**Apostila de SQL**

1

**1. Introdução**

A primeira versão da linguagem SQL, chamada SEQUEL (Structured Query English Language), surgiu em 1974 nos laboratórios da IBM (Califórnia). Entre 1976 e 1977 ela foi revisada e ampliada, tendo então o seu nome alterado para SQL.

Devido ao sucesso da nova forma de consulta e manipulação de dados dentro de um ambiente de banco de dados, sua utilização tornou-se cada vez maior. Vários SGBD’s atuais utilizam o SQL como a linguagem padrão para o acesso

às bases de dados. Entre eles podemos citar:

DB2 da IBM

ORACLE da Oracle Corporation; RDB da Digital

SYBASE da Sybase INC SQL Server da Microsoft

Ingres da Computer Associates

Em 1982 o American National Standard Institute (ANSI) tornou a SQL a linguagem padrão para a manipulação de dados em ambiente relacional.

**2. A Linguagem SQL**

A linguagem SQL pode ter vários enfoques:

**Linguagem interativa de consulta (query AdHoc)**

Através de comandos SQL os usuários podem montar consultas poderosas, sem a necessidade da criação de um programa, podendo utilizar ferramentas front-end para a montagem de relatórios.

**Linguagem de programação para acesso às bases de dados**

Comandos SQL embutidos em programas de aplicação (escritos em C, C++, Java, Visual Basic e etc) acessam os dados armazenados em uma base de dados relacional.

**Linguagem de administração de banco de dados**

O responsável pela administração do banco de dados (DBA) pode utilizar comandos SQL para realizar tarefas relacionadas com a manutenção dos schemas do banco de dados.

**Linguagem de consulta em ambiente cliente/servidor**

Os programas sendo processados nos computadores dos clientes (front ends) usam comandos SQL para se comunicarem, através de uma rede, com um SGBD sendo processado em uma máquina servidora (back end);

**Linguagem para bancos de dados distribuídos**

A linguagem SQL é também a linguagem padrão para a manipulação de dados em uma base de dados distribuída.

**Linguagem de definição de dados (DDL)**

Permite ao usuário a definição da estrutura e organização dos dados armazenados, e das relações existentes entre eles.

**Linguagem de manipulação de dados (DML)**

Permite a um usuário, ou a um programa de aplicação, a execução de operações de inclusão, remoção, seleção ou atualização de dados previamente armazenados na base de dados.

**Controle de acesso**

Protege os dados de manipulações não autorizadas.

**Integridade dos dados**

Auxilia no processo de definição da integridade dos dados, protegendo contra corrupções e inconsistências geradas por falhas do sistema de computação, ou por erros nos programas de aplicação.

**3. Vantagens e Desvantagens da Linguagem SQL**

Podemos apontar as seguintes vantagens no uso da linguagem SQL:

**Independência de fabricante**

A linguagem SQL é adotada por praticamente todos os SGBD’s relacionais existentes no mercado, além de ser uma linguagem padronizada (ANSI). Com isso, pelo menos em tese, posso mudar de SGBD sem me preocupar em alterar os programas de aplicação.

**Portabilidade entre plataformas de hardware e software**

Pode ser utilizada tanto em máquinas Intel rodando Windows, passando por workstations RISC rodando UNIX, até mainframes rodando sistemas operacionais proprietários.

**Redução dos custos com treinamento**

Com base no item anterior, as aplicações podem se movimentar de um ambiente para o outro sem que seja necessária uma reciclagem da equipe de desenvolvimento.

**Usa inglês estruturado de alto nível**

O SQL é formado por um conjunto bem simples de sentenças em inglês, oferecendo um rápido e fácil entendimento.

**Permite consultas interativas**

Permite aos usuários acesso fácil e rápido aos dados a partir de um front end que permita a edição e a submissão de comandos SQL.

**Múltiplas visões dos dados**

Permite ao criador do banco de dados levar diferentes visões dos dados aos diferentes usuários.

**Definição dinâmica dos dados**

Através da linguagem SQL pode-se alterar, expandir ou incluir, dinamicamente, as estruturas dos dados armazenados, com máxima flexibilidade.

Porém, existem também algumas desvantagens no uso da linguagem SQL: Críticas (segundo C.J. Date)

Falta de ortogonalidade nas expressões, funções embutidas, variáveis indicadoras, referência a dados correntes, constante NULL, conjuntos vazios, e etc;

Definição formal da linguagem após sua criação;

Discordância com as linguagens hospedeiras (geralmente procedurais e orientadas para registros e não para conjuntos);

Falta de algumas funções;

Não dá suporte a alguns aspectos do modelo relacional (join explícito, domínios, e etc.)

**4. Tabelas dos Exemplos**

As tabelas a seguir serão usadas nos exemplos que se seguem:

N

N

N

N

1

PRODUTO

ITEM-PEDIDO

PEDIDO

VENDEDOR

1

CLIENTE

N

N

N

N

1

PRODUTO

ITEM-PEDIDO

PEDIDO

VENDEDOR

N

VENDEDOR

PEDIDO

ITEM-PEDIDO

N

1

PRODUTO

N

N

1

1

1

CLIENTE

1

CLIENTE (cpf\_cli, nome\_cli, endereco, cidade, cep, uf) VENDEDOR (cpf\_vend, nome\_vend, sal\_fixo, faixa\_comiss) PEDIDO ( num\_ped, prazo\_entr, cpf\_cli, cpf\_vend) ITEM\_PEDIDO (no\_ped, cd\_prod, qtd\_ped, seq-item)

PRODUTO (cod\_prod, unid\_prod, desc\_prod, val\_unit)

DICIONÁRIO DE DADOS

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | CLIENTE |  |  |  |  |  |  |  |  |
|  | **atributo** | **pk (ID)** | **fk (ce)-tabela** | **tipo** | **obrigat.** | **tamanho** | **decimais** | **dominio** | **obser. (regra de negócio)** |
|  | Nome-cli |  |  | varchar | s | 80 |  | texto | Exceto caracteres especiais |
|  | Endereço |  |  | varchar | n | 90 |  | texto |  |
|  | Dt-nasc |  |  | date | S |  |  |  | <= Data hj |
|  | Sexo |  |  | varchar | s | 1 |  |  | M=masculino, F=feminino |
|  | cpf-cli | S |  | double | S | ? |  | Receita Federal |  |
|  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | VENDEDOR |  |  |  |  |  |  |  |  |
|  | **atributo** | **pk** | **fk (ce)-tabela** | **tipo** | **obrigat.** | **tamanho** | **decimais** | **dominio** | **obser.** |
|  | Nome |  |  | varchar | s | 80 |  | texto | exceto caract especiais |
|  | Cpf-vend | S |  | double | s |  |  | numerico |  |
|  | tx-comissao |  |  | numeric | s | 8 | 2 | numerico |  |
|  | Sexo |  |  | varchar | s | 1 |  |  | M=masculino, F=feminino |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | PEDIDO |  |  |  |  |  |  |  |  |
|  | **atributo** | **pk** | **fk (ce)-tabela** | **tipo** | **obrigat.** | **tamanho** | **decimais** | **dominio** | **obser.** |
|  | Num-pedido | S |  | integer | s |  |  |  |  |
|  | Dt-pedido |  |  | Date | s |  |  | calendario | QQ data valida acima de 20-10-2013 |
|  | End-entrega |  |  | varchar | n | 90 |  |  |  |
|  | Dt-entrega |  |  | Date | s |  |  | calendario |  |
|  | cpf-cliente |  | Cliente | double | s |  |  | receita Federal |  |
|  | cpf-vend |  | Vendedor | double | s |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
|  | ITEM-PEDIDO |  |  |  |  |  |  |  |  |
|  | **atributo** | **pk** | **fk (ce)-tabela** | **tipo** | **obrigat.** | **tamanho** | **decimais** | **dominio** | **obser.** |
|  | Num-ped | s | Pedido | integer | s |  |  |  |  |
|  | Seq-item | s |  | integer | s |  |  |  |  |
|  | Cod-prod |  | Produto | text | s | 10 |  |  |  |
|  | Qtde-pedida |  |  | numeric | s | 6 | 2 |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  | PRODUTO |  |  |  |  |  |  |  |  |
|  | **atributo** | **pk** | **fk (ce)-tabela** | **tipo** | **obrigat.** | **tamanho** | **decimais** | **dominio** | **obser.** |
|  | Cod-produto | s |  | text | s | 10 |  |  |  |
|  | Desc-produto |  |  | text | s | 50 |  |  |  |
|  | Val-produto |  |  | numeric | s | 10 | 2 |  |  |
|  | Unid-Medida |  |  | text | s | 10 |  |  |  |

**5. Criação e Destruição de Tabelas**

O comando CREATE TABLE é usado para criar uma tabela. A sua forma geral é:

CREATE TABLE <nome\_tabela> (<descrição das colunas>,

<descrição das chaves>);

onde:

**<nome\_tabela>** dever ser substituído pelo nome da tabela a ser criada.

**<descrição das colunas>** deve ser substituída pela relação das colunas da tabela e seus respectivos tipos de dados (por exemplo, smallint, char, varchar, integer, number, float e etc).

