Valores nulos, chaves primárias

Prof. Dr. Nazareno de Oliveira Pacheco
nazareno.pacheco@prof.sc.senac.br

Exemplo

- Imagine a seguinte tabela:

```
CREATE TABLE curriculo (
    nomeCandidato VARCHAR(100),
    descricaoHabilidades TEXT,
    pretensaoSalarial NUMERIC(8,2),
    PRIMARY KEY (codigo)
);
```

Exemplo

- É necessário cadastrar dois currículos:
  - Hermenegildo Norton de Bragança sabe programar em Java, JSP, SQL, C, C#, C++, Delphi, PHP, .Net e HTML. Sua pretensão salarial é R$10000
  - Asdroaldo Jenkins de Souza sabe Java, JSP e conhece um pouco de SQL. Não informou pretensão salarial
- Como efetuar o cadastro destes currículos na tabela?

Experimente!

```
INSERT INTO CURRICULO (nomeCandidato, descricaoHabilidades,
 pretensaoSalarial)
VALUES ('Hermenegildo Norton de Bragança',
        'Sabe programar em Java, JSP, SQL, C, C#, C++, Delphi,
 PHP, .Net e HTML',
        10000);

INSERT INTO CURRICULO (nomeCandidato, descricaoHabilidades,
 pretensaoSalarial)
VALUES ('Asdroaldo Jenkins de Souza',
        'Sabe Java, JSP e conhece um pouco de SQL',
        ????);
--ele não informou pretensao!!! Como fazer?
```

Valores nulos

- A solução ideal neste caso é utilizar o valor null
- null representa um valor desconhecido
- O que null não é:
  - null não é 0
  - null não é '' (Texto vazio)
- Quando queremos manter colunas de forma não preenchida, podemos utilizar este valor

Experimente!

```
INSERT INTO CURRICULO (nomeCandidato, descricaoHabilidades,
 pretensaoSalarial)
VALUES ('Asdroaldo Jenkins de Souza',
        'Sabe Java, JSP e conhece um pouco de SQL',
        null);
```

Colunas nulas e não nulas

- Por padrão, qualquer coluna aceita valores nulos
- Porém, existem casos em que não queremos aceitar valores nulos
  - Por exemplo: podemos aceitar candidatos sem pretensão salarial, **mas não podemos aceitar candidatos sem nome!**
- Existe uma forma de impedir que uma coluna aceite valores nulos durante a criação de uma tabela
- É possível associar o atributo NOT NULL a uma determinadas colunas

Exemplo

```
CREATE TABLE curriculo (
    nomeCandidato         VARCHAR(100) NOT NULL,
    descricaoHabilidades  TEXT,
    pretensaoSalarial     NUMERIC(8,2),
    PRIMARY KEY (codigo)
);
```

Identificando registros

- Até o momento, nada nos impedia de inserir dois registros idênticos em uma tabela
- Isso pode ser problemático
- O ideal é identificar cada registro de forma única

Chaves Primárias

- No momento em que uma tabela é criada, é possível definir uma Chave Primária (PRIMARY KEY, ou PK)
  - Uma chave primária é uma restrição (CONSTRAINT) definida sobre uma ou mais colunas de uma tabela
  - Esta restrição impede que sejam adicionados dois registros em uma tabela cujo os valores da PK sejam idênticos
- **Toda tabela deve ter uma chave primária!**

Experimente!

```
 --rode um comando de cada vez
CREATE TABLE usuario (
   codigo      INTEGER,
   nome        VARCHAR(100),
   PRIMARY KEY (codigo)
);

INSERT INTO usuario (codigo, nome)
VALUES (1, 'Charles'); --OK

INSERT INTO usuario (codigo, nome)
VALUES (2, 'Leonardo'); --OK

INSERT INTO usuario (codigo, nome)
VALUES (1, 'Alessandra'); --ERRO! PK duplicada!
```

Identificando registros

- Uma chave primária serve para identificar um registro de uma tabela de forma única
- No exemplo anterior, se eu quiser solicitar o usuário com código = 1, posso ter certeza absoluta que só existe um usuário com este código
- **Chaves primárias não podem ser nulas!**

Escolhendo uma chave primária

- A escolha da coluna (ou colunas) que forma a chave primária deve ser feita com cuidado
- Uma recomendação comum é que a chave primária de uma tabela seja sempre composta por colunas numéricas
  - **Isto facilita a comparação de informações no banco, tornando as consultas mais rápidas**

