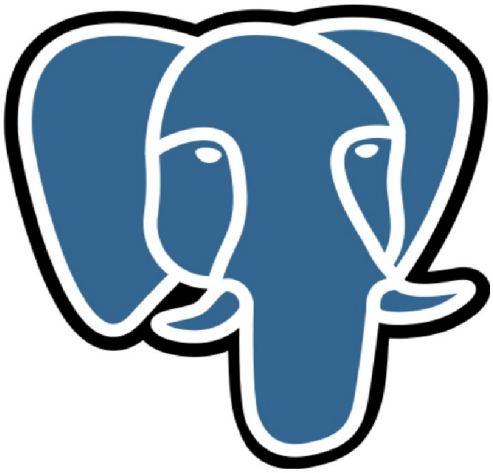


# PostgreSQL Essentials

POLIBRÁS NET

## Objetivos do Curso

*** Restrições e integridade referencial**
- **Conceito de integridade referencial**
- **Controle de restrições(CONSTRAINT)**
- **Constraint CHECK**
- **Constraint NOT NULL**
- **Constraint UNIQUE**
- **Constraint PRIMARY KEY**
- **Constraint FOREIGN KEY**

*** Seleção de dados**
- **Introdução**
- **Utilização de expressões e constantes**
- **Eliminação de linhas duplicadas**
- **Manipulação de resultados**
- **Comando CASE**
- **Substituição de valores nulos**
- **Funções para lista de valores**
- **Consultas Recursivas**

*** Operações de conjuntos**
- **Introdução**
- **União**
- **Intersecção**
- **Subtração**

# 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.**

# Capítulo 6

# Restrições e Integridade Referencial

## O que são Restrições?

- A grande parte dos sistemas de informação necessita criar restrições, baseadas em regras de negócios, sobre os dados armazenados nas tabelas.
- O banco de dados PostgreSQL oferece o recurso de controlar as restrições ao nível de banco de dados através da cláusula CONSTRAINT.
- O uso de restrições em colunas permite o grau de controle que se deseja sobre o conteúdo dos dados.
- Quando uma restrição é violada, a operação é abortada
- Existem os seguintes tipos de constraints no PostgreSQL:

  Check
  Not null
  Unique
  Primary key
  Foreign key

## TABELA EXEMPLO

```
CREATE TYPE type_condicoes AS enum ('novo', 'seminovo', 'conservado', 'precisa de conserto') ;

CREATE TABLE ema_carros (
    id_carro        serial PRIMARY KEY,
    marca           text,
    modelo          text,
    combustivel     text CHECK (combustivel in ('alcool','diesel','gasolina','gnv','bi-combustivel','tri-combustivel')),
    ano             integer,
    condicoes       type_condicoes,
    valor           double precision,
    opcionais       text[],
    data_inc        timestamp without time zone,
    observacoes text
) ;
```

## CHECK CONSTRAINT

- A CONSTRAINT CHECK especifica que o valor da coluna deve obedecer uma determinada expressão.

- Por exemplo, para garantir que a arrecadação seja um valor positivo:

```
ALTER TABLE ema_carros ADD CHECK (valor >= 0);
```

- É possível nomear a constraint:

```
ALTER TABLE ema_carros
ADD   CONSTRAINT    ema_carros_valor_positivo
CHECK (valor >= 0);
```

- É possível adicionar ou remover uma constraint com o comando ALTER TABLE:

```
ALTER TABLE ema_carros
DROP CONSTRAINT ema_carros_valor_positivo;
```

## NOT NULL CONSTRAINT

- Esta restrição impede que o valor de uma coluna seja nulo.

- Por exemplo, os valores do código do produto e o seu nome não podem ser nulos:

```
CREATE TABLE produtos (
    produto_id integer NOT NULL,
    nome text NOT NULL,
    preco numeric);
```

- Funcionalmente, a restrição NOT NULL é equivalente à restrição CHECK (coluna IS NOT NULL). Entretanto, a primeira forma é mais eficiente.

- Não é possível nomear restrições NOT NULL.

## NOT NULL CONSTRAINT

- É possível adicionar ou remover uma constraint NOT NULL após a tabela estar criada:

```
ALTER TABLE  produtos
ALTER COLUMN preco SET NOT NULL;

ALTER TABLE  produtos
ALTER COLUMN preco DROP NOT NULL;
```

## UNIQUE CONSTRAINT

- A restrição UNIQUE garante que o conteúdo de uma coluna, ou grupo de colunas, é único em relação à tabela. É criado um índice implícito para garantir a unicidade.

- Uma restrição UNIQUE admite valores nulos.

