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*

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 Objeto de Estudo do Projeto

O projeto se baseia na arquitetura de persistência de dados para uma plataforma de e-commerce, denominada **"DataDriven Store"**. O sistema deve ser capaz de gerenciar dados com diferentes naturezas e demandas, incluindo: transações financeiras e de pedidos que exigem alta consistência; um catálogo de produtos com atributos variados; dados de sessão de usuário e carrinhos de compra com exigência de baixa latência; um grande volume de logs de eventos para análise de comportamento; e uma rede de relacionamentos entre entidades para sistemas de recomendação.

3 Escopo e Requisitos do Projeto

Cada grupo deverá projetar e implementar um protótipo da camada de dados para o sistema proposto, utilizando obrigatoriamente as cinco tecnologias de banco de dados abordadas. A tarefa central é justificar a escolha e modelar os dados para cada tecnologia de acordo com sua função no sistema. A distribuição de responsabilidades sugerida é a seguinte:

Banco de Dados Relacional (PostgreSQL)

- **Responsabilidade:** Manter a consistência transacional (ACID) para os dados críticos do negócio.
- **Dados a serem modelados:** Clientes (dados cadastrais essenciais), Pedidos, Itens de Pedido e Transações Financeiras.
- **Justificativa:** A estrutura rígida e as garantias transacionais do modelo relacional são adequadas para dados onde a integridade é um requisito fundamental.

Banco de Dados de Documento (MongoDB)

- **Responsabilidade:** Armazenar dados com estrutura flexível ou semiestruturada.
- **Dados a serem modelados:** Catálogo de Produtos, onde cada item pode possuir um conjunto distinto de atributos. Perfis estendidos de usuários (preferências, dados demográficos complementares).
- **Justificativa:** A flexibilidade do esquema (`schema-on-read`) facilita a evolução do catálogo e dos perfis de usuário sem a necessidade de migrações complexas na estrutura de dados.

Banco de Dados Chave-Valor (Redis)

- **Responsabilidade:** Gerenciar dados voláteis que requerem acesso com latência mínima.
- **Dados a serem modelados:** Sessões de usuário, carrinhos de compra e cache de dados frequentemente acessados.
- **Justificativa:** Acesso a dados em memória proporciona o desempenho necessário para operações em tempo real que impactam diretamente a experiência do usuário.

Banco de Dados Colunar (ClickHouse)

- **Responsabilidade:** Ingestão e processamento de grandes volumes de dados para consultas analíticas (OLAP).
- **Dados a serem modelados:** Logs de eventos da aplicação, como visualizações de produtos, cliques, termos de busca e outras interações do usuário.
- **Justificativa:** A arquitetura colunar é otimizada para agregações e varreduras em larga escala, sendo ideal para a geração de relatórios e dashboards analíticos.

Banco de Dados de Grafo (Neo4j)

- **Responsabilidade:** Mapear e consultar relacionamentos complexos, incluindo a implementação de um sistema de recomendação.
- **Dados a serem modelados:** Grafo de recomendações, contendo nós como `Cliente`, `Produto`, `Marca`, e `Categoria`, e arestas representando interações como `COMPROU`, `AVALIOU` e `VISUALIZOU`. A implementação deve habilitar a execução de algoritmos de filtragem colaborativa.
- **Justificativa:** A estrutura de grafo é eficiente para executar consultas baseadas em relacionamentos complexos, essenciais para sistemas de recomendação.

4 Entregáveis

O trabalho será executado em duas fases. **Nota importante:** O foco é a camada de persistência. Não é necessária a criação de uma aplicação com interface de usuário ou APIs. A entrega consiste em um conjunto de scripts que simulam as interações que uma aplicação real faria com os bancos de dados.

A) Entrega Parcial (Até Sábado, 28/06, 17h00)

1. **Composição dos Grupos:** Envio da lista de integrantes.
2. **Documento de Arquitetura (PDF, 1-2 páginas):** Contendo o diagrama conceitual da arquitetura e a descrição do modelo de dados de cada banco com suas justificativas.

