Aspectos Práticos

Em relação ao trabalho prático

- Alguma dúvida?
- 10 minutos de consultoria pra cada projeto caso achem necessário
- Se não acharem necessário VAMOS EM FRENTE.

Algumas Funcionalidades

comandos complexos

chaves estrangeiras

gatilhos

visões

integridade transacional

controle de simultaneidade
multiversão

Mecanismos de Extensão

- tipos de dado
- funções
- operadores
- funções de agregação
- métodos de índice
- linguagens procedurais

Criando um banco

createdb

Apagando um banco

dropdb

Acessando um banco

psql

Visões

```
CREATE VIEW minha_visao AS
    SELECT cidade, temp_min, temp_max, prcp, data, localizacao
        FROM clima, cidades
        WHERE cidade = nome;

SELECT * FROM minha_visao;
```

Chave Estrangeira

```
CREATE TABLE cidades (
        cidade      varchar(80) primary key,
        localizacao point
);

CREATE TABLE clima (
        cidade     varchar(80) references cidades(cidade),
        temp_min   int,
        temp_max   int,
        prcp       real,
        data       date
);
```

Herança

```
CREATE TABLE cidades (
  nome       text,
  populacao  real,
  altitude   int     -- (em pés)
);

CREATE TABLE capitais (
  estado      char(2)
) INHERITS (cidades);
```

```
SELECT nome, altitude
    FROM ONLY cidades
    WHERE altitude > 500;
```

```
   nome     | altitude
------------+----------
 Las Vegas  |     2174
 Mariposa   |     1953
(2 linhas)
```

Definição de Dados

Criando tabela

```
CREATE TABLE minha_primeira_tabela (
    primeira_coluna text,
    segunda_coluna integer
);
```

Excluindo tabela

```
DROP TABLE minha_primeira_tabela;
DROP TABLE produtos;
```

Valor padrão

```
CREATE TABLE produtos (
    cod_prod   integer PRIMARY KEY,
    nome       text,
    preco      numeric DEFAULT 9.99
);
```

```
CREATE TABLE produtos (
    cod_prod integer DEFAULT nextval('produtos_cod_prod_seq'),
    ...
);
```

Restrições

```
CREATE TABLE produtos (
    cod_prod   integer,
    nome       text,
    preco      numeric CHECK (preco > 0)
);
```

```
CREATE TABLE produtos (
    cod_prod  integer,
    nome      text,
    preco     numeric CONSTRAINT chk_preco_positivo CHECK (preco > 0)
);
```

```
CREATE TABLE produtos (
    cod_prod           integer,
    nome               text,
    preco              numeric CHECK (preco > 0),
    preco_com_desconto numeric CHECK (preco_com_desconto > 0),
    CHECK (preco > preco_com_desconto)
);
```

Alteração da seqüência da coluna serial

```
=> ALTER SEQUENCE t_c1_seq RESTART WITH 1000;

=> INSERT INTO t VALUES (DEFAULT, 'Primeira linha');

=> SELECT * FROM t;

  c1  |       c2
------+----------------
 1000 | Primeira linha
(1 linha)
```

Declarando Matriz

```
CREATE TABLE jogo_da_velha (
    casa     integer[3][3]
);

INSERT INTO sal_emp
    VALUES ('Bill',
    '{10000, 10000, 10000, 10000}',
    '{{"reunião", "almoço"}, {"treinamento", "apresentação"}}');

INSERT INTO sal_emp
    VALUES ('Carol',
    '{20000, 25000, 25000, 25000}',
    '{{"café da manhã", "consultoria"}, {"reunião", "almoço"}}');

SELECT nome FROM sal_emp WHERE pagamento_semanal[1] <> pagamento_semanal[2];

 UPDATE sal_emp SET pagamento_semanal = '{25000,25000,27000,27000}'
     WHERE nome = 'Carol';

 UPDATE sal_emp SET pagamento_semanal[4] = 15000
```

