Atividade avaliativa: Níveis de Isolamento em Sistemas Distribuídos

1) Descrição

Você é funcionário do Kabum, uma famosa loja de comércio eletrônico especializada em equipamentos de informática. Durante a Black Friday, o Kabum disponibiliza produtos com grandes descontos, mas com estoque limitado, conforme ilustrado na Figura 1.

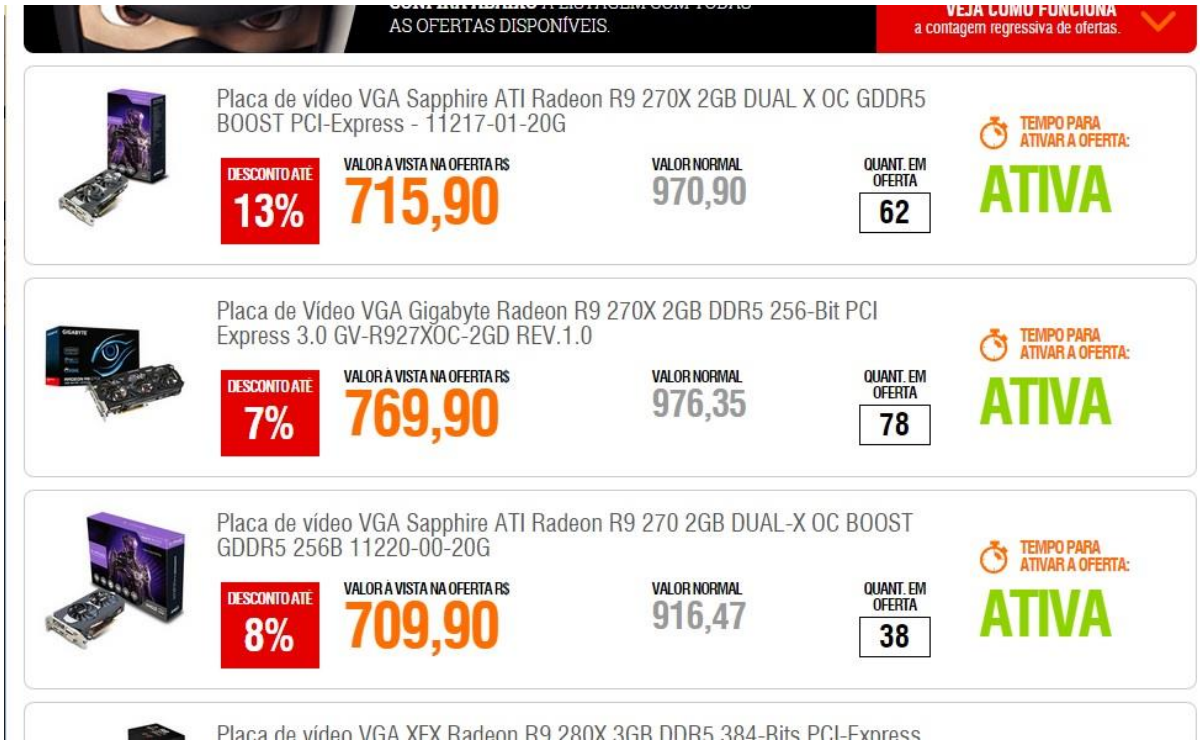

Figura 1 – Tela no site do Kabum mostrando produtos em oferta e seus estoques disponíveis.

Devido ao intenso fluxo de vendas nesse período, o Kabum enfrenta problemas de concorrência, como exibição de estoques incorretos ou até negativos (não é possível ter estoque inferior a zero). Esses problemas podem levar à venda de produtos sem estoque, inviabilizando a entrega no prazo prometido, causando insatisfação dos clientes. Com a projeção de novos descontos na próxima Black Friday, espera-se um aumento significativo no número de acessos simultâneos aos servidores.

Para resolver o problema, vamos aplicar os conceitos de **Lock Otimista** e **Lock Pessimista** no controle de estoque dos produtos. No entanto, precisamos analisar as vantagens e desvantagens de cada abordagem. Para isso, definimos os seguintes cenários de teste**:**

1) Servidor de Pedidos recebendo 100 requisições simultâneas para comprar o mesmo produto
2) Servidor de Pedidos recebendo 1.000 requisições simultâneas para comprar o mesmo produto
3) Servidor de Pedidos recebendo 10.000 requisições simultâneas para comprar o mesmo produto

Serão criados três endpoints REST, cada um representando uma abordagem diferente:

Sem Locks: http://localhost:8080/pedido/novo
Endpoint sem controle explícito de concorrência.

Com Lock Otimista: http://localhost:8080/pedido_otimista/novo
Endpoint implementando controle de concorrência com Lock Otimista.

Com Lock Pessimista: http://localhost:8080/pedido_pessimista/novo
Endpoint implementando controle de concorrência com Lock Pessimista.

2) Cenário de Teste

- **Produto**: PlayStation 5, vendido por **R$ 1999,99**, com estoque inicial de **1000 unidades**.
- **Requisições**: Os pedidos serão gerados aleatoriamente, comprando entre **1 a 5** unidades por requisição.
- **Processo**: Cada requisição deve:
  - Buscar o estoque no banco de dados
  - Retornar uma resposta de erro (**409 conflict**) caso não haja estoque suficiente para atender o pedido.
  - Caso tenha estoque, salvar o pedido com os itens comprados.
  - Debitar o estoque do produto.
  - Tanto o salvamento quanto o débito de estoque devem ser operações atômicas dentro da mesma transação.