- No exemplo utilizado, o código do produto deve ser único:

```
CREATE TABLE produtos_2 (
produto_id integer UNIQUE,
nome text,
preco numeric);
```

- Quando existem mais de uma coluna na restrição, usamos:

```
CREATE TABLE exemplo (
a integer,
b integer,
c integer,
UNIQUE (a, c)
);

CREATE TABLE cidades (
id integer,
nome text,
uf char(2)
);
```

## UNIQUE CONSTRAINT

- Também é possível adicionar ou remover uma constraint UNIQUE após a tabela estar criada:

```
ALTER TABLE  produtos_2
ADD CONSTRAINT uni_produtos_2_nome UNIQUE (nome);

ALTER TABLE produtos_2
DROP CONSTRAINT uni_produtos_2_nome;
```

- Uma constraint UNIQUE pode ter múltiplas colunas:

```
ALTER TABLE cidades
ADD CONSTRAINT uni_cidades_23 UNIQUE (nome, uf);
ALTER TABLE cidades DROP CONSTRAINT uni_cidades_23;
```

## PRIMARY KEY CONSTRAINT

- Tecnicamente, a restrição PRIMARY KEY é uma combinação das restrições UNIQUE e NOT NULL.

- É criado um índice implícito para garantir a unicidade.

- Exemplo:

```
CREATE TABLE produtos (
produto_id integer PRIMARY KEY,
nome text,
preco numeric);
```

- A teoria relacional diz que toda tabela deve ter uma PRIMARY KEY. Entretanto, o PostgreSQL não exige que esta regra seja cumprida.

## PRIMARY KEY CONSTRAINT

- Também é possível adicionar ou remover uma constraint PRIMARY KEY após a tabela estar criada:

```
ALTER TABLE  produtos
ADD CONSTRAINT pk_produtos PRIMARY KEY (produto_id);
ALTER TABLE produtos DROP CONSTRAINT pk_produtos;
```

Ou

```
ALTER TABLE produtos ADD PRIMARY KEY (produto_id);
ALTER TABLE produtos DROP CONSTRAINT produtos_pkey;
```

- Uma constraint PRIMARY KEY pode ter múltiplas colunas:

```
ALTER TABLE produto_fornecedor  ADD CONSTRAINT
pk_produto_fornecedor PRIMARY KEY (produto_id, fornecedor_id);

ALTER TABLE produto_fornecedor
DROP CONSTRAINT pk_produto_fornecedor;
```

## FOREIGN KEY CONSTRAINT

- Uma restrição FOREIGN KEY especifica que os valores numa coluna (ou grupo de colunas) devem casar com valores contidos em alguma linha de outra tabela relacionada.
- Esta relação é conhecida também como integridade referencial.
- Por exemplo:

```
CREATE TABLE estados (
uf char(2) PRIMARY KEY,
estado text
);

CREATE TABLE cidades (
codigo int4 NOT NULL,
nome varchar NOT NULL,
uf char(2) NOT NULL,
FOREIGN KEY (uf) REFERENCES estados(uf)
);
```

*As colunas referenciadas devem ser PRIMARY KEY ou UNIQUE KEY.*

## Sintaxe de FOREIGN KEY

- Sintaxe de constraint de coluna:

REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL |
MATCH PARTIAL | MATCH SIMPLE ]
[ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED |
INITIALLY IMMEDIATE ]

- Uma CONSTRAINT também pode ser adicionada sobre mais de uma coluna. Neste caso, ela é chamada de constraint de tabela e a restrição REFERENCES muda para:

FOREIGN KEY ( column_name [, ... ] ) REFERENCES reftable
[ ( refcolumn [, ... ] ) ]
[ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
[ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED |
INITIALLY IMMEDIATE ]

- Onde:

FOREIGN KEY (column [,...] )
REFERENCES reftable [ ( refcolumn [,...] ) ]

Valores da(s) coluna(s) são checados aos valores da tabela referenciada.

MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]

Habilita ou impede que partes dos valores das colunas sejam nulos.

[ ON DELETE action ] [ ON UPDATE action ]
[ DEFERRABLE | NOT DEFERRABLE ]

A opção DEFERRABLE permite que a ação seja postergada e executada no final da transação e não em cada comando.

A opção NOT DEFERRABLE (padrão) executa a verificação a cada comando.

*A opção MATCH PARTIAL não está implementada.*

## Sintaxe de FOREIGN KEY

- É também possível adicionar ou remover uma constraint

FOREIGN KEY após a tabela estar criada:

```
ALTER TABLE cidades
ADD CONSTRAINT cidades_uf_fkey
FOREIGN KEY (uf) REFERENCES estados (uf);
```

- Uma constraint FOREIGN KEY também pode ter múltiplas colunas.