Tipos Compostos

```
CREATE TYPE complexo AS (
    r       double precision,
    i       double precision
);

CREATE TYPE catalogo AS (
    nome            text,
    id_fornecedor   integer,
    preco           numeric
);

CREATE TABLE estoque (
    item     catalogo,
    contador integer
);

INSERT INTO estoque VALUES (ROW('dados de pano', 42, 1.99), 1000);

SELECT (item).nome FROM estoque WHERE (item).preco > 9.99;

UPDATE minha_tabela SET coluna_complexa = ROW(1.1,2.2) WHERE ...;

UPDATE minha_tabela SET coluna_complexa.r = (coluna_complexa).r + 1 WHERE ...;
```

Tipos Compostos

```
CREATE TYPE complexo AS (
    r       double precision,
    i       double precision
);

CREATE TYPE catalogo AS (
    nome            text,
    id_fornecedor   integer,
    preco           numeric
);

CREATE TABLE estoque (
    item     catalogo,
    contador integer
);

INSERT INTO estoque VALUES (ROW('dados de pano', 42, 1.99), 1000);

SELECT (item).nome FROM estoque WHERE (item).preco > 9.99;

UPDATE minha_tabela SET coluna_complexa = ROW(1.1,2.2) WHERE ...;

UPDATE minha_tabela SET coluna_complexa.r = (coluna_complexa).r + 1 WHERE ...;
```

Funções e operadores

Operadores matemáticos

Operador	Descrição	Exemplo	Resultado
+	adição	2 + 3	5
-	subtração	2 - 3	-1
*	multiplicação	2 * 3	6
/	divisão (divisão inteira trunca o resultado)	4 / 2	2
%	módulo (resto)	5 % 4	1
^	exponenciação	2.0 ^ 3.0	8
|/	raiz quadrada	|/ 25.0	5
||/	raiz cúbica	||/ 27.0	3
!	fatorial	5 !	120
!!	fatorial (operador de prefixo)	!! 5	120
@	valor absoluto	@ -5.0	5
&	AND bit a bit	91 & 15	11
|	OR bit a bit	32 | 3	35
#	XOR bit a bit	17 # 5	20
~	NOT bit a bit	~1	-2
<<	deslocamento à esquerda bit a bit	1 << 4	16
>>	deslocamento à direita bit a bit	8 >> 2	2

Funções matemáticos

`mod(y, x)`	(same as argument types)	resto de `y/x`	`mod(9,4)`	`1`
`pi()`	`dp`	constante "π"	`pi()`	`3.14159265358979`
`power(a dp, b dp)`	`dp`	`a` elevado a `b`	`power(9.0, 3.0)`	`729`
`power(a numeric, b numeric)`	`numeric`	`a` elevado a `b`	`power(9.0, 3.0)`	`729`
`radians(dp)`	`dp`	graus para radianos	`radians(45.0)`	`0.785398163397448`
`random()`	`dp`	valor randômico entre 0.0 e 1.0	`random()`	

Funções trigonométricas

Função	Descrição
`acos(x)`	arco cosseno
`asin(x)`	arco seno
`atan(x)`	arco tangente
`atan2(x, y)`	arco tangente de x/y
`cos(x)`	cosseno
`cot(x)`	cotangente
`sin(x)`	seno
`tan(x)`	tangente

Funções e operadores para cadeia de caracteres

Exemplo	Resultado
`'Post' || 'greSQL'`	`PostgreSQL`
`bit_length( 'José')`	`32`
`char_length( 'José')`	`4`
`position( 'om' in 'Thomas')`	`3`
`substring( 'Thomas' from 2 for 3)`	`hom`

Operadores bit a bit

Operador	Descrição	Exemplo	Resultado
||	concatenação	B'10001' || B'011'	10001011
&	AND bit a bit	B'10001' & B'01101'	00001
|	OR bit a bit	B'10001' | B'01101'	11101
#	XOR bit a bit	B'10001' # B'01101'	11100
~	NOT bit a bit	~ B'10001'	01110
<<	deslocamento à esquerda bit a bit	B'10001' << 3	01000
>>	deslocamento à direita bit a bit	B'10001' >> 2	00100

