

---

# SQL - Structured Query Language

## Unidade 2 – Manipulação Básica de Dados – Parte 3

Prof. Daniel Callegari  
Escola Politécnica – PUCRS

---

Para manter bancos de dados com qualidade, é necessário garantir a integridade dos dados armazenados. Nesta parte, vamos aprender sobre Integridade de Entidade, Integridade de Domínio e Integridade Referencial.

### 1. Integridade de Entidade

A integridade de entidade define uma linha (registro) como entidade exclusiva de uma determinada tabela.

Por exemplo, não podemos permitir que exista mais de uma pessoa com o mesmo CPF ou mais de um veículo com uma mesma placa. Para isso, devemos escolher, para cada tabela, o conjunto mínimo de colunas (atributos) que unicamente identifica cada registro. Na tabela PESSOAS, a coluna 'cpf' deve ter valores únicos. Na tabela VEICULOS, a coluna 'placa' deve ter valores únicos. Em alguns casos será necessário usar duas ou mais colunas para garantir unicidade (isso será visto mais adiante). Chamamos esses conjuntos de atributos de "chave".

Às vezes uma tabela pode apresentar mais de uma alternativa para essa escolha. Considere, por exemplo, uma tabela de ALUNOS contendo os campos 'nroMatricula' e 'cpf'. Nesse caso os dois campos, individualmente, devem ser únicos. Nos bancos de dados, chamamos cada um desses casos de chaves candidatas.

Depois que definimos todas as chaves candidatas para uma determinada tabela, escolhemos apenas uma delas como sendo a principal. Por exemplo, se uma tabela apresenta três chaves candidatas, escolhemos uma delas como sendo a principal (arbitrariamente, já que qualquer uma delas serviria para esse propósito).

Chamamos a chave principal de chave primária (ou, em inglês, PRIMARY KEY - PK). As outras chaves candidatas, então, se tornam chaves alternativas (ou, em inglês, ALTERNATE KEYS - AKs).


Em SQL usamos as restrições PRIMARY KEY (para a principal) e UNIQUE (para as demais AKs). Naturalmente, toda PK é também NOT NULL (por quê?). Por exemplo:

```
CREATE TABLE ALUNOS
(
    nroMatricula  VARCHAR(10)  PRIMARY KEY,
    cpf           VARCHAR(20)  UNIQUE,
    email         VARCHAR(100) UNIQUE,
    nome          VARCHAR(150) NOT NULL,
    anoIngresso   NUMBER(4)    NOT NULL,
    endereco      VARCHAR(150) NULL,
    sexo          CHAR(1)      NOT NULL
);
```

Essas declarações de restrições PRIMARY KEY e UNIQUE estão na forma *inline*, ou seja, são especificadas junto com a criação de cada campo no comando CREATE TABLE.

A forma alternativa que usa a palavra CONSTRAINT (“restrição”) a seguir é preferível por ser mais flexível, como veremos mais adiante:

```
CREATE TABLE ALUNOS
(
    nroMatricula  VARCHAR(10)  NOT NULL,
    cpf           VARCHAR(20)  NOT NULL,
    email         VARCHAR(100) NULL,
    nome          VARCHAR(150) NOT NULL,
    anoIngresso   NUMBER(4)    NOT NULL,
    endereco      VARCHAR(150) NULL,
    sexo          CHAR(1)      NOT NULL,
    CONSTRAINT PK_ALUNOS PRIMARY KEY (nroMatricula),
    CONSTRAINT AK1_ALUNOS UNIQUE (cpf),
    CONSTRAINT AK2_ALUNOS UNIQUE (email)
);
```

 Experimente agora tentar inserir registros diferentes, mas com o mesmo número de matrícula ou com um mesmo número de CPF para ver como o SGBD impede a operação e relata um erro de duplicidade de chave.

