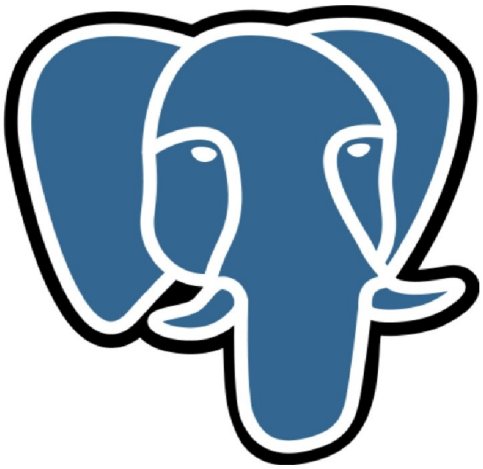

PostgreSQL Essentials

POLIBRÁS NET

Objetivos do Curso

Filtros na seleção de dados

- o Utilização da cláusula WHERE
- o Operador LIKE e ILIKE
- o Operador BETWEEN
- o Operador IN
- o Comparações agrupadas
- o Comparações com NULL
- o Cláusula ORDER BY

Seleção de dados de várias tabelas

- o Apelidos de tabelas
- o Cruzamento de dados entre tabelas
- o Tipos de Junções
- o CROSS JOIN
- o INNER e OUTER JOINs
- o NATURAL JOIN

Operadores

- o Tipos de operadores
- o Operadores de texto
- o Expressões regulares
- o Operadores matemáticos
- o Conversão de tipos

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 9

Restringindo a Seleção de Dados

Utilizando a Cláusula WHERE

- A cláusula WHERE permite o uso de condições lógicas que devem ser satisfeitas para que linhas sejam selecionadas. Podem ser usadas na cláusula WHERE valores de colunas, literais, expressões aritméticas e funções.

- A cláusula WHERE corresponde ao operador de restrição da álgebra relacional.

- O WHERE contém uma ou mais condições que as linhas devem satisfazer para serem selecionadas. É possível adicionar várias condições lógicas ligadas pelos operadores lógicos AND, OR e NOT. Os operadores lógicos podem ser agrupados entre parênteses para fins de organização e legibilidade.

- Exemplos:

```
SELECT s_codcliente, s_nome FROM t_cliente WHERE d_lim_cred > 0;

SELECT s_fantasia, s_atividade, s_emails FROM t_cliente WHERE s_emails IS NOT NULL LIMIT 10;
```

```
       s_fantasia        |          s_atividade           |          s_emails
-------------------------+--------------------------------+---------------------------
 PREJUVENATO SAO GERALDO | ORGANIZAÇÃO RELIGIOSA (NORMAL) | 1
 ASSOCIACAO              | OUTROS (NORMAL)                | 2
 UNIVERSSO DA EMPADA     | LANCHONETE (NORMAL)            | 1
 VAL                     | OUTROS (ESPECIAL)              | VALERIAMELO16@YAHOO.COM.BR
 DELLA NONNA             | PADARIA (NORMAL)               | 1
 PRODUTOR RURAL          | OUTROS (NORMAL)                | 1
 PRODUTOR RURAL          | OUTROS (NORMAL)                | 1
 MERCADO ESQUINAO LTD    | OUTROS (NORMAL)                | 1
 NORIVAL                 | PRODUTOR RURAL (NORMAL)        | 2
 VIDAL                   | OUTROS (NORMAL)                | 1
```

Comandos LIKE e ILIKE

- O operador LIKE busca um padrão de texto utilizando coringas % (zero ou mais caracteres) e _ (exatamente um).

- Selecionar as cidades que começam com a letra “S”:

SELECT s_nome FROM t_clientes WHERE s_nome LIKE 'S%';

- Selecionar todas as cidades com 8 letras:

SELECT nome FROM cidades WHERE nome LIKE '_ _ _ _ _ _ _ _ ';

- Existe também o operador ILIKE que desconsidera a caixa dos caracteres (case insensitive).

