Atividade TCL / *Nykolas Guimarães Isler*

1. Introdução à Transaction Control Language (TCL)

O que é TCL?
Transaction Control Language (TCL) é um conjunto de comandos utilizados para gerenciar transações em sistemas de bancos de dados. Ela assegura que as operações no banco sejam realizadas de maneira segura, respeitando os princípios da integridade dos dados.

Importância da TCL:
TCL desempenha um papel crucial em cenários onde consistência e integridade de dados são primordiais, como em sistemas bancários, e-commerce e outros sistemas críticos. Ela permite controlar se as alterações realizadas.

Principais comandos da TCL:

1. COMMIT

Confirma todas as alterações realizadas em uma transação, tornando-as permanentes no banco de dados.

Exemplo:

```sql
START TRANSACTION;

INSERT INTO cliente (id_cliente, nome) VALUES (1, 'Maria Santos');

-- Confirma as mudanças realizadas
COMMIT;
```

2. ROLLBACK

Reverte todas as alterações feitas na transação desde seu início ou até um ponto de salvamento especificado.

Exemplo:

```sql
START TRANSACTION;

INSERT INTO cliente (id_cliente, nome) VALUES (2, 'Carlos Souza');

-- Algo deu errado; desfazer todas as alterações
ROLLBACK;
```

3. SAVEPOINT

Define um ponto dentro da transação que permite reverter somente até ele, sem desfazer toda a transação.

Exemplo:

```sql
START TRANSACTION;

INSERT INTO cliente (id_cliente, nome) VALUES (3, 'Ana Costa');

-- Define um ponto de salvamento
SAVEPOINT cliente_inserido;

-- Tentativa de inserir dados em outra tabela
INSERT INTO pedido (id_pedido, id_cliente, data_pedido) VALUES (201, 3, '2024-12-01');

-- Se houver um erro, reverte até o ponto de salvamento
ROLLBACK TO SAVEPOINT cliente_inserido;

-- Confirma o restante
COMMIT;
```

4.SET TRANSACTION

Define propriedades específicas para a transação, como o nível de isolamento.

Exemplo:

```sql
-- Define o nível de isolamento da transação
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

START TRANSACTION;

INSERT INTO cliente (id_cliente, nome) VALUES (4, 'João Lima');

-- Confirma a transação
COMMIT;
```

Níveis de isolamento mais comuns:

- **READ UNCOMMITTED:** Permite leitura de dados não confirmados por outras transações.
- **READ COMMITTED:** Permite leitura apenas de dados confirmados.
- **REPEATABLE READ:** Garante que as leituras dentro de uma transação sejam consistentes.
- **SERIALIZABLE:** Transações são completamente isoladas umas das outras.

Como TCL interage com ACID?

O modelo ACID é a base para o funcionamento confiável das transações em bancos de dados. Ele garante:

1. **Atomicidade:** Todas as operações da transação são concluídas ou nenhuma delas é aplicada.
    - Garantido pelo uso de **`ROLLBACK`** para desfazer mudanças em caso de falhas.
2. Consistência: Após a execução de uma transação, o banco permanece em um estado válido.
    - **`COMMIT`** assegura que as alterações são feitas de forma consistente.
3. **Isolamento:** Transações simultâneas não interferem umas nas outras.
    - Controlado por **`SET TRANSACTION`** e seus níveis de isolamento.
4. **Durabilidade:** Alterações confirmadas são permanentes mesmo após falhas no sistema.
    - Assegurado pelo **`COMMIT`.**

---

2. Como as Transações Funcionam

O que é uma transação em um banco de dados?

Uma transação é uma sequência de operações realizadas no banco de dados que são tratadas como uma única unidade lógica.

- Se todas as operações dentro da transação forem concluídas com sucesso, as alterações são confirmadas **(COMMIT).**
- Caso alguma operação falhe, todas as alterações feitas até então são revertidas **(ROLLBACK).**

Propriedades principais (ACID):

1. **Atomicidade:** A transação é "tudo ou nada".
2. **Consistência:** Garante que o banco de dados passe de um estado válido para outro.
3. **Isolamento:** Transações simultâneas não interferem entre si.
4. **Durabilidade:** Após confirmada, a transação é permanente.

Como o MySql lida com as transações de banco de dados?

