Manipulação de Dados em SQL

DML - INSERT, UPDATE, DELETE

"Um banco de dados só tem valor quando os dados estão lá."

INSERT

UPDATE

DELETE

Aula 07 - UC3 | Ciência de Dados

Contexto e Objetivos da Aula

"Um banco de dados só tem valor quando os dados estão lá."

O **DML (Data Manipulation Language)** é o conjunto de comandos SQL (INSERT, UPDATE, DELETE) usados para gerenciar e modificar os dados dentro das tabelas, que foram criadas com o DDL nas aulas anteriores. Dominar o DML é essencial para garantir que o banco de dados funcione como um sistema de informação válido e confiável.

Objetivos de Aprendizagem

Objetivo 1

Compreender e aplicar o comando **INSERT** para adicionar novos registros às tabelas, respeitando tipos de dados e restrições.

Objetivo 2

Dominar o comando **UPDATE** com a cláusula WHERE para modificar dados existentes com segurança e precisão.

Objetivo 3

Aplicar o comando **DELETE** com consciência de integridade referencial e boas práticas de segurança.

Objetivo 4

Entender a importância da **Integridade de Dados** e como as restrições (NOT NULL, UNIQUE, CHECK, FK) protegem a validade dos dados.

Fundamentos do DML e Integridade de Dados

INSERT

Usado para **adicionar novos registros (linhas)** às tabelas.

Sintaxe Básica:

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

Regras Críticas:

- Número de valores = número de colunas
- Valores devem respeitar **tipos de dados**
- Colunas **NOT NULL** devem ter valores
- Colunas **UNIQUE** não podem ter duplicatas

UPDATE

Usado para **modificar dados existentes** nas tabelas.

Sintaxe Básica:

```
UPDATE Tabela
SET coluna = valor
WHERE condição;
```

Regras Críticas:

- A cláusula **WHERE é obrigatória**
- **UPDATE sem WHERE** altera TODOS os registros!
- Valores devem respeitar tipos de dados
- Restrições (UNIQUE, CHECK) devem ser respeitadas

DELETE

Usado para **remover registros** das tabelas.

Sintaxe Básica:

```
DELETE FROM Tabela
WHERE condição;
```

Regras Críticas:

- A cláusula **WHERE é obrigatória**
- **DELETE sem WHERE** remove TODOS os registros!
- Respeitar **Integridade Referencial** (FOREIGN KEY)
- Deletar na **ordem inversa** das dependências

INSERT: Inserção de Dados - Sintaxe e Regras

Sintaxe Básica do INSERT

Forma Completa (Recomendada):

```
INSERT INTO Clientes
  (id, nome, email, telefone)
VALUES
  (1, 'João Silva',
   'joao@email.com',
   '11-98765-4321');
```

Forma Resumida (Menos Segura):

```
INSERT INTO Clientes
VALUES
  (1, 'João Silva',
   'joao@email.com',
   '11-98765-4321');
```

Regras Críticas do INSERT

Número de Valores

O número de **valores** deve ser **EXATAMENTE igual** ao número de **colunas** especificadas.

Tipos de Dados

Os valores devem respeitar os **tipos de dados** definidos na tabela (INT, VARCHAR, DATE, etc.).

Restrição NOT NULL

Colunas com **NOT NULL** **obrigatoriamente** devem receber um valor.

Restrição UNIQUE

Colunas com **UNIQUE** não podem ter valores **duplicados** na tabela.

INSERT: Exemplos Práticos

Exemplo 1: Todas as Colunas

Inserir um novo cliente especificando todas as colunas (mais seguro e explícito).

```sql
INSERT INTO Clientes
  (id, nome, email, telefone)
VALUES
  (1, 'Maria Santos',
   'maria@email.com',
   '11-97654-3210');
```

✓ **Resultado:** Cliente com todas as colunas preenchidas é inserido com sucesso.

Exemplo 2: Colunas Selecionadas

Inserir apenas algumas colunas; as outras receberão NULL ou DEFAULT.

```sql
INSERT INTO Clientes
  (nome, email)
VALUES
  ('Pedro Costa',
   'pedro@email.com');
```

✓ **Resultado:** id (auto-increment), nome, email inseridos. telefone recebe NULL.

Exemplo 3: Múltiplas Linhas

Inserir vários registros em uma única instrução (mais eficiente).

```sql
INSERT INTO Produtos
  (nome, preco, estoque)
VALUES
  ('Notebook', 3500.00, 5),
  ('Mouse', 50.00, 25),
  ('Teclado', 150.00, 15);
```

✓ **Resultado:** 3 produtos inseridos em uma única operação.

