Transações em banco de dados

O conceito **ACID** é um conjunto de propriedades que garantem a confiabilidade e a integridade das transações em um sistema de banco de dados. Esses princípios são fundamentais para o funcionamento seguro de sistemas de banco de dados, principalmente em operações complexas ou críticas, onde a integridade dos dados é vital.

Cada letra do termo **ACID** representa uma propriedade importante:

1. **A - Atomicidade** (*Atomicity*):
   A atomicidade garante que uma transação seja tratada como uma única unidade indivisível. Isso significa que, dentro de uma transação, ou todas as operações são executadas com sucesso, ou nenhuma delas será aplicada. Se houver algum erro no meio do processo, todas as mudanças feitas até aquele ponto são revertidas (com um rollback).
   Exemplo: Em uma transferência bancária, se o débito em uma conta for bem-sucedido mas o crédito na outra falhar, a transação inteira é revertida.
2. **C - Consistência** (*Consistency*):
   A propriedade de consistência assegura que o banco de dados permaneça em um estado válido antes e após uma transação. Qualquer transação deve levar o banco de dados de um estado consistente para outro, respeitando todas as regras de integridade e restrições do banco.
   Exemplo: Se houver uma restrição de integridade referencial, como uma chave estrangeira, a transação deve garantir que ela seja respeitada. Inserções ou atualizações que violam essa regra não serão permitidas.
3. **I - Isolamento** (*Isolation*):
   O isolamento garante que as operações de uma transação sejam isoladas das de outras transações simultâneas. Isso significa que, enquanto uma transação estiver em andamento, outras não poderão interferir nos dados que estão sendo manipulados. Dependendo do nível de isolamento configurado, pode haver menor ou maior interação entre transações simultâneas.
   Exemplo: Em um banco de dados de inventário, duas transações que tentam atualizar a quantidade de um produto ao mesmo tempo devem ser isoladas para evitar discrepâncias.
4. **D - Durabilidade** (*Durability*):
   A durabilidade assegura que, uma vez que uma transação é concluída (commit), as alterações são permanentes no banco de dados, mesmo em casos de falhas no sistema (como queda de energia). Os dados comprometidos serão preservados e recuperáveis após qualquer falha, garantindo que o banco de dados reflita o estado final da transação.
   Exemplo: Depois que uma venda é confirmada, o registro da transação deve persistir no sistema, mesmo que haja uma falha de energia logo em seguida.

Para criar uma transação no PostgreSQL, você pode utilizar os comandos SQL para iniciar, manipular e concluir a transação. Uma transação é um conjunto de operações que são executadas de maneira atômica, ou seja, todas as operações devem ser bem-sucedidas, ou nenhuma delas será aplicada ao banco de dados. As principais etapas para criar uma transação no PostgreSQL são:

1. **Iniciar a transação**: `BEGIN;`
2. **Executar comandos SQL**: Executar as instruções de inserção, atualização, deleção ou consulta.
3. **Concluir a transação**: Se todas as operações foram bem-sucedidas, você pode usar `COMMIT;` para aplicar as mudanças. Caso contrário, utilize `ROLLBACK;` para desfazer as alterações feitas durante a transação.

Exemplo de transação básica

```
-- Iniciar a transação
BEGIN;

-- Inserir uma nova linha na tabela "usuarios"
INSERT INTO usuarios (nome, email) VALUES ('João Silva',
'joao@email.com');

-- Atualizar uma linha existente na tabela "usuarios"
UPDATE usuarios SET email = 'joaosilva@email.com' WHERE nome = 'João
Silva';

-- Se tudo correr bem, aplicar as mudanças
COMMIT;
```

Exemplo com ROLLBACK

```
-- Iniciar a transação
BEGIN;

-- Inserir um novo usuário
INSERT INTO usuarios (nome, email) VALUES ('Maria Souza',
'maria@email.com');

-- Simular um erro, como tentar inserir um usuário com um email
duplicado
```

```
INSERT INTO usuarios (nome, email) VALUES ('Maria Souza',
'maria@email.com');

-- Desfazer todas as alterações da transação
ROLLBACK;
```

Transações em código (Node.js com pg)

Se você estiver utilizando Node.js para se conectar ao PostgreSQL, você pode gerenciar transações utilizando a biblioteca pg. Um exemplo de como isso pode ser feito:

```
const { Client } = require('pg');

async function executarTransacao() {
  const client = new Client({
    user: 'seu_usuario',
    host: 'localhost',
    database: 'seu_banco_de_dados',
    password: 'sua_senha',
    port: 5432,
  });

  try {
    await client.connect();

    // Iniciar a transação
    await client.query('BEGIN');

    // Executar algumas consultas dentro da transação
    await client.query('INSERT INTO usuarios(nome, email) VALUES($1,
$2)', ['João Silva', 'joao@email.com']);
    await client.query('UPDATE usuarios SET email = $1 WHERE nome =
$2', ['joaosilva@email.com', 'João Silva']);

    // Finalizar a transação
    await client.query('COMMIT');
    console.log('Transação bem-sucedida');
  } catch (err) {
    // Se algo der errado, desfazer a transação
    await client.query('ROLLBACK');
    console.error('Erro na transação, rollback realizado', err);
  } finally {
    // Fechar a conexão
    await client.end();
```

```
    }
}

executarTransacao();
```

Esse exemplo em Node.js mostra como iniciar uma transação, executar consultas SQL e fazer o commit ou rollback, dependendo de sucesso ou falha.

