Comandos DDL no SQL Server e Tipos de Dados

O que são comandos DDL?

Os comandos **DDL (Data Definition Language - Linguagem de Definição de Dados)** são usados para **criar, modificar e excluir** estruturas no banco de dados, como **tabelas, índices e esquemas**.

✅ Os comandos DDL **não manipulam dados diretamente**, mas afetam a estrutura do banco.
✅ Eles são **autocommit**, ou seja, qualquer alteração feita é **permanente** e não pode ser revertida com um **ROLLBACK**.

🔹 Principais comandos DDL

Comando	Descrição
CREATE	Cria novos objetos no banco de dados (tabelas, índices, visões, etc.).
ALTER	Modifica objetos já existentes.
DROP	Remove permanentemente objetos do banco de dados.

O que é e para que serve CREATE TABLE, DROP TABLE e ALTER TABLE?

🔹 CREATE TABLE

O comando CREATE TABLE é usado para **criar uma nova tabela**, definindo suas colunas, tipos de dados e restrições.

📌 Exemplo de CREATE TABLE com IDENTITY

```
CREATE TABLE Alunos (
    AlunoID INT IDENTITY(1,1),
    Nome VARCHAR(100) NOT NULL,
    DataNascimento DATE,
    Email VARCHAR(150) UNIQUE,
    CONSTRAINT PK_ALUNOS PRIMARY KEY (AlunoID)
);
```

✅ **Explicação**:

- AlunoID INT IDENTITY(1,1) → A coluna AlunoID será gerada **automaticamente**, começando em **1** e incrementando de **1 em 1**.
- Nome VARCHAR(100) NOT NULL → Nome obrigatório com até 100 caracteres.
- DataNascimento DATE → Armazena a data de nascimento do aluno.
- Email VARCHAR(150) UNIQUE → Garante que cada e-mail seja único.
- CONSTRAINT PK_ALUNOS PRIMARY KEY (AlunoID) → A coluna AlunoID é declara como chave primária da tabela Aluno.

🔹 DROP TABLE

O comando DROP TABLE remove completamente uma tabela, excluindo **todos os dados e sua estrutura**.

📌 Exemplo de DROP TABLE

```
DROP TABLE Alunos;
```

⚠ **Cuidado!** Esse comando **não pode ser revertido**.

🔹 ALTER TABLE

O comando ALTER TABLE é usado para **modificar a estrutura de uma tabela existente**, permitindo **adicionar, modificar ou excluir colunas** e **criar ou remover restrições**.

📌 Exemplo 1 - Adicionar uma coluna

ALTER TABLE Alunos ADD Telefone VARCHAR(20);

✅ Adiciona a coluna Telefone à tabela Alunos.

📌 Exemplo 2 - Modificar o tipo de uma coluna

ALTER TABLE Alunos ALTER COLUMN Nome VARCHAR(150);

✅ Altera o tamanho do campo Nome de 100 para 150 caracteres.

📌 Exemplo 3 - Remover uma coluna

ALTER TABLE Alunos DROP COLUMN Telefone;

✅ Remove a coluna Telefone da tabela Alunos.

O que é PRIMARY KEY, para que serve e como funciona?

A **chave primária (PRIMARY KEY)** é uma **restrição que garante que cada registro em uma tabela seja único**. Ela é usada para **identificar de forma exclusiva cada linha de uma tabela**.

🔹 Regras da PRIMARY KEY

1. **Valores devem ser únicos** (não pode haver dois registros com a mesma chave primária).
2. **Não aceita valores nulos (NULL)**.
3. **Cada tabela pode ter apenas uma chave primária**.
4. **Uma chave primária pode ser composta por uma ou mais colunas**.

🔹 O que é o IDENTITY e como funciona?

O **IDENTITY** é uma propriedade no SQL Server que permite a geração automática de valores numéricos sequenciais em uma coluna. Normalmente, é usado em **chaves primárias (PRIMARY KEY)** para evitar a necessidade de inserção manual do identificador.

📌 Formato do IDENTITY

```
IDENTITY (valor_inicial, incremento)
```

- valor_inicial → Número inicial da sequência.
- incremento → Valor a ser somado ao último número gerado.

Por padrão, o SQL Server começa a numeração com **1** e incrementa em **1**, ou seja, IDENTITY(1,1).

📌 Exemplo de PRIMARY KEY com IDENTITY

```
CREATE TABLE Funcionarios (

    FuncionarioID INT IDENTITY(1,1),

    Nome VARCHAR(100) NOT NULL,

    Cargo VARCHAR(50),

    Salario DECIMAL(10,2),

    CONSTRAINT PK_FUNCIONARIOS PRIMARY KEY (FuncionarioID)

);
```

✅ **Explicação**:

- FuncionarioID INT PRIMARY KEY IDENTITY(1,1) → A coluna FuncionarioID será **gerada automaticamente**, começando em 1 e aumentando de 1 em 1.