Exemplo 4: Valores DEFAULT

Inserir registros deixando colunas com DEFAULT receberem seus valores padrão.

```sql
INSERT INTO Produtos
  (nome, preco)
VALUES
  ('Webcam', 200.00);

-- estoque recebe DEFAULT (0)
-- data_cadastro recebe CURRENT_DATE
```

✓ **Resultado:** Produto inserido com estoque=0 (DEFAULT) e data_cadastro=hoje.

INSERT: Erros Comuns e Restrições

Erro 1: Número de Valores Incorreto

```
INSERT INTO Clientes
  (nome, email)
VALUES
  ('João', 'joao@email.com',
   '11-98765-4321');
```

Motivo: 2 colunas especificadas, mas 3 valores fornecidos. O número de valores deve ser igual ao número de colunas.

Mensagem de Erro:

```
Column count doesn't match value
count at row 1
```

Erro 2: Violação de NOT NULL

```
INSERT INTO Clientes
  (email, telefone)
VALUES
  ('teste@email.com',
   '11-98765-4321');
```

Motivo: A coluna 'nome' é NOT NULL e não foi fornecida. Todas as colunas NOT NULL devem ter valores.

Mensagem de Erro:

```
Field 'nome' doesn't have a
default value
```

Erro 3: Violação de UNIQUE

```
INSERT INTO Clientes
  (nome, email)
VALUES
  ('Novo Cliente',
   'joao@email.com');
```

Motivo: O email 'joao@email.com' já existe na tabela. A restrição UNIQUE não permite valores duplicados.

Mensagem de Erro:

```
Duplicate entry 'joao@email.com'
for key 'email'
```

Erro 4: Violação de CHECK

```
INSERT INTO Produtos
  (nome, preco)
VALUES
  ('Produto X', -100.00);
```

Motivo: O preço é negativo, violando a restrição CHECK (preco > 0). A restrição CHECK valida regras de negócio.

Mensagem de Erro:

```
Check constraint 'preco' is violated
```

UPDATE: Atualização de Dados - Parte 1

O comando **UPDATE** modifica dados existentes nas tabelas. A cláusula **WHERE** é **essencial** para especificar quais registros serão atualizados. Sem WHERE, **TODOS** os registros serão modificados!

Exemplo 1: UPDATE Básico

Atualizar uma coluna de um registro específico usando WHERE.

```
UPDATE Clientes
SET email = 'novo@email.com'
WHERE id = 1;
```

Resultado: O email do cliente com id=1 é atualizado para 'novo@email.com'

Exemplo 2: Múltiplas Colunas

Atualizar várias colunas no mesmo comando separadas por vírgula.

```
UPDATE Clientes
SET email = 'maria@email.com',
    telefone = '11-99999-9999'
WHERE nome = 'Maria Santos';
```

Resultado: Email e telefone são atualizados para Maria Santos

Exemplo 3: Expressão Matemática

Usar expressões matemáticas para calcular novos valores baseados nos valores atuais.

```
UPDATE Produtos
SET estoque = estoque - 5
WHERE id = 1;
```

Resultado: O estoque do produto com id=1 é reduzido em 5 unidades

Exemplo 4: Múltiplas Condições

Usar AND/OR para atualizar registros que atendem a múltiplas condições.

```
UPDATE Produtos
SET preco = preco * 1.10
WHERE categoria = 'Eletrônicos'
  AND estoque > 0;
```

Resultado: Preço aumenta 10% para produtos eletrônicos com estoque > 0

UPDATE: A Importância Crítica do WHERE

PERIGO CRÍTICO: UPDATE SEM WHERE

Executar um UPDATE **sem a cláusula WHERE** é **CATASTRÓFICO**. Todos os registros da tabela serão atualizados, causando **perda ou corrupção de dados**.

```
-- NUNCA EXECUTE ISSO! UPDATE Clientes SET email = 'novo@email.com'; -- Resultado: TODOS os clientes terão o mesmo email!
```

Consequência: Perda de dados irreversível. Todos os emails originais são sobrescritos!

Exemplos Corretos: UPDATE com WHERE

UPDATE com Condição Simples

```
UPDATE Clientes
SET email = 'novo@email.com'
WHERE id = 5;
```

✓ **Correto:** Apenas o cliente com id=5 é atualizado. Outros clientes não são afetados.

UPDATE com Múltiplas Colunas

```
UPDATE Produtos
SET preco = 150.00,
    estoque = 10
WHERE categoria = 'Eletrônicos';
```

✓ **Correto:** Apenas produtos eletrônicos são atualizados. Outras categorias permanecem inalteradas.

