Trabalho Banco de Dados

Transaction Control Language (TCL) no MySQL

1. Transaction Control Language (TCL)

Definição e Importância

A Transaction Control Language (TCL) é um conjunto de comandos SQL utilizados para gerenciar transações em um banco de dados. Sua importância é fundamental pois permite garantir a integridade e consistência dos dados durante operações complexas que envolvem múltiplas alterações no banco de dados.

Características Fundamentais da TCL:

- Controle granular sobre mudanças no banco de dados
- Capacidade de reverter operações em caso de erro
- Garantia de consistência em operações relacionadas
- Suporte a operações concorrentes seguras

Interação com o Modelo ACID

A TCL trabalha diretamente com os princípios ACID:

- Atomicidade: Garante que todas as operações sejam executadas com sucesso ou nenhuma seja realizada
- Consistência: Mantém o banco de dados em estado válido antes e após a transação
- Isolamento: Assegura que transações concorrentes não interfiram entre si
- Durabilidade: Garante que alterações confirmadas sejam permanentes

Principais Comandos TCL

1. **COMMIT**
    - Finaliza uma transação, salvando permanentemente todas as alterações realizadas
    - Libera todos os locks de dados adquiridos durante a transação
    - Torna as alterações visíveis para outras sessões
2. **ROLLBACK**
    - Desfaz todas as alterações realizadas desde o início da transação
    - Retorna o banco de dados ao estado anterior ao início da transação
    - Libera todos os locks adquiridos durante a transação
3. **SAVEPOINT**
    - Cria um ponto de salvamento dentro de uma transação
    - Permite realizar rollback parcial até um ponto específico
    - Oferece maior granularidade no controle de transações
4. **SET TRANSACTION**
    - Define características da transação
    - Permite configurar o nível de isolamento
    -

2. Funcionamento das Transações

Definição de Transação

Uma transação é uma unidade lógica de trabalho que contém uma ou mais operações de banco de dados. Representa uma mudança consistente no estado do banco de dados.

Estados de uma Transação

1. Ativa
2. Parcialmente commitada
3. Commitada
4. Falha
5. Abortada

Gerenciamento de Transações no MySQL

Início de Transação

```
START TRANSACTION;
-- ou
BEGIN WORK;
```

Controle de Autocommit

```
SET autocommit = 0; -- Desabilita autocommit
SET autocommit = 1; -- Habilita autocommit
```

Tratamento de Erros

```
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
BEGIN
    ROLLBACK;
    -- Tratamento de erro
END;
```

Garantia de Integridade

Locks e Bloqueios

- Shared Locks (S-Locks)
- Exclusive Locks (X-Locks)
- Intention Locks
- Record Locks
- Gap Locks

Níveis de Isolamento

READ UNCOMMITTED

- Menor nível de isolamento
- Permite dirty reads
- Maior performance

READ COMMITTED

- Previne dirty reads
- Permite non-repeatable reads
- Nível padrão em muitos SGBD

REPEATABLE READ

- Previne non-repeatable reads
- Permite phantom reads
- Padrão no MySQL

SERIALIZABLE

- Maior nível de isolamento
- Previne todos os fenômenos de concorrência
- Menor performance

3. Estudo de Caso Expandido: Sistema Bancário

Cenário Detalhado

Sistema de transferência bancária com:

- Verificação de saldo
- Registro de transações
- Notificação de clientes
- Auditoria

Implementação com Transações

```
START TRANSACTION;

-- Verificação de saldo
SELECT @saldo := saldo FROM contas WHERE conta_id = @origem FOR UPDATE;

IF @saldo >= @valor_transferencia THEN
    -- Débito na conta origem
    UPDATE contas SET saldo = saldo - @valor_transferencia
    WHERE conta_id = @origem;

    -- Crédito na conta destino
    UPDATE contas SET saldo = saldo + @valor_transferencia
    WHERE conta_id = @destino;

    -- Registro da transação
    INSERT INTO transferencias (origem, destino, valor, data)
    VALUES (@origem, @destino, @valor_transferencia, NOW());

    COMMIT;
ELSE
    ROLLBACK;
END IF;
```

