Guia Completo de Comandos DML no MariaDB

A DML (Linguagem de Manipulação de Dados) é o subconjunto de SQL usado para adicionar, modificar e remover dados das tabelas. Diferente da DDL, que lida com a *estrutura*, a DML lida com o *conteúdo*.

Os três comandos essenciais da DML são:

- **INSERT:** Adiciona novos registros (linhas) a uma tabela.
- **UPDATE:** Modifica registros existentes em uma tabela.
- **DELETE:** Remove registros existentes de uma tabela.

Uma característica fundamental dos comandos DML é que eles operam dentro de **transações**, o que significa que as alterações podem ser confirmadas (COMMIT) ou desfeitas (ROLLBACK).

Cenário de Exemplo

Para todos os exemplos, vamos continuar usando o banco de dados empresa com as tabelas setores e funcionarios.

Estrutura das Tabelas:

```
CREATE TABLE IF NOT EXISTS setores (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  nome VARCHAR(100) NOT NULL UNIQUE  
);
```

```
CREATE TABLE IF NOT EXISTS funcionarios (  
  matricula INT PRIMARY KEY,  
  nome_completo VARCHAR(255) NOT NULL,  
  data_admissao DATE NOT NULL,  
  salario DECIMAL(10, 2),  
  setor_id INT,  
  CONSTRAINT fk_setor  
  FOREIGN KEY (setor_id) REFERENCES setores(id)  
);
```

1. INSERT: Adicionando Novos Dados

O comando INSERT é usado para popular as tabelas com novos registros.

Varição 1: INSERT Básico (Ordem das Colunas)

Esta é a forma mais simples, onde você fornece os valores na mesma ordem em que as colunas foram declaradas na tabela.

Sintaxe:

```
INSERT INTO nome_da_tabela VALUES (valor1, valor2, valor3, ...);
```

Exemplo: Adicionando um setor. A tabela setores tem as colunas id (auto-incremento) e nome. Como o id é gerado automaticamente, podemos passar NULL ou DEFAULT para ele.

```
INSERT INTO setores VALUES (NULL, 'Recursos Humanos');
```

- NULL: Informa ao banco para usar o valor padrão da coluna, que no caso do AUTO_INCREMENT, é o próximo número da sequência.

Varição 2: INSERT Especificando as Colunas (Recomendado)

Esta é a forma **mais segura e recomendada**. Você especifica explicitamente para quais colunas está enviando valores. Isso torna seu código mais legível e resiliente a mudanças na estrutura da tabela (como adicionar uma nova coluna no futuro).

Sintaxe:

```
INSERT INTO nome_da_tabela (coluna1, coluna2) VALUES (valor1, valor2);
```

Exemplo: Adicionando um funcionário.

-- Primeiro, vamos garantir que o setor 'Recursos Humanos' tenha o ID 1

-- Supondo que a tabela estava vazia, o ID será 1.

```
INSERT INTO funcionarios (matricula, nome_completo, data_admissao, salario, setor_id)
VALUES (101, 'Ana Clara Souza', '2023-05-10', 4500.00, 1);
```

- A ordem dos valores corresponde exatamente à ordem das colunas que especificamos.

Varição 3: INSERT de Múltiplas Linhas

Para maior eficiência, você pode inserir vários registros com um único comando, separando os conjuntos de valores por vírgulas.

Sintaxe:

```
INSERT INTO nome_da_tabela (coluna1, coluna2) VALUES
```

```
(valor1_linha1, valor2_linha1),  
(valor1_linha2, valor2_linha2),  
(valor1_linha3, valor2_linha3);
```

Exemplo: Adicionando mais setores de uma só vez.

```
INSERT INTO setores (nome) VALUES
```

```
('Tecnologia da Informação'),  
(Financeiro),  
(Marketing);
```

- Note que não precisamos mais passar NULL para o id, pois especificamos apenas a coluna nome.

Exemplo: Adicionando mais funcionários.

-- Supondo que os IDs dos setores são: TI=2, Financeiro=3, Marketing=4

```
INSERT INTO funcionarios (matricula, nome_completo, data_admissao, salario, setor_id)  
VALUES
```

```
(102, 'Bruno Costa', '2022-08-01', 8200.00, 2),  
(103, 'Carlos Dias', '2024-01-15', 6100.00, 3),  
(104, 'Daniela Rocha', '2021-11-20', 7500.00, 2);
```

Variação 4: INSERT ... SELECT

Esta variação poderosa permite inserir dados em uma tabela a partir do resultado de uma consulta SELECT em outra tabela. É ideal para copiar ou arquivar dados.