## Exemplos

```
INSERT INTO estados VALUES ('CE', 'Ceará'),('SP','São Paulo'),('PI', 'Piauí'),('RJ','Rio de Janeiro') ;

SELECT * FROM estados ;

INSERT INTO cidades VALUES (1, 'Fortaleza', 'CE') ;
INSERT INTO cidades VALUES (1, 'Caucaia', 'CE') ;
INSERT INTO cidades VALUES (1, 'Caruaru', 'PE') ;

ERROR:  insert or update on table "cidades" violates foreign key constraint "cidades_uf_fkey"
DETAIL:  Key (uf)=(PE) is not present in table "estados".
```

## Ações sobre FOREIGN KEYS

CASCADE: remove/atualiza as linhas que referenciam a linha removida.
SET NULL: assinala o valor NULL para as colunas
SET DEFAULT: assinala o valor padrão definido para as colunas

- O comportamento padrão é o NO ACTION, ou seja, a transação é abortada no final, por tentar remover uma linha que violaria uma constraint.

## A Ação CASCADE

- Usando a opção CASCADE:

```
ALTER TABLE cidades DROP CONSTRAINT cidades_uf_fkey ;
ALTER TABLE cidades
ADD CONSTRAINT cidades_uf_fkey
FOREIGN KEY (uf) REFERENCES estados (uf)
ON UPDATE CASCADE ON DELETE CASCADE;
```

- Vamos verificar a situação atual da tabela aposta_numeros:

```
SELECT * FROM estados ;
SELECT * FROM cidades ;
```

- Atualizando linhas da tabela apostas:

```
UPDATE estados SET uf = 'PE' WHERE uf = 'CE';
SELECT * FROM cidades ;
```

- Removendo linhas da tabela apostas:

```
DELETE FROM estados WHERE uf = 'PE';
SELECT * FROM cidades ;
```

- Removendo a constraint:

```
ALTER TABLE cidades
DROP CONSTRAINT cidades_uf_fkey;
```

## A Ação SET NULL

- Usando a opção SET NULL:

```
ALTER TABLE cidades DROP CONSTRAINT cidades_uf_fkey ;
ALTER TABLE cidades
ADD CONSTRAINT cidades_uf_fkey
FOREIGN KEY (uf) REFERENCES estados (uf)
ON UPDATE SET NULL ON DELETE SET NULL;
```

# Capítulo 7

# Selecionando Dados

## Selecionando Dados com SELECT

- Sintaxe:

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    * | expression [ [ AS ] output_name ] [, ...]
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY expression [, ...] ]
    [ HAVING condition [, ...] ]
    [ WINDOW window_name AS ( window_definition ) [, ...] ]
    [ { UNION | INTERSECT | EXCEPT } [ ALL ] select ]
    [ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS { FIRST
| LAST } ] [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start [ ROW | ROWS ] ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
    [ FOR { UPDATE | SHARE } [ OF table_name [, ...] ] [ NOWAIT ] [...] ]
```

## Um SELECT Simples

- Uma query simples:

```
SELECT s_codcliente, s_nome, s_cnpj_cpf FROM t_cliente LIMIT 10;
```

```
 s_codcliente |             s_nome               |     s_cnpj_cpf
--------------+----------------------------------+--------------------
 10000        | JOSE APARECIDO MOREIRA SOUZA     | 027.685.928-62
 10001        | MARCELINO CARDOSO DE QUEIROZ     | 858.739.262-04
 10002        | ANDREIA MODY DE LIMA             | 517.186.692-91
 10003        | LUIZ SALATI                      | 09.059.240/0001-36
 10004        | ISANDRO DA SILVA SEIXAS          | 719.735.592-34
 10005        | ISAJIM MERCAINHO LTDA            | 08.855.370/0001-12
 10006        | JOSE FRANK LANE SOUZA DA SILVA   | 727.584.902-00
 10007        | MANUEL NEVES DE SOUZA            | 240.213.262-00
 10008        | VALDEMILSON DA SILVA ROCHA       | 874.917.373-15
 1001         | ANTONIO AMARO SANTANA SERASA     | 34.506.782/0001-57
```

*O comando SELECT * retorna todas as colunas de uma tabela.*

## O comando TABLE

- A partir da versão 8.4, podemos simplificar o SELECT em toda a tabela com o comando TABLE:

```
TABLE t_cliente;
```

## Selecionando Dados com SELECT

- Podemos especificar as colunas a serem consultadas:

```
SELECT s_codcliente, s_nome, s_cnpj_cpf FROM t_cliente LIMIT 2;
```

```
 s_codcliente |            s_nome              |   s_cnpj_cpf
--------------+--------------------------------+----------------
 10000        | JOSE APARECIDO MOREIRA SOUZA   | 027.685.928-62
 10001        | MARCELINO CARDOSO DE QUEIROZ   | 858.739.262-04
```

- Podemos renomear as colunas:

```
SELECT s_codcliente AS “Codigo Cliente”,
        s_nome AS “Nome Cliente”,
        s_cnpj_cpf AS “Cnpj/Cpf”
FROM t_cliente
LIMIT 10;
```

```
 Codigo Cliente |          Nome Cliente            |      Cnpj/Cpf
----------------+----------------------------------+--------------------
 10000          | JOSE APARECIDO MOREIRA SOUZA     | 027.685.928-62
 10001          | MARCELINO CARDOSO DE QUEIROZ     | 858.739.262-04
 10002          | ANDREIA MODY DE LIMA             | 517.186.692-91
 10003          | LUIZ SALATI                      | 09.059.240/0001-36
 10004          | ISANDRO DA SILVA SEIXAS          | 719.735.592-34
 10005          | ISAJIM MERCAINHO LTDA            | 08.855.370/0001-12
 10006          | JOSE FRANK LANE SOUZA DA SILVA   | 727.584.902-00
 10007          | MANUEL NEVES DE SOUZA            | 240.213.262-00
 10008          | VALDEMILSON DA SILVA ROCHA       | 874.917.373-15
 1001           | ANTONIO AMARO SANTANA SERASA     | 34.506.782/0001-57
```

# Consultas Recursivas

- Consultas recursivas são executadas através da cláusula WITH RECURSIVE.
- Essas consultas eliminam boa parte da programação PL/PGSQL.
- A forma geral de um consulta recursiva é sempre formada por termo não-recursivo, a união (ou UNION ALL), e em seguida, um termo recursivo.

```
WITH RECURSIVE t(n) AS (
    VALUES (1)
    UNION ALL
    SELECT n+1 FROM t WHERE n < 100
 )
 SELECT sum(n) FROM t;

WITH RECURSIVE t(n) AS (
    VALUES (1)
    UNION ALL
    SELECT n+1 FROM t WHERE n < 100
 )
 SELECT n, '-> '||n||' + 1' FROM t;

WITH RECURSIVE t(n) AS (
    VALUES (1)
--  UNION ALL
--  SELECT n+1 FROM t WHERE n < 100
 )
 SELECT n, '-> '||n||' + 1' FROM t;
```

# Consultas Recursivas

```
WITH recursive Fib (i, j) AS (
 VALUES (0, 1)
UNION ALL
SELECT (i + j), (i + j) + j FROM Fib WHERE (i + j) < 100)
SELECT i FROM Fib UNION ALL SELECT j FROM Fib ORDER BY i;
```

```
 i
---
   0
   1
   1
   2
   3
   5
   8
  13
  21
  34
  55
  89
```

```
WITH recursive Fib (i, j) AS (
 VALUES (0, 1)
UNION ALL
SELECT (i + j), (i + j) + j FROM Fib WHERE (i + j) < 100)
SELECT i,'('||i||' + '||j||'), ('||i||' + '||j||') + '||j FROM Fib UNION ALL SELECT j,'('||i||' + '||j||'), ('||i||'
+ '||j||') + '||j FROM Fib ORDER BY i;
```

```
 i  |          ?column?
----+---------------------------
  0 | i
  1 | j
  1 | (0 + 1), (0 + 1) + 1
  2 |
  3 | (1 + 2), (1 + 2) + 2
  5 |
  8 | (3 + 5), (3 + 5) + 5
 13 |
 21 | (8 + 13), (8 + 13) + 13
 34 |
 55 | (21 + 34), (21 + 34) + 34
 89 |
```

```
 i  |          ?column?
----+---------------------------
  0 | i = 0
  1 | j = 1
  1 | i = (0 + 1)
  2 | j = (0 + 1) + 1
  3 | i = (1 + 2)
  5 | j = (1 + 2) + 2
  8 | i = (3 + 5)
 13 | j = (3 + 5) + 5
 21 | i = (8 + 13)
 34 | j = (8 + 13) + 13
 55 | i = (21 + 34)
 89 | j = (21 + 34) + 34
```

```
WITH recursive Fib (i, j) AS (
 VALUES (0, 1)
UNION ALL
SELECT (i + j), (i + j) + j FROM Fib WHERE (i + j) < 100)
SELECT i,'('||i||' + '||j||')' FROM Fib ORDER BY i;
```

```
 i  |         ?column?
----+--------------------------
  0 |  0
  1 | (0 + 1)
  3 | (1 + 2)
  8 | (3 + 5)
 21 | (8 + 13)
 55 | (21 + 34)
```