3) Coleta de Dados

Após realizar os testes, os seguintes dados devem ser coletados para as três abordagens implementadas (**sem locks**, **lock otimista** e **lock pessimista**):

1) Tempo médio de resposta para cada requisição: Calcular o tempo médio necessário para que cada requisição seja processada e respondida.

2) Tempo total para responder todas as requisições: Medir o tempo total necessário para processar e responder todas as requisições enviadas ao servidor.

3) Status das Requisições: Contabilizar a quantidade de requisições bem-sucedidas (200 OK) e as que retornaram erro:

    - 409 (Conflict) para situações de estoque insuficiente (no caso do lock otimista e lock pessimista).
    - Outros códigos de erro, se aplicável.

4) **Inconsistências Detectadas**: Verificar e reportar quaisquer inconsistências no sistema, como:
    - Estoque negativo.
    - Pedidos processados com quantidade de estoque incorreta.
    - Variações no saldo final de estoque entre as abordagens.

4) Entrega

Entregar o projeto privado no Github colocando carloseduardoxp como colaborador. O código-fonte deve ser feito usando Spring Boot, com escolha livre para tecnologias dentro deste framework (Maven, Jpa, etc.). A coleta de dados pode ser entregue dentro do projeto, dentro da pasta docs.

5) Banco de Dados

A Figura 2 apresenta um modelo de banco de dados como exemplo. No entanto, trata-se apenas de uma sugestão. Algumas tabelas, como "categorias", não precisam necessariamente ser implementadas. O foco principal está no fluxo de pedidos e no controle de estoque.

Embora cada pedido esteja associado a um cliente, não é necessário criar um endpoint REST para clientes. Basta que o pedido esteja vinculado a um cliente que já exista no banco de dados.

Para cada chamada ao endpoint REST, o sistema deve:

1. Salvar o pedido na tabela de pedidos.
2. Salvar os detalhes do pedido na tabela correspondente.
3. Atualizar a coluna UnidadesEmEstoque da tabela de produtos, refletindo a quantidade reduzida devido ao pedido.

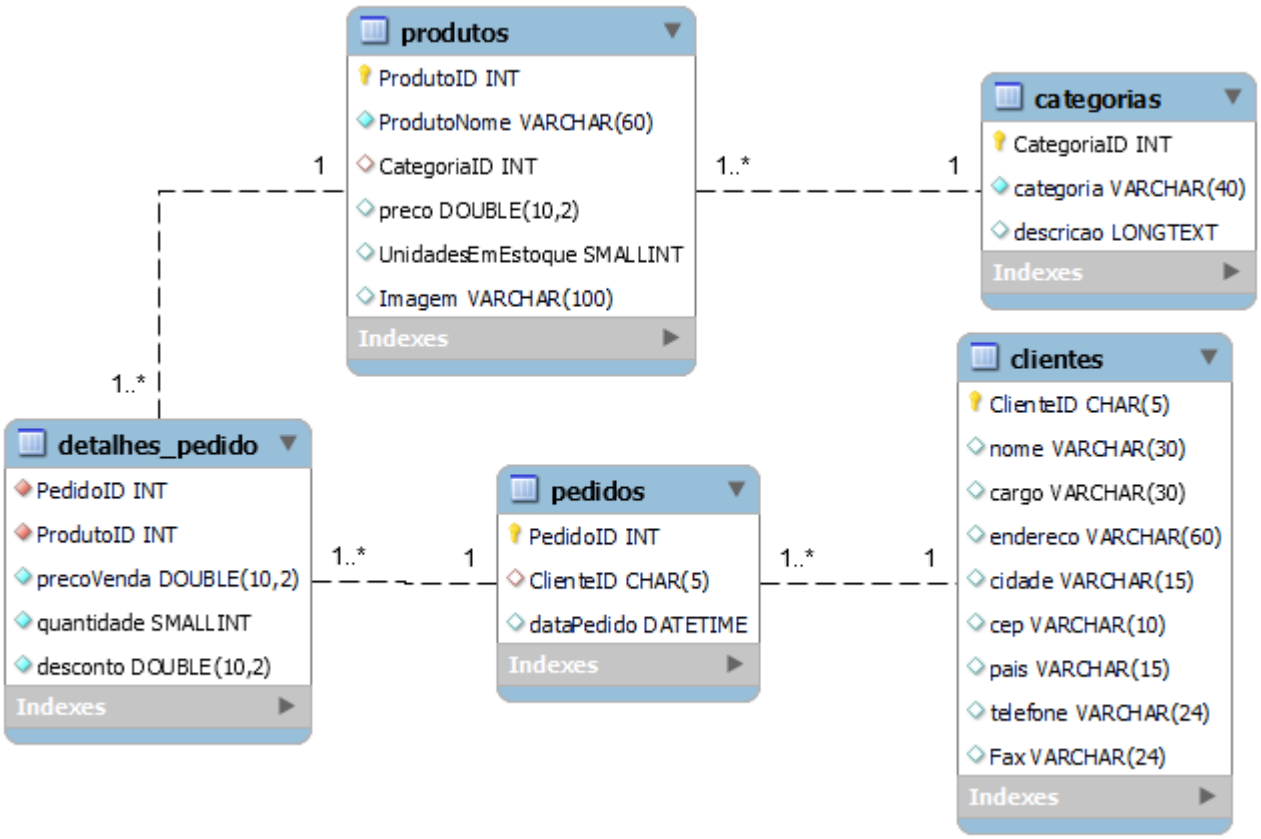

Figura 2 – Exemplo de Banco de Dados para um sistema de pedidos.