# DML – Comandos para manipulação de dados

Prof. Bruno Augusto Nassif Travençolo

# SQL

- Linguagem estruturada de consulta
- Duas classes importantes de comandos
  - DDL: Data definition language
    - Comandos para a definição de dados
  - DML: Data manipulation language
    - Comandos para a manipulação de dados
      - Exemplos: INSERT,  DELETE, UPDATE,SELECT

# SQL INSERT INTO

- Tabela Exemplo

```
CREATE TABLE produtos (
    cod_prod   integer,
    nome       text,
    preco      numeric
);
```

- Inserindo uma tupla (linha)

```
INSERT INTO produtos VALUES (1, 'Queijo', 9.99);
```

  - Dados colocados na ordem de criação das colunas da tabela
  - É necessário, neste caso, conhecer a ordem das colunas
    - Fique atento pois esta ordem pode ser alterada!

- Fonte: http://pgdocptbr.sourceforge.net/pg80/dml.html

# SQL INSERT INTO

```
CREATE TABLE produtos (
    cod_prod   integer,
    nome       text,
    preco      numeric);
```

- Inserindo uma tupla (linha)

```
INSERT INTO produtos (cod_prod, nome, preco) VALUES (1, 'Queijo', 9.99);
INSERT INTO produtos (nome, preco, cod_prod) VALUES ('Queijo', 9.99, 1);
```

  - Neste caso, indica-se as colunas
  - Essa sintaxe é fortemente recomendada
    - A ordem das colunas de uma tabela pode ser alterada em uma eventual manutenção no BD.

- Pode-se omitir uma coluna

```
INSERT INTO produtos (cod_prod, nome) VALUES (1, 'Queijo');
```

  - As colunas não indicadas são preenchidas com o seus respectivos valores DEFAULT

- Fonte: http://pgdocptbr.sourceforge.net/pg80/dml.html

# SQL INSERT INTO

```
CREATE TABLE produtos (
    cod_prod   integer,
    nome       text,
    preco      numeric);
```

- Especificando um valor DEFAULT

```
INSERT INTO produtos (cod_prod, nome, preco) VALUES (1, 'Queijo', DEFAULT);
```

- Inserindo toda a linha com valores DEFAULT

```
INSERT INTO produtos DEFAULT VALUES;
```

- Especificando mais de uma linha

```
INSERT INTO produtos (cod_prod, nome, preco) VALUES (1, 'Queijo', DEFAULT),
                                                    (2, 'Goiabada', 10.00),
```

- Fonte: http://pgdocptbr.sourceforge.net/pg80/dml.html

# Inserindo dados no banco

- Comando SQL: INSERT

```
INSERT INTO table [ ( column [, ...] ) ]
    { DEFAULT VALUES | VALUES ( { expression | DEFAULT } [, ...] ) [, ...]
     | query }
    [ RETURNING * | output_expression [ [ AS ] output_name ] [, ...] ]
```

http://www.postgresql.org/docs/8.4/static/sql-insert.html

```
CREATE TABLE produtos (
    cod_prod   integer,
    nome       text,
    preco      numeric);
```

# Modificação de dados

- A operação de modificação/atualização de dados que já estão no banco de dados é conhecida como UPDATE
- Para atualizar alguma tupla é necessário informar
  - O nome da *tabela* e da *coluna* a ser atualizada
  - O novo valor para a *coluna*
  - Quais tuplas (*linhas*) serão atualizadas
- Exemplo

```
UPDATE produtos SET preco = 10 WHERE preco = 5;
```

  - Atualiza a coluna preco da tabela produtos, modificando o preço para 10 das linhas que possuem preço igual a 5

# Modificação de dados

```
CREATE TABLE produtos (
    cod_prod   integer,
    nome       text,
    preco      numeric);
```

- Exemplo

```
UPDATE produtos SET preco = preco*1.10
```

  - Atualiza a coluna preco da tabela produtos, aumentando o preço em 10% para *todos* os produtos

- Mais de uma coluna pode ser atualizada

```
UPDATE minha_tabela SET a = 5, b = 3, c = 1 WHERE a > 0;
```

# SQL UPDATE

- Comando SQL: UPDATE

```
UPDATE [ ONLY ] table [ [ AS ] alias ]
    SET { column = { expression | DEFAULT } |
          ( column [, ...] ) = ( { expression | DEFAULT } [, ...] ) } [, ...]
    [ FROM fromlist ]
    [ WHERE condition | WHERE CURRENT OF cursor_name ]
    [ RETURNING * | output_expression [ [ AS ] output_name ] [, ...] ]
```

