CEUB

EDUCAÇÃO SUPERIOR

ceub.br

BANCO DE DADOS II

AULA 14

TRANSAÇÕES

Prof. Leonardo R. de Deus

1. TRANSAÇÃO

O que é uma TRANSAÇÃO?

Conjunto de operação que o banco de dados trata como uma única unidade de trabalho.

Ou acontece tudo, ou não acontece nada.

1. TRANSAÇÃO

O que é uma TRANSAÇÃO?

Conjunto de operação que o banco de dados trata como uma única unidade de trabalho.

Ou acontece tudo, ou não acontece nada.

Considerando nosso exemplo:

Em **transferência bancária** — se debitar de uma conta, precisa **obrigatoriamente** creditar na outra.

Se der erro no meio do caminho, **nenhuma das duas operações deve ser confirmada.**

2. PROPRIEDADES DE UMA TRANSAÇÃO

Para que um SGBD seja considerado confiável é preciso garantir que cada transação atenda a 4 propriedades:

A - Atomicidade

C - Consistência

I - Isolamento

D - Durabilidade

Criação da Stored Procedures para encapsular a transação de efetivar um novo pedido

ETAPAS

1. Incluir novo pedido na tabela de pedidos
2. Incluir novo registro do item pedido na tabela item_pedido
3. Atualizar a quantidade de estoque na tabela produto

2. PROPRIEDADES DE UMA TRANSAÇÃ

Criação da Stored Procedures para encapsular a transação de efetivar um novo pedido

CRIAR A PROCEDURE NO BANCO

```sql
CREATE OR REPLACE PROCEDURE loja.sp_registrar_venda_completa(
    p_id_cliente INT,
    p_id_produto INT,
    p_quantidade INT
)
AS $$
DECLARE
    v_estoque_atual INT;
    v_preco_unitario DECIMAL;
    v_id_pedido_novo INT;
    v_numero_pedido_novo VARCHAR(50);
BEGIN

    -- Passo 1: verificar se o cliente existe
    IF NOT EXISTS (SELECT 1 FROM loja.tb01_cliente WHERE id_cliente = p_id_cliente) THEN
        RAISE EXCEPTION 'Cliente com ID % não existe.', p_id_cliente;
    END IF;

    -- Passo 2: verificar se o produto desejado existe
    IF NOT EXISTS (SELECT 1 FROM loja.tb02_produto WHERE id_produto = p_id_produto) THEN
        RAISE EXCEPTION 'Produto com ID % não existe', p_id_produto;
    END IF;

    -- Passo 3: Selecionar dados necessário nas etapas seguintes
    SELECT estoque, preco
    INTO v_estoque_atual, v_preco_unitario
    FROM loja.tb02_produto
    WHERE id_produto = p_id_produto
    FOR UPDATE; -- bloqueio para travar atualização concorrente para a mesma tupla de dados

    -- Passo 4:  verificar se o produto desejado tem estoque disponível
    IF v_estoque_atual < p_quantidade THEN
        RAISE EXCEPTION 'Estoque insuficiente para o produto ID %. Disponível: %', p_id_produto, v_estoque_atual;
    END IF;

    -- Passo 5: Gerar o número do novo pedido
    SELECT 'PED-2025-' || LPAD((MAX(id_pedido) + 1)::TEXT, 5, '0')
    INTO v_numero_pedido_novo
    FROM loja.tb04_pedido;

    -- Passo 6: atualizar a tabela pedido, item_pedido e produto
    INSERT INTO loja.tb04_pedido (numero_pedido, data_pedido, id_cliente, id_status_pedido)
    VALUES (v_numero_pedido_novo, CURRENT_DATE, p_id_cliente, 2) -- Status 2 = Pagamento Aprovado
    RETURNING id_pedido INTO v_id_pedido_novo; -- Captura o ID do novo pedido

    INSERT INTO loja.tb05_item_pedido (id_pedido, id_produto, quantidade, valor_unitario)
    VALUES (v_id_pedido_novo, p_id_produto, p_quantidade, v_preco_unitario);

    UPDATE loja.tb02_produto
    SET estoque = estoque - p_quantidade
    WHERE id_produto = p_id_produto;

    -- Se tudo deu certo a transação é concretizada
    RAISE NOTICE 'Venda registrada com sucesso! Pedido N°: %', v_numero_pedido_novo;

EXCEPTION
    -- Se alguma etapa anterior falhar, a execução entra neste bloco e é feito Rollback
    WHEN OTHERS THEN
        RAISE NOTICE 'Ocorreu um erro: %. A transação não foi concluída (ROLLBACK).', SQLERRM;
END;
$$ LANGUAGE plpgsql;
```

1. Atomicidade

A transação é uma unidade indivisível.