**<descrição das chaves>** deve ser substituída pela lista das colunas que são tratadas como chaves estrangeiras.

Algumas colunas podem receber o valor NULL (nulo), e a coluna definida como chave primária, além de não poder receber NULL, deve ser uma coluna UNIQUE (sem repetições; isto é, chave primária)

**Script de Criação das Tabelas dos Exemplos**

create table cliente

(

cod\_cli smallint not null, nome\_cli varchar(40) not null, endereco varchar(40) null, cidade varchar(20) null, cep char(08) null, uf char(02) null, primary key (cod\_cli));

create table vendedor

(

cod\_vend smallint not null, nome\_vend varchar(40) not null, sal\_fixo number(9,2) not null,

faixa\_comiss char(01) not null, primary key (cod\_vend));

create table produto

(

cod\_prod smallint not null, unid\_prod char(03) not null, desc\_prod varchar(20) not null, val\_unit number(9,2) not null, primary key (cod\_prod));

create table pedido

(

num\_ped smallint not null, prazo\_entr smallint not null, cd\_cli smallint not null

REFERENCES CLIENTE (cod\_cli),

cd\_vend smallint not null

REFERENCES VENDEDOR (cod\_vend), primary key (num\_ped));

create table item\_pedido

(

no\_ped smallint not null

REFERENCES PEDIDO (num\_ped),

cd\_prod smallint not null

REFERENCES PRODUTO (cod\_prod),

qtd\_ped float not null);

Para excluirmos uma tabela existente devemos usar o comando DROP TABLE. A sua forma geral é:

DROP TABLE <nome\_tabela>;

onde:

**<nome\_tabela>** dever ser substituído pelo nome da tabela a ser excluída.

**Exemplos**

drop table item\_pedido;

drop table pedido; drop table vendedor; drop table produto; drop table cliente;

**6. Executando Consultas sobre as Tabelas**

**6.1 Selecionando Colunas Específicas de uma Tabela**

SELECT <lista\_de\_colunas> FROM <nome\_tabela>;

Problema:

Listar todos os produtos com as respectivas descrições, unidades e valores unitários.

**select desc\_prod,unid\_prod,val\_unit from produto;**

DESC\_PROD UNI VAL\_UNIT

-------------------- --- ---------- Chapa de Aco kg 2,5

Cimento kg 4,5 parafuso 3.0X10.5 mm kg 2

Fio plastico m ,2

Solvente PRW l 5

5 linhas selecionadas.

Problema:

Listar os nomes dos clientes, as cidade e os estados onde eles estão localizados.

**select nome\_cli,cidade,uf from cliente;**

NOME\_CLI CIDADE UF

---------------------------------------- -------------------- --

|  |  |  |  |
| --- | --- | --- | --- |
| Supermercado | Carrefour | rio de janeiro | rj |
| Supermercado | Baratao | rio de janeiro | rj |
| Supermercado | Arariboia | niteroi | rj |
| UFF |  | niteroi | rj |
| CSN |  | volta redonda | rj |
| Pegeout |  | resende | rj |
| Ind. Quimicas Paulistas | | sao paulo | sp |
| Ford Caminhoes | | sao paulo | sp |
| Riocel Celulose | | guaiba | rs |
| Elevadores Sur | | guaiba | rs |

**6.2 Selecionando todas as Colunas de uma Tabela**

SELECT \*

FROM <nome\_tabela>;

Problema:

Listar o conteúdo de todas as tabelas da base de dados dos exemplos.

**select \* from cliente;**

COD\_CLI NOME\_CLI ENDERECO CIDADE CEP UF

---------- ---------------------------------------- ---------------------

------------------- -------------------- -------- --

1000 Supermercado Carrefour Av. das Americas rio de janeiro 20000001 rj

2000 Supermercado Baratao Rua Rolando Lero rio de janeiro 20000002 rj

3000 Supermercado Arariboia Rua Itaoca niteroi 20000003 rj

4000 UFF Cidade Univers. niteroi 20000004 rj

5000 CSN Rua do Aco

volta redonda 20000005 rj

6000 Pegeout Rodovia Pres. Dutra resende 20000006 rj

7000 Ind. Quimicas Paulistas Rua Tuiuti sao paulo 11000001 sp

8000 Ford Caminhoes Rua Henry Ford sao paulo 11000002 sp

9000 Riocel Celulose Rua Gen. Arouca guaiba 30000001 rs

10000 Elevadores Sur Rua Planejada guaiba 30000001 rs

**select \* from produto;**

COD\_PROD UNI DESC\_PROD VAL\_UNIT

---------- --- -------------------- ----------

100 kg Chapa de Aco 2,5

200 kg Cimento 4,5

300 kg parafuso 3.0X10.5 mm 2

|  |  |  |  |
| --- | --- | --- | --- |
| 400 | m | Fio plastico | ,2 |
| 500 | l | Solvente PRW | 5 |

**select \* from vendedor;**

COD\_VEND NOME\_VEND SAL\_FIXO F

---------- ---------------------------------------- ---------- -

11 Paulo Alberto 1500 b

12 Ana Cristina 2100 a

13 Cassia Andrade 900 c

14 Armando Pinto 2500 a

15 Maria Paula 900 c

**select \* from pedido;**

|  |  |  |  |
| --- | --- | --- | --- |
| NUM\_PED | PRAZO\_ENTR | CD\_CLI | CD\_VEND |
| ---------- | ---------- | ---------- | ---------- |
| 1111 | 10 | 1000 | 11 |
| 1112 | 5 | 1000 | 11 |
| 1113 | 30 | 1000 | 15 |
| 2111 | 15 | 3000 | 14 |
| 2112 | 18 | 3000 | 15 |
| 2113 | 3 | 3000 | 12 |
| 3111 | 13 | 4000 | 12 |
| 3112 | 7 | 4000 | 11 |
| 4111 | 7 | 6000 | 11 |
| 4112 | 7 | 6000 | 14 |
| 5111 | 10 | 8000 | 14 |
| 6111 | 30 | 9000 | 14 |
| 6112 | 60 | 9000 | 12 |
| 7111 | 20 | 10000 | 15 |

**select \* from item\_pedido;**

|  |  |  |
| --- | --- | --- |
| NO\_PED | CD\_PROD | QTD\_PED |
| ---------- | ---------- | ---------- |
| 1111 | 100 | 100 |
| 1111 | 200 | 100 |
| 1111 | 300 | 200 |
| 1112 | 400 | 100 |
| 1112 | 500 | 1000 |
| 1113 | 100 | 300 |
| 2111 | 100 | 500 |
| 2111 | 500 | 400 |
| 2112 | 200 | 100 |
| 2112 | 300 | 200 |
| 2113 | 500 | 500 |
| 3111 | 400 | 300 |
| 3112 | 100 | 400 |
| 3112 | 200 | 600 |
| 4111 | 300 | 700 |
| 4112 | 500 | 1000 |
| 4112 | 500 | 500 |
| 5111 | 200 | 100 |
| 5111 | 300 | 500 |
| 6111 | 400 | 100 |
| 6112 | 300 | 400 |
| 6112 | 400 | 200 |
| 7111 | 100 | 500 |

**6.3 Selecionando Apenas Alguns Registros da Tabela**

SELECT <lista\_de\_colunas> FROM <nome\_tabela>

WHERE <condição\_de\_seleção>;

Onde a cláusula WHERE tem a seguinte forma:

**WHERE <nome\_da\_coluna> <operador> <valor>**

6.3.1 Operadores Relacionais:

= igual

<> ou != diferente

< menor que

> maior que

>= maior ou igual a

<= menor ou igual a

Quando a coluna é do tipo **caracter**, o <valor> deve estar entre aspas simples

(').

Exemplo: 'parafuso'

**Observação:**

Na linguagem SQL existe diferença entre caracteres maiúsculos e minúsculos;

logo, 'PARAFUSO' é diferente de 'parafuso'.

Problema:

Listar o número do pedido, o código do produto e a quantidade pedida dos itens de um pedido, onde a quantidade pedida seja igual a 500.

**select no\_ped,cd\_prod,qtd\_ped from item\_pedido**

**where qtd\_ped = 500;**

|  |  |  |
| --- | --- | --- |
| NO\_PED | CD\_PROD | QTD\_PED |
| ---------- | ---------- | ---------- |
| 2111 | 100 | 500 |
| 2113 | 500 | 500 |
| 4112 | 500 | 500 |
| 5111 | 300 | 500 |
| 7111 | 100 | 500 |

5 linhas selecionadas.

Problema:

Quais são os clientes localizados na cidade de Niterói?

**select nome\_cli from cliente**

**where cidade = 'niteroi';**

NOME\_CLI

---------------------------------------- Supermercado Arariboia

UFF

2 linhas selecionadas.

6.3.2 Operadores Lógicos

AND conjunção OR disjunção NOT negação

Problema:

Quais são os produtos que têm unidade igual a 'kg' e valor unitário maior do que R$ 2,00?

**select desc\_prod from produto**

**where unid\_prod = 'kg' and val\_unit > 2.00;**

DESC\_PROD

-------------------- Chapa de Aco

Cimento

2 linhas selecionadas.

Problema:

Liste todos os clientes localizados na cidade de São Paulo ou que tenham CEP

entre 20000005 e 20000010.

**select nome\_cli,cidade,cep from cliente**

**where cidade = 'sao paulo' or (cep>='20000005' and cep<='20000010');**

NOME\_CLI CIDADE CEP

---------------------------------------- -------------------- -------- CSN volta redonda 20000005

Pegeout resende 20000006

Ind. Quimicas Paulistas sao paulo 11000001

Ford Caminhoes sao paulo 11000002

4 linhas selecionadas.

**Observação:**

A prioridade do operador AND é maior do que a prioridade do operador OR;

logo, neste exemplo, a utilização dos parênteses é opcional.

Problema:

Mostrar todos os pedidos que não tenham prazo de entrega superior a 15 dias.

**select num\_ped from pedido**

**where not (prazo\_entr > 15);**

NUM\_PED

----------

1111

1112

2111

2113

3111

3112

4111

4112

5111

9 linhas selecionadas.

**VEJAM FUNÇÕES ESPECIFICAS PARA STRINGS, DATA OUTRAS NOS APÊNDICES NO FINAL DA APOSTILA**

6.3.3 Operadores BETWEEN e NOT BETWEEN

**WHERE <nome\_coluna> BETWEEN <valor1> AND <valor2> WHERE <nome\_coluna> NOT BETWEEN <valor1> AND <valor2>**

Este operador possibilita a seleção de uma faixa de valores sem a necessidade do uso dos operadores >=, <= e AND.