Escolhendo uma chave primária

- Alguns enganos comuns são:
  - Utilizar CPF: esta não é uma boa escolha de chave primária pelas seguintes razões:
    - Nem todas as pessoas gostam de informar o CPF
  - Utilizar nomes: além de não serem informações numéricas, é muito comum registros com nomes repetidos (Ex. João da Silva)

Escolhendo uma chave primária

- Uma das práticas mais recomendadas é a criação de um campo próprio em sua tabela que sirva como chave primária (p. ex.: codigo)
  - **Isto evita que a gente esbarre com limitações que outros valores podem apresentar**

Auto incremento

- Um problema com a utilização de chaves primárias próprias é que nós mesmos temos que garantir que o banco de dados utilize um valor diferente para cada registro a ser inserido
  - No primeiro INSERT, precisamos usar um valor
  - No próximo INSERT, precisamos usar outro valor diferente

Auto incremento

- Uma forma simples de resolver esta questão é utilizar valores sequenciais:
  - O primeiro registro tem codigo = 1,
  - O segundo registro tem codigo = 2, ...
- A maior parte dos bancos de dados possuem uma funcionalidade que facilita a criação desta sequência
- Esta funcionalidade é comumente chamada de **auto incremento**

Auto incremento

- O auto incremento é implementado de forma diferente em cada SGBD (não existe um padrão)
- No MySQL, ele é implementado utilizando os tipos ***serial*** e ***bigserial***
  - O tipos *serial* e *bigserial* são na verdade “pseudo-tipos”. Internamente, são equivalentes ao *integer* e *bigint*.
  - Ao criar uma coluna com este tipo, o MySQL automaticamente cria uma “sequência”, que é associada a este campo

Experimente!

```
DROP TABLE usuario;
--apaga a tabela gerada anteriormente

CREATE TABLE usuario (
   codigo    SERIAL,
   nome      VARCHAR(100),
   PRIMARY KEY (codigo)
);
```

Inserindo registros em tabelas com auto incremento

- Existem duas formas de inserir registro em uma tabela com um campo com auto incremento

- ```
  INSERT INTO usuario (codigo, nome)
  VALUES (DEFAULT, 'Charles');
  ```
  - Utilizamos a palavra default para deixar que o banco de dados utilize o valor padrão para esta coluna (que é o valor sequencial que o banco vai gerar)

- ```
  INSERT INTO usuario (nome)
  VALUES ('Charles')
  ```
  - O nome e o valor da coluna com auto incremento são omitidos intencionalmente. Isto vai fazer com que o valor padrão (a sequência) seja utilizado.

Experimente!

```
INSERT INTO usuario (nome)
VALUES ('Charles');

INSERT INTO usuario (codigo, nome)
VALUES (DEFAULT, 'Leonardo');

INSERT INTO usuario (nome)
VALUES ('Alessandra');
```

E se...

- E se eu informar explicitamente o valor de uma coluna com autoincremento?
  - Se a chave primária da tabela não for violada, não ocorrerá nenhum erro
  - Porém, o banco de dados não vai incrementar a sequência, o que pode causar erros mais tarde
  - Portanto, não faça isso!
  - Uma vez que uma coluna foi criada com auto incremento, utilize sempre o valor default

Experimente!

```
--rode um comando por vez
DROP TABLE usuario;

CREATE TABLE usuario (
   codigo    SERIAL,
   nome      VARCHAR(100),
   PRIMARY KEY (codigo)
);

INSERT INTO usuario (nome)
VALUES ('Charles'); --codigo = 1

INSERT INTO usuario (codigo, nome)
VALUES (DEFAULT, 'Leonardo'); --codigo = 2
```

Experimente!

```
INSERT INTO usuario (codigo, nome)
VALUES (3, 'Alessandra');
--valor explicito, a sequencia nao foi incrementada!

INSERT INTO usuario (codigo, nome)
VALUES (DEFAULT, 'Aline');
--ERRO, mas mesmo assim a sequencia é incrementada

INSERT INTO usuario (codigo, nome)
VALUES (DEFAULT, 'Aline');
--OK

SELECT * FROM usuario;
```

Exercícios (9)

- Refaça os exercícios anteriores criando chaves primárias para cada uma das tabelas