```
WITH recursive Fib (i, j) AS (
 VALUES (0, 1)
UNION ALL
SELECT (i + j), (i + j) + j FROM Fib WHERE (i + j) < 100)
SELECT j,'('||i||' + '||j||'), ('||i||' + '||j||') + '||j FROM Fib ORDER BY i;
```

```
 j  |         ?column?
----+--------------------------
  1 | 1
  2 | (0 + 1) + 1
  5 | (1 + 2) + 2
 13 | (3 + 5) + 5
 34 | (8 + 13) + 13
 89 | (21 + 34) + 34
```

```
WITH recursive Fib (i, j) AS (
 VALUES (0, 1)
UNION ALL
SELECT (i + j),j+j FROM Fib WHERE (i + j) < 100)
SELECT i,j FROM Fib  ORDER BY i;
```

```
 i  | j
----+----
  0 |  1
  1 |  2
  3 |  4
  7 |  8
 15 | 16
 31 | 32
 63 | 64
```

# Consultas Recursivas

- Exemplo de consulta recursiva com window function:

```
pagila=# SELECT * FROM
(
WITH cte as (
  SELECT first_name, last_name, store_id, sum(amount) as
total FROM payment JOIN customer using (customer_id) GROUP BY
first_name, last_name, store_id
)
SELECT first_name, last_name, store_id, total, rank() OVER
(PARTITION BY store_id ORDER BY total desc) FROM cte ) x
where rank <= 3;
```

```
  first_name     | last_name   |     store_id | total   | rank
  ---------------+-------------+--------------+---------+---------
  ELEANO   R     | HUNT        |            1 | 216.54  |       1
  CLARA          | SHAW        |            1 | 195.58  |       2
  TOMMY          | COLLAZO     |            1 | 186.62  |       3
  KARL           | SEAL        |            2 | 221.55  |       1
  MARION         | SNYDER      |            2 | 194.61  |       2
  RHONDA         | KENNEDY     |            2 | 194.61  |       2
```

(6 rows)

# Utilizando Expressões e Constantes

- SELECT com expressões e apelidos em colunas:

  SELECT s_codcliente, s_nome, (d_coef_ajuste * 0.25) as ajuste FROM t_cliente;

```
 s_codcliente |                  s_nome                   |  ajuste
--------------+-------------------------------------------+----------
 11080        | NELCICLEI GALVAO DO NASCIMENTO            | 0.000000
 11233        | ELIANE LEAL DE SOUZA                      | 0.750000
 1676         | MARIA MARLUCE DA SILVA                    | 2.000000
 1940         | J C CARVALHO                              | 0.500000
 2262         | LIMEIRA E ALCANTARA LTDA                  | 0.250000
 2302         | JOSE CORACY DE SOUZA                      | 0.000000
```

- SELECT com constantes e concatenação:

```
SELECT nome||' - '||uf FROM cidades;

          ?column?
------------------------
 São Paulo - SP
 Sorocaba - SP
 Osasco - SP
 Campinas - SP
```

## Removendo Linhas Duplicadas no SELECT

- Em uma determinada coluna podem haver valores duplicados.
- Exemplo sem o DISTINCT:

```
SELECT uf FROM cidades ;

 uf
----
 SP
 SP
...
 RJ
 RJ
```

- Exemplo com o DISTINCT :

```
SELECT DISTINCT uf FROM cidades ;

 uf
----
 MG
 RJ
 SC
 SP
```

## Limitando o Resultado de um SELECT

- Se a tabela contém muitos registros, podemos selecionar um número restrito com o LIMIT.
- Para selecionar 10 linhas da tabela aposta_numeros, a partir da primeira linha, utilizamos:

```
SELECT * FROM t_produto LIMIT 10;
```

- Para selecionar 10 linhas, a partir da décima primeira, ou seja, os registros 11 a 20, utilizamos:

```
SELECT * FROM t_produto LIMIT 10 OFFSET 10;
```

**EXEMPLO**

```
SELECT u_pkey,
      u_orgvenda,
      s_codproduto,
      d_estoque,
      s_descricao
FROM t_produto ORDER BY u_pkey
LIMIT 30;
```

| u_pkey | u_orgvenda | s_codproduto | d_estoque | s_descricao |
|---|---|---|---|---|
| 18 | 1 | 1388 | 94.0000 | CHIC T LINE MENTA 12X12G |
| 20 | 1 | 1410 | 5368.0000 | PIRUL BIG BIG T FRUT 700G |
| 22 | 1 | 1412 | 2594.0000 | PIRUL BIG BIG MOR 700GR |
| 23 | 1 | 1414 | 3158.0000 | PIRUL BIG BIG SORT 700G |
| 26 | 1 | 1429 | 3754.0000 | SM BALA RECH DE FRUTAS 150G |
| 40 | 1 | 1462 | 164.0000 | POTE BALA KIDS LEITE 800G |
| 41 | 1 | 1463 | 283.0000 | SM BALA KIDS LEITE 170G |
| 43 | 1 | 1465 | 38.0000 | BALA 7 BELO IOGURTE BS 1K |
| 44 | 1 | 1466 | 1890.0000 | SM BALA 7 BELO FRAMB 200G |
| 45 | 1 | 1467 | 535.0000 | SM BALA 7 BELO IOGURTE 200G |
| 46 | 1 | 1468 | 513.0000 | SM BALA 7 BELO MACA VD 200G |
| 49 | 1 | 1474 | 651.0000 | SM BALA KIDS AMENDOIM 170G |
| 50 | 1 | 1475 | 1406.0000 | SM BALA KIDS HORTELA 170G |
| 51 | 1 | 1476 | 0.0000 | BALA KIDS HORTELA BS 800G |
| 55 | 1 | 1511 | 120.0000 | BOMBOM SAMBA LEITE BS 500G |
| 57 | 1 | 1513 | 93.0000 | BOMBOM SAMBA BRIGAD BS 500G |
| 60 | 1 | 1517 | 348.0000 | BOMBOM SAMBA MIX 200G |
| 63 | 1 | 1522 | 3.0000 | CHOC TABL AO LEITE 12X160G |
| 64 | 1 | 1523 | 161.0000 | CHOC TABL BCO 12X160G |
| 65 | 1 | 1524 | 5.0000 | CHOC TABL CROC 12X140G |
| 66 | 1 | 1526 | 232.0000 | CHOC TABL BRIGAD 12X160G |
| 67 | 1 | 1527 | 42.0000 | CHOC TABL MEIO AMARGO 12X160G |
| 69 | 1 | 1530 | 17.0000 | CHOC TABL TWIST CROC 24X20G |
| 70 | 1 | 1532 | 0.0000 | CHOC TABL TWIST BRIGAD 24X20G |
| 71 | 1 | 1533 | 988.0000 | SM CHOC TORT BAUN 72G |
| 72 | 1 | 1536 | 130.0000 | SM CHOC TORT CH BCO 72G |
| 76 | 1 | 1545 | 10.0000 | CHOC TORT.LT RECH MOR 432G |
| 77 | 1 | 1546 | 60.0000 | SM CHOC TORT.BRIGAD 76G |
| 78 | 1 | 1548 | 0.0000 | CHOC TORT BRIGAD 24X19G |
| 84 | 1 | 1563 | 6.0000 | CONF ROCKLETS LEITE 12X40G |

```
SELECT u_pkey,
      u_orgvenda,
      s_codproduto,
      d_estoque,
      s_descricao
FROM t_produto ORDER BY u_pkey
LIMIT 30 OFFSET 10;
```

| u_pkey | u_orgvenda | s_codproduto | d_estoque | s_descricao |
|---|---|---|---|---|
| 46 | 1 | 1468 | 513.0000 | SM BALA 7 BELO MACA VD 200G |
| 49 | 1 | 1474 | 651.0000 | SM BALA KIDS AMENDOIM 170G |
| 50 | 1 | 1475 | 1406.0000 | SM BALA KIDS HORTELA 170G |
| 51 | 1 | 1476 | 0.0000 | BALA KIDS HORTELA BS 800G |
| 55 | 1 | 1511 | 120.0000 | BOMBOM SAMBA LEITE BS 500G |
| 57 | 1 | 1513 | 93.0000 | BOMBOM SAMBA BRIGAD BS 500G |
| 60 | 1 | 1517 | 348.0000 | BOMBOM SAMBA MIX 200G |
| 63 | 1 | 1522 | 3.0000 | CHOC TABL AO LEITE 12X160G |
| 64 | 1 | 1523 | 161.0000 | CHOC TABL BCO 12X160G |
| 65 | 1 | 1524 | 5.0000 | CHOC TABL CROC 12X140G |