B) Entrega Final (Até Quinta-feira, 03/07, 23h59)

1. **Repositório de Código:** Link para um repositório Git com todos os artefatos do projeto.
2. **Configuração do Ambiente:** Um arquivo `docker-compose.yml` (recomendado) ou um guia claro de como configurar o ambiente e as dependências manualmente.
3. **Scripts de Estrutura e Carga:** Scripts para criação do `schema` e povoamento com dados de exemplo em cada banco.
4. **Scripts de Demonstração de Consultas:** Um arquivo por banco de dados contendo, no mínimo, 5 consultas ou operações que demonstrem seu uso específico na arquitetura. Sugestões de consultas de complexidade média são fornecidas na seção a seguir.
5. **Relatório Técnico Final (PDF):** Documentação da arquitetura final, detalhes da implementação, desafios e conclusões.

5 Sugestões de Consultas para a Entrega Final

Para garantir a profundidade técnica do trabalho, seguem exemplos do tipo de consulta esperado para cada tecnologia.

- **PostgreSQL:**
    1. Transação atômica para criar um pedido, seus itens e atualizar o estoque.

2. Listar os 5 clientes com maior faturamento nos últimos 6 meses.
3. Gerar um relatório de faturamento mensal, agrupado por categoria.
4. Identificar produtos com estoque abaixo de um limiar.
5. Listar todos os pedidos de um cliente, incluindo o valor total.

- **MongoDB:**
  1. Usar o Aggregation Framework para calcular a média de preço por marca.
  2. Buscar produtos com atributos específicos (e.g., ‘processador: "i7"‘) e com preço em uma faixa definida.
  3. Adicionar um novo campo a todos os produtos de uma categoria.
  4. Listar as avaliações de um produto, ordenadas por data.
  5. Encontrar usuários que tenham uma preferência específica em seu perfil.
- **Redis:**
  1. Simular login de usuário (comando `SET` com expiração).
  2. Gerenciar um carrinho de compras (comandos para `HASH` ou `LIST`).
  3. Implementar cache de produtos (verificar se existe; se não, simular busca no DB principal e popular o cache).
  4. Manter um ranking de produtos mais vistos (usando `Sorted Set`).
  5. Contar visualizações de página de um produto (usando `INCR`).
- **ClickHouse:**
  1. Consulta de funil de conversão (visualizou -> adicionou ao carrinho -> comprou).
  2. Calcular o número de eventos de "visualização"por dia na última semana.
  3. Identificar os 10 termos de busca mais utilizados.
  4. Calcular a taxa de cliques (CTR) de uma campanha.
  5. Listar usuários que vieram de uma ‘utm_source‘ específica e realizaram compra.
- **Neo4j:**
  1. **Filtragem Colaborativa (Item-Item):** "Dado um produto X, encontre outros produtos frequentemente comprados juntos".
  2. **Filtragem Colaborativa (User-User):** "Encontre clientes com histórico de compra similar ao do cliente Y e recomende produtos que eles compraram, mas Y não".
  3. Encontrar o caminho mais curto entre dois produtos através de suas categorias.
  4. Identificar clientes "influenciadores"(cujas avaliações positivas se correlacionam com mais vendas).
  5. Recomendar produtos de categorias que um cliente visualizou, mas das quais ainda não comprou.

6 Critérios de Avaliação

- **Adequação da Modelagem (30%):** Correta aplicação dos conceitos de cada modelo de banco de dados ao problema.
- **Qualidade da Implementação Técnica (40%):** Funcionalidade do ambiente, dos scripts e da organização do código. A complexidade e correção das consultas serão avaliadas aqui.
- **Profundidade da Análise no Relatório (20%):** Clareza na documentação, justificativas técnicas e conclusões.
- **Cumprimento dos Prazos (10%):** Pontualidade na entrega das duas fases do trabalho.

7 Ferramentas Sugeridas

- **Bancos:** PostgreSQL, MongoDB, Redis, ClickHouse, Neo4j.
- **Orquestração/Ambiente:** Docker e Docker Compose (recomendado), ou configuração manual com guia.
- **Controle de Versão:** Git.

Estarei disponível para esclarecimentos e orientação durante o período de aula no sábado.

Referências