Comando BETWEEN

- O operador BETWEEN é equivalente à >= e <=.

Vantagem de economia de código e legibilidade

```
SELECT s_codcliente, s_nome, d_lim_cred FROM t_cliente
WHERE d_lim_cred BETWEEN 1000 AND 2000;

SELECT numero, arrecadacao FROM concurso
WHERE d_lim_cred >= 1000 AND d_lim_cred <= 2000;
```

Utilizando o IN

- O operador IN verifica se um elemento pertence a um conjunto de dados.

- O seu uso pode reduzir o tamanho da query, quando comparamos com o uso do OR.

```
SELECT s_descricao FROM t_produto
WHERE d_unidade IN ('PC', 'UN', 'FD');
```

- Tem o mesmo significado da seguinte query:

```
SELECT s_descricao FROM t_produto
WHERE d_unidade = 'PC' OR
      d_unidade = 'UN' OR
      d_unidade = 'FD';
```

Comparação Agrupada

- É possível agrupar várias expressões de comparação em uma única expressão.

- Sintaxe:

(expr [, expr ...]) operador (expr [, expr ...])

- O primeiro SELECT é equivalente ao segundo:

```
SELECT * FROM t_produto
WHERE s_descricao = 'CUMBUCA PLAST.MP-36 CRISTAL-PCTE C/400 -MIL PLAST'
    AND d_unidade = 1;

SELECT * FROM t_produto
WHERE = (s_descricao , d_unidade) = ('CUMBUCA PLAST.MP-36 CRISTAL-PCTE C/400 -MIL
PLAST' , 1);
```

Comparação com NULL

- O resultado de qualquer expressão com NULL será sempre falso.
- Por exemplo:

coluna = NULLfalso
coluna > NULL falso
coluna != NULL falso
NULL = NULL falso

- Para lidar com expressões lógicas com colunas NULL, utilizamos o operador IS:

```
SELECT * FROM concurso
WHERE acertadores1 IS NULL;

SELECT * FROM concurso
WHERE acertadores1 IS NOT NULL;
```

A Cláusula ORDER BY

- A ordenação de saída de um SELECT é feita de acordo com a ordem dos dados na tabela. Para obtermos o resultado numa determinada ordem utilizamos a cláusula ORDER BY.
- O ORDER BY aceita como parâmetros uma lista de colunas (ou expressões baseadas em colunas) separadas por vírgula.
- Para cada parâmetro é possível estabelecer o critério de ordenação através do ASC (padrão) e do DESC.

SELECT * FROM cidades ORDER BY 3 DESC, nome;

```
codigo |           ome          | uf
-------+------------------------+----
     4 | Campinas               | SP
     3 | Osasco                 | SP
     1 | São Paulo              | SP
```

- Também é possível passar números referentes às colunas do SELECT.
- A partir da versão 8.3 também é possível definir a ordenação dos valores nulos, através da cláusula NULLS que tem como opção FIRST e LAST, isto é, FIRST ordena primeiro os nulos enquanto que LAST ordena os nulos por último.

```
ORDER BY expression [ ASC | DESC | USING operator ]
[ NULLS { FIRST | LAST } ] [, ...] ]
```

Capítulo 10

Selecionando Dados de Várias Tabelas

Apelidos de Tabelas

- Quando trabalhamos com várias tabelas, podem existir colunas que possuem o mesmo nome nas tabelas selecionadas. Isto gera conflito no momento de parsing do comando. Isto é resolvido de duas maneiras:

- Prefixando a coluna com o nome da tabela (trabalhoso em alguns casos)

- Através do uso de apelidos de tabelas

- Exemplo:

```
SELECT c.nome, e.nome
FROM cidades c, estados e
WHERE c.uf = e.uf;
```

Cruzando Dados entre Tabelas

- Na maior parte das consultas, é necessário selecionar dados de várias tabelas e associá-los através das chaves definidas na modelagem de dados.

