

# PostgreSQL Essentials



## Objetivos do Curso

### \* Alteração de dados

Adição de dados com INSERT

Inserção de dados através do SELECT

Inserção de dados com COPY

Recuperação de dados com COPY

COPY no formato CSV

Modificação de dados com UPDATE

Remoção de dados com DELETE

Remoção de dados com TRUNCATE

### \* Herança entre tabelas

Conceito de herança em banco de dados

Aplicação no PostgreSQL

Cláusula INHERITS

Cláusula ONLY

## Notação Utilizada no Curso

***Itálico*** : Utilizamos para nomes de objetos de banco de dados, arquivos e outros elementos que devem ser fornecidos pelo usuário.

**Courier New** : Os comandos, palavras-chave e saídas de comandos são grafados no tipo Courier.

**Courier New** : Utilizamos Courier New em negrito para enfatizar uma palavra-chave apresentada pela primeira vez e valores padrão.

**[ ]** : Utilizamos colchetes sempre que uma palavra-chave for opcional.

**{ }** : Utilizamos chaves para delimitar uma lista de itens onde um deles deva ser escolhido.

**...** : Utilizamos três pontos para indicar que aquele item pode se repetir. Quando utilizado nos exemplos, indica uma parte não importante da informação foi removida.

# CAPITULO 1

## Adicionando dados com INSERT

Tabelas Exemplo.

```
CREATE TABLE teste (  
  a INT,  
  b INT,  
  c TEXT,  
  d TEXT,  
  e TIMESTAMP WITH TIME ZONE  
);
```

Sintaxe:

\h INSERT INTO

## Exemplos de INSERT

Inclusão de registro incompleto:

```
INSERT INTO a (a,b,c)  
VALUES (1,1,'Ola');
```

Adicionando dados com o SELECT e retornando os registros incluídos:

```
INSERT INTO a (a,b,c,d,e)  
SELECT 1,2,'Meu nome','Meu Cargo',timenow() ;
```

Note que podemos também utilizar a cláusula NULL para atribuir valores nulos a determinados campos.

A inclusão incompleta, ou inclusão de nulos, somente é permitida para campos que não tenham sido criados com a cláusula NOT NULL.

Exemplo:

```
INSERT INTO a (a,b,c,d,e)  
VALUES (1,1,'Ola', NULL, NULL);
```

Quando uma coluna tem um valor default e um outro valor é passado, este valor sobrepõe o valor default.

## **Adicionando dados com SELECT**

Os registros adicionados consistirão no resultado de um SELECT.

Sintaxe:

```
INSERT INTO tabela SELECT ....
```

Como exemplo, vamos criar o resultado do concurso 2 a partir da aposta 3:

```
INSERT INTO a  
SELECT 1,2,'Meu nome','Meu Cargo',timenow() ;
```

## **Inserindo Dados com o COPY**

O COPY é mais eficiente que o INSERT ao inserir grandes quantidades de registros. Isto se dá porque o COPY insere os dados numa única transação.

Uma maneira prática e eficiente de inserir dados em tabelas de arquivos externos é utilizar o comando COPY.

Sintaxe:

```
COPY tablename [ ( column [, ...] ) ]  
FROM { 'filename' | STDIN }  
[ [ WITH ]  
[ BINARY ]  
[ OIDS ]  
[ DELIMITER [ AS ] 'delimiter' ]  
[ NULL [ AS ] 'null string' ] ]
```

O separador padrão de campos é o TAB.

Caso haja algum erro numa das linhas, toda a operação de inserção será abortada.

*O COPY é mais eficiente que o INSERT ao inserir grandes quantidades de registros. Isto se dá porque o COPY insere os dados numa única transação.*

## **Inserindo Dados com o COPY**

Exemplo:

```
COPY aposta FROM '/tmp/EXEMPLO1.txt';
```

O arquivo apostas.txt:

```
1 4 2006-06-27 11:00 1
2 4 2006-06-27 11:10 1
3 3 2006-06-27 11:15 1
4 6 2006-06-27 12:30 1
5 2 2006-06-27 13:01 1
```

Podemos mudar, no arquivo apostas.txt, o delimitador de campos (o padrão é o TAB).