- CUIDADO: Sempre teste a clausura WHERE de um comando UPDATE antes de executá-lo. Um simples erro no WHERE pode resultar na alteração de muitas, senão todas, linhas da tabela, atribuindo a elas o mesmo valor

# Remoção de dados

- Para deletar dados do BD usa-se o comando DELETE
- Exemplo

```
DELETE FROM produtos WHERE preco = 10;
```

  - Todas as tuplas cujos produtos custam 10 serão eliminadas

- Removendo todas as tuplas da tabela produtos

```
DELETE FROM produtos
```

- CUIDADO: é muito fácil apagar os dados das tabelas

# SQL DELETE

- Comando SQL: DELETE

```
DELETE FROM [ ONLY ] table [ [ AS ] alias ]
    [ USING usinglist ]
    [ WHERE condition | WHERE CURRENT OF cursor_name ]
    [ RETURNING * | output_expression [ [ AS ] output_name ] [, ...] ]
```

# Observações
# Inserindo campos de data

```
DROP TABLE IF EXISTS d;
CREATE TABLE d (
 d DATE);

INSERT INTO d(d) VALUES ('11/12/2010'); -- ok funciona, mas qual o mês? 11 ou 12?

-- vamos tentar inserir 30 de dez de 2010
INSERT INTO d(d) VALUES ('30/12/2010'); -- ok!, no meu computador funcionou. Pode ser que no seu não funcione este comando, mas funcione o próximo

-- agora vou tentar inserir a mesma data, só que em outro formato (mês, dia, ano)
INSERT INTO d(d) VALUES ('12/30/2010'); -- ERRO:  valor do campo date/time está fora do intervalo: "12/30/2010“
```

# Observações
# Inserindo campos de data

```
-- descobrindo o formato atual configurado no postgresql
-- ver aquivo postgresql.conf. O meu é datestyle = 'iso, dmy'
-- formato ISO: YYYY-MM-DD (ano,mês,dia)
-- formato dmy: DD/MM/YYYYY
SHOW datestyle;

-- mudando a configuração do sistema para aceitar a data no formato anterior
SET datestyle TO 'mdy';
Obs: para o povoamento disponível no moodle o formato deve ser  'mdy';

-- conferindo a mudança
SHOW datestyle;

-- tentando inserir a data que deu erro anteriormente
INSERT INTO d(d) VALUES ('12/30/2010'); -- agora está ok!
```

# Observações
# Inserindo campos de data

```
-- É possível inserir dados independentemente do tipo configurado no
-- postgresql. Para isso use o comando to_date

-- inserindo o dia 30 de dezembro em dois formatos diferentes
INSERT INTO d(d) VALUES (to_date('12/30/2010','MM/DD/YYYY'));
INSERT INTO d(d) VALUES (to_date('30/12/2010','DD/MM/YYYY'));
```

- Comando EXTRACT
  - Para obter somente o ano de um campo de data

```
SELECT extract(year from datanasc)
FROM empregado
```

# Lidando com o desconhecido: NULLs

- Quando não sabemos qual o valor preencher em um determinado atributo, podemos utilizar um valor especial chamado NULL
  - NULL significa que o valor é desconhecido no momento
    - Ex: hora da chegada de um voo que ainda está no ar
  - Ou pode significar algo não relevante para uma determinada tupla
    - Ex:  Número do apartamento no cadastro do endereço. Muitas pessoas não moram em prédios.
- NULL não quer dizer 0 (zero)
- NULL não quer dizer ‘’ (string vazia)

# NULL

- Qual o resultado da comparação de dois valores NULL?
  - É indeterminado,  ou seja NULL

    **SELECT NULL** = **NULL**
- Algumas operações com NULL também são indefinidas
  - **SELECT** 2 + **NULL**
- O SQL provê uma maneira para fazer testes com NULL, por meio do teste IS NULL e IS NOT NULL
  - Ex: buscar no banco quais as pessoas que não possuem nome do meio.
  - FORMA ERRADA:

**SELECT** *

**FROM** empregado

**WHERE** minicial = **NULL**

  - (consulta retorna vazio)

# NULL

- FORMA CORRETA:

**SELECT** *

**FROM** empregado

**WHERE** minicial **IS NULL**

- Obtendo a lista de pessoas que possuem o nome do meio
- FORMA CORRETA:

**SELECT** *

**FROM** empregado

**WHERE** minicial **IS NOT NULL**

# Outros tópicos

- Nas práticas e outras aulas veremos alguns desses tópicos
  - Utilizando o comando \copy
  - Inserindo dados a partir de outras tabelas
  - Atualizando dados a partir de outras tabelas
  - Utilizando o comando TRUNCATE (não é padrão SQL)