  - No SQL, este processo de associar dados de várias tabelas se chama JOIN.

- O conceito por trás do JOIN é agrupar um ou mais conjuntos de dados que, quando agrupados, formam um novo conjunto de dados contendo as colunas das tabelas participantes.

  - A fundamentação do JOIN está no produto cartesiano entre todas as possíveis combinações entre os conjuntos de dados.

  - O conjunto resultante pode ser refinado através de critérios de seleção (cláusula WHERE) definidos na sintaxe do JOIN.

Tipos de JOINs

- Existem 4 tipos genéricos de JOINS:

  - CROSS JOIN: Cria um produto cartesiano entre conjuntos de dados das tabelas. Ele não cria relacionamento entre os conjuntos e retorna cada combinação possível entre as linhas.

  TODOS PARA TODOS

  - INNER JOIN: Cria um subconjunto de um produto cartesiano entre conjuntos de dados. Requer cláusulas condicionais para especificar um critério no qual dois registros são ligados. A cláusula deve retornar um valor lógico que determina se os registros serão relacionados.

  - OUTER JOIN: Similar ao INNER JOIN no que se refere à ligação lógica entre os dois conjuntos, entretanto, retorna no mínimo uma linha de um determinado conjunto. Pode ser tanto o conjunto esquerdo (LEFT OUTER JOIN), como o direito (RIGHT OUTER JOIN) ou ambos (FULL OUTER JOIN). A parte restante de uma relação é preenchida com NULL.

  - NATURAL JOIN: ele é similar ao INNER JOIN, entretanto utiliza como cláusula todas as colunas que possuam o mesmo nome nas tabelas envolvidas na consulta com a operação de igualdade.

Sintaxe

- Na prática, os tipos mais usuais de JOINS utilizados são o INNER e OUTER, pois eles requerem uma ligação qualificada entre os conjuntos de dados.

- Sintaxe:

  source1 join_type source2 [ON (condition [, ...]) | USING (column [, ...])]

  Onde:

  source1: identifica a primeira tabela ou sub-consulta

  join_type: [INNER] JOIN, LEFT JOIN, RIGHT JOIN ou FULL JOIN

  source2: identifica a segunda tabela ou sub-consulta

  ON ...: especifica o relacionamento entre colunas

  USING ...: especifica colunas com nomes iguais nas duas tabelas

Comparação entre JOINs

- Usando a cláusula WHERE:

```
SELECT a.aposta, a.data, an.numero
FROM aposta a, aposta_numeros an
WHERE a.aposta = an.aposta;
```

- Usando o INNER JOIN:

```
SELECT a.aposta, a.data, an.numero
  FROM aposta a
  INNER JOIN aposta_numeros an ON (a.aposta = an.aposta);

SELECT a.aposta, a.data, an.numero
  FROM aposta a
  INNER JOIN aposta_numeros an USING (aposta);
```

- Usando o NATURAL JOIN:

```
SELECT a.aposta, a.data, an.numero
FROM aposta a NATURAL JOIN aposta_numeros an;
```

- O resultado de todos é idêntico.

OUTER JOIN

- Suponha que desejamos obter uma lista completa dos agentes lotéricos e das apostas realizadas (usando o WHERE):

```
nome            |  aposta
----------------+--------
Cobra Ligeira   |       5
Coelho Vivo     |       3
Tiro Certo      |       2
Mira na Sorte   |       4
```

- Onde estão os outros agentes? Não foram recuperados, pois não existem apostas neles.

- Para resolver este problema, instruímos o gerenciador para considerar também os registros de uma tabela para os quais a outra tabela não tenha contrapartida, utilizando o OUTER JOIN.