Arquivo apostas.txt, com ponto-e-vírgula:

```
1;4;2006-06-27 11:00;1
2;4;2006-06-27 11:10;1
3;3;2006-06-27 11:15;1
4;6;2006-06-27 12:30;1
5;2;2006-06-27 13:01;1
```

Mudando o delimitador no comando SQL:

```
COPY aposta FROM '/tmp/EXEMPLO2.txt' WITH DELIMITER ';' ;
```

## **Exportando Dados com o COPY**

Também é possível utilizar o COPY de uma maneira simétrica para extrair dados de tabelas e enviar para um arquivo.

Sintaxe:

```
COPY { tablename [ ( column [, ...] ) ] | ( query ) }
TO { 'filename' | STDOUT }
[ [ WITH ]
[ BINARY ]
[ HEADER ]
[ OIDS ]
[ DELIMITER [ AS ] 'delimiter' ]
[ NULL [ AS ] 'null string' ] ]
```

## **COPY (CSV)**

O comando COPY a partir da versão 8.0 suporta arquivos no formato CSV (Comma Separated Value).  
Abaixo segue a sintaxe:

para copiar os dados de um arquivo para uma tabela nesse formato.

```
COPY tablename [ ( column [, ...] ) ]  
FROM { 'filename' | STDIN }  
[ CSV [ HEADER ]  
[ QUOTE [ AS ] 'quote' ]  
[ ESCAPE [ AS ] 'escape' ]  
[ FORCE NOT NULL column [, ...] ]
```

Onde:

quote: caracter utilizado para englobar strings vazios.

escape: caracter que deve aparecer antes do caracter de quote.

## **COPY TO (CSV)**

Exportando dados de tabelas para o formato CSV.

```
COPY { tablename [ ( column [, ...] ) ] | ( query ) }  
TO { 'filename' | STDOUT }  
[ CSV [ HEADER ]  
[ QUOTE [ AS ] 'quote' ]  
[ ESCAPE [ AS ] 'escape' ]  
[ FORCE QUOTE column [, ...] ]
```

Onde:

quote: caracter utilizado para englobar strings vazios.

escape: caracter que deve aparecer antes do caracter de quote.

force quote: força que as colunas especificadas com valores não nulos sejam englobadas em caracteres de quote.

## **Modificando dados com UPDATE**

O comando UPDATE não aceita mais de uma tabela para alteração.

Sintaxe:

```
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  
] [, ...] ]
```

O comando UPDATE não aceita mais de uma tabela para alteração.

## **Modificando dados com UPDATE**

Onde:

SET coluna=expressao

A cláusula SET é seguida por uma expressão para uma ou mais colunas da tabela

FROM lista\_de\_tabelas

Extensão específica do PostgreSQL para permitir que colunas de outras tabelas sejam utilizadas na cláusula WHERE.

WHERE condicao

A cláusula WHERE descreve a condição para que a expressão definida por SET seja aplicada.

RETURNING

Lista dos registros modificados.

## Exemplos de UPDATE

Exemplo:

```
SELECT * FROM agente WHERE codigo = 11;
codigo | cidade | nome
-----+-----+-----
      11 |      8 | Cobra
```

```
UPDATE agente SET nome = 'Cobra Cega'
WHERE codigo = 11;
UPDATE 1
```

Informa o número de linhas atualizadas

```
SELECT * FROM agente WHERE codigo = 11;
codigo | cidade | nome
-----+-----+-----
      11 |      8 | Cobra Cega
```

## Sub-consultas com UPDATE

Sub-consultas também podem ser usadas nos comandos de UPDATE.

Exemplo:

Atualizar a arrecadação do concurso com base nas apostas registradas (supondo um valor de R\$2 por aposta):

```
UPDATE concurso SET arrecadacao =
(SELECT count(*) FROM aposta a
WHERE a.concurso = concurso.numero) * 2;
```



## Removendo dados com DELETE

As linhas de uma tabela podem ser removidas com o comando DELETE.

Sintaxe:

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

Onde:

**ONLY**

Esta cláusula previne que registros inseridos via tabelas herdadas sejam removidos. Ou seja, restringe a operação somente para a tabela passada como parâmetro. WHERE condição

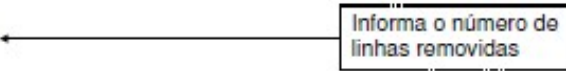
A cláusula WHERE impõe a condição para que o registro seja removido; se esta cláusula for omitida, TODOS os registros da tabela serão removidos.