UPDATE com Condição AND

```
UPDATE Pedidos
SET status = 'Processado'
WHERE id_cliente = 1
  AND data_pedido > '2025-01-01';
```

✓ **Correto:** Apenas pedidos do cliente 1 após 2025-01-01 são atualizados.

UPDATE com Condição OR

```
UPDATE Clientes
SET status = 'Inativo'
WHERE telefone IS NULL
   OR email IS NULL;
```

✓ **Correto:** Apenas clientes sem telefone OU sem email são marcados como inativos.

UPDATE: Boas Práticas de Verificação

Boas Práticas de Verificação Antes de UPDATE

- **1. Verifique a condição:** Sempre escreva a cláusula WHERE **ANTES** de escrever SET. Isso garante que você sempre especifique a condição.
- **2. Teste com SELECT:** Use **SELECT** com a mesma condição para ver **quantos registros** serão afetados antes de executar o UPDATE.
- **3. Use transações:** Comece com **START TRANSACTION**, execute o UPDATE, revise os resultados e execute **COMMIT** ou **ROLLBACK**.
- **4. Faça backup:** Sempre faça **backup** antes de executar UPDATE em produção. Dados atualizados incorretamente podem ser irrecuperáveis.

DELETE: Remoção de Dados - Parte 1

O Comando DELETE

O DELETE é usado para **remover registros (linhas) das tabelas**. Como o UPDATE, o DELETE é um comando poderoso que requer cuidado extremo, especialmente com a cláusula WHERE.

Sintaxe Básica

```
DELETE FROM Tabela
WHERE condição;
```

Componentes:

- **DELETE FROM:** Inicia o comando
- **Tabela:** Nome da tabela
- **WHERE:** Especifica qual(is) registro(s)

Exemplos Práticos

Exemplo 1: Deletar por ID

```
DELETE FROM Clientes WHERE id = 5;
```

✓ Remove cliente com id=5

Exemplo 2: Deletar por Condição

```
DELETE FROM Produtos WHERE estoque = 0;
```

✓ Remove produtos sem estoque

Exemplo 3: Múltiplas Condições

```
DELETE FROM Pedidos WHERE status = 'Cancelado' AND data_pedido <
'2025-01-01';
```

✓ Remove pedidos antigos cancelados

Integridade Referencial e FOREIGN KEY

⚠️ Problema: Violação de FK

Se um cliente tem **pedidos associados**, tentar deletar o cliente resultará em **erro de integridade referencial**. O SGBD protege a relação entre tabelas.

✓ Solução: Ordem Correta

Passo 1: Deletar os pedidos do cliente
Passo 2: Deletar o cliente
Sempre delete na **ordem inversa** das dependências!

DELETE: Integridade Referencial

Violação de Integridade Referencial

Quando um registro pai tem registros filhos associados (via FOREIGN KEY), tentar deletar o registro pai resulta em **erro de integridade referencial**. O SGBD protege a relação entre tabelas.

```
-- Tentativa de deletar cliente com pedidos
DELETE FROM Clientes WHERE id = 1;

-- Erro esperado:
-- Cannot delete or update a parent row:
-- a foreign key constraint fails
```

Erro: O cliente tem pedidos associados. Não pode ser deletado sem deletar os pedidos primeiro!

ON DELETE CASCADE vs ON DELETE RESTRICT

ON DELETE RESTRICT (Padrão)

Comportamento: Impede a deleção do registro pai se houver registros filhos associados.

```
CREATE TABLE Pedidos (
  id INT PRIMARY KEY,
  id_cliente INT,
  FOREIGN KEY (id_cliente)
    REFERENCES Clientes(id)
    ON DELETE RESTRICT
);
```

Resultado: Tentar deletar um cliente com pedidos gera **erro**. Você deve deletar os pedidos primeiro, depois o cliente.

ON DELETE CASCADE

Comportamento: Deleta automaticamente todos os registros filhos quando o registro pai é deletado.

```
CREATE TABLE Pedidos (
  id INT PRIMARY KEY,
  id_cliente INT,
  FOREIGN KEY (id_cliente)
    REFERENCES Clientes(id)
    ON DELETE CASCADE
);
```

Resultado: Deletar um cliente **automaticamente deleta seus pedidos**. Cuidado: pode resultar em perda de dados!

DELETE: Boas Práticas de Segurança

Boas Práticas de Segurança no DELETE

Usar Transações

Comece com **START TRANSACTION**, execute o DELETE, revise os resultados e execute **COMMIT** ou **ROLLBACK**.