<valor1> e <valor2> têm que ter o mesmo tipo de dado que <nome\_coluna>.

Problema:

Liste o código e a descrição dos produtos que tenham o valor unitário na faixa de R$ 0,10 a R$ 3,00.

**select cod\_prod,desc\_prod from produto**

**where val\_unit between 0.10 and 3.00;**

COD\_PROD DESC\_PROD

---------- --------------------

100 Chapa de Aco

300 parafuso 3.0X10.5 mm

400 Fio plastico

3 linhas selecionadas.

6.3.4 Operadores LIKE e NOT LIKE **WHERE <nome\_coluna> LIKE <valor> WHERE <nome\_coluna> NOT LIKE <valor>**

Aplicáveis apenas a colunas dos tipos CHAR e VARCHAR.

Funcionam de modo análogo aos operadores = e <>, porém o poder dos operadores LIKE e NOT LIKE está na utilização dos símbolos **%** e **\_**, que podem fazer o papel de “coringa”:

% substitui uma palavra

\_ substitui um caracter qualquer

Exemplos:

'apis%' se aplicaria às seguintes cadeias de caracteres:

'lapis preto'

'lapis cera'

'lapis borracha'

'broca n\_' se aplicaria às seguintes cadeias de caractere:

'broca n1'

'broca n9'

'broca n3'

Problema:

Listar todos os produtos que tenham a sua unidade começando por k (lembre- se de que a coluna unid\_prod foi definida como char(03)).

**select cod\_prod,desc\_prod from produto**

**where unid\_prod like 'k ';**

COD\_PROD DESC\_PROD

---------- --------------------

100 Chapa de Aco

200 Cimento

300 parafuso 3.0X10.5 mm

3 linhas selecionadas.

Problema:

Listar todos os vendedores cujos os nome não comecem por 'A'.

**select cod\_vend,nome\_vend from vendedor**

**where nome\_vend not like 'A%';**

COD\_VEND NOME\_VEND

---------- ----------------------------------------

11 Paulo Alberto

13 Cassia Andrade

15 Maria Paula

3 linhas selecionadas.

6.3.5 Operadores IN e NOT IN

**WHERE <nome\_coluna> IN <lista\_de\_valores> WHERE <nome\_coluna> NOT IN (<lista\_de\_valores>)**

Seleciona as linhas cujo o valor da coluna <nome\_coluna> pertença ao conjunto <lista\_de\_valores>.

Problema:

Listar todos os vendedores cujas as faixas de comissão sejam 'a' ou 'b'.

**select cod\_vend,nome\_vend from vendedor**

**where faixa\_comiss in ('a','b');**

COD\_VEND NOME\_VEND

---------- ----------------------------------------

11 Paulo Alberto

12 Ana Cristina

14 Armando Pinto

3 linhas selecionadas.

6.3.6 Operadores IS NULL e IS NOT NULL **WHERE <nome\_coluna> IS NULL WHERE <nome\_coluna> IS NOT NULL**

A utilização do valor nulo (NULL) é problemática, pois as diversas implementações da linguagem SQL podem adotar qualquer representação para o valor nulo.

Problema:

Mostrar os clientes que não tenham endereço cadastrado.

**select nome\_cli from cliente**

**where endereco is null;**

NOME\_CLI

----------------------------------------

0 linhas selecionadas.

**6.4 Ordenando os Dados Selecionandos**

SELECT <lista\_de\_colunas> FROM <nome\_tabela>

WHERE <condição\_de\_seleção>

ORDER BY {<nome\_coluna>|<num\_col> [ASC|DESC]}

Onde <nome\_coluna> se refere à coluna segundo a qual as linhas serão ordenadas, e <num\_col> se refere à posição relativa da coluna na

<lista\_de\_colunas> projetadas, contada da esquerda para a direita, e não à posição na tabela original.

As cláusulas ASC e DESC denotam ordenação ascendente e descendente

respectivamente. A forma ascendente de ordenação é assumida caso nenhuma opção seja informada explicitamente.

Problema:

Mostrar em ordem alfabética a lista de vendedores e seus respectivos salários fixos.

**select nome\_vend,sal\_fixo from vendedor**

**order by nome\_vend;**

NOME\_VEND SAL\_FIXO

---------------------------------------- ---------- Ana Cristina 2100

Armando Pinto 2500

Cassia Andrade 900

Maria Paula 900

Paulo Alberto 1500

5 linhas selecionadas.

Problema:

Listar os nomes, as cidades e os estados de todos os clientes, ordenados por estado e cidade de forma descendente.

**select nome\_cli,cidade,uf from cliente**

**order by uf desc,cidade desc;**

NOME\_CLI CIDADE UF

---------------------------------------- -------------------- -- Ind. Quimicas Paulistas sao paulo sp Ford Caminhoes sao paulo sp Riocel Celulose guaiba rs Elevadores Sur guaiba rs CSN volta redonda rj Supermercado Carrefour rio de janeiro rj Supermercado Baratao rio de janeiro rj Pegeout resende rj Supermercado Arariboia niteroi rj UFF niteroi rj

10 linhas selecionadas.

Problema:

Mostrar a descrição e o valor unitário de todos os produtos que tenham unidade 'kg' em ordem ascendente de valor unitário.

**select desc\_prod,val\_unit from produto**

**where unid\_prod = 'kg' order by 2;**

DESC\_PROD VAL\_UNIT

-------------------- ---------- parafuso 3.0X10.5 mm 2

Chapa de Aco 2,5

Cimento 4,5

3 linhas selecionadas.

**6.5 Realizando Cálculos sobre a Informação Selecionada**

Podemos criar dinamicamente um campo que não pertença à tabela original através de operações executadas sobre os campos projetados.

Problema:

Exibir o novo salário fixo dos vendedores da faixa de comissão ‘C’, calculado com base no reajuste de 75% sobre o salário atual acrescido de R$ 120,00 de bonificação. Ordene a relação resultante pelo nome do vendedor.

**select nome\_vend,((sal\_fixo\*1.75)+120) as from vendedor**

**where faixa\_comiss='c' order by nome\_vend;**

NOME\_VEND NOVO\_SAL

---------------------------------------- ---------- Cassia Andrade 1695

Maria Paula 1695

2 linhas selecionadas.

6.5.1 Máximos (MAX) e Mínimos (MIN) Problema:

Mostrar o menor e o maior salário entre os vendedores.

**select MIN(sal\_fixo),MAX (sal\_fixo)**

**from vendedor;**

MIN(SAL\_FI MAX(SAL\_FI

---------- ----------

900 2500

1 linha selecionada.

6.5.2 Totalizando Colunas (SUM) Problema:

Mostrar a quantidade total pedida para o produto cimento, de código 200.

**select SUM(qtd\_ped) from item\_pedido where cd\_prod=200;**

SUM(QTD\_PE

----------

900

1 linha selecionada.

6.5.3 Calculando Médias (AVG) Problema:

Qual é a média dos salários fixos dos vendedores?

**select AVG(sal\_fixo)**

**from vendedor;**

AVG(SAL\_FI

----------

1580

1 linha selecionada.

6.5.4 Contando as Linhas (COUNT) Problema:

Quantos vendedores ganham acima de R$ 2.000,00 de salário fixo?

**select COUNT(\*)**

**from vendedor**

**where sal\_fixo>2000;**

COUNT(\*)

----------

2

1 linha selecionada.

6.5.5 A Palavra-Chave DISTINCT

Várias linhas de uma tabela podem conter os mesmos valores para as suas colunas (duplicidade), com exceção da chave primária. Quando desejarmos eliminar a duplicidade, podemos inserir a palavra-chave DISTINCT após a palavra-chave SELECT.

Problema:

Em que cidades as nossa empresa possui clientes?

**select DISTINCT cidade from cliente;**

CIDADE

-------------------- guaiba

niteroi resende

rio de janeiro sao paulo

volta redonda

6 linhas selecionadas.

6.5.6 Agrupando a Informação Selecionada (GROUP BY)

Existem ocasiões em que desejamos aplicar uma função de agregação não somente a um conjunto de tuplas, mas também organizar a informação em determinadas categorias. Isto é possível através do uso da cláusula GROUP BY.

Problema:

Listar o número de itens existente em cada pedido.

**select no\_ped, count(\*) as total\_itens from item\_pedido**

**group by no\_ped;**

NO\_PED TOTAL\_ITEN

---------- ----------

1111 3

1112 2

1113 1

2111 2

2112 2

2113 1

3111 1

3112 2

4111 1

4112 2

5111 2

|  |  |
| --- | --- |
| 6111 | 1 |
| 6112 | 2 |
| 7111 | 1 |

14 linhas selecionadas.

Inicialmente as linha são agrupadas através do(s) atributo(s) fornecido(s) na cláusula GROUP BY; neste caso, no\_ped. Em um segundo passo, é aplicada a operação COUNT(\*) para cada grupo de linhas que tenha o mesmo número de pedido. Após a operação de contagem de cada grupo, o resultado da consulta é apresentado.

Normalmente, a cláusula GROUP BY é utilizada em conjunto com as operações COUNT e AVG.

6.5.7 A Cláusula HAVING

Às vezes temos que definir condições e aplicá-las aos grupos ao invés de fazê- lo a cada linha separadamente. Por exemplo, suponha que desejemos listar todos os pedidos que possuam mais de um item. Esta condição não se aplica a uma única linha separadamente, mas a cada grupo definido pela cláusula GROUP BY. Para exprimir tal consulta, usamos a cláusula HAVING. A condição da cláusula HAVING é aplicada após a formação dos grupos; logo, podemos usar funções de agregação na construção das condições da cláusula HAVING.

Problema:

Listar os pedidios que possuam mais de um item.