📌 Inserindo valores na tabela Funcionarios

```
INSERT INTO Funcionarios (Nome, Cargo, Salario)
VALUES ('Carlos Silva', 'Analista', 4500.00);

INSERT INTO Funcionarios (Nome, Cargo, Salario)
VALUES ('Maria Oliveira', 'Gerente', 8500.50);

INSERT INTO Funcionarios (Nome, Cargo, Salario)
VALUES ('José Santos', 'Estagiário', 2000.00);
```

📌 Tabela Funcionarios com registros preenchidos

FuncionarioID (PK)	Nome	Cargo	Salario
1	Carlos Silva	Analista	4500.00
2	Maria Oliveira	Gerente	8500.50
3	José Santos	Estagiário	2000.00

✅ Perceba que FuncionarioID foi gerado automaticamente! O valor não foi informado no comando do INSERT.

📌 Se tentarmos inserir um valor para FuncionarioID, o SQL Server retornará erro.

INSERT INTO Funcionarios (FuncionarioID, Nome, Cargo, Salario)

VALUES (10, 'Ana Costa', 'Supervisor', 5200.00);

❌ Erro! O IDENTITY impede a inserção manual de valores na coluna FuncionarioID.

O que é o comando CHECK e para que ele serve?

O CHECK é uma restrição (constraint) no SQL Server usada para definir regras sobre os valores que podem ser inseridos em uma coluna. Ele impede que valores inválidos sejam armazenados, garantindo a integridade dos dados na tabela.

Como o CHECK funciona?

O CHECK verifica se um valor atende a uma condição específica antes de ser inserido ou atualizado. Se o valor não cumprir a condição, a operação será bloqueada e o SQL Server retornará um erro.

Principais vantagens do CHECK

✔ Evita inserção de dados incorretos.
✔ Reduz a necessidade de verificações manuais no código da aplicação.
✔ Mantém a consistência dos dados na tabela.

Exemplo Prático do CHECK

Vamos supor que estamos criando uma tabela Empregados, e queremos garantir que:

- A idade do empregado seja de no mínimo 18 anos.
- O salário seja maior ou igual a R$ 1.500,00.
- O cargo seja limitado a opções pré-definidas (Analista, Gerente, Estagiário).

📌 Criando a tabela Empregados com CHECK

```
CREATE TABLE Empregados (

    EmpregadoID INT IDENTITY(1,1),

    Nome VARCHAR(100) NOT NULL,

    Idade TINYINT CONSTRAINT CK_EMPREGADOS_IDADE CHECK (Idade >= 18), -- Garante que a idade seja no mínimo 18 anos

    Salario DECIMAL(10,2) CONSTRAINT CK_EMPREGADOS_SALARIO CHECK (Salario >= 1500), -- Salário mínimo permitido

    Cargo VARCHAR(50) CONSTRAINT CK_EMPREGADOS_CARGO CHECK (Cargo IN ('Analista', 'Gerente', 'Estagiário')), -- Apenas cargos válidos

    CONSTRAINT PK_EMPREGADOS PRIMARY KEY (EmpregadoID)

);
```

Relacionamento N:M no Banco de Dados

Em um banco de dados relacional, o **relacionamento N:M (muitos para muitos)** ocorre quando múltiplos registros de uma tabela podem se relacionar com múltiplos registros de outra tabela. Diferente dos relacionamentos 1:1 e 1:N, o relacionamento **N:M não pode ser representado diretamente** sem o uso de uma **tabela intermediária**.

🔹 **Por que uma terceira tabela é necessária?**

Os bancos de dados relacionais não permitem a criação direta de um relacionamento **N:M** entre duas tabelas. Para resolver isso, criamos uma **tabela intermediária**, que geralmente contém **duas chaves estrangeiras** representando as tabelas originais.

Essa terceira tabela tem a função de:

- **Registrar cada associação entre os registros** das tabelas envolvidas.
- **Garantir que múltiplos registros possam se relacionar entre si**.
- **Facilitar consultas e manipulação de dados**.

📌 Exemplo: Relacionamento entre Filmes e Gêneros

Vamos considerar um sistema de streaming onde **um filme pode pertencer a vários gêneros** e **um gênero pode estar associado a vários filmes**.

🎬 Modelo Conceitual

- **Tabela Filme**: Contém os dados dos filmes (ex: título, duração, ano de lançamento).
- **Tabela Genero**: Contém os diferentes gêneros (ex: Ação, Drama, Comédia).
- **Tabela FilmeGenero**: Tabela intermediária que associa os filmes aos seus gêneros.