Ela (a transação) deve ser realizada em sua totalidade ou não ser realizada de forma alguma.

Uma transação não pode ser parcialmente concluída.

1. **Atomicidade**

Transação realizada

```
CALL loja.sp_registrar_venda_completa(1,1,1);
```

Todos os passos são concluídos com sucesso.

Transação é efetivada!

1. **Atomicidade**

Cliente inexistente

Produto inexistente

Estoque insuficiente

Um passo falhou?

Transação não é efetivada!

2. PROPRIEDADES DE UMA TRANSAÇÃO

1. **Atomicidade**

Cliente inexistente

Produto inexist

Estoque insuficiente

Vamos chamar a PROCEDURE com dados inexistentes para confirmar o funcionamento.

2. Consistência

Uma transação deve preservar a consistência de um banco, quando ela for completamente executada do início ao fim.

Ou seja, **a transação deve levar o banco de um estado válido para outro estado válido.**

Uma transação não pode quebrar as regras e restrições de um banco.

Exemplo: uma transação não pode incluir na tabela item_pedido um id_produto que não exista na tabela produto.

2. Consistência

Mesmo sem uma operação explícita para verificar se o “Cliente” e o “Produto” existem no banco, a transação falha se o “Cliente” e “Produto” não existirem no banco.

Por que?

2. Consistência

Mesmo sem uma operação explícita para verificar se o “Cliente” e o “Produto” existem no banco, a transação falha se o “Cliente” e “Produto” não existirem no banco.

Por que?

Vamos Confirmar?

3. Isolamento

O isolamento garante que transações concorrentes não interfiram umas nas outras.

Mesmo que várias transações ocorram ao mesmo tempo, cada transações acredita que é a única em execução.

Exemplo: Se o "Cliente A" está no meio de uma transação para comprar o último item do estoque, o "Cliente B" (em outra transação) não pode ver o estoque "quase" atualizado. Ele só verá o estado de antes da compra ou o estado de depois, mas nunca o estado intermediário.

3. Isolamento

O isolamento garante que transações concorrentes não interfiram umas nas outras.

Mesmo que várias transações ocorram ao mesmo tempo, cada transações acredita que é a única em execução.

Exemplo: Se o "Cliente A" está no meio de uma transação para comprar o último item do estoque, o "Cliente B" (em outra transação) não pode ver o estoque "quase" atualizado. Ele só verá o estado de antes da compra ou o estado de depois, mas nunca o estado intermediário.

3. Isolamento

Níveis de isolamento PostgreSQL

- **READ COMMITTED:** dentro de uma transação, só é possível visualizar dados que foram confirmados (committed);

  Os dados mudam a todo tempo, à medida que novas transações são efetivadas.

  Este é o nível de isolamento default do banco.

Exemplo: Funcionário fazendo uma consulta no banco, enquanto uma novo pedido é processado.

3. Isolamento

Funcionário fazendo um relatório de que executa várias ações

```
BEGIN;

SELECT SUM(fato_faturamento_total) FROM
loja.vw_fato_vendas;
                    .
                    .
                    .
```

Inclusão de um novo pedido

```
CALL loja.sp_registrar_venda_completa(3,5,2);
```

3. Isolamento

Funcionário fazendo um relatório de que executa várias ações

```
BEGIN;

SELECT SUM(fato_faturamento_total) FROM
loja.vw_fato_vendas;
                    .
                    .
                    .
```

altera o resultado de ações que ainda não tinham sido executadas

Inclusão de um novo pedido

```
CALL loja.sp_registrar_venda_completa(3,5,2);
```

comando executado antes que todos os comandos da transação anterior sejam finalizados

3. Isolamento

Níveis de isolamento PostgreSQL

- **REPEATABLE READ:** dentro de uma transação, os dados não podem mudar, até a conclusão da transação;

  Ao definir este nível de isolamento é tirado um “snapshot” do banco, e todas as ações dentro da transação estarão vendo a mesma versão do banco.