**select no\_ped,count(\*) as total\_itens from item\_pedido**

**group by no\_ped having count(\*)>1;**

NO\_PED TOTAL\_ITEN

---------- ----------

1111 3

1112 2

2111 2

2112 2

3112 2

4112 2

5111 2

6112 2

8 linhas selecionadas.

**6.6 Recuperando Dados de Várias Tabelas (JOINS)**

Algumas consultas necessitam acessar simultaneamente várias tabelas, o que leva à realização de junções (JOINS) entre as tabelas para poder extrair as informações necessárias para a consulta formulada.

6.6.1 Qualificadores de Nomes

Um qualificador de nome consiste do nome da tabela, seguido de um ponto, seguido por um nome de uma coluna da tabela. Por exemplo, o qualificador da coluna DESC\_PROD da tabela PRODUTO será PRODUTO.DESC\_PROD.

Os qualificadores de nome são utilizados em uma consulta para efetivar a junção (JOIN) entre as tabelas.

Problema:

Faça uma junção da tabela de clientes com a de pedidos, exibindo o nome do cliente, o código do cliente e o número do pedido.

**select cod\_cli,nome\_cli,pedido.num\_ped from cliente,pedido;**

COD\_CLI NOME\_CLI NUM\_PED

---------- ---------------------------------------- ----------

|  |  |  |  |
| --- | --- | --- | --- |
| 1000 | Supermercado | Carrefour | 1111 |
| 2000 | Supermercado | Baratao | 1111 |
| 3000 | Supermercado | Arariboia | 1111 |
| 4000 | UFF |  | 1111 |
| 5000 | CSN |  | 1111 |
| 6000 | Pegeout |  | 1111 |
| 7000 | Ind. Quimicas | Paulistas | 1111 |
| 8000 | Ford Caminhoes | | 1111 |
| 9000 | Riocel Celulose | | 1111 |
| 10000 | Elevadores Sur | | 1111 |
| 1000 | Supermercado | Carrefour | 1112 |
| 2000 | Supermercado | Baratao | 1112 |
| 3000 | Supermercado | Arariboia | 1112 |
| 4000 | UFF |  | 1112 |
| 5000 | CSN |  | 1112 |
| 6000 | Pegeout |  | 1112 |
| 7000 | Ind. Quimicas | Paulistas | 1112 |
| 8000 | Ford Caminhoes | | 1112 |
| 9000 | Riocel Celulose | | 1112 |
| 10000 | Elevadores Sur | | 1112 |
| 1000 | Supermercado Carrefour | | 1113 |
| 2000 | Supermercado Baratao | | 1113 |

|  |  |  |  |
| --- | --- | --- | --- |
| 3000 | Supermercado | Arariboia | 1113 |
| 4000 | UFF |  | 1113 |
| 5000 | CSN |  | 1113 |
| 6000 | Pegeout |  | 1113 |
| 7000 | Ind. Quimicas Paulistas | | 1113 |
| 8000 | Ford Caminhoes | | 1113 |
| 9000 | Riocel Celulose | | 1113 |

..............................................................

|  |  |  |  |
| --- | --- | --- | --- |
| 1000 | Supermercado C | arrefour | 6112 |
| 2000 | Supermercado B | aratao | 6112 |
| 3000 | Supermercado A | rariboia | 6112 |
| 4000 | UFF |  | 6112 |
| 5000 | CSN |  | 6112 |
| 6000 | Pegeout |  | 6112 |
| 7000 | Ind. Quimicas | Paulistas | 6112 |
| 8000 | Ford Caminhoes | | 6112 |
| 9000 | Riocel Celulose | | 6112 |
| 10000 | Elevadores Sur | | 6112 |
| 1000 | Supermercado C | arrefour | 7111 |
| 2000 | Supermercado B | aratao | 7111 |
| 3000 | Supermercado A | rariboia | 7111 |
| 4000 | UFF |  | 7111 |
| 5000 | CSN |  | 7111 |
| 6000 | Pegeout |  | 7111 |
| 7000 | Ind. Quimicas | Paulistas | 7111 |
| 8000 | Ford Caminhoes | | 7111 |
| 9000 | Riocel Celulose | | 7111 |
| 10000 | Elevadores Sur | | 7111 |
| 140 linhas | selecionadas. | |  |

Neste exemplo foi executado um produto cartesiano das tabelas CLIENTE e PEDIDO, seguido de uma projeção das colunas exibidas. Neste caso, poucas informações úteis podem ser extraídas da relação resultante. Devemos então aplicar critérios de seleção à junção para podermos obter algum resultado concreto.

Problema:

A que clientes estão associados os pedidos existentes? Listar pelos nomes dos clientes.

**select nome\_cli,pedido.cd\_cli,pedido.num\_ped from cliente,pedido**

**where cliente.cod\_cli=pedido.cd\_cli;**

|  |  |  |
| --- | --- | --- |
| NOME\_CLI | CD\_CLI | NUM\_PED |
| ---------------------------------------- | ---------- | ---------- |
| Supermercado Carrefour | 1000 | 1111 |
| Supermercado Carrefour | 1000 | 1112 |

|  |  |  |  |
| --- | --- | --- | --- |
| Supermercado | Carrefour | 1000 | 1113 |
| Supermercado | Arariboia | 3000 | 2111 |
| Supermercado | Arariboia | 3000 | 2112 |
| Supermercado | Arariboia | 3000 | 2113 |
| UFF |  | 4000 | 3111 |
| UFF |  | 4000 | 3112 |
| Pegeout |  | 6000 | 4111 |
| Pegeout |  | 6000 | 4112 |
| Ford Caminhoes | | 8000 | 5111 |
| Riocel Celulose | | 9000 | 6111 |
| Riocel Celulose | | 9000 | 6112 |
| Elevadores Sur | | 10000 | 7111 |
| 14 linhas selecionadas. | |  |  |

A equação apresentada na cláusula WHERE é chamada de EQUAÇÃO DE JUNÇÃO.

Podemos utilizar os operadores LIKE, NOT LIKE, IN, NOT IN, NULL, NOT NULL, os operadores relacionais e operadores AND, OR e NOT, na cláusula

WHERE de uma junção de tabelas.

Problema:

Quais são os clientes que têm pedidos com prazos de entrega superiores a 15 dias e que estão localizados nos estados de São Paulo ou do Rio de Janeiro?

**select nome\_cli,uf,pedido.num\_ped,pedido.prazo\_entr from cliente,pedido**

**where cliente.cod\_cli=pedido.cd\_cli and uf in ('rj','sp') and prazo\_entr>15;**

NOME\_CLI UF NUM\_PED PRAZO\_ENTR

---------------------------------------- -- ---------- ---------- Supermercado Carrefour rj 1113 30

Supermercado Arariboia rj 2112 18

2 linhas selecionadas.

Problema:

Mostrar os pedidos dos clientes e seus respectivos prazos de entrega, ordenados do maior para o menor.

**select nome\_cli,pedido.num\_ped,pedido.prazo\_entr from cliente,pedido**

**where cliente.cod\_cli=pedido.cd\_cli order by prazo\_entr DESC;**

|  |  |  |
| --- | --- | --- |
| NOME\_CLI | NUM\_PED | PRAZO\_ENTR |
| ---------------------------------------- | ---------- | ---------- |
| Riocel Celulose | 6112 | 60 |
| Supermercado Carrefour | 1113 | 30 |
| Riocel Celulose | 6111 | 30 |
| Elevadores Sur | 7111 | 20 |
| Supermercado Arariboia | 2112 | 18 |
| Supermercado Arariboia | 2111 | 15 |
| UFF | 3111 | 13 |
| Supermercado Carrefour | 1111 | 10 |
| Ford Caminhoes | 5111 | 10 |
| UFF | 3112 | 7 |
| Pegeout | 4111 | 7 |
| Pegeout | 4112 | 7 |
| Supermercado Carrefour | 1112 | 5 |
| Supermercado Arariboia | 2113 | 3 |

14 linhas selecionadas.

6.6.2 Sinônimos

Para que não seja necessário escrever o nome da tabela nas qualificações de nomes, podemos utilizar ALIASES definidos na própria consulta.

A definição dos ALIASES é feita na cláusula FROM, sendo então utilizada nas outras cláusulas (WHERE, ORDER BY, GROUP BY, HAVING e

SELECT) de uma consulta.

Problema:

Exibir os vendedores (ordenados por nome) que emitiram pedidos com prazos de entrega superiores a 15 dias e que tenham salários fixos iguais ou superiores a R$ 1.000,00.

**select distinct nome\_vend,prazo\_entr from vendedor V, pedido P**

**where V.cod\_vend=P.cd\_vend and**

**V.sal\_fixo>1000 and prazo\_entr>15 order by nome\_vend;**

NOME\_VEND PRAZO\_ENTR

---------------------------------------- ---------- Ana Cristina 60

Armando Pinto 30

2 linhas selecionadas.

**6.6 Junções Envolvendo Três ou mais Tabelas**

Problema:

Exiba a relação dos clientes localizados no Rio de Janeiro (ordenados alfabeticamente) que têm pedidos do produto **Chapa de Aco** com prazos de entrega superirores a 15 dias.

**select nome\_cli,desc\_prod,no\_ped,prazo\_entr**

**from cliente C,pedido P,item\_pedido IP,produto PR**

**where C.cod\_cli=P.cd\_cli and P.num\_ped=IP.no\_ped and IP.cd\_prod=PR.cod\_prod and PR.desc\_prod='Chapa de Aco' and P.prazo\_entr>15 and**

**C.uf='rj'**

**order by C.nome\_cli;**

|  |  |  |
| --- | --- | --- |
| NOME\_CLI | DESC\_PROD | NO\_PED |
| PRAZO\_ENTR |  |  |
| ---------------------------------------- | -------------------- | ---------- |
| ---------- |  |  |
| Supermercado Carrefour | Chapa de Aco | 1113 |
| 30 |  |  |
| 1 linha selecionada. |  |  |

Problema:

Mostre os nome de todos os vendedores que venderam **Chapa de Aco** em quantidade superior a 300 Kg.