```
SELECT u_pkey,
      u_orgvenda,
      s_codproduto,
      d_estoque,
      s_descricao
FROM t_produto ORDER BY u_pkey
LIMIT 30 OFFSET 20;
```

| u_pkey | u_orgvenda | s_codproduto | d_estoque | s_descricao |
|---|---|---|---|---|
| 66 | 1 | 1526 | 232.0000 | CHOC TABL BRIGAD 12X160G |
| 67 | 1 | 1527 | 42.0000 | CHOC TABL MEIO AMARGO 12X160G |
| 69 | 1 | 1530 | 17.0000 | CHOC TABL TWIST CROC 24X20G |
| 70 | 1 | 1532 | 0.0000 | CHOC TABL TWIST BRIGAD 24X20G |
| 71 | 1 | 1533 | 988.0000 | SM CHOC TORT BAUN 72G |
| 72 | 1 | 1536 | 130.0000 | SM CHOC TORT CH BCO 72G |
| 76 | 1 | 1545 | 10.0000 | CHOC TORT.LT RECH MOR 432G |
| 77 | 1 | 1546 | 60.0000 | SM CHOC TORT.BRIGAD 76G |
| 78 | 1 | 1548 | 0.0000 | CHOC TORT BRIGAD 24X19G |
| 84 | 1 | 1563 | 6.0000 | CONF ROCKLETS LEITE 12X40G |

# O Comando CASE

- O comando SQL CASE é uma expressão condicional semelhante ao IF/THEN em linguagens procedurais.

- O CASE pode ser utilizado em qualquer lugar que uma expressão seja válida.

- Sintaxe:

```
CASE expression
    WHEN value THEN result
    [WHEN ...]
[ELSE result]
END;
```

- Exemplo:

```
SELECT s_codproduto,
    s_descricao,
    CASE s_descricao_unidade  WHEN 'CX' THEN 'CAIXA'
       ELSE 'OUTRO'
    END FROM t_produto;
```

```
SELECT s_codproduto,
        s_descricao,
        CASE WHEN s_descricao_unidade  = 'CX' THEN 'CAIXA'
                WHEN s_descricao_unidade  = 'PC' THEN 'PACOTE'
                WHEN s_descricao_unidade  = 'UN' THEN 'UNIDADE'
                WHEN s_descricao_unidade  = 'FD' THEN 'FARDO'
                WHEN s_descricao_unidade  = 'LT' THEN 'LATA'
               ELSE 'OUTRO'
        END FROM t_produto;
```

# Substituindo Valores Nulos

- Quando uma coluna possui valores nulos e queremos substituí-los por valores padrão, utilizamos o comando COALESCE.

- O COALESCE devolve o primeiro valor não nulo da lista de parâmetros passada.

- Sintaxe:

```
COALESCE( valor1 [,...] )
```

# Substituindo Valores Nulos

- Exemplos:

```
 SELECT s_codproduto as "Codigo Produto",
d_estoque,
d_unidade,
        d_estoque / d_unidade AS "Estoque em embalagens"
   FROM t_produto;
```

```
 Codigo Produto | d_estoque  | d_unidade | Estoque em embalagens
----------------+------------+-----------+-----------------------
 101965         |     7.0000 |   12.0000 | 0.58333333333333333333
 102851         |    17.0000 |   12.0000 |     1.4166666666666667
 103447         |     0.0000 |   16.0000 | 0.00000000000000000000
 1388           |    94.0000 |   12.0000 |     7.8333333333333333
 1410           |  5368.0000 |   50.0000 |   107.36000000000000000
 1412           |  2594.0000 |   50.0000 |    51.88000000000000000
 1414           |  3158.0000 |    1.0000 |  3158.00000000000000000
 1429           |  3754.0000 |    1.0000 |  3754.00000000000000000
 1433           |  1170.0000 |    1.0000 |  1170.00000000000000000
 1462           |   164.0000 |    1.0000 |   164.00000000000000000
 1463           |   283.0000 |    1.0000 |   283.00000000000000000
 1465           |    38.0000 |    1.0000 |    38.00000000000000000
 1466           |  1890.0000 |    1.0000 |  1890.00000000000000000
```

```
 SELECT s_codproduto as "Codigo Produto",
d_estoque,
d_unidade,
        d_estoque / COALESCE(d_unidade, 0) AS "Estoque em embalagens"
   FROM t_produto;
```

```
 Codigo Produto | d_estoque  | d_unidade | Estoque em embalagens
----------------+------------+-----------+-----------------------
 101965         |     7.0000 |   12.0000 | 0.58333333333333333333
 102851         |    17.0000 |   12.0000 |     1.4166666666666667
 103447         |     0.0000 |   16.0000 | 0.00000000000000000000
 1388           |    94.0000 |   12.0000 |     7.8333333333333333
 1410           |  5368.0000 |   50.0000 |   107.36000000000000000
 1412           |  2594.0000 |   50.0000 |    51.88000000000000000
 1414           |  3158.0000 |    1.0000 |  3158.00000000000000000
 1429           |  3754.0000 |    1.0000 |  3754.00000000000000000
 1433           |  1170.0000 |    1.0000 |  1170.00000000000000000
 1462           |   164.0000 |    1.0000 |   164.00000000000000000
 1463           |   283.0000 |    1.0000 |   283.00000000000000000
 1465           |    38.0000 |    1.0000 |    38.00000000000000000
 1466           |  1890.0000 |    1.0000 |  1890.00000000000000000
```

## Funções para lista de valores

- A função NULLIF retorna nulo se os valores 1 e 2 forem iguais,
- caso contrário retorna o valor1:

```
SELECT nullif(1,1);

SELECT nullif(1,2);
```

- A função GREATEST retorna o maior valor de uma lista de valores:

```
SELECT greatest(1,4,3,7,2,9,8);

greatest
--------
9
```

- A função LEAST retorna o menor valor de uma lista de valores:

```
SELECT least(1,4,3,7,2,9,8);

least
--------
1
```

# Capítulo 8

## Operações de Conjunto

```
CREATE TABLE tab1 (col1 text, col2 int) ;
CREATE TABLE tab2 (col1 text, col2 int) ;

INSERT INTO tab1 VALUES ('A', 1),('B', 2),('C', 3),('D', 4),('E', 5) ;
INSERT INTO tab2 VALUES ('A', 1),('B', 1),('D', 4),('E', 5),('H', 6) ;
```

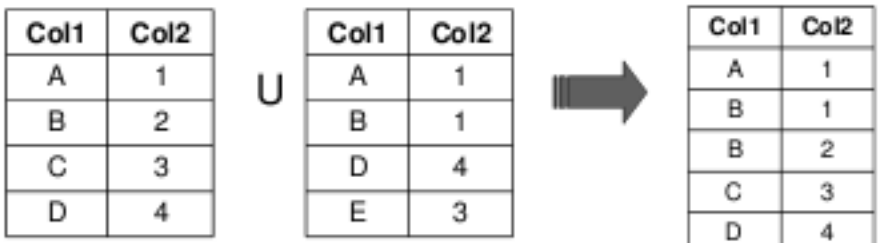
- Uma tabela pode ser encarada como um conjunto.
- Os elementos do conjunto são as linhas da tabela:


- Existem três operações básicas de conjunto no PostgreSQL:

União (UNION)
Intersecção (INTERSECT)
Diferença (EXCEPT)

- Nas duas consultas, o número de colunas selecionadas deve ser idêntico e os tipos de dados das colunas também devem ser iguais.
- Cria um novo conjunto de dados com a união das linhas das duas consultas

.

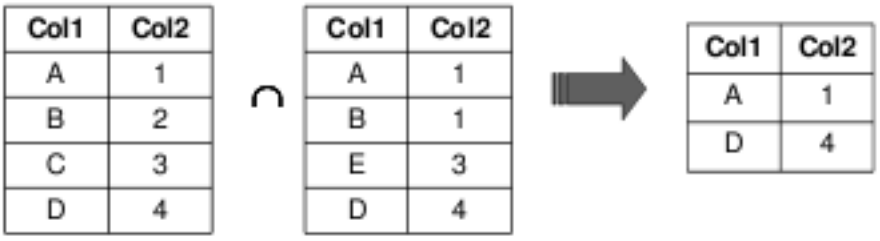



```
SELECT col1, col2 FROM tab1
UNION [ ALL ]
SELECT col1, col2 FROM tab2
```

- As linhas duplicadas nas tabelas são removidas a menos que se utilize a opção ALL.
- Quando não temos as colunas em todas as tabelas, podemos complementar com NULL.

## Intersecção

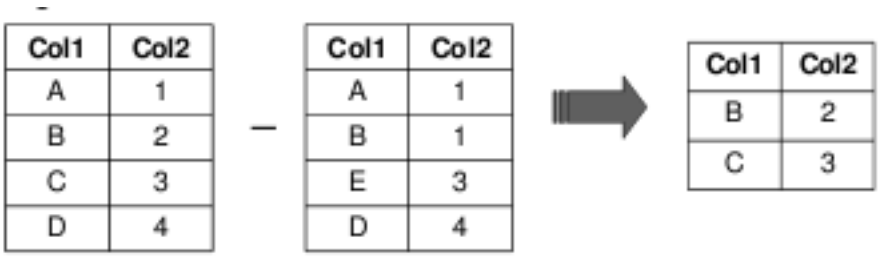
- A intersecção seleciona os valores que existem nas duas tabelas.




```
SELECT col1, col2 FROM tab1
INTERSECT
SELECT col1, col2 FROM tab2
```

## Subtração

- Seleciona os valores que estejam na primeira tabela, mas não na segunda.




```
SELECT col1, col2 FROM tab1
EXCEPT
SELECT col1, col2 FROM tab2
```