Funções de formatação

Função	Tipo retornado	Descrição	Exemplo
`to_char(timestamp, text)`	`text`	converte carimbo do tempo (`time stamp`) em cadeia de caracteres	`to_char(current_timestamp, 'HH12:MI:SS')`
`to_char(interval, text)`	`text`	converte intervalo em cadeia de caracteres	`to_char(interval '15h 2m 12s', 'HH24:MI:SS')`
`to_char(int, text)`	`text`	converte inteiro em cadeia de caracteres	`to_char(125, '999')`
`to_char(double precision, text)`	`text`	converte real e precisão dupla em cadeia de caracteres	`to_char(125.8::real, '999D9')`
`to_char(numeric, text)`	`text`	converte numérico em cadeia de caracteres	`to_char(-125.8, '999D99S')`
`to_date(text, text)`	`date`	converte cadeia de caracteres em data	`to_date('05 Dec 2000', 'DD Mon YYYY')`
`to_timestamp(text, text)`	`timestamp with time zone`	converte cadeia de caracteres em carimbo do tempo	`to_timestamp('05 Dec 2000', 'DD Mon YYYY')`
`to_number(text, text)`	`numeric`	converte cadeia de caracteres em numérico	`to_number('12,454.8-', '99G999D9S')`

Operadores para data e hora

Operador	Exemplo	Resultado
+	date '2001-09-28' + integer '7'	date '2001-10-05'
+	date '2001-09-28' + interval '1 hour'	timestamp '2001-09-28 01:00'
+	date '2001-09-28' + time '03:00'	timestamp '2001-09-28 03:00'
+	interval '1 day' + interval '1 hour'	interval '1 day 01:00'
+	timestamp '2001-09-28 01:00' + interval '23 hours'	timestamp '2001-09-29 00:00'
+	time '01:00' + interval '3 hours'	time '04:00'
-	- interval '23 hours'	interval '-23:00'
-	date '2001-10-01' - date '2001-09-28'	integer '3'
-	date '2001-10-01' - integer '7'	date '2001-09-24'
-	date '2001-09-28' - interval '1 hour'	timestamp '2001-09-27 23:00'
-	time '05:00' - time '03:00'	interval '02:00'
-	time '05:00' - interval '2 hours'	time '03:00'
-	timestamp '2001-09-28 23:00' - interval '23 hours'	timestamp '2001-09-28 00:00'
-	interval '1 day' - interval '1 hour'	interval '23:00'

Funções informação do sistema

Nome	Tipo retornado	Descrição
`current_database()`	`name`	nome do banco de dados corrente
`current_schema()`	`name`	nome do esquema corrente
`current_schemas(boolean)`	`name[]`	nomes dos esquemas no caminho de procura incluindo, opcionalmente, os esquemas implícitos
`current_user`	`name`	nome do usuário do contexto de execução corrente
`inet_client_addr()`	`inet`	endereço da conexão remota
`inet_client_port()`	`int4`	porta da conexão remota
`inet_server_addr()`	`inet`	endereço da conexão local
`inet_server_port()`	`int4`	port da conexão local
`session_user`	`name`	nome do usuário da sessão
`user`	`name`	equivale a `current_user`
`version()`	`text`	informação da versão do PostgreSQL

Correspondência com padrão

Like

```
'abc' LIKE 'abc'     verdade
'abc' LIKE 'a%'      verdade
'abc' LIKE '_b_'     verdade
'abc' LIKE 'c'       falso
```

Expressões regulares

```
'abc' SIMILAR TO 'abc'       verdade
'abc' SIMILAR TO 'a'         falso
'abc' SIMILAR TO '%(b|d)%'   verdade
'abc' SIMILAR TO '(b|c)%'    falso
```

Correspondência com padrão

case

```
SELECT a,
    CASE a WHEN 1 THEN 'um'
           WHEN 2 THEN 'dois'
           ELSE 'outro'
    END AS caso
FROM teste;
```