**select distinct nome\_vend**

**from vendedor V, pedido P, item\_pedido IP, produto PR**

**where V.cod\_vend=P.cd\_vend and p.num\_ped=IP.no\_ped and IP.cd\_prod=PR.cod\_prod and IP.qtd\_ped>300 and PR.desc\_prod='Chapa de Aco';**

NOME\_VEND

---------------------------------------- Armando Pinto

Maria Paula

Paulo Alberto

3 linhas selecionadas.

Problema:

Quantos clientes fizeram pedidos com a vendedora **Ana Cristina**?

**select count(distinct cod\_cli)**

**from cliente C, pedido P, vendedor V**

**where C.cod\_cli=P.cd\_cli and p.cd\_vend=V.cod\_vend and V.nome\_vend='Ana Cristina';**

COUNT(COD\_

----------

3

1 linha selecionada.

Problema:

Quantos clientes das cidades do Rio de Janeiro e Niterói tiveram seus pedidos tirados com a vendedora **Ana Cristina**?

**select C.cidade,count(distinct cod\_cli) as num\_clientes from cliente C, pedido P, vendedor V**

**where C.cod\_cli=P.cd\_cli and**

**C.cidade in ('rio de janeiro','niteroi') and p.cd\_vend=V.cod\_vend and**

**V.nome\_vend='Ana Cristina' group by C.cidade;**

CIDADE NUM\_CLIENT

-------------------- ---------- niteroi 2

1 linha selecionada.

**6.7 Utilizando Consultas Aninhadas (Subqueries)**

Chamamos de consulta aninhada à consulta cujo o resultado é utilizado por outra consulta, de forma encadeada e contida no mesmo comando SQL.

Problema:

Que produtos estão incluídos em um pedido qualquer com a quantidade pedida igual a 100?

**select desc\_prod from produto**

**where cod\_prod IN (select cd\_prod**

**from item\_pedido where qtd\_ped=100);**

DESC\_PROD

-------------------- Chapa de Aco

Cimento

Fio plastico

3 linhas selecionadas.

Problema:

Quais vendedores ganham um salário fixo abaixo da média?

**select nome\_vend from vendedor where sal\_fixo <**

**(select avg(sal\_fixo)**

**from vendedor);**

NOME\_VEND

---------------------------------------- Paulo Alberto

Cassia Andrade

Maria Paula

3 linhas selecionadas.

|  |  |  |
| --- | --- | --- |
| Problema: |  | |
| Quais os vendedores que só venderam quilogramas (Kg)? | produtos comercializados | em |
| **select distinct cod\_vend,nome\_vend** |  |  |

**from vendedor V**

**where 'kg'=ALL (select unid\_prod**

**from pedido P,item\_pedido IP,produto PR**

**where P.num\_ped=IP.no\_ped and IP.cd\_prod=PR.cod\_prod and P.cd\_vend=V.cod\_vend);**

COD\_VEND NOME\_VEND

---------- ----------------------------------------

15 Maria Paula

1 linha selecionada.

Problema:

Quais clientes realizaram mais de dois pedidos?

**select nome\_cli from cliente C where exists**

**(select count(\*)**

**from pedido**

**where cd\_cli=C.cod\_cli having count(\*)>2);**

NOME\_CLI

---------------------------------------- Supermercado Carrefour

Supermercado Arariboia

2 linhas selecionadas.

Problema:

Quais os produtos que não estão presentes em nenhum pedido?

**select cod\_prod,desc\_prod from produto P**

**where not exists**

**(select \***

**from item\_pedido**

**where cd\_prod=P.cod\_prod);**

COD\_PROD DESC\_PROD

---------- --------------------

0 linhas selecionadas.

**7. Inserindo, Modificando e Excluindo Registros**

**7.1 Inserindo Registros em uma Tabela**

INSERT INTO <nome\_tabela> (<lista\_de\_colunas>) VALUES (<lista\_de\_valores);

Problema:

Inserir o produto **Tinta de PVC** na tabela de produtos.

**insert into produto**

**values (600,’l’,’Tinta PVC’,15.80);**

1 linha processada.

**7.2 Inserindo Registros Usando um SELECT**

INSERT INTO <nome\_tabela> (<lista\_de\_colunas>) SELECT <lista\_de\_colunas>

FROM <nome\_tabela>

WHERE <condição\_de\_seleção>;

Problema:

Cadastrar como clientes os vendedores que emitiram mais de 3 pedidos. Usar para o código de cliente o mesmo código do vendedor. O restante das colunas devem ser preenchidas com um espaço em branco.

**insert into cliente (cod\_cli, nome\_cli,endereco,cidade,cep,uf)**

**select cod\_vend,nome\_vend,' ', ' ',' ',' ' from vendedor V**

**where 3<(select count(\*)**

**from pedido P**

**where V.cod\_vend=P.cd\_vend);**

2 linhas processadas.

**7.3 Modificando um Registro**

UPDATE <nome\_tabela>

SET {<nome\_coluna> = <expressão>} WHERE <condição\_de\_seleção>;

Problema:

Alterar o preço unitário do **Cimento** para R$ 5,00.

**update produto**

**set val\_unit=5.00**

**where desc\_prod=’Cimento’;**

1 linha processada.

Problema:

Atualizar o salário fixo de todos os vendedores em 27% mais uma bonificação de R$ 100,00.

**update vendedor**

**set sal\_fixo=(sal\_fixo\*1.27+100.00);**

5 linhas processadas.

Problema:

Acrescentar 2,5% ao preço unitário dos produtos que estejam abaixo da média dos preços dos produtos comprados a quilo.

**update produto**

**set val\_unit=val\_unit\*1.025**

**where val\_unit < (select avg(val\_unit)**

**from produto**

**where unid\_prod=’kg’);**

3 linhas processadas.

**7.4 Excluindo Registros**

DELETE FROM <nome\_tabela> WHERE <condição\_de\_seleção>;

Problema:

Excluir todos o itens de pedido que tenham quantidade pedida inferior a 200.

**delete from item\_pedido where qtd\_ped < 200;**

6 linhas processadas.

Problema

Excluir todos os vendedores que não realizaram nenhum pedido.

**delete from vendedor V**

**where 0=(select count(\*)**

**from vendedor,pedido**

**where V.cod\_vend=cd\_vend);**

1 linha processada.

**8. Transações**

A execução de um programa que inclui operações de acesso a um banco de dados é chamada de **transação**. Se as operações em questão não alteram os dados do banco de dados, a transação é chamada de **read-only transaction**. Neste capítulo estamos interessados apenas nas transações que executam operações de atualização. Logo, a palavra **transação** se refere a um programa que realiza operações que alteram registros de um banco de dados.

**8.1 Propriedades das Transações**

Uma transação deve possuir as seguintes propriedades (chamadas de propriedades **ACID**):

1. **Atomicidade**: uma transação é uma unidade atômica de processamento;

ou ela é executada na sua totalidade, ou então nada é executado.

2. **Consistência**: a execução de uma transação deve manter a consistência de um banco de dados.

3. **Isolamento**: uma transação não deve tornar visível para outras transações

as modificações feitas em um banco de dados até que ela seja encerrada com sucesso.

4. **Durabilidade**: uma vez executada com sucesso, as alterações feitas por uma transação devem persistir, mesmo se houver falhas subseqüentes no

sistema.

**8.2 Definição de Transações em SQL**

O padrão SQL especifica que uma transação deve começar de modo subentendido. As transações são terminadas por um dos seguintes comandos SQL:

1. **commit**: efetiva a transação corrente e inicia uma nova.

2. **rollback**: aborta a transação corrente.

**Exemplo**

**update vendedor**

**set sal\_fixo=1000.00;**

**rollback;**

No exemplo acima, as alterações feitas nos salários fixos dos vendedores não serão efetivadas, pois o comando **rollback** é executado antes que a transação seja terminada com sucesso. Logo, os salários fixos dos vendedores permanecerão com os mesmos valores que possuíam imediatamente antes do início da transação.

**9. Visões**

Considerações sobre segurança podem exigir que determinados dados não estejam disponíveis para alguns usuários. Logo, não é desejável que o modelo lógico possa ser acessado indiscriminadamente. Com base nas questões de segurança, podemos criar uma coleção de relações personalizadas que se ajustem melhor às necessidades dos usuários e que levem em conta as questões de segurança. Tais relações são chamadas de **visões**.

Uma **visão**, na terminologia SQL, é uma tabela que é derivada de outras tabelas. Estas outras tabelas podem ser tabelas-base (criadas através do

comando **CREATE TABLE**) ou outras visões previamente definidas. Uma

**visão** não está, necessariamente, fisicamente armazenada no banco de dados; ela pode existir apenas virtualmente, em contraste com as tabelas-base, cujas as tuplas se encontram fisicamente armazenadas no banco de dados.

**9.1 Definindo Visões**

Em SQL, uma visão é definida através do comando **create view**, cuja a forma geral é:

CREATE VIEW <nome\_visão> (<nomes das colunas>) AS <expressão\_de\_consulta>;

Problema:

Criar uma visão que contenha apenas os produtos vendidos em quilograma.

**create view prod\_kg**

**(codigo,descricao,unidade)**

**as select cod\_prod,desc\_prod,unid\_prod from produto**

**where unid\_prod=’kg’;**

Instrução processada.

Problema

Criar uma visão contendo o código do vendedor, o seu nome e o salário fixo anual.

**create view salario\_anual**

**(codigo,nome,sal\_anual)**

**as select cod\_vend,nome\_vend,sal\_fixo\*12 from vendedor;**

Instrução processada.

Problema

Criar uma visão contendo os vendedores, seus pedidos efetuados e os respectivos produtos.