## 2. Integridade de Domínio

A integridade de domínio visa garantir que os dados armazenados respeitem determinados valores permitidos. Podemos restringir o intervalo de dados permitido para um campo. Alguns exemplos de restrição de domínio são:


- garantir que o preço de um produto não pode ser zero ou ter um valor negativo;
- garantir que o campo *status* de um pedido tenha somente um dos seguintes valores: 'ABERTO', 'PENDENTE', 'FECHADO'.
- garantir que o campo *genero* somente aceite os valores 'M' ou 'F'.

Esses casos podem ser garantidos usando a restrição CHECK. Por exemplo, vamos garantir que o campo 'anoIngresso' possua sempre um valor superior a 2000 e que o campo sexo permita apenas os valores 'M' ou 'F' (em maiúsculas):

```
ALTER TABLE ALUNOS
ADD CONSTRAINT CK_AnoIngr CHECK (anoIngresso > 2000);

ALTER TABLE ALUNOS
ADD CONSTRAINT CK_sexo CHECK (sexo IN ('M', 'F'));
```

Observação: se já houver algum registro na tabela que não atenda à restrição CHECK que estamos tentando adicionar, o SGBD retorna um erro (por quê?).

 Que restrições de integridade de domínio poderiam ser aplicadas às tabelas VEICULOS e PESSOAS?

### 3. Integridade Referencial

A integridade referencial é usada entre duas tabelas para garantir que os dados de uma coluna da primeira tabela se referem aos dados registrados em uma coluna da segunda tabela.


Por exemplo, suponha uma tabela que registra os estados (unidades federativas) do país:

```
CREATE TABLE ESTADOS
(
    uf      CHAR(2)      NOT NULL,
    nome    VARCHAR(40)  NOT NULL,
    regioao CHAR(2)      NOT NULL,
    CONSTRAINT PK_ESTADOS PRIMARY KEY (uf)
);
```

Suponha agora outra tabela que registra as cidades, vinculando-as aos estados:

```
CREATE TABLE CIDADES
(
    cod_cidade NUMBER(4)      NOT NULL,
    nome       VARCHAR(60)  NOT NULL,
    uf         CHAR(2)      NOT NULL,
    CONSTRAINT PK_CIDADES PRIMARY KEY (cod_cidade)
);
```

Como não há verificação de consistência (integridade) entre os dados das duas tabelas, seria perfeitamente possível cadastrar uma cidade especificando sua 'uf' como 'XX', ou ainda especificando sua 'uf' com uma sigla de estado ainda não cadastrada na tabela de estados.

 Experimente inserir dados nas duas tabelas forçando esse tipo de inconsistência. Depois, elimine os registros inconsistentes para podermos prosseguir com o exemplo.

Então, como podemos garantir que, ao incluir uma nova cidade, o seu campo 'uf' se refere a um estado que realmente existe (ou seja, já tenha sido anteriormente cadastrado)?


A resposta é: através de uma chave estrangeira (em inglês, FOREIGN KEY – FK)


O comando a seguir cria um vínculo entre as duas tabelas, definindo uma regra de integridade referencial do campo 'uf' da tabela de cidades para o campo 'uf' da tabela de estados (ou seja, para a chave primária da tabela 'pai'):


```
ALTER TABLE CIDADES
ADD
(
    CONSTRAINT FK_EST_CID
    FOREIGN KEY (uf)
    REFERENCES ESTADOS (uf)
);
```

Observação: se houver algum registro que não atenda à restrição que estamos tentando adicionar, o SGBD retorna um erro (por quê?).

Observação: o nome "FK\_EST\_CID" é um identificador arbitrário escolhido pelo projetista do banco de dados (assim como escolhemos nomes de variáveis em programas). É uma boa prática definir um padrão para esses identificadores. O mesmo raciocínio ocorre com os demais identificadores nas seções anteriores: AK..., CK..., PK...

 Experimente agora inserir dados nas duas tabelas depois de enviar o comando ALTER TABLE para o banco de dados.

 Tente também forçar uma inconsistência de dados, como por exemplo, inserir uma cidade para um estado que ainda não esteja cadastrado.