Fazer Backup

Sempre faça **backup** antes de executar DELETE em produção. Dados deletados são **permanentemente perdidos**.

Testar com SELECT

Use **SELECT** com a mesma condição do DELETE para ver quantos registros serão afetados antes de executar.

Ordem Correta de Deleção

Sempre delete na **ordem inversa** das dependências. Filhos antes de pais.

Atividade Prática: DML em Ação

Trabalhem em **duplas** para completar as seguintes atividades usando as tabelas **Clientes**, **Produtos** e **Pedidos**. Cada atividade deve incluir código SQL comentado e testes de validação.

Atividade 1: INSERT

Objetivo: Inserir novos registros respeitando restrições.

Tarefas:

- **Tarefa 1.1:** Inserir 3 novos clientes na tabela Clientes
- **Tarefa 1.2:** Um cliente deve ter email UNIQUE válido
- **Tarefa 1.3:** Tentar inserir um cliente sem nome (NOT NULL) para ver o erro
- **Tarefa 1.4:** Inserir 2 produtos com preço > 0 (CHECK)

```
-- Exemplo: Inserir cliente
INSERT INTO Clientes
  (nome, email, telefone)
VALUES
  ('João Silva',
   'joao@email.com',
   '11-98765-4321');

-- Exemplo: Inserir produto
INSERT INTO Produtos
  (nome, preco, estoque)
VALUES
  ('Notebook', 3500.00, 5);
```

Atividade 2: UPDATE

Objetivo: Atualizar dados com WHERE específico.

Tarefas:

- **Tarefa 2.1:** Atualizar email de um cliente específico
- **Tarefa 2.2:** Atualizar múltiplas colunas de um produto
- **Tarefa 2.3:** Reduzir estoque de um produto em 5 unidades
- **Tarefa 2.4:** Atualizar status de pedidos com múltiplas condições (AND)

```
-- Exemplo: Atualizar email
UPDATE Clientes
SET email = 'novo@email.com'
WHERE id = 1;

-- Exemplo: Reduzir estoque
UPDATE Produtos
SET estoque = estoque - 5
WHERE id = 1;
```

Atividade 3: DELETE

Objetivo: Remover registros com segurança e integridade.

Tarefas:

- **Tarefa 3.1:** Deletar um produto sem pedidos associados
- **Tarefa 3.2:** Tentar deletar um cliente com pedidos (erro esperado)
- **Tarefa 3.3:** Deletar pedidos de um cliente, depois o cliente
- **Tarefa 3.4:** Documentar a ordem correta de deleção

```
-- Exemplo: Deletar produto
DELETE FROM Produtos
WHERE id = 5 AND estoque = 0;

-- Ordem correta:
-- 1. DELETE FROM Pedidos WHERE id_cliente =
1;
-- 2. DELETE FROM Clientes WHERE id = 1;
```

Síntese e Consolidação Final

Resumo dos Três Comandos DML

INSERT

Adiciona novos registros às tabelas, respeitando tipos de dados e restrições (NOT NULL, UNIQUE, CHECK).

- Número de valores = colunas
- Respeitar tipos de dados
- Validar restrições

UPDATE

Modifica dados existentes com a cláusula WHERE obrigatória para especificar quais registros atualizar.

- WHERE é OBRIGATÓRIO
- Testar com SELECT antes
- Usar transações

DELETE

Remove registros respeitando integridade referencial e ordem correta de deleção.

- WHERE é OBRIGATÓRIO
- Respeitar FOREIGN KEY
- Deletar filhos antes de pais

A Importância do DML para o Sistema de Informação

O DML é o **"motor" do sistema de informação**. Sem INSERT, UPDATE e DELETE, o banco de dados seria apenas um repositório estático de dados. Dominar o DML com **segurança** (WHERE, transações, backups) e **consciência de integridade** (restrições, FOREIGN KEY) é o que garante que o banco de dados funcione como um sistema válido, confiável e eficiente.

Checklist de Boas Práticas Essenciais

- **INSERT:** Validar tipos de dados, NOT NULL, UNIQUE e CHECK antes de inserir
- **DELETE:** Respeitar FOREIGN KEY, deletar na ordem inversa das dependências
- **Backup:** Sempre fazer backup antes de operações em produção
- **Testes:** Testar em ambiente de desenvolvimento antes de produção
- **UPDATE:** Sempre escrever WHERE antes de SET, testar com SELECT
- **Transações:** Usar START TRANSACTION, COMMIT e ROLLBACK para operações críticas
- **Documentação:** Comentar scripts DML com explicações claras
- **Integridade:** Sempre respeitar restrições e relacionamentos entre tabelas