aula_criacao_tabelas_sql

July 25, 2025

1 Aula: Introdução à Criação de Tabelas em SQL

1.1 Objetivos

- Entender a estrutura básica do comando `CREATE TABLE`
- Conhecer os principais tipos de dados em SQL
- Adotar boas práticas de nomenclatura
- Compreender o papel das tabelas no modelo relacional

1.2 1.1 – Estrutura do Comando `CREATE TABLE`

```
CREATE TABLE nome_tabela (
    nome_coluna1 tipo_dado restrição,
    nome_coluna2 tipo_dado restrição
);
```

- `CREATE TABLE`: comando para criação da tabela
- `nome_tabela`: nome da nova tabela
- `colunas`: atributos da entidade
- `tipo_dado`: tipo de dado aceito pela coluna
- `restrição`: regras como NOT NULL, PRIMARY KEY, etc.

1.3 1.2 – Tipos de Dados Mais Usados

Tipo	Descrição	Exemplo
`INT`	Número inteiro	10, 0, -50
`DECIMAL`	Número com casas decimais	12345.67
`VARCHAR(n)`	Texto com até n caracteres	'João', 'Produto X'
`DATE`	Data	'2025-07-25'
`BOOLEAN`	Verdadeiro ou falso	TRUE, FALSE

2 1.2.2

2.1 Restrições (Constraints) em SQL

As **restrições (constraints)** são regras que garantem a **integridade, validade e consistência** dos dados armazenados nas tabelas.

2.1.1 PRIMARY KEY

- Garante que os valores da coluna (ou combinação de colunas) **sejam únicos e não nulos**
- Define a **chave primária** da tabela

```
CREATE TABLE clientes (
    id INT PRIMARY KEY,
    nome VARCHAR(100)
);
```

Apenas **uma PRIMARY KEY** é permitida por tabela.

2.1.2 FOREIGN KEY

- Garante a **referência válida** entre tabelas
- Cria um vínculo entre uma coluna e a **PRIMARY KEY** de outra tabela

```
CREATE TABLE pedidos (
    id INT PRIMARY KEY,
    cliente_id INT,
    FOREIGN KEY (cliente_id) REFERENCES clientes(id)
);
```

Mantém a **integridade referencial** entre registros relacionados.

2.1.3 NOT NULL

- Impede que o campo receba valor nulo
- Ideal para campos obrigatórios

```
CREATE TABLE produtos (
    id INT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL
);
```

Útil para evitar registros incompletos.

2.1.4 UNIQUE

- Garante que **os valores de uma coluna sejam únicos**
- Pode ser aplicado em múltiplas colunas

```
CREATE TABLE usuarios (
    id INT PRIMARY KEY,
    email VARCHAR(100) UNIQUE
);
```

Pode haver **múltiplas UNIQUE constraints** numa tabela.

2.1.5 DEFAULT

- Define um **valor padrão** para a coluna, caso não seja informado no `INSERT`

```
CREATE TABLE clientes (
    id INT PRIMARY KEY,
    ativo BOOLEAN DEFAULT TRUE
);
```

Evita valores nulos e simplifica inserções.

2.1.6 CHECK

- Define uma **regra lógica** que o valor precisa obedecer

```
CREATE TABLE funcionarios (
    id INT PRIMARY KEY,
    salario DECIMAL(10,2),
    CHECK (salario >= 1320)
);
```

Rejeita inserções ou atualizações que **violam a condição**.

2.1.7 COMBINANDO RESTRIÇÕES

```
CREATE TABLE alunos (
    id INT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    matricula VARCHAR(20) UNIQUE NOT NULL,
    idade INT CHECK (idade >= 16),
    ativo BOOLEAN DEFAULT TRUE
);
```

2.2 Observações Importantes

- As constraints ajudam a **automatizar validações no próprio banco**
- Melhor prevenir erro com uma constraint do que confiar só na aplicação
- Podem ser **criadas ou removidas** após a criação da tabela com `ALTER TABLE`

2.3 1.3 – Boas Práticas de Nomenclatura

- Use `snake_case`: `nome_cliente`, `data_pedido`
- Nomes sempre no singular: `cliente`, `produto`
- Evite espaços, acentos e caracteres especiais
- Evite prefixos redundantes como `tbl_` ou `col_`

```sql
-- Ruim:
CREATE TABLE Tabela1 (Nome Cliente TEXT);
-- Bom:
CREATE TABLE cliente (nome_cliente VARCHAR(100));
```

2.4 1.4 – Pensando como Modelador de Dados

Antes de criar uma tabela, pense: - Qual entidade será representada? - Quais atributos são obrigatórios? - Há algum campo que deve ser único? - Existe relação com outra tabela? **Exemplo de modelagem de clientes:** - `id` (chave primária) - `nome` (obrigatório) - `email` (único) - `nascimento` (data) - `ativo` (booleano)

2.5 1.5 – Papel das Tabelas no Modelo Relacional

- Tabelas são representações de entidades
- Cada linha representa um registro (tupla)
- Cada coluna é um atributo
- Devem conter uma chave primária (`PRIMARY KEY`)
- Permitem relacionamentos com outras tabelas por meio de chaves estrangeiras (`FOREIGN KEY`)

2.6 1.6 – Segurança com `IF NOT EXISTS`

```sql
CREATE TABLE IF NOT EXISTS clientes (
    id INT PRIMARY KEY,
    nome VARCHAR(100)
);
```

- Evita erro se a tabela já existir
- Boa prática para scripts versionados

2.7 1.7 – Discussão

- Quando usar `NOT NULL`?
- `CPF` deveria ser `VARCHAR` ou `INT`?
- Toda tabela deve ter uma `PRIMARY KEY`?

2.8 Respostas

2.8.1 Quando usar `NOT NULL`?

Resposta: Use `NOT NULL` quando o campo **é obrigatório** para o sistema funcionar corretamente. Exemplo:

- Nome do cliente: não faz sentido ter um cliente sem nome.
- Data de cadastro: é necessária para auditoria ou controle.
- Senha: essencial para login.

  `NOT NULL` **garante integridade** e evita dados "incompletos" que podem causar erro nos sistemas.

2.8.2 `CPF` deve ser `VARCHAR` ou `INT`?

Resposta curta: `VARCHAR`. Por quê?

- CPF **pode começar com zero**, e inteiros não armazenam zeros à esquerda.
- Pode conter **pontuação** se você quiser exibir formatado (`123.456.789-00`).
- Não é uma informação numérica que será usada em somas ou médias.

  Apesar de ser composto por números, **CPF é um identificador, não um número matemático**.

2.8.3 Toda tabela deve ter uma `PRIMARY KEY`?

Resposta: Sim, **toda tabela deveria ter uma chave primária** — é uma **boa prática fundamental** em bancos relacionais.

- Garante **unicidade** e **identificação clara** de cada linha
- É usada como referência em **chaves estrangeiras**
- Facilita a integridade referencial e a criação de índices

  Tabelas sem chave primária correm risco de ter dados duplicados, inconsistentes e difíceis de manipular.

2.9 Recapitulando

- `CREATE TABLE` estrutura os dados no banco
- Tipos de dados garantem consistência
- Nomear com clareza é essencial
- Integridade dos dados começa na definição das tabelas