Vantagens das Transações neste Cenário

1. Garantia de consistência do saldo
2. Prevenção de débitos duplicados
3. Rastreabilidade completa
4. Recuperação automática em caso de falhas

4. Exemplo Prático Expandido

Cenário de E-commerce

```sql
-- Estrutura expandida das tabelas
CREATE TABLE clientes (
    id_cliente INT PRIMARY KEY,
    nome VARCHAR(100),
    saldo DECIMAL(10,2),
    limite_credito DECIMAL(10,2),
    status VARCHAR(20)
);

CREATE TABLE pedidos (
    id_pedido INT PRIMARY KEY,
    id_cliente INT,
    valor DECIMAL(10,2),
    status VARCHAR(20),
    data_pedido TIMESTAMP,
    FOREIGN KEY (id_cliente) REFERENCES clientes(id_cliente)
);

CREATE TABLE itens_pedido (
    id_item INT PRIMARY KEY,
    id_pedido INT,
    id_produto INT,
    quantidade INT,
    valor_unitario DECIMAL(10,2),
    FOREIGN KEY (id_pedido) REFERENCES pedidos(id_pedido)
);

CREATE TABLE estoque (
    id_produto INT PRIMARY KEY,
    quantidade INT,
    reservado INT
);

-- Transação completa de pedido
DELIMITER $$
CREATE PROCEDURE criar_pedido(
    IN p_cliente_id INT,
    IN p_produto_id INT,
    IN p_quantidade INT
)
BEGIN
    DECLARE v_valor_total DECIMAL(10,2);
    DECLARE v_estoque_disponivel INT;

    START TRANSACTION;

    -- Verificar estoque
    SELECT quantidade - reservado INTO v_estoque_disponivel
    FROM estoque WHERE id_produto = p_produto_id FOR UPDATE;

    IF v_estoque_disponivel >= p_quantidade THEN
        -- Criar pedido
        INSERT INTO pedidos (id_cliente, valor, status, data_pedido)
        VALUES (p_cliente_id, v_valor_total, 'PENDENTE', NOW());

        SET @id_pedido = LAST_INSERT_ID();

        -- Reservar estoque
        UPDATE estoque
        SET reservado = reservado + p_quantidade
        WHERE id_produto = p_produto_id;

        -- Inserir itens do pedido
        INSERT INTO itens_pedido (id_pedido, id_produto, quantidade, valor_unitario)
        VALUES (@id_pedido, p_produto_id, p_quantidade,
                (SELECT preco FROM produtos WHERE id_produto = p_produto_id));

        COMMIT;
    ELSE
        ROLLBACK;
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Estoque insuficiente';
    END IF;
END $$
DELIMITER ;
```

5. Conclusão

Importância da TCL em Diferentes Contextos

Sistemas Financeiros

- Garantia de operações atômicas
- Rastreabilidade de transações
- Conformidade regulatória
- Recuperação de desastres

E-commerce

- Gestão de estoque
- Processamento de pedidos
- Integração com pagamentos
- Experiência do usuário

Sistemas Empresariais

- Integração entre sistemas
- Processamento em lote
- Migração de dados
- Backup e recuperação

Tendências Futuras

- Transações distribuídas
- Microsserviços e consistência eventual
- NoSQL e transações
- Blockchain e smart contracts

Melhores Práticas

1. **Planejamento de Transações**
    - Manter transações curtas
    - Minimizar locks
    - Definir pontos de salvamento estratégicos
2. **Tratamento de Erros**
    - Implementar handlers adequados
    - Logging de erros
    - Notificação de falhas
3. **Performance**
    - Escolher nível de isolamento apropriado
    - Otimizar queries dentro de transações
    - Monitorar deadlocks
4. **Segurança**
    - Controle de acesso
    - Auditoria de transações
    - Backup e recovery