Exercício 1: Controle de Estoque com Transações

Imagine que você está criando um sistema de controle de estoque para uma loja. A loja possui uma tabela de produtos (`tb_produto`) e uma tabela de pedidos (`tb_pedido`). Quando um pedido é realizado, o sistema deve:

1. Verificar se a quantidade solicitada de um produto está disponível no estoque.
2. Atualizar a quantidade disponível do produto, descontando a quantidade vendida.
3. Inserir o pedido na tabela de pedidos.
4. Se algum desses passos falhar, a transação deve ser revertida (rollback) para garantir que nenhuma atualização parcial seja aplicada.

Estrutura das Tabelas

Tabela `tb_produto`

Coluna	Tipo	Descrição
id_produto	SERIAL	Chave primária
nome	VARCHAR	Nome do produto
quantidade	INTEGER	Quantidade disponível no estoque

Tabela `tb_pedido`

Coluna	Tipo	Descrição
id_pedido	SERIAL	Chave primária
id_produto	INTEGER	ID do produto
quantidade	INTEGER	Quantidade solicitada
data_pedido	TIMESTAMP	Data e hora do pedido

Passos do Exercício

1. **Criação das tabelas:** Crie as tabelas `tb_produto` e `tb_pedido` usando os campos acima.
2. **Insira alguns produtos na tabela `tb_produto`:** Adicione, por exemplo, um produto com `quantidade = 100`.
3. **Criar uma função de transação:** Implemente uma função que:
    - Receba `id_produto` e `quantidade` como parâmetros.
    - Verifique se a quantidade solicitada está disponível. Se não houver estoque suficiente, a transação deve falhar.
    - Atualize a quantidade de produtos no estoque na tabela `tb_produto`.
    - Insira o pedido na tabela `tb_pedido`.
4. **Usar transação para segurança dos dados:** Coloque todas as operações de verificação, atualização e inserção dentro de uma transação.

Exercício2: Transferência Bancária entre Contas com Controle de Saldo

Imagine que você está desenvolvendo uma funcionalidade de transferência de valores entre contas em um sistema bancário. O sistema possui uma tabela de **contas bancárias** e uma tabela de **transações**. Quando um usuário transfere dinheiro de uma conta para outra, o sistema deve:

1. Verificar se o saldo da conta de origem é suficiente para a transferência.
2. Debitar o valor da conta de origem.
3. Creditar o valor na conta de destino.
4. Registrar a transação na tabela de transações.
5. Em caso de erro em qualquer etapa, reverter toda a transação para garantir que nenhuma operação parcial seja aplicada.

Estrutura das Tabelas

Tabela `conta_bancaria`

Coluna	Tipo	Descrição
id_conta	SERIAL	Chave primária
nome_titular	VARCHAR	Nome do titular da conta
saldo	NUMERIC	Saldo atual da conta

Tabela `transacao`

Coluna	Tipo	Descrição
id_transacao	SERIAL	Chave primária
id_conta_origem	INTEGER	ID da conta de origem (chave estrangeira)
id_conta_destino	INTEGER	ID da conta de destino (chave estrangeira)
valor	NUMERIC	Valor transferido
data_transacao	TIMESTAMP	Data e hora da transação

Passos do Exercício

1. **Criação das Tabelas:** Crie as tabelas `conta_bancaria` e `transacao` usando a estrutura acima.
2. **Inserção de Dados Iniciais:** Insira alguns registros de contas na tabela `conta_bancaria` com saldos variados para simular transferências.
3. **Criação da Função de Transação:** Implemente uma função para transferir valores entre duas contas que:
    - Receba `id_conta_origem`, `id_conta_destino` e `valor` como parâmetros.
    - Verifique se o saldo da conta de origem é suficiente. Se não houver saldo suficiente, a transação deve falhar.
    - Debite o valor da conta de origem.
    - Credite o valor na conta de destino.
    - Registre a transação na tabela `transacao`.
4. **Isolamento e Reversão:** Coloque todas as operações dentro de uma transação, usando `BEGIN`, `COMMIT` e `ROLLBACK` para garantir que qualquer erro reverta todas as operações.