Exemplo: Funcionário fazendo uma consulta no banco, enquanto uma novo pedido é processado.

3. Isolamento

Funcionário fazendo um relatório de que executa várias ações

```
BEGIN;
SET TRANSACTION ISOLATION LEVEL
REPEATABLE READ;
SELECT SUM(fato_faturamento_total) FROM
loja.vw_fato_vendas;
                    .
                    .
                    .
```

Inclusão de um novo pedido

```
CALL loja.sp_registrar_venda_completa(3,5,2);
```

3. Isolamento

Funcionário fazendo um relatório de que executa várias ações

```
BEGIN;
SET TRANSACTION ISOLATION LEVEL
REPEATABLE READ;
SELECT SUM(fato_faturamento_total) FROM
loja.vw_fato_vendas;
                    .
                    .
```

não altera os dados vistos pela ações dentro da transação

Inclusão de um novo pedido

```
CALL loja.sp_registrar_venda_completa(3,5,2);
```

comando executado antes que todos os comandos da transação anterior sejam finalizados

3. Isolamento

Controle de Concorrência e Bloqueio - FOR UPDATE

Como controlar o processo de venda quando temos dois clientes comprando o mesmo produto ao mesmo tempo?

Cliente 1: comprando 2 itens do produto id = 1

```
CALL loja.sp_registrar_venda_completa(1,1,2);
```

Cliente 2: comprando 1 item do produto id = 1

```
CALL loja.sp_registrar_venda_completa(2,1,1);
```

3. Isolamento

Controle de Concorrência e Bloqueio - UPDATE

Quando ocorre um impasse, como o banco resolver? **DEADLOCK**

Exemplo: o Funcionário 1 vai atualizar o Produto 1 e depois o Produto 2. Ao mesmo tempo o Funcionário 2 vai atualizar o Produto 2 e depois o Produto 1.

Funcionário 1:

```
BEGIN;
UPDATE loja.tb02_produto SET estoque = 10 WHERE id_produto = 1;
```

Funcionário 2:

```
BEGIN;
UPDATE loja.tb02_produto SET estoque = 20 WHERE id_produto = 2;
```

Funcionário 1:

```
UPDATE loja.tb02_produto SET estoque = 30 WHERE id_produto = 2;
```

Funcionário 2:

```
UPDATE loja.tb02_produto SET estoque = 40 WHERE id_produto = 1;
```

4. Durabilidade

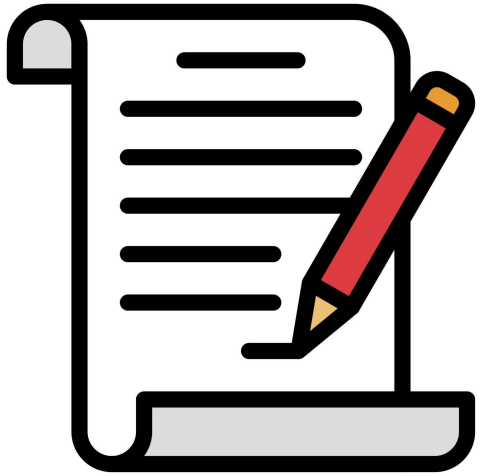
As mudanças aplicadas no banco por uma transação que foi confirmada (commit) não podem ser perdidas, mesmo que ocorra alguma falha.

Para garantir a durabilidade o SGDB escreve as alterações em um log de transações antes de escrever as alterações nas tabelas específicas.

3. TAREFA

Faça um texto, em formato PDF, explicando com suas palavras as propriedades que uma transação executada no SGBD deve garantir.

Entrega no final da aula.



OBRIGADO
A TODOS!

CEUB

EDUCAÇÃO SUPERIOR