**create view lista\_pedidos**

**as select nome\_vend,num\_ped,desc\_prod**

**from vendedor V,pedido P,item\_pedido I, produto PR**

**where V.cod\_vend=P.cd\_vend and P.num\_ped=I.no\_ped and I.cd\_prod=PR.cod\_prod;**

Instrução processada.

**9.2 Utilizando Visões**

Problema

Com base na visão **salario\_anual**, mostrar os vendedores que têm salário fixo anual superior a R$ 20.000,00.

**select nome,sal\_anual from salario\_anual**

**where sal\_anual > 20000;**

NOME SAL\_ANUAL

---------------------------------------- ---------- Ana Cristina 25200

Armando Pinto 30000

2 linhas selecionadas.

Problema

Inserir uma linha na visão **prod\_kg** com código **800**, descrição **Cimento**

**Branco**, e unidade **kg**.

**insert into prod\_kg**

**values (800,’Cimento Branco’,’kg’);**

ORA-01400: não é possível inserir NULL em ("xxx"."PRODUTO"."VAL\_UNIT")

Problema

Excluir da visão salario\_anual todos os vendedores que têm salário anual abaixo de R$ 15.000,00.

**delete from salario\_anual where sal\_anual<15000;**

ORA-02292: restrição de integridade (xxx.SYS\_C001449) violada - registro filho localizado

**9.3 Eliminando Visões**

Podemos eliminar uma visão através do comando **drop view**, cuja a forma geral é:

DROP VIEW <nome\_visão>;

Problema:

Eliminar a visão **salario\_anual**;

**drop view salario\_anual;**

Instrução processada.

**10. Índices**

Os SGBD’s utilizam índices para tornar mais eficiente a recuperação dos dados de um banco de dados.

Um índice é uma estrutura de dados, onde são armazenados valores e ponteiros organizados de forma ascendente ou descendente, que torna possível

localizar rapidamente as linhas de uma tabela nas quais o valor desejado está armazenado.

Os índices são utilizados internamente pelo SGBD, ficando totalmente transparente ao usuário a sua utilização.

**10.1 Criando Índices**

Um índice é construído sobre uma ou mais colunas de uma determinada tabela. Em SQL, um índice é criado através do comando **create index**, cuja a forma geral é:

CREATE [UNIQUE] INDEX <nome\_índice> ON <nome\_tabela> (<lista\_de\_colunas>);

A cláusula UNIQUE é opcional, e a sua inclusão assegura a não existência de valores duplicados no índice a ser criado.

Problema:

Criar um índice chamado **produto\_desc\_prod\_idx** sobre a coluna **desc\_prod**

da tabela **produto**.

**create index produto\_desc\_prod\_idx on produto(desc\_prod);**

Instrução processada.

Problema:

Criar um índice único chamado **cliente\_nome\_cli\_idx** sobre a coluna

**nome\_cli** da tabela **cliente**.

**create unique index cliente\_nome\_cli\_idx on cliente(nome\_cli);**

Instrução processada.

Problema:

Criar um índice único chamado **pedido\_num\_ped\_cd\_vend\_idx** sobre as colunas **num\_ped** e **cd\_vend** da tabela **pedido**.

**create unique index pedido\_num\_ped\_cd\_vend\_idx on pedido(num\_ped,cd\_vend);**

Instrução processada.

Problema:

Criar um índice único chamado **cliente\_cod\_cli\_idx** sobre a coluna **cod\_cli** da tabela **cliente**.

**create unique index cliente\_cod\_cli\_idx on cliente(cod\_cli);**

ORA-01408: esta lista de colunas já está indexada

Obs: um índice único é automaticamente construído quando uma coluna é definida como **chave primária** no comando CREATE TABLE.

**10.2 Eliminado Índices**

Podemos eliminar uma índice através do comando **drop index**, cuja a forma geral é:

DROP INDEX <nome\_índice>;

Problema:

Eliminar os índices criados nos problemas acima.

**drop index produto\_desc\_prod\_idx;**

Instrução processada.

**drop index cliente\_nome\_cli\_idx;**

Instrução processada.

**drop index pedido\_num\_ped\_cd\_vend\_idx;**

Instrução processada.

**11. Segurança**

A informação é vital para o sucesso de um negócio, mas quando ela é tratada de forma incorreta ou quando cai em mão erradas, pode tornar-se um sério obstáculo para se atingir o sucesso.

Visando garantir a segurança dos dados, os SGBD’s disponíveis no mercado fornecem um série de facilidades para salvaguardar as informações por eles mantidas. Tais facilidades são implementadas concedendo-se e retirando-se privilégios dos usuários sobre os dados mantidos por um SGBD.

**11.1 O Comando GRANT**

Quando uma tabela ou uma visão é criada, o nome do usuário que a criou é anexado implicitamente ao nome da tabela ou da visão.

Exemplo:

Se a tabela **produto** foi criada pelo usuário **pedro**; então, internamente, ela será conhecida como **pedro.produto**.

O criador de uma tabela ou de uma visão tem todos os privilégio sobre o objeto criado, podendo inclusive conceder tais privilégios para outros

usuários.

Em SQL, os privilégios são concedidos através do comando **grant**, cuja a forma geral é:

GRANT <lista\_de\_privilégios> ON <nome\_objeto> TO <lista\_de\_usuários>;

Os privilégios concedidos são os seguintes: select permite executar consultas insert permite inserir registros

delete permite excluir registros update permite modificar registros

all permite executar qualquer operação

**Observações:**

a) Os usuário que irão receber os privilégios têm que estar previamente cadastrados no banco de dados.

b) Podemos conceder privilégios para todos os usuários do banco de dados colocando a palavra **PUBLIC** no lugar reservado para a

**<lista\_de\_usuários>**.

Exemplos:

**GRANT select on produto to pedro;**

Permite apenas consultas ao usuário Pedro sobre a tabela **produto**.

**GRANT select, insert, update on pedido to ana;**

Concede ao usuário Ana os privilégios de consulta, inclusão e alteração sobre a tabela **pedido**.

**GRANT select on cliente to ana,pedro;**

Concede aos usuários Ana e Pedro o privilégio de consulta sobre a tabela

**cliente**.

**GRANT all on vendedor to PUBLIC;**

Permite todos os privilégios a todos os usuários cadastrados sobre a tabela

**vendedor**.

Problema:

Disponibilizar para seleção, a todos os usuários, a visão **salario\_anual**.

**grant select on salario\_anual to public;**

Instrução processada.

Problema:

Disponibilizar para alteração as colunas endereco, cidade, cep e uf, da tabela

**cliente**, para todos os usuários.

**grant update (endereco,cidade,cep,uf) on cliente to public;**

Instrução processada.

Podemos passar nossa concessão de privilégio a outros usuários através da cláusula WITH GRANT OPTION.

Exemplo:

Conceder ao usuário Ana o poder de dar a concessão de todos os privilégios sobre a tabela **pedido** a outros usuários.

**grant all on pedido to ana with grant option;**

Instrução processada.

**11.2 O Comando REVOKE**

Da mesma forma que podemos conceder privilégios de acesso a outros usuários, podemos também retirá-los através do comando REVOKE, cuja a forma geral é:

REVOKE <lista\_de\_privilégios> ON <nome\_objeto> FROM <lista\_de\_usuários>;

Exemplo:

Retirar o privilégio de seleção sobre a tabela **pedido** do usuário Ana.

REVOKE select on pedido from ana;

Problema:

Retirar todos os privilégios concedidos a todos os usuários sobre a visão

**salario\_anual**.

**revoke all on salario\_anual from public;**

Instrução processada.

Problema:

Retirar os privilégios de atualização e inserção concedidos ao usuário Ana sobre a tabela **pedido**.

**revoke insert, update on pedido from ana;**

Instrução processada.

**APÊNDICES**

5.2 – Funções Matemáticas[[editar](https://pt.wikibooks.org/w/index.php?title=PostgreSQL_Pr%C3%A1tico/Fun%C3%A7%C3%B5es_Internas/Matem%C3%A1ticas&veaction=edit&section=1" \o "Editar secção: 5.2 – Funções Matemáticas) | [editar código-fonte](https://pt.wikibooks.org/w/index.php?title=PostgreSQL_Pr%C3%A1tico/Fun%C3%A7%C3%B5es_Internas/Matem%C3%A1ticas&action=edit&section=1)]

**Operadores Matemáticos**

+, -, \*, /, % (módulo, resto de divisão de inteiros), ^(potência), !(fatorial), @(valor absoluto)

| / - rais quadrada ( | / 25.0 = 5)

| | / - raiz cúbica ( | | / 27.0 = 3)

**Algumas funções Matemáticas**

ABS(x) - valor absoluto de x

CEIL(numeric) - arredonda para o próximo inteiro superior

DEGREES(valor) - converte valor de radianos para graus

FLOOR(numeric) - arredonda para o próximo inteiro inferior

MOD(x,y) - resto da divisão de x por y

PI() - constante PI (3,1415...)

POWER(x,y) - x elevado a y

RADIANS(valor) - converte valor de graus para radianos

RANDOM() - valor aleatório entre 0 e 1

ROUND(numeric) - arredonda para o inteiro mais próximo

ROUND(v, d) - arredonda v com d casas decimais

SIGN(numeric) - retorna o sinal da entrada, como -1 ou +1

SQRT(X) - Raiz quadrada de X

TRUNC (numeric) - trunca para o nenhuma casa decimal

TRUNC (v numeric, s int) - trunca para s casas decimais

**Operadores Lógicos:**

AND, OR e NOT. TRUE, FALSE e NULL

**Operadores de Comparação:**

<, >, <=, >=, =, <> ou !=

a BETWEEN x AND y

a NOT BETWEEN x AND y

expressão IS NULL

expressão IS NOT NULL

expressão IS TRUE

expressão IS NOT TRUE

expressão IS FALSE

expressão IS NOT FALSE

expressão IS UNKNOWN

expressão IS NOT UNKNOWN

**OPERADOR NULL**

Em SQL NULL é para valores inexistentes. Regra geral: NULL se propaga, o que significa que com quem NULL se combina o resultado será um NULL. NULL não zero, não é string vazia nem string de comprimento zero.