Sintaxe:

```
INSERT INTO tabela_destino (coluna1, coluna2)  
SELECT coluna_origem1, coluna_origem2 FROM tabela_origem WHERE condicao;
```

Exemplo: Vamos criar uma tabela para ex-funcionários e mover um registro para lá.

-- 1. Criar a tabela de arquivo morto

```
CREATE TABLE IF NOT EXISTS funcionarios_demitidos (  
    matricula INT,  
    nome_completo VARCHAR(255),
```

```
data_saida DATE,  
ultimo_salario DECIMAL(10,2)  
);
```

-- 2. Copiar os dados do funcionário com matrícula 103 para a nova tabela

```
INSERT INTO funcionarios_demitidos (matricula, nome_completo, ultimo_salario,  
data_saida)
```

```
SELECT matricula, nome_completo, salario, CURDATE()
```

```
FROM funcionarios
```

```
WHERE matricula = 103;
```

- CURDATE(): É uma função que retorna a data atual.
- A estrutura da SELECT (as colunas selecionadas) deve corresponder à estrutura do INSERT (as colunas de destino).

2. UPDATE: Modificando Dados Existentes

O comando UPDATE é usado para alterar os valores em registros que já existem na tabela.

ALERTA MÁXIMO: O componente mais crítico do UPDATE é a cláusula **WHERE**. Se você esquecer a cláusula WHERE, **TODAS as linhas da tabela serão atualizadas!** Sempre verifique sua condição WHERE.

Sintaxe:

```
UPDATE nome_da_tabela
```

```
SET coluna1 = novo_valor1, coluna2 = novo_valor2
```

```
WHERE condicao;
```

Exemplo 1: Atualizando um único registro Vamos promover a funcionária Ana Clara e aumentar seu salário.

```
UPDATE funcionarios
```

```
SET salario = 5200.00
```

```
WHERE matricula = 101;
```

Exemplo 2: Atualizando múltiplos registros Vamos dar um aumento de 10% para todos os funcionários do setor de 'Tecnologia da Informação' (ID 2).

```
UPDATE funcionarios
```

```
SET salario = salario * 1.10
```

```
WHERE setor_id = 2;
```

- Note que podemos usar o valor atual da coluna (salario) para calcular o novo valor.

Exemplo 3: Atualizando múltiplas colunas O funcionário Bruno Costa mudou de setor e recebeu uma promoção.

```
UPDATE funcionarios
```

```
SET
```

```
    setor_id = 4, -- Mudou para o Marketing
```

```
    salario = 9000.00
```

```
WHERE matricula = 102;
```

3. DELETE: Removendo Dados

O comando DELETE remove um ou mais registros de uma tabela.

ALERTA MÁXIMO: Assim como no UPDATE, a cláusula **WHERE** é fundamental. Se você esquecer a cláusula WHERE, **TODOS os registros da tabela serão excluídos!**

Sintaxe:

```
DELETE FROM nome_da_tabela WHERE condicao;
```

Exemplo 1: Removendo um único registro O funcionário Carlos Dias foi movido para a tabela de demitidos, agora vamos removê-lo da tabela principal.

```
DELETE FROM funcionarios WHERE matricula = 103;
```

Exemplo 2: Removendo múltiplos registros Vamos supor que um setor inteiro foi desativado e todos os funcionários precisam ser removidos.

-- CUIDADO: Este comando removeria TODOS os funcionários do setor de Marketing (ID 4)

```
DELETE FROM funcionarios WHERE setor_id = 4;
```

DELETE vs. TRUNCATE vs. DROP - Resumo Final

- **DELETE (DML):** Remove linhas baseadas na cláusula WHERE. Lento, pode ser desfeito (rollback), dispara gatilhos (triggers).

- **TRUNCATE (DDL):** Esvazia a tabela inteira. Rápido, não pode ser desfeito, reinicia AUTO_INCREMENT.
- **DROP (DDL):** Exclui a tabela inteira, incluindo sua estrutura. A tabela deixa de existir.