 Que tipo(s) de verificação de integridade seria(m) necessária(s) para (e entre) as tabelas PESSOAS e VEICULOS dos exemplos anteriores?

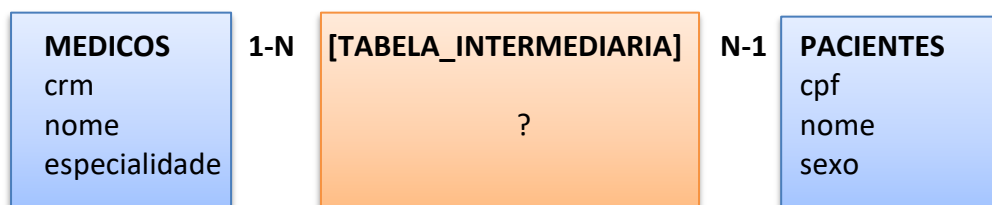
## 4. Relacionamentos do tipo muitos-para-muitos

Dizemos que o relacionamento existente entre as tabelas ESTADOS e CIDADES é do tipo um-para-muitos (ou 1-N). Em outras palavras, estamos dizendo que cada estado pode ter várias cidades, mas que cada cidade pertence a apenas um estado. O mesmo tipo de relacionamento ocorre entre as tabelas de PESSOAS e VEICULOS.

✎ Quais seriam as alterações nas tabelas PESSOAS e VEICULOS para estabelecer tal relacionamento?

No entanto, em alguns casos precisamos representar relacionamentos do tipo muitos-para-muitos (ou N-N). Esse é o caso dos relacionamentos entre MEDICOS e PACIENTES ou então entre PESSOAS e PROJETOS, por exemplo.

Bem, não há como representar relacionamentos N-N entre (usando apenas) duas tabelas em um banco de dados relacional. No entanto, podemos resolver esse problema criando uma terceira tabela (ou seja, uma tabela intermediária entre as duas tabelas originais) e criando dois relacionamentos 1-N entre as tabelas:



## 5. Prática

✎ Crie uma tabela MEDICOS com os campos 'crm', 'nome' e 'especialidade'.

✎ Crie uma tabela PACIENTES com os campos 'cpf', 'nome' e 'sexo'.

✎ Crie uma tabela chamada MEDICOSPACIENTES com os campos 'crm' e 'cpf'. Esta tabela servirá para saber quais médicos tratam de quais pacientes (e, por consequência, quais pacientes são atendidos por quais médicos).

✎ Defina uma FK entre MEDICOSPACIENTES.crm e MEDICOS.crm.

✎ Defina outra FK entre MEDICOSPACIENTES.cpf e PACIENTES.cpf.

✎ Qual será a chave primária da tabela MEDICOSPACIENTES?

✎ Desafio: E se quiséssemos armazenar o histórico de consultas entre médicos e pacientes? Crie uma tabela de consultas, que contenha a data de cada consulta.

### Dicas finais desta aula

- Lembre-se de que toda chave primária (PK) é implicitamente não nula.
- Um campo de chave estrangeira (FK), no entanto, pode ser nulo. Ou seja, nem todo registro de uma tabela filha deve necessariamente estar vinculado a um registro da tabela pai.
- É possível especificar as restrições de chaves estrangeiras (FKs) dentro do próprio comando de criação de uma tabela (CREATE TABLE).
- Para relacionamentos do tipo N-N precisamos criar uma tabela intermediária entre as duas tabelas originais.

### Fechamento

Parabéns! Você aprendeu alguns conceitos fundamentais sobre integridade:

1. Há maneiras diferentes de garantir diferentes tipos de integridade entre os dados;
2. Nesta parte, você aprendeu sobre Integridade de Entidade, Integridade de Domínio e Integridade Referencial;
3. Um banco de dados bem estruturado não irá permitir inconsistências entre os dados, o que aumenta a qualidade da informação armazenada.

-X-