No MySQL, o suporte a transações depende do mecanismo de armazenamento utilizado. O mais popular, **InnoDB**, é totalmente compatível com transações.

Funcionamento básico:

1. **Início:**
    - Transações começam com o comando **`START TRANSACTION`.**
    - Por padrão, operações no **MySQL** são automáticas **(autocommit)**. Para gerenciar manualmente, o autocommit deve ser desativado **(`SET autocommit = 0;`).**
2. **Durante a transação:**
    - Todas as alterações feitas ficam temporárias até serem confirmadas **(`COMMIT`)** ou revertidas **(`ROLLBACK`).**
3. **Finalização:**

- **Se bem-sucedida:**
  Todas as alterações são gravadas permanentemente com o comando **`COMMIT`.**
- **Se houver falha:**
  As alterações são descartadas com o comando **`ROLLBACK`**, garantindo que o banco de dados volte ao estado anterior.

Como garantir a integridade dos dados usando transações?

Uso de **ROLLBACK** em caso de erros:

- Ao detectar falhas em qualquer operação da transação, o uso de **`ROLLBACK`** evita que alterações incompletas sejam aplicadas.

SAVEPOINT para controle granular:

- Permite definir pontos intermediários, possibilitando reverter alterações apenas até um estado específico, sem desfazer toda a transação.

Níveis de isolamento:

- O comando **`SET TRANSACTION ISOLATION LEVEL`** ajusta como as transações interagem entre si, evitando problemas como leituras sujas **(dirty reads)** ou leituras repetidas.

Evitar interrupções não planejadas:

- Sempre finalize transações abertas para evitar bloqueios ou inconsistências.

---

3. Estudo de Caso

Exemplo: Processamento de Pagamentos em um Sistema Bancário

Imagine um sistema bancário onde um cliente transfere dinheiro de sua conta para a conta de outro cliente. Esse processo envolve duas operações:

1. Debitar o valor da conta de origem.
2. Creditar o valor na conta de destino.

Se qualquer uma dessas etapas falhar, o banco pode ficar com dados inconsistentes, como dinheiro debitado sem ser creditado na conta de destino ou valores incorretos nos saldos.

Como as transações são vantajosas nesse cenário?

Ao usar transações, garantimos que ambas as operações sejam tratadas como uma única unidade lógica. Isso significa que:

- Se ambas as etapas forem bem-sucedidas, as alterações são confirmadas com um **`COMMIT`.**

- Se uma das etapas falhar, todas as alterações são revertidas com um **ROLLBACK**, deixando o banco de dados no estado anterior à transação.

Como as transações garantem a consistência?

1. **Início da transação:**
   O sistema inicia uma transação com o comando **START TRANSACTION.**
2. **Execução das operações:**
   As operações de débito e crédito são realizadas sequencialmente.
3. **Verificação de sucesso:**
   - Se ambas as operações forem bem-sucedidas, a transação é finalizada com um **COMMIT.**
   - Se houver falha (por exemplo, saldo insuficiente ou problema técnico), a transação é revertida com um **ROLLBACK.**
4. **Isolamento e consistência:**
   - Níveis de isolamento são aplicados para evitar que outras transações acessem dados intermediários, garantindo a consistência.

---

4. Criação de um exemplo prático

Cenário:

Inserir dados em duas tabelas relacionadas **(`cliente` e `pedido`).**

- A tabela **`cliente`** armazena informações dos clientes.
- A tabela **`pedido`** registra os pedidos feitos por esses clientes.

Estrutura das tabelas:

```sql
-- Criar a tabela 'cliente'
CREATE TABLE cliente (
    id_cliente INT PRIMARY KEY,
    nome VARCHAR(100)
);

-- Criar a tabela 'pedido'
CREATE TABLE pedido (
    id_pedido INT PRIMARY KEY,
    id_cliente INT,
    data_pedido DATETIME,
    FOREIGN KEY (id_cliente) REFERENCES cliente(id_cliente)
);
```