🔹 **Estrutura das tabelas**

1️⃣**Tabela Filme**

FilmeID	Título	Ano
1	O Poderoso Chefão	1972
2	Vingadores	2012
3	Toy Story	1995

2️⃣**Tabela Gênero**

GeneroID	Nome
1	Ação
2	Drama
3	Animação

3️⃣**Tabela FilmeGenero** (associando filmes a gêneros)

FilmeGeneroID	FilmeID	GeneroID
1	1	2
2	2	1
3	2	2
4	3	3

📌 **Explicação do exemplo**:

- O filme **"O Poderoso Chefão" (FilmeID 1)** pertence ao gênero **Drama (GeneroID 2)**.
- O filme **"Vingadores" (FilmeID 2)** pertence aos gêneros **Ação (GeneroID 1) e Drama (GeneroID 2)**.
- O filme **"Toy Story" (FilmeID 3)** pertence ao gênero **Animação (GeneroID 3)**.

🔹 **Benefícios do uso da terceira tabela (FilmeGenero)**:

✅ **Evita duplicação de dados**, pois os gêneros não precisam ser armazenados diretamente na tabela Filme.
✅ **Flexibilidade**, permitindo adicionar/remover associações facilmente.
✅ **Facilidade de manutenção**, pois novos gêneros ou filmes podem ser adicionados sem impactar a estrutura do banco de dados.

O que é uma Chave Estrangeira (FOREIGN KEY)?

A **chave estrangeira (FOREIGN KEY)** é uma **restrição (constraint)** no banco de dados que estabelece um **relacionamento entre duas tabelas**. Ela garante que os valores de uma coluna (ou conjunto de colunas) em uma tabela sejam **referenciados** por uma chave primária (PRIMARY KEY) em outra tabela.

Para que serve a FOREIGN KEY?

✔ **Mantém a integridade referencial**: impede que sejam inseridos valores na tabela que não existam na tabela relacionada.
✔ **Evita registros órfãos**: se um valor for referenciado em outra tabela, não pode ser excluído sem um tratamento adequado.
✔ **Facilita a navegação e junção de dados (JOINs)**.
✔ **Melhora a organização e normalização do banco de dados**.

Quando utilizar uma FOREIGN KEY?

- Sempre que for necessário estabelecer um **relacionamento entre duas tabelas**.
- Quando for preciso garantir que um valor inserido em uma tabela **já existe em outra tabela**.
- Em **relacionamentos 1:N (um para muitos)** e **N:M (muitos para muitos, através de uma tabela intermediária).**

Exemplo Prático de FOREIGN KEY

Vamos supor que temos um sistema de **cadastro de pedidos**, onde:

- A tabela Clientes armazena os clientes do sistema.
- A tabela Pedidos armazena os pedidos feitos pelos clientes.

📌 **Cada pedido pertence a um único cliente**, logo, a tabela Pedidos deve ter uma chave estrangeira (FOREIGN KEY) que referencia o ClienteID da tabela Clientes.

Criando as tabelas com FOREIGN KEY

```
CREATE TABLE Clientes (

    ClienteID INT IDENTITY(1,1),

    Nome VARCHAR(100) NOT NULL,

    Email VARCHAR(150) UNIQUE,

    CONSTRAINT PK_CLIENTES PRIMARY KEY (ClienteID)

);
```

```
CREATE TABLE Pedidos (

    PedidoID INT IDENTITY(1,1),

    ClienteID INT, -- Chave estrangeira

    DataPedido DATE NOT NULL,

    Total DECIMAL(10,2) CONSTRAINT CK_PEDIDOS_TOTAL CHECK (Total >= 0),

    CONSTRAINT PK_PEDIDOS PRIMARY KEY (PedidoID),

    CONSTRAINT FK_PEDIDOS_CLIENTES FOREIGN KEY (ClienteID) REFERENCES
Clientes(ClienteID) -- Definição da chave estrangeira

);
```

✅ **Explicação**:

- ClienteID na tabela Clientes é a **chave primária (PRIMARY KEY)**.
- ClienteID na tabela Pedidos é uma **chave estrangeira (FOREIGN KEY)** que **referencia** ClienteID na tabela Clientes.
- Isso **impede que um pedido seja associado a um cliente inexistente**.

Criando uma FOREIGN KEY em uma Tabela Existente

Se a tabela Pedidos **já foi criada sem a chave estrangeira**, podemos adicioná-la depois com ALTER TABLE.

📌 **Adicionando a FOREIGN KEY em uma tabela existente**

```
ALTER TABLE Pedidos

ADD CONSTRAINT FK_PEDIDOS_CLIENTES

FOREIGN KEY (ClienteID) REFERENCES Clientes(ClienteID);
```

✅ Agora a tabela Pedidos está corretamente relacionada à tabela Clientes.