Um exemplo: num cadastro de alunos, para o aluno que ainda não se conhece a nota, não é correto usar zero para sua nota, mas sim NULL. Não se pode efetuar cálculos de expressões onde um dos elementos é NULL.

**COMPARANDO NULLs**

NOT NULL com NULL -- Unknown

NULL com NULL -- Unknown

**CONVERSÃO DE/PARA NULL**

NULLIF() e COALESCE()

NULLIF(valor1, valor2)

NULLIF – Retorna NULL se, e somente se, valor1 e valor2 forem iguais, caso contrário retorna valor1.

Algo como:

if (valor1 == valor2){

then NULL

else valor1;

**Retorna valor1 somente quando valor1 == valor2.**

**COALESCE** – retorna o primeiro de seus argumentos que não for NULL. Só retorna NULL quando todos os seus argumentos forem NULL.

Uso: mudar o valor padrão cujo valor seja NULL.

create table nulos(nulo int, nulo2 int, nulo3 int);

insert into nulos values (1,null,null);

select coalesce(nulo, nulo2, nulo3) from nulos; - - Retorna 1, valor do campo nulo;

select coalesce(nulo2, nulo3) from nulos; - - Retorna NULL, pois ambos são NULL.

**GREATEST** - Retorna o maior valor de uma lista - SELECT GREATEST(1,4,6,8,2); - - 8

**LEAST** - Retorna o menor valor de uma lista.

Todos os valores da lista devem ser do mesmo tipo e nulos são ignorados.

Obs.: Ambas as funções acima não pertencem ao SQL standard, mas são uma extensão do PostgreSQL.

**CONCATENANDO NULLs**

A regra é: NULL se propaga. Qualquer que concatene com NULL gerará NULL.

STRING || NULL -- NULL

Usos:

- Como valor default para campos que futuramente receberão valor.

- Valor default para campos que poderão ser sempre inexistentes.

### Arredondar por aproximidade:

Essa é uma das funções mais utilizadas para arredondar as casas decimais, pois o valor que será arredondado será sempre para o valor inteiro mais próximo da expressão. Veja como fica a aplicação da função **ROUND**:

**SELECT** **ROUND**(10.19)  
> 10

**SELECT** **ROUND**(10.91)  
> 11

**SELECT** **ROUND**(10.5)  
> 11

**SELECT** **ROUND**(10.111111)  
> 10

Uma vantagem ainda do **ROUND** é que você tem a possibilidade de informar quantas casas decimais você deseja de aproximação. Isso é bastante útil quando você quer trbalhar com o formato moeda por exemplo, ou ainda que o seu banco do banco de dados seja do tipo **DECIMAL(X, 2)** ou **Double Precision**. Nesse caso, veja como você pode usar a função e como fica o seu retorno:

**SELECT** **ROUND**(10.1231, 3)  
> 10.123

**SELECT** **ROUND**(10.1239, 3)  
> 10.124

Como você pode constar, o valor .1231 é mais próximo de .123 (contando somente as 3 casas de aproximação), já o valor .1239 é mais próximo de .124.

### Arredondamento por corte:

É uma maneira pouco usual de se fazer esse procedimento, e o resultado se assemelha muito a função **FLOOR** (arredondar para baixo). Nesse caso, estou me referindo ao **TRUNCATE**, que deve ser informado também a quantidade de casas decimais que você deseja na resposta, veja os exemplos:

**SELECT** **TRUNCATE**(10.19, 0)  
> 10

**SELECT** **TRUNCATE**(10.19, 1)  
> 10.1

**SELECT** **TRUNCATE**(10.19, 2)  
> 10.19

**SELECT** **TRUNCATE**(10.19, 3)  
> 10.190

# 14 manipulações com String no Postgres

Durante o ciclo de desenvolvimento de uma procedure ou script Transact-SQL, quantas vezes você já precisou manipular uma String no Postgres, e se perguntou qual seria a função ou comando mais adequado? Como exemplo, imagine que você tenha uma string com o nome completo e seja necessário listar apenas o primeiro nome. Ou ainda precisa identificar a posição inicial de um texto.

Felizmente, assim como em outras linguagens de programação, o Transact-SQL do Postgres oferece inúmeras funções que torna mais fácil a manipulação de strings. Vamos listar 14 delas.

#### ****1. SUBSTR() ou SUBSTRING()****

Tanto o SUBSTR(), quanto o SUBSTRING() são funções que tem o mesmo objetivo, retirar parte de uma String, a partir de uma determinada posição.

Declarações das funções: ***substr(string text, posicao integer, quantidade integer)*** e ***substring(string text, posicao integer, quantidade integer)***.

Exemplo: Em ambos os exemplos abaixo, o resultado será ***“brasileiro”***.

[?](https://www.gagno.com/14-manipulacoes-com-string-no-postgres/)

|  |  |
| --- | --- |
| 1  2  3 | SELECT SUBSTR('Eu sou brasileiro', 8, 10);  SELECT SUBSTRING('Eu sou brasileiro', 8, 10);  SELECT SUBSTRING('Eu sou brasileiro' FROM 8 FOR 10); |

#### ****2. REPEAT()****

Como o próprio nome diz, a função REPEAT(), repete uma string quantas vezes for necessário.

Declaração da função: ***repeat(string text, numero integer)***.

Exemplo: Neste exemplo, teremos a string ‘BRASIL’ se repetindo por 3 vezes, resultado a string **‘BRASILBRASILBRASIL’**.

[?](https://www.gagno.com/14-manipulacoes-com-string-no-postgres/)

|  |  |
| --- | --- |
| 1 | SELECT REPEAT('BRASIL', 3); |

#### ****3. REPLACE()****

Para substituir todas as ocorrências na string inicial, da string de origem pela de destino utilizaremos esta função.

Declaração da função: **replace(string\_inicial text, origem text, destino text)**.

Exemplo: No exemplo a seguir, vamos substituir o texto **‘Argentina’** por **‘Brasil’** na String ***‘Argentina penta campeão do mundo’***. O resultado será ***‘Brasil penta campeão do mundo’***.

[?](https://www.gagno.com/14-manipulacoes-com-string-no-postgres/)

|  |  |
| --- | --- |
| 1 | SELECT REPLACE('Argentina penta campeão do mundo', 'Argentina', 'Brasil'); |

#### 4. POSITION() ou STRPOS()

As funções position() e strpos() servem para retornar um número que corresponde a posição inicial da substring em uma string.

Declaração da função: ***position(substring text in string text)*** e ***strpos(string text, substring text)***.

Exemplo: O resultado no exemplo a baixo é 4.

[?](https://www.gagno.com/14-manipulacoes-com-string-no-postgres/)

|  |  |
| --- | --- |
| 1  2 | SELECT POSITION('no' in 'Gagno.com');  SELECT STRPOS('Gagno.com', 'no'); |

#### ****5. TRIM()****

A função trim() tem como objetivo remover caracteres (espaço por padrão) da extremidade inicial, final ou ambas as extremidades da string, podendo atingir o resultado de duas formas para cada tipo. Caso use a função trim é importante identificar o uso do primeiro parâmetro que pode ser:

**leading -> para remoção à esquerda -> que corresponde ao LTRIM();**

**trailing -> para remoção à direita -> que corresponde ao RTRIM();**

**both -> para remoção em ambos os lados**.

Declaração da função:

***ltrim(string text [, caracter text]), rtrim(string text [, caracter text]), trim(string text [, caracter text]) e trim([leading | trailing | both]  [string text] FROM string text).***

Exemplo: No exemplo, será demonstrado o funcionamento da função para remover espaço em branco e também o caracter x de ambas as formas. Veja:

[?](https://www.gagno.com/14-manipulacoes-com-string-no-postgres/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | -- Removendo à esquerda  SELECT LTRIM(' Testando '); -- Removendo espaço em branco.  SELECT TRIM(leading from ' Testando '); -- Removendo espaço em branco.  SELECT LTRIM('xxTestandox', 'x'); -- Removendo caracter x.  SELECT TRIM(leading 'x' from 'xxTestandox'); -- Removendo caracter x.  -- Removendo à direita  SELECT RTRIM(' Testando '); -- Removendo espaço em branco.  SELECT TRIM(trailing from ' Testando '); -- Removendo espaço em branco.  SELECT RTRIM('xxTestandox', 'x'); -- Removendo caracter x.  SELECT TRIM(trailing 'x' from 'xxTestandox'); -- Removendo caracter x.  -- Removendo em ambas as extremidades  SELECT TRIM(' Testando '); -- Removendo espaço em branco.  SELECT TRIM(both from ' Testando '); -- Removendo espaço em branco.  SELECT TRIM('xxTestandox', 'x'); -- Removendo caracter x.  SELECT TRIM(both 'x' from 'xxTestandox'); -- Removendo caracter x. |

#### ****6. LOWER() e UPPER()****

A função lower() converte os caracteres da string para minúsculo, enquanto a função upper() converse os caracteres da string para maiúsculo.

Declarações das funções: ***lower(string text)*** e ***upper(string text)***.

Exemplo: Veremos no exemplo a conversão das strings. Veja:

[?](https://www.gagno.com/14-manipulacoes-com-string-no-postgres/)

|  |  |
| --- | --- |
| 1  2 | SELECT LOWER('EXEMPLO');  SELECT UPPER('exemplo'); |

#### ****7. DOIS || (pipes)****

Para concatenar strings, basta fazer uso de dois pipes.

Exemplo: Vamos concatenar as strings resultando a string ‘abcde’.

[?](https://www.gagno.com/14-manipulacoes-com-string-no-postgres/)

|  |  |
| --- | --- |
| 1 | SELECT 'ab' || 'cd' || 'e'; |

#### ****8. OVERLAY()****

Essa função é usada para substituir parte dos caracteres de uma string (sobreposição).

