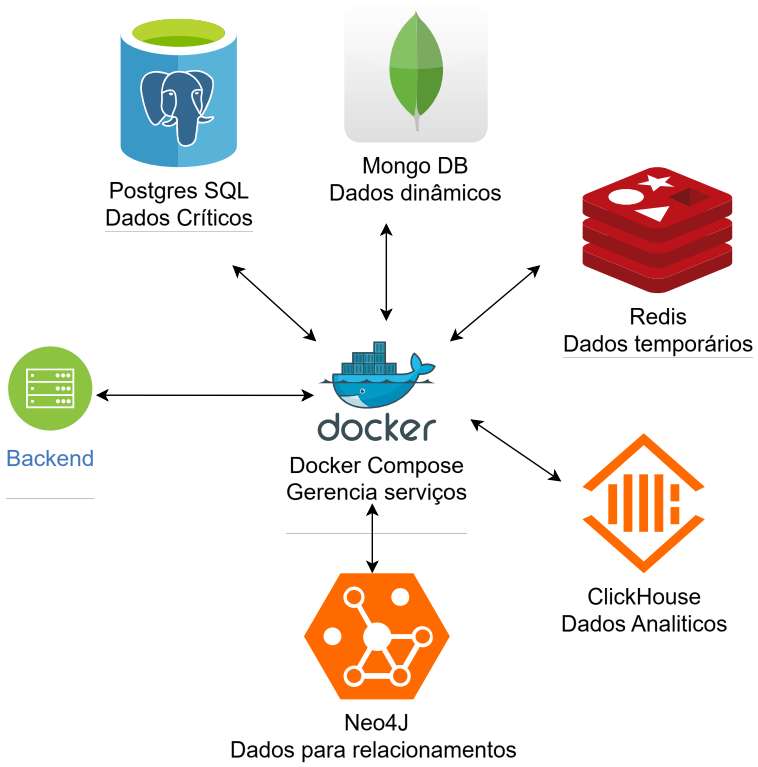

Figura 6: Arquitetura final da camada de persistência

3 Implementação

Para implementar a camada de persistência deste projeto foi utilizado o Docker para gerenciar todos os bancos de dados individualmente. Para iniciar os containers com os bancos de dados, acessamos o repositório do projeto e executamos o comando:

```
docker compose up -d
```

Ao executar este comando todos os contêineres são iniciados e prontos para serem acessados e executados. Para o Postgres SQL foi utilizado o *Beekeeper Studio* para acessar o banco de dados e executar os scripts de criação de tabelas e população de dados e posteriormente as consultas, todos os scripts estão disponibilizados no diretório scripts, e podem ser reproduzidos. Posteriormente para o MongoDB foi utilizado o Studio3T para acessar o banco e executar os comandos de criação das coleções e inserção dos documentos.

Para o Redis foi utilizado a ferramenta *Redis Insight* onde ser tornou possível popular as chaves definidas com valores de exemplos e posteriormente executar os comandos de consulta. Para o *ClickHouse* foi utilizado a própria interface de gerenciamento web que o próprio serviço do banco no docker disponibiliza. O *Neo4J* também foi acessado através da própria interface web oferecida pelo serviço para execução de comandos.

A implementação das diversas consultas no projeto seguiu uma abordagem que capitaliza as forças e otimizações nativas de cada sistema de gerenciamento de banco de dados (SGBD), alinhando a natureza da consulta ao tipo de persistência mais adequado. Para o PostgreSQL, as consultas foram projetadas para garantir a integridade transacional e a manipulação eficiente de dados relacionais e financeiros. Isso envolveu a construção de transações atômicas para

operações críticas como a criação de pedidos e a atualização de estoque, assegurando que todas as etapas ocorram com sucesso ou que nenhuma alteração seja persistida, prevenindo inconsistências.

Adicionalmente, a capacidade do PostgreSQL em realizar agregações complexas e filtros sobre dados estruturados foi amplamente utilizada para relatórios de faturamento mensal e para identificar tendências de clientes e estoque, aproveitando as funções SQL padrão e a otimização de índices para garantir performance em operações analíticas sobre volumes moderados de dados transacionais.