Removendo uma FOREIGN KEY

Se for necessário remover uma FOREIGN KEY, usamos ALTER TABLE com DROP CONSTRAINT.

📌 **Removendo a restrição de chave estrangeira**

```
ALTER TABLE Pedidos

DROP CONSTRAINT FK_PEDIDOS_CLIENTES;
```

Tipos de Dados no SQL Server

Tipo de Dado	Uso Recomendado	Tamanho	Faixa de Valores / Limites	Exemplo de Uso
Numéricos Inteiros				
TINYINT	Valores pequenos	1 byte	0 a 255	Contador de tentativas, status binário
SMALLINT	Valores pequenos e médios	2 bytes	-32.768 a 32.767	Quantidade de produtos em um pedido
INT	Identificadores, contadores	4 bytes	-2.147.483.648 a 2.147.483.647	Chaves primárias (ID)
BIGINT	Números inteiros muito grandes	8 bytes	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	Registro de grandes quantidades
Numéricos Decimais e Ponto Flutuante				
DECIMAL(p,s) / NUMERIC(p,s)	Valores financeiros e precisos	5 a 17 bytes	p: 1 a 38 (total de dígitos), s: 0 a p (casas decimais). Faixa de valores: -10^{38} +1 a 10^{38} -1	Valores monetários (Preço, Salário)
FLOAT(24) (4 bytes)	Valores decimais aproximados	4 bytes	$\pm 1.18 \times 10^{-38}$ a $\pm 3.4 \times 10^{38}$	Medidas de sensores (Temperatura)
FLOAT(53) (8 bytes)	Valores decimais aproximados	8 bytes	$\pm 2.23 \times 10^{-308}$ a $\pm 1.79 \times 10^{308}$	Cálculos científicos
Texto e Caracteres				
CHAR(n)	Texto fixo	Máx. 8.000	Sempre ocupa n caracteres	Códigos padronizados (CEP, CPF)

		caracteres		
VARCHAR(n)	Texto variável	Máx. 8.000 caracteres	Usa apenas o espaço necessário	Nomes, descrições curtas
TEXT	Textos muito longos (desaconselhado)	Máx. 2GB	Recomendado apenas para grandes textos	Campos de observações extensas
Data e Hora				
DATE	Apenas datas	3 bytes	0001-01-01 a 9999-12-31	Data de nascimento
DATETIME	Data e hora combinadas	8 bytes	1753-01-01 00:00:00 a 9999-12-31 23:59:59	Registro de pedidos
TIME	Apenas horas	5 bytes	00:00:00 a 23:59:59.9999999	Horário de funcionamento
Lógicos (Booleanos)				
BIT	Sim/Não, Verdadeiro/Falso	1 bit	0 (Falso), 1 (Verdadeiro)	Status de ativação
Binários e Armazenamento de Arquivos				
BINARY(n)	Dados binários fixos	Máx. 8.000 bytes	Armazena valores binários	Hashes, assinaturas digitais
VARBINARY(n)	Dados binários variáveis	Máx. 2GB	Armazena arquivos	Imagens, PDFs, documentos

Exemplo Completo de CREATE TABLE Utilizando Diferentes Tipos de Dados

Aqui está um exemplo prático com diferentes tipos de dados para ilustrar o uso correto.

```
CREATE TABLE Produtos (
    ProdutoID INT IDENTITY(1,1),  -- Identificador único gerado automaticamente
    Nome VARCHAR(100) NOT NULL,               -- Nome do produto (até 100 caracteres)
    QuantidadeEstoque SMALLINT CONSTRAINT CK_PRODUTOS_QUANTIDADE_ESTOQUE CHECK (QuantidadeEstoque >= 0),  -- Garante que o estoque não seja negativo
    Preco DECIMAL(10,2) CONSTRAINT CK_PRODUTOS_PRECO CHECK (Preco >= 0),   -- Armazena valores financeiros com 2 casas decimais
    CodigoBarras CHAR(13) UNIQUE,             -- Código de barras fixo de 13 caracteres
    DataCadastro DATE,                         -- Armazena a data de cadastro
    UltimaVenda DATETIME,                      -- Data e hora da última venda
    HorarioAbertura TIME,                      -- Apenas horário de funcionamento
    Disponivel BIT DEFAULT 1,                  -- 1 = Disponível, 0 = Indisponível
    ImagemProduto VARBINARY(MAX),              -- Armazena imagens do produto
    Descricao TEXT,                             -- Descrição detalhada do produto

    CONSTRAINT PK_PRODUTOS PRIMARY KEY (ProdutoID)
);
```