- Usando o OUTER JOIN:

```
SELECT ag.nome, ap.aposta FROM agente ag
LEFT OUTER JOIN aposta ap ON ag.codigo = ap.agente;
```

```
          nome         | aposta
-----------------------+--------
 Pavão da Sorte        |
 Cobra Ligeira         |      5
 Coelho Vivo           |      3
 Tiro Certo            |      2
 Tiro Certo            |      1
 Alvo do Alvarez       |
 Mira na Sorte         |     4
 Tutu que Volta        |
 Dindin                |
 Roda da Fortuna       |
 Corrente da Sorte     |
 Cobra                 |
...
```

FULL OUTER JOIN

- O FULL OUTER JOIN é equivalente à soma do LEFT e RIGHT OUTER JOIN.

- Vamos inserir uma aposta sem estar relacionada a um agente:

```
INSERT INTO aposta VALUES (6, null, '2006-06-26', 1);
```

- Usando o FULL OUTER JOIN:

```
SELECT ag.nome, ap.aposta
FROM agente ag
FULL OUTER JOIN aposta ap
ON ag.codigo = ap.agente;
```

FULL OUTER JOIN

- Resultado:

nome	aposta
Pavão da Sorte	
Cobra Ligeira	5
Coelho Vivo	3
Tiro Certo	1
Tiro Certo	2
Alvo do Alvarez	
Mira na Sorte	4
	6
Tutu que Volta	
Dindin	
Roda da Fortuna	
Corrente da Sorte	
Cobra	
Pé de Coelho	
Ferradura de Coelho	
Só Sorte	
Casa da Sorte	
Pulo do Gato	
Dinheiro Certo	

Capítulo 11

Operadores

- Operadores são utilizados para operações/comparações entre valores, ou pares de valores.

- Tipos de Operadores:

  - Caractere
  - Numérico
  - Lógico

A lista de todos os operadores do PostgreSQL pode ser obtida com \do no psql.

Operadores de Texto

- Operadores de caracteres:

= expr1 = expr2

Retorna true se expr1 e expr2 são idênticas

!= expr1!= expr2

Retorna true se expr1 e expr2 não forem idênticas

<> expr1 <> expr2

O mesmo que o operador !=

<= expr1 <= expr2

Retorna true se expr1 pode ser alfabeticamente ordenada antes

de expr2, ou se são idênticas

>= expr1 >= expr2

Retorna true se expr1 pode ser alfabeticamente ordenada depois

de expr2, ou se são idênticas

Operadores de Texto

- Exemplos:

```
SELECT * FROM agente WHERE nome = 'Pavão da Sorte';
 codigo | cidade |      nome
--------+--------+----------------
      1 |      2 | Pavão da Sorte

SELECT * FROM agente WHERE nome > 'S';
 codigo | cidade |      nome
--------+--------+----------------
      4 |      5 | Tiro Certo
      7 |      7 | Tutu que Volta
     14 |      8 | Só Sorte

SELECT * FROM cidades WHERE uf <> 'SP';
 codigo |        nome        | uf
--------+--------------------+----
      5 | Belo Horizonte     | MG
      6 | Ouro Preto         | MG
      7 | Congonhas          | MG
      8 | Balneário Camboriú | SC
      9 | Florianópolis      | SC
     10 | Blumenau           | SC
     11 | Rio de janeiro     | RJ
     12 | Angra dos Reis     | RJ
```

- Concatenação de strings

O operador que realiza a concatenação de string é o ||.

- Exemplo:

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

         ?column?
-------------------------
 São Paulo - SP
 Sorocaba - SP
 Osasco - SP
 Campinas - SP
 Belo Horizonte - MG
 Ouro Preto - MG
 Congonhas - MG
 Balneário Camboriú - SC
 Florianópolis - SC
 Blumenau - SC
 Rio de janeiro - RJ
 Angra dos Reis - RJ
```

Expressões Regulares

- Utilizando expressões regulares

~ string ~ expressao

Retorna true se a expressão casa

!~ string !~ expressao

Retorna true se a expressão não casa

~* string ~* expressao

Retorna true se a expressão casa (case insensitive)

!~* string !~* expressao
Retorna true se a expressão não casa (case insensitive)

O uso de expressões regulares é especialmente útil em aplicações que necessitam de manipulações complexas de texto.

Operadores com Expressões Regulares

~ string ~ 'expr' : true se a expressão casa

!~ string !~ 'expr' : true se a expressão não casa

~* string ~* 'expr' : true se a expressão casa (case insensitive)

!~* string !~* 'expr' : true se a expressão não casa (case insensitive)

^ string ~ '^expr' : true se a expressão começa com 'expr'

$ string ~ 'expr$' : true se a expressão acaba com 'expr'

[] string ~ [abc] : true se contem algum dos caracteres

[-] string ~ [0-9] : true se contem algum dos caracteres do intervalo

[^-] string ~ ^[a-z] : true se a string começa com algum dos caracteres do intervalo

| expr | expr : true se alguma das expressões regulares é verdadeira

Operadores com Expressões Regulares

- Registros que contenham a string ‘Paulo‘ no seu conteúdo:

```
SELECT * FROM cidades WHERE nome ~ 'Paulo';

 codigo |   nome    | uf
--------+-----------+----
      1 | São Paulo | SP
```

- Registros que contenham a string ‘campinas‘ (sem considerar a caixa):

SELECT * FROM cidades WHERE nome ~* 'Campinas';

```
 codigo |   nome   | uf
--------+----------+----
      4 | Campinas | SP
```

- Registros que comecem por ‘S' ou ‘s':

```
SELECT * FROM cidades WHERE nome ~* '^S';

 codigo |   nome    | uf
--------+-----------+----
      1 | São Paulo | SP
      2 | Sorocaba  | SP
```

Operadores com Expressões Regulares

- Mais exemplos:

- Registros que comecem com 'A' ou ‘R':

```
SELECT * FROM cidades WHERE nome ~ '^[AR]';

 codigo |        nome        | uf
--------+--------------------+----
     11 | Rio de janeiro     | RJ
     12 | Angra dos Reis     | RJ
```

  - Registros que comecem por vogais, maiúsculas ou minúsculas:

```
SELECT * FROM cidades WHERE nome ~* '^[aeiou]';

 codigo |        nome         | uf
--------+---------------------+----
      3 | Osasco              | SP
      6 | Ouro Preto          | MG
     12 | Angra dos Reis      | RJ
```

Operadores Matemáticos

- Os operadores matemáticos podem ser utilizados na cláusula WHERE de um comando SELECT ou em qualquer lugar em que um resultado numérico seja possível.

Operador Uso Descrição

+	a+b	Adição
-	a-b	Subtração
*	a*b	Multiplicação
/	a/b	Divisão
%	a%b	Resto da divisão de a por b
^	a^b	Exponencial (a elevado a b)
|/	|/a	Raiz quadrada de a
||/	||/a	Raiz cúbica de a
!	a!	Fatorial de a
@	@a	Valor absoluto de a

Conversão de Tipos

- Utilizamos o operador “::" quando precisamos converter o tipo de um determinado campo.

- O SELECT abaixo executa uma operação (numérica) mas o valor de retorno é convertido para caracter.

```
SELECT (111+1)::varchar;
```

- O SELECT abaixo concatena duas strings, converte o resultado para número e então executa uma soma:

```
SELECT (('11'||'22')::integer)+1;
```

- O operador “::” não está no padrão SQL, entretanto existe a função CAST que também converte tipos e encontra-se no padrão SQL92.

- Os dois comandos abaixo tem o mesmo efeito:

```
SELECT (111+1)::varchar;