**RETURNING**

Esta cláusula exibe os registros que foram removidos.

```
SELECT count(*) FROM aposta_numeros;
count
-----
  6050
(1 row)

DELETE FROM aposta_numeros WHERE aposta > 5;
DELETE 6000
```



```
SELECT count(*) FROM aposta_numeros;
count
-----
   50
(1 row)
```

```
DELETE FROM films USING producers
WHERE producer_id = producers.id AND producers.name = 'foo';
```

```
DELETE FROM films
WHERE producer_id IN (SELECT id FROM producers WHERE name = 'foo');
```

## **Sub-consultas com DELETE**

Sub-consultas também podem ser usadas nos comandos

DELETE

Para eliminar as apostas sem números:

```
DELETE FROM aposta WHERE aposta NOT IN  
(SELECT aposta FROM aposta_numeros);
```

## **Removendo dados com TRUNCATE**

A cláusula ONLY está implícita quando utilizamos o TRUNCATE, ou seja, somente registros da tabela alvo serão removidos.

O comando TRUNCATE remove todos os registros de uma tabela.

Sintaxe:

```
TRUNCATE [ TABLE ] [ ONLY ] name [, ... ]  
[ RESTART IDENTITY | CONTINUE IDENTITY ]  
[ CASCADE | RESTRICT ]
```

O comando é mais rápido do que o DELETE, pois não faz varredura na tabela.

A cláusula CASCADE torna possível aplicá-lo em tabelas que possuem chave estrangeira.

A cláusula ONLY está implícita quando utilizamos o TRUNCATE, ou seja, somente registros da tabela alvo serão removidos.

RESTART IDENTITY | CONTINUE IDENTITY (DEFAULT)

VOLTA UM VALOR DE UMA SEQUENCE CASO ESTEJA ATRIBUÍDO A UMA COLUNA.

## **Herança entre Tabelas**

### Herança de Tabelas

- O PostgreSQL permite a especificação de uma ou mais tabelas das quais uma nova tabela herdar a estrutura de colunas.
- O recurso de herança permite que uma tabela herde colunas de mais de uma tabela.
- A tabela criada possuirá todas as colunas da tabela pai, além das colunas específicas da tabela.

Ao realizar uma consulta numa tabela que possua tabelas filhas (que herdaram suas colunas), podemos selecionar somente as linhas da própria tabela ou as linhas da tabela mais as linhas das tabelas filhas.

```
SELECT * FROM ONLY [tabela];
```

## **Criando Tabelas com Herança**

Criamos uma tabela filha e utilizamos INHERITS para especificar a(s) tabela(s) que contém a estrutura de colunas que serão herdadas.

A cláusula INHERITS pode indicar uma ou mais tabelas.

Sintaxe:

```
CREATE TABLE tab_filha ( colunas_especificas ) INHERITS ( tab_pai [,...] )
```

As linhas inseridas na tabela-pai não são visíveis para consultas na tabela-filha. Por outro lado, a tabela pai enxerga as linhas inseridas na tabela filha.

*Pode haver uma coluna com valor duplicado, mesmo com índice único, na tabela pai. Isto acontece quando uma inserção é feita na tabela pai e outra é feita na tabela filha (com o mesmo valor da chave única).*

## **Exemplos de Herança**

Exemplo:

```
CREATE TABLE b (h integer)
INHERITS (a);
SELECT * FROM a;
SELECT * FROM b;
INSERT INTO a VALUES (99, 99, 'Registro novo', 'Agora', timenow());
INSERT INTO b (9900, 9900, 'Registro novo de novo', 'Agora', timenow())
SELECT * FROM a;
SELECT * FROM b;
```

## **Acessando Dados da Tabela Pai**

É possível acessar exclusivamente os dados da tabela pai nas operações de SELECT, UPDATE e DELETE. Para isto, usamos a opção ONLY.

Exemplos:

```
SELECT * FROM ONLY a;
DELETE FROM ONLY a
WHERE codigo = 1;
```

```
UPDATE ONLY a
SET c = 'Não atualiza capitais!'
WHERE a = 9900;
```

## **Obrigado!**