at_S10_A3_SL06_backend

Roteiro de Atividade Prática

Nome: _____ Turma: _____

Título da atividade: Transações e consistência

Objetivo:

Implementar **transações** em um banco de dados para garantir a **consistência dos dados**, utilizando operações atômicas para que múltiplas ações sejam realizadas com sucesso ou totalmente revertidas em caso de falha.

Conceito técnico: transações e consistência

As **transações** são um conjunto de operações executadas como uma única unidade de trabalho, em que todas as operações devem ser bem-sucedidas para que a transação seja confirmada. Caso qualquer operação falhe, a transação inteira é revertida (rollback), garantindo que o sistema de banco de dados permaneça em um estado consistente. Esse conceito é frequentemente relacionado às propriedades **ACID**:

- **atomicidade:** todas as operações da transação devem ser concluídas ou nenhuma será aplicada;
- **consistência:** as transações levam o banco de dados de um estado consistente para outro estado consistente;
- **isolamento:** transações concorrentes devem ser executadas de maneira que não interfiram umas nas outras;
- **durabilidade:** uma vez confirmada, a transação deve persistir mesmo em caso de falhas no sistema.

No contexto de bancos de dados NoSQL, nem todos seguem as propriedades ACID de forma completa, especialmente em sistemas distribuídos que priorizam **disponibilidade** e **particionamento** (CAP theorem). Contudo, muitos bancos de dados, como **MongoDB** (a partir da versão 4.0), oferecem suporte a

transações multi-documento, garantindo consistência em operações mais complexas.

Objetivo da atividade:
O participante irá implementar uma transação utilizando **MongoDB** para garantir a consistência dos dados em um sistema de e-commerce, em que múltiplas coleções precisam ser atualizadas de maneira atômica.

Enunciado:
Você foi encarregado de implementar uma operação de compra em um sistema de e-commerce utilizando transações. A operação de compra envolve a atualização de múltiplas coleções:

1. atualizar o inventário do produto comprado;
2. registrar o pedido na coleção de pedidos;
3. registrar o histórico de transações do cliente.

A transação deve garantir que todas essas operações sejam realizadas com sucesso ou que nenhuma delas seja aplicada em caso de erro.

Código de programação:
Instalação do MongoDB e pymongo:
Se ainda não tiver o MongoDB instalado, siga as instruções de instalação ou utilize um serviço de banco de dados MongoDB na nuvem (como MongoDB Atlas).

Instale a biblioteca **pymongo** para Python:
pip install pymongo

Exemplo em Python com pymongo:

```
from pymongo import MongoClient
from pymongo.errors import PyMongoError

# Conectar ao MongoDB
client = MongoClient('mongodb://localhost:27017/')
db = client['ecommerce']
produtos_collection = db['produtos']
pedidos_collection = db['pedidos']
```

```
historico_collection = db['historico_transacoes']

# Função para processar compra com transação
def processar_compra(cliente_id, produto_id, quantidade):
    session = client.start_session()
    try:
        with session.start_transaction():
            # Atualizar o estoque do produto
            produto = produtos_collection.find_one({"_id": produto_id},
session=session)
            if produto["estoque"] < quantidade:
                raise Exception("Estoque insuficiente!")

            produtos_collection.update_one(
                {"_id": produto_id},
                {"$inc": {"estoque": -quantidade}},
                session=session
            )

            # Registrar o pedido
            pedido = {
                "cliente_id": cliente_id,
                "produto_id": produto_id,
                "quantidade": quantidade,
                "status": "Confirmado"
            }
            pedidos_collection.insert_one(pedido, session=session)

            # Atualizar histórico de transações do cliente
            historico = {
                "cliente_id": cliente_id,
                "produto_id": produto_id,
                "quantidade": quantidade,
                "tipo_transacao": "Compra"
            }
            historico_collection.insert_one(historico, session=session)
```

```
        session.commit_transaction()
        print("Transação realizada com sucesso!")

    except (PyMongoError, Exception) as e:
        session.abort_transaction()
        print(f"Erro na transação: {e}")

    finally:
        session.end_session()

# Simulação de uma compra
processar_compra(cliente_id="cliente123", produto_id="produto456",
quantidade=2)
```

Explicação técnica:

- **Sessão e transação:** utilizamos o método **start_session()** para iniciar uma sessão de transação. As operações realizadas dentro do bloco **with session.start_transaction()** são tratadas como uma única transação.
- **Atomicidade:** se qualquer operação dentro da transação falhar, a função **abort_transaction()** é chamada, revertendo todas as mudanças feitas até o ponto de falha. Caso todas as operações sejam bem-sucedidas, **commit_transaction()** é chamado para confirmar as mudanças.
- **Consistência:** a transação garante que o estoque, o pedido e o histórico sejam atualizados de forma consistente. Se algo der errado, nenhum dado será gravado ou alterado.

Perguntas para conclusão da atividade:

- Explique como a propriedade de atomicidade é aplicada em transações e por que isso é importante para a consistência dos dados.
- Descreva como o uso de transações em MongoDB pode garantir a consistência dos dados em um ambiente NoSQL.

- Por que a operação abort_transaction() é importante em transações, e o que ela faz quando chamada?