SELECT CAST ((111+1) AS varchar); -- padrão SQL
```

Capítulo 12

Funções

- Uma função é um identificador que instrui o PostgreSQL a realizar uma operação dentro do comando SQL.

- O PostgreSQL possui diversos tipos de funções pré-definidas:

Funções matemáticas
Manipulação de data e hora
Manipulação de string
Conversão de tipos
Funções de sistema
Funções agregadas

Funções Matemáticas

- Funções matemáticas mais usuais:

abs(x) valor absoluto
ceil(x) menor inteiro maior ou igual ao argumento
floor(x) maior inteiro menor ou igual ao argumento
cos(x) cosseno
sin(x) seno
log(x) logaritmo de x na base 10
mod(x,y) resto inteiro da divisão de x por y
pow(x,y) eleva o número x à potência y
random() retorna um número aleatório entre 0.1 e 1.0
round(x [,y]) arredonda para a precisão informada
sign(x) retorna o sinal do argumento (-1, 0, 1)
sqrt(x) raiz quadrada
trunc(x [,y]) trunca o número até a casa decimal especificada

- Funções Matemáticas (exemplos):

```
SELECT pi(), log(53);
        pi         |        log
-------------------+-------------------
 3.14159265358979 | 1.72427586960079


SELECT sin(30), pow(2, 5);
        sin         | pow
--------------------+-----
 -0.988031624092862 |  32


SELECT random(), random(), random();
    random      |      random      |      random
----------------+------------------+----------------
0.449414308857 | 0.842947382938 | 0.260392530291
```

Exemplos de Funções

- Funções matemáticas:

```
SELECT 9.4,ceil(9.4),round(9.4),sqrt(9.4);

?column?  | ceil | round | sqrt
----------+------+-------+------------------
      9.4 |   10 |     9 | 3.06594194335118
```

- Também podemos utilizar “funções aninhadas”:

```
SELECT trunc(sqrt(floor(9.4)),2)^2;

?column?
--------------------
9.0000000000000000
```

Funções de Data e Hora

- Funções de manipulação de data e hora mais usuais:

current_date	retorna a data atual no formato DATE
current_time	retorna a data atual no formato TIME
current_timestamp	retorna a data atual no formato TIMESTAMP
to_char(dt, masc)	converte a data para a máscara dada
date_part(s,t)	retorna a data ou hora t no formato s
date_trunc(s,t)	retorna a data ou hora t truncada no nível s
now()	retorna data e hora no formato TIMESTAMP
timeofday()	retorna data e hora no formato texto
age(dt, dt)	calcula o intervalo de tempo entre as datas

- Argumentos do date_part e date_trunc:

day	century	hour
minute	month	second
millisecond	week	year

Máscaras de Data e Hora

- Formato de máscaras de data e hora mais usuais:

HH	hora do dia (01-12)
HH12	hora do dia (01-12)
HH24	hora do dia (00-23)
MI	minuto (00-59)
SS	segundo (00-59)
MS	milisegundo (000-999)
US	microsegundo (000000-999999)
SSSS	segundo depois da meia-noite (0-86399)
AM ou PM	indicador do meridiano
YYYY	ano
YY	ano 2 últimos dígitos
Y	último dígito
BC ou AD	indicador de era
MONTH	nome completo do mês (maiúsculo)
MM	mês (01-12)
DAY	nome completo do dia (maiúsculo)
DD	dia do mês (01-31)
D	dia da semana (1-7, domingo=1)
DDD	dia do ano (1-366)
WW	semana do ano (1-53)

Usando Funções de Data e Hora

- Exemplo de funções de data e hora:

```
SELECT current_date + 1;

    date
------------
 2003-03-08

SELECT now();

              now
-------------------------------
 2003-03-08 12:01:32.057475-03

SELECT date_trunc('hour', now());

       date_trunc
-------------------------------
 2006-06-08 12:00:00-03

SELECT to_char(now(), 'DD-MM-YY HH24:MI:SS');

     to_char
 -------------------
 08-06-06 12:13:22
```

Usando Funções de Data e Hora

- Exemplo de funções de data e hora:

```
    SELECT aposta, to_char(age(now(), data), 'DD "dias" HH "horas e" MI
"minutos"')
    FROM aposta;

     aposta |            to_char
    --------+-------------------------------
          1 | 08 dias 02 horas e 40 minutos
          2 | 08 dias 02 horas e 39 minutos
          3 | 08 dias 02 horas e 39 minutos

    SELECT current_date - '1 month'::interval;

          ?column?
    ---------------------
     2006-06-05 00:00:00


    SELECT current_date - '1 month 8 hours'::interval;

          ?column?
    ---------------------
     2006-06-04 16:00:00
```

Funções de Texto

- Funções de manipulação de texto mais usuais:

ascii(s)	retorna o código ASCII do caracter s
trim(s [,t])	elimina o caracter t (ou branco) no início e no final da string s
chr(n)	retorna o caracter correspondente ao número n
initcap(s)	torna maiúscula a primeira letra de cada palavra da string s
length(s)	comprimento da string s
lower(s)	retorna a string s em minúsculo
upper(s)	retorna a string s em maiúsculo
overlay(s placing b from x for y)	substitui substring da posição x com o comprimento y por b
position(b in s)	retorna a posição da string b na string s
repeat(s,n)	repete o caracter s n vezes

- Funções de manipulação de texto mais usuais:

lpad(s, n, b)	preenche a esquerda a string s com a substring b até o comprimento n
rpad(s, n, b)	preenche a direita a string s com a substring b até o comprimento n
split_part(s, d, n)	divide a string s pelo delimitador d e devolve o campo da coluna n
strpos(s, b)	retorna a posição da substring b na string s
substr(s,n,l)	retorna a string s começando em n e com comprimento l
translate(s, o, n)	retorna a string s substituindo as ocorrências de o por n

Funções de Texto

- Funções de manipulação de strings:

```
SELECT length(nome), upper(nome) FROM agente ORDER BY 1;

 length |         upper
--------+----------------------
      5 | COBRA
      6 | DINDIN
...

SELECT substr(nome, 1, 3) FROM estados ;

 substr
--------
 São
 Rio
...

SELECT translate(nome,  '444331100775522')
 FROM cidades ;
     translate
--------------------
 540 P4ul0
 50r0c4b4
```

Funções de Texto

- Funções de manipulação de strings:

```
SELECT split_part('Curso:PostgreSQL:Essencial', ':', 2);

 split_part
------------
 PostgreSQL

SELECT trim('*******Linux*******', '*');

 btrim
-------
 Linux

SELECT nome, lpad(nome, 20, ' ')
 FROM agente;

        nome          |          lpad
----------------------+----------------------
 Pavão da Sorte       |        Pavão da Sorte
 Cobra Ligeira        |         Cobra Ligeira
 Coelho Vivo          |           Coelho Vivo
...
```

Funções de Conversão de Tipos

- O PostgreSQL oferece funções específicas para conversão de tipos de dados:

to_char(dt, masc)	converte data para texto conforme masc
to_char(num, masc)	converte número para texto conforme masc
to_timpestamp(str, masc)	converte texto para data conforme masc
to_number(str, masc)	converte texto para número conforme masc

- Máscaras para conversão numérica:

9	Valor com o número determinado de dígitos
0	Valor preenchendo início com zeros
D	Separador decimal
G	Separador de milhar
RN	Converte para romano
S	Sinal negativo
SG	Sinal negativo ou positivo

Funções de Conversão de Tipos

- Exemplos:

```
SELECT numero, to_char(data, 'dd/mm/yyyy')
 FROM concurso;
 numero |  to_char
--------+------------
      1 | 27/06/2006
      2 | 04/07/2006

SELECT numero, to_char(arrecadacao, '000G000G000D99SG')
 FROM concurso;
 numero |     to_char
--------+-----------------
      1 | 000,000,004,00+
      2 | 000,000,000,00+

SELECT    numero,    to_char(numero,    'RN')    FROM
concurso_numeros;
 numero |     to_char
--------+-----------------
     10 |               X
     12 |             XII
     11 |              XI
```