Declaração da função: ***overlay(string placing string from integer [for integer])***.

Exemplo: No exemplo abaixo, vamos demonstrar o uso da função, inclusive sem o último parâmetro.

[?](https://www.gagno.com/14-manipulacoes-com-string-no-postgres/)

|  |  |
| --- | --- |
| 1  2 | SELECT OVERLAY('Argentina penta campeão do mundo' placing 'Brasil' from 1 for 9);  SELECT OVERLAY('Brasil' placing 'ão' from 5); |

#### ****9. INITCAP()****

Esta função converte a primeira letra de cada palavra da string em maiúscula e as outras em minúsculo.

Descrição da função: ***initcap(text)***.

Exemplo: Repare a string gerada na aplicação da função (“Brasil Penta Campeão Do Mundo”).

[?](https://www.gagno.com/14-manipulacoes-com-string-no-postgres/)

|  |  |
| --- | --- |
| 1 | SELECT INITCAP('brasil penta campeão do mundo'); |

#### ****10. CHR()****

Esta função retorna o caracter correspondente ao código ASCII informado.

Declaração da função: ***chr(integer)***.

Exemplo: No exemplo abaixo, vamos exibir a string ‘AB’.

[?](https://www.gagno.com/14-manipulacoes-com-string-no-postgres/)

|  |  |
| --- | --- |
| 1 | SELECT CHR(65) || CHR(66); |

#### ****11. LPAD() e RPAD()****

Essa função preenche a string até o comprimento adicionando os caracteres de preenchimento (espaço por padrão) à esquerda -> lpad ou à direita -> rpad(). Se a string for mais longa que o comprimento então é truncada (à direita).

Declaração da função: ***lpad(string text, comprimento integer [, preenchimento text])*** e ***rpad(string text, comprimento integer [, preenchimento text])***.

Exemplo: No exemplo abaixo, repare que no primeiro e terceiro caso será adicionado espaço na string.

[?](https://www.gagno.com/14-manipulacoes-com-string-no-postgres/)

|  |  |
| --- | --- |
| 1  2  3  4 | SELECT LPAD('oi', 5); -- resultado "   oi".  SELECT LPAD('oi', 5, 'ae'); -- resultado "aeaoi".  SELECT RPAD('oi', 5); -- resultado "oi   ".  SELECT RPAD('oi', 5, 'ae'); -- resultado "oiaea". |

#### ****12. MD5()****

Essa função serve para calcular o md5 da string e retorna em hexadecimal.

Declaração da função: ***md5(string text)***.

Exemplo:

[?](https://www.gagno.com/14-manipulacoes-com-string-no-postgres/)

|  |  |
| --- | --- |
| 1 | SELECT MD5('TESTEmd5'); |

#### ****13. SPLIT\_PART()****

Essa função divide a string utilizando o delimitador, retornando o campo especificado (contado a partir de 1).

Declaração da função: ***split\_part(string text, delimitador text, posicao integer)***.

Exemplo: O retorno da query será a string “oub”.

[?](https://www.gagno.com/14-manipulacoes-com-string-no-postgres/)

|  |  |
| --- | --- |
| 1 | SELECT SPLIT\_PART('aei~#~oub~#~cdf', '~#~', 2); |

#### ****14. TRANSLATE()****

Nessa função todo caracter da string que corresponde a um caracter do conjunto origem, é substituído pelo caracter correspondente do conjunto destino.

Declaração da função: ***translate(string text, origem text, destino text)***.

Exemplo:

[?](https://www.gagno.com/14-manipulacoes-com-string-no-postgres/)

|  |  |
| --- | --- |
| 1 | SELECT TRANSLATE('12345', '25', 'zm'); |

#### ****E VOCÊ, TEM ALGUMA OBSERVAÇÃO A FAZER?****

Se você já precisou utilizar alguma dessas funções ou utilizou alguma que não contém neste post. Partilhe sua experiência. Comente.

Até já!

Tags:[chr()](https://www.gagno.com/tag/chr/), [initcap()](https://www.gagno.com/tag/initcap/), [lower()](https://www.gagno.com/tag/lower/), [lpad()](https://www.gagno.com/tag/lpad/), [ltrim()](https://www.gagno.com/tag/ltrim/), [md5()](https://www.gagno.com/tag/md5/), [overlay()](https://www.gagno.com/tag/overlay/), [pipes](https://www.gagno.com/tag/pipes/), [position()](https://www.gagno.com/tag/position/), [Postgres](https://www.gagno.com/tag/postgres/), [repeat()](https://www.gagno.com/tag/repeat/), [replace()](https://www.gagno.com/tag/replace/), [rpad()](https://www.gagno.com/tag/rpad/), [rtim()](https://www.gagno.com/tag/rtim/), [split\_part()](https://www.gagno.com/tag/split_part/), [strpos()](https://www.gagno.com/tag/strpos/), [substr()](https://www.gagno.com/tag/substr/), [substring()](https://www.gagno.com/tag/substring/), [translate()](https://www.gagno.com/tag/translate/), [trim()](https://www.gagno.com/tag/trim/), [upper()](https://www.gagno.com/tag/upper/)

## Funções de Data/Hora[[editar](https://pt.wikibooks.org/w/index.php?title=PostgreSQL_Pr%C3%A1tico/Fun%C3%A7%C3%B5es_Internas/Data_e_Hora&veaction=edit&section=1" \o "Editar secção: 5.4 – Funções de Data/Hora) | [editar código-fonte](https://pt.wikibooks.org/w/index.php?title=PostgreSQL_Pr%C3%A1tico/Fun%C3%A7%C3%B5es_Internas/Data_e_Hora&action=edit&section=1)]

**Operações com datas:**

timestamp '2001-09-28 01:00' + interval '23 hours' -> timestamp '2001-09-29 00:00'

date '2001-09-28' + interval '1 hour' -> timestamp '2001-09-28 01:00'

date '01/01/2006' – date '31/01/2006'

time '01:00' + interval '3 hours'time -> '04:00'

interval '2 hours' - time '05:00' -> time '03:00:00'

Função age (retorna Interval) - **Diferença entre datas**

age(timestamp)interval (**Subtrai de hoje**)

age(timestamp '1957-06-13') -> 43 years 8 mons 3 days

age(timestamp, timestamp)interval Subtrai os argumentos

age('2001-04-10', timestamp '1957-06-13') -> 43 years 9 mons 27 days

Função extract (retorna double) Extrai parte da data**: ano, mês, dia, hora, minuto, segundo.**

select extract(year from age('2001-04-10', timestamp '1957-06-13'))

select extract(month from age('2001-04-10', timestamp '1957-06-13'))

select extract(day from age('2001-04-10', timestamp '1957-06-13'))

**Data e Hora atuais** (retornam data ou hora)

SELECT CURRENT\_DATE;

SELECT CURRENT\_TIME;

SELECT CURRENT\_TIME(0);

SELECT CURRENT\_TIMESTAMP;

SELECT CURRENT\_TIMESTAMP(0);

**Somar dias e horas a uma data**:

SELECT CAST('06/04/2006' AS DATE) + INTERVAL '27 DAYS' AS Data;

**Função now** (retorna timestamp with zone)

now() - Data e hora corrente (timestamp with zone);

Não usar em campos somente timestamp.

**Função date\_part** (retorna double)

SELECT date\_part('day', TIMESTAMP '2001-02-16 20:38:40');

Resultado: 16 (day é uma string, diferente de extract)

**Obtendo o dia da data atual**:

SELECT DATE\_PART('DAY', CURRENT\_TIMESTAMP) AS dia;

**Obtendo o mês da data atual:**

SELECT DATE\_PART('MONTH', CURRENT\_TIMESTAMP) AS mes;

**Obtendo o ano da data atual:**

SELECT DATE\_PART('YEAR', CURRENT\_TIMESTAMP) AS ano;

**Função date\_trunc** (retorna timestamp)

SELECT date\_trunc('year', TIMESTAMP '2001-02-16 20:38:40');

Retorna 2001-02-16 00:00:00

**Convertendo (CAST)**

select to\_date('1983-07-18', 'YYYY-MM-DD')

select to\_date('19830718', 'YYYYMMDD')

**Função timeofday** (retorna texto)

select timeofday() -> Fri Feb 24 10:07:32.000126 2006 BRT

**Interval**

interval [ (p) ]

to\_char(interval '15h 2m 12s', 'HH24:MI:SS')

date '2001-09-28' + interval '1 hour'

interval '1 day' + interval '1 hour'

interval '1 day' - interval '1 hour'

900 \* interval '1 second'

Interval trabalha com as unidades: second, minute, hour, day, week, month, year, decade, century, millenium ou abreviaturas ou plurais destas unidades.

Se informado sem unidades '13 10:38:14' será devidamente interpretado '13 days 10 hours 38 minutes 14 seconds'.

**CURRENTE\_DATE** - INTERVAL '1' day;

TO\_TIMESTAMP('2006-01-05 17:56:03', 'YYYY-MM-DD HH24:MI:SS')

**Tipos Geométricos:**

CREATE TABLE geometricos(ponto POINT, segmento LSEG, retangulo BOX, poligono POLYGON, circulo CIRCLE);

ponto (0,0),

segmento de (0,0) até (0,1),

retângulo (base inferior (0,0) até (1,0) e base superior (0,1) até (1,1)) e

círculo com centro em (1,1) e raio 1.

INSERT INTO geometricos VALUES ('(0,0)','((0,0),(0,1))', '((0,0),(0,1))', '((0,0),(0,1),(1,1),(1,0))','((1,1),1)');

[This document was created with Win2PDF available at http://www.daneprairie.com.](http://www.daneprairie.com) The unregistered version of Win2PDF is for evaluation or non-commercial use only.