Código do exemplo prático:

```sql
-- Iniciar a transaçao
START TRANSACTION;

-- Inserindo dados na tabela 'cliente'
INSERT INTO cliente (id_cliente, nome) VALUES (1, 'João Silva');

-- Definir um ponto de salvamento
SAVEPOINT cliente_inserted;

-- Tentativa de inserir dados na tabela 'pedido'
INSERT INTO pedido (id_pedido, id_cliente, data_pedido) VALUES (101, 1, NOW());

-- Simular um erro: tentar inserir um pedido com cliente inexistente
INSERT INTO pedido (id_pedido, id_cliente, data_pedido) VALUES (102, 999, NOW());

-- Reverter para o ponto de salvamento caso algo falhe
ROLLBACK TO SAVEPOINT cliente_inserted;

-- Confirmar as alterações feitas até o ponto de salvamento
COMMIT;

```

Explicação do Fluxo

1. **Início da Transação:**
   - Usamos **`START TRANSACTION`** para começar uma nova transação no banco de dados.
2. **Inserção na Tabela `cliente`:**
   - Um registro é adicionado à tabela **`cliente` com `id_cliente = 1` e `nome = 'João Silva'`.**
3. **Definição de um SAVEPOINT:**
   - Criamos um ponto de salvamento chamado **`cliente_inserted`** com o comando `SAVEPOINT`. Isso nos permite reverter para este ponto se necessário.
4. **Inserção na Tabela `pedido`:**
   - Primeiro, adicionamos um pedido válid**o (`id_pedido = 101`**) associado ao cliente **`id_cliente = 1`.**
5. **Erro Simulado:**
   - Tentamos adicionar outro pedido, mas com um **`id_cliente = 999`**, que não existe na tabela `cliente`. Esse erro viola a integridade referencial e é tratado na transação.
6. **ROLLBACK TO SAVEPOINT:**
   - Usamos **`ROLLBACK TO SAVEPOINT cliente_inserted`** para desfazer a inserção problemática na tabela `pedido`, mas manter as alterações na tabela `cliente`.
7. **Confirmação da Transação:**
   - Finalmente, usamos **`COMMIT`** para confirmar as alterações realizadas até o ponto de salvamento.

---

5. Conclusão

A importância da TCL

A Transaction Control Language (TCL) é essencial para a gestão de transações em sistemas de banco de dados, pois garante que as operações sejam realizadas de forma consistente e segura. Seus comandos como **COMMIT**, **ROLLBACK**, **SAVEPOINT**, e **SET TRANSACTION** permitem que desenvolvedores controlem o comportamento de transações, assegurando a integridade dos dados mesmo em cenários de falha.

A TCL é fundamental para implementar o modelo **ACID** (Atomicidade, Consistência, Isolamento, Durabilidade), que é o alicerce para confiabilidade e integridade em transações complexas.

Vantagens do uso de transações em sistemas críticos

1. **Consistência dos Dados**
    - Em sistemas críticos, como bancos e e-commerces, erros ou falhas podem causar inconsistências nos dados. As transações garantem que operações como transferências bancárias ou atualizações de estoques sejam realizadas completamente ou não sejam feitas.
2. **Recuperação em Caso de Falha**
    - Usando **ROLLBACK**, é possível desfazer alterações em caso de falhas, evitando corrupção ou perda de dados.
3. **Controle de Concorrência**
    - Em ambientes com múltiplos usuários acessando o banco simultaneamente, transações ajudam a evitar problemas como leituras sujas (dirty reads) ou atualizações perdidas.
4. **Isolamento de Operações**
    - Níveis de isolamento, configurados com **SET TRANSACTION**, garantem que transações concorrentes não interfiram umas nas outras, mantendo a consistência.
5. **Segurança e Confiabilidade**
    - Em sistemas bancários, onde milhares de transações ocorrem por segundo, a TCL garante que os dados permaneçam seguros e consistentes mesmo em situações de alta carga ou falhas técnicas.
6. **Experiência do Usuário**
    - Em plataformas de vendas online, transações garantem que um pedido seja registrado apenas se o pagamento for processado, evitando problemas como cobranças indevidas ou pedidos inconsistentes.

Exemplo de Aplicação em Sistemas Críticos

- **Bancos:** Garantir que uma transferência de dinheiro debite uma conta e credite outra, de forma integrada.
- **E-commerce:** Registrar um pedido e reduzir a quantidade do produto no estoque simultaneamente, ou desfazer ambas as ações em caso de falha.
- **Sistemas de Reserva:** Confirmar ou cancelar uma reserva de passagem aérea ou quarto de hotel dependendo do sucesso de todas as etapas