No que tange aos SGBDs especializados, a implementação das consultas explorou as características únicas de cada um. No MongoDB, o foco esteve na flexibilidade para consultas sobre dados semiestruturados e na poderosa capacidade do Aggregation Framework para transformar e agregar dados, como no cálculo de médias de preço por marca ou na busca por produtos com atributos variados. As operações no Redis foram desenvolvidas para máxima velocidade, utilizando suas estruturas de dados em memória para funções de alta demanda, como simulação de login com expiração, gestão dinâmica de carrinhos de compras, implementação de cache para reduzir a carga nos bancos de dados primários e a manutenção de rankings em tempo real.

Já o ClickHouse, como banco de dados colunar, foi a escolha para consultas analíticas complexas sobre grandes volumes de eventos. Sua performance é evidente na construção de funis de conversão e no cálculo de métricas comportamentais e de campanha em tempo real, onde a agregação e filtragem de vastas quantidades de dados de eventos são processadas de forma extremamente eficiente devido à sua arquitetura otimizada para OLAP. Por fim, no Neo4j, as consultas foram modeladas para explorar a riqueza dos relacionamentos entre entidades, aplicando algoritmos de grafo para filtragem colaborativa (seja Item-Item ou User-User) e identificação de padrões de influência ou caminhos de conexão, gerando recomendações contextuais e personalizadas que seriam ineficientes ou impraticáveis em bancos de dados relacionais.

4 Desafios encontrados

Um dos desafios notáveis encontrados durante a fase de implementação deste projeto foi a rigidez na modelagem de dados no momento da execução. Apesar de um planejamento inicial cuidadoso, o processo de desenvolvimento, muitas vezes, revelou requisitos adicionais ou nuances que só se tornaram aparentes quando a codificação e os testes foram executados. A identificação tardia da necessidade de novos atributos em tabelas ou coleções existentes, ou a reavaliação da forma como certas informações deveriam ser organizadas, exigiu ajustes consideráveis.

Desta forma foi necessário realizar revisões na estrutura dos bancos de dados. Esta situação destacou a importância de uma análise de requisitos ainda mais aprofundada antes da implementação e a consideração de abordagens que permitam maior flexibilidade inicial na estrutura dos dados, minimizando o impacto de modificações futuras.

A arquitetura que utiliza múltiplos tipos de bancos de dados, embora vantajosa por otimizar o desempenho para usos específicos, introduziu complexidades na forma como os dados são integrados e coordenados entre as diferentes bases. Uma situação prática dessa dificuldade surgiu quando uma função importante no PostgreSQL necessitava de informações

que estavam armazenadas exclusivamente no MongoDB. Para resolver essa questão, foi preciso desenvolver mecanismos de sincronização ou métodos de comunicação indireta.

Esse desafio enfatizou a necessidade de um planejamento minucioso sobre como os dados seriam trocados entre os diferentes sistemas de persistência, buscando um equilíbrio entre a especialização de cada banco e a complexidade das operações que envolvem mais de um deles.

5 Conclusão

A arquitetura de persistência deste projeto adota uma abordagem de persistência poliglota, selecionando tecnologias de banco de dados otimizadas para atender a requisitos específicos de dados e cargas de trabalho. O PostgreSQL foi empregado para gerenciar dados críticos e transacionais, garantindo alta consistência e integridade. Paralelamente, o MongoDB, um banco de dados NoSQL orientado a documentos, oferece flexibilidade para o armazenamento de dados semiestruturados e não estruturados.

Para otimização de performance e gerenciamento de dados efêmeros, o Redis atuou como um cache em memória e para manipulação de dados temporários. No domínio da análise de dados em larga escala, o ClickHouse, um banco de dados colunar, foi dedicado à ingestão e processamento eficiente de eventos, habilitando capacidades analíticas em tempo real. Adicionalmente, o Neo4j, um banco de dados de grafo, foi utilizado para a modelagem e consulta de relacionamentos complexos, fundamental para o funcionamento de sistemas de recomendação.

A orquestração e gerenciamento de todos esses serviços foram facilitados pelo Docker Compose, que promove a conteinerização, portabilidade e escalabilidade do ambiente. Desta forma, esta arquitetura foi desenhada para ser robusta, escalável e capaz de suportar diversas demandas de dados, desde transações de alta consistência até análises complexas e inteligência de recomendação.