Modificando Tabelas

Adicionando uma nova coluna

```
ALTER TABLE piloto ADD COLUMN observacao text;
```

Removendo uma coluna

```
ALTER TABLE piloto DROP COLUMN observacao;
```

Adicionando restrição

```
ALTER TABLE produtos ADD CHECK (nome <> '');
ALTER TABLE produtos ADD CONSTRAINT unq_cod_prod UNIQUE
(cod_prod);
ALTER TABLE produtos ADD FOREIGN KEY (fk_grupo_produtos)
REFERENCES grupo_produtos;
```

Removendo restrição

```
ALTER TABLE produtos DROP CONSTRAINT nome_da_restrição;
```

Modificando Tabelas

Alterando o tipo de uma coluna

```
ALTER TABLE produtos ALTER COLUMN preco TYPE
numeric(10,2);
```

Alterando o nome de uma coluna

```
ALTER TABLE produtos RENAME COLUMN cod_prod TO
cod_produto;
```

Alterando o nome de tabela

```
ALTER TABLE produtos RENAME TO equipamentos;
```

Alguns Tipos de Dados

Tipos Geométricos

Nome	Tamanho de Armazenamento	Descrição	Representação
`point`	16 bytes	Ponto no plano	(x,y)
`line`	32 bytes	Linha infinita (não totalmente implementado)	((x1,y1),(x2,y2))
`lseg`	32 bytes	Segmento de linha finito	((x1,y1),(x2,y2))
`box`	32 bytes	Caixa retangular	((x1,y1),(x2,y2))
`path`	16+16n bytes	Caminho fechado (semelhante ao polígono)	((x1,y1),...)

Definindo Funções

```
CREATE FUNCTION limpar_emp () RETURNS void AS '
DELETE FROM emp
WHERE emp.salario <= 0;
' LANGUAGE SQL;
SELECT limpar_emp();


CREATE FUNCTION um() RETURNS integer AS '
SELECT 1;
' LANGUAGE SQL;

CREATE FUNCTION somar(integer, integer) RETURNS
integer AS $$
SELECT $1 + $2;
$$ LANGUAGE SQL;
```

Definindo Funções

```
CREATE FUNCTION debitar (integer, numeric) RETURNS
integer AS $$
UPDATE conta_corrente
SET saldo = saldo - $2
WHERE numero_da_conta = $1;
SELECT 1;
$$ LANGUAGE SQL;

CREATE FUNCTION debitar (integer, numeric) RETURNS
numeric AS $$
UPDATE conta_corrente
SET saldo = saldo - $2
WHERE numero_da_conta = $1;
SELECT saldo FROM conta_corrente WHERE
numero_da_conta = $1;
$$ LANGUAGE SQL;
```

Definindo Funções

Recebendo um tipo composto

```
CREATE FUNCTION dobrar_salario(emp) RETURNS numeric AS $$
SELECT $1.salario * 2 AS salario;
$$ LANGUAGE SQL;

SELECT nome, dobrar_salario(emp.*) AS sonho
FROM emp;
```

```
 nome | sonho
------+-------
 José |  8400
(1 linha)
```

```
SELECT nome, dobrar_salario(ROW(nome, salario*1.1, idade,
baia)) AS sonho
FROM emp;
```

```
 nome | sonho
------+--------
 João | 4840.0
 José | 9240.0
(2 linhas)
```

Definindo Funções

Retornando um tipo composto

```
CREATE FUNCTION novo_empregado() RETURNS emp AS $$
SELECT text 'Nenhum' AS nome,
1000.0 AS salario,
25 AS idade,
point '(2,2)' AS baia;
$$ LANGUAGE SQL;


CREATE OR REPLACE FUNCTION novo_empregado() RETURNS emp AS
$$
SELECT ROW('Nenhum', 1000.0, 25, '(2,2)')::emp;
$$ LANGUAGE SQL;

SELECT novo_empregado();

SELECT * FROM novo_empregado();
```

Definindo Funções

Retornando um tipo composto

```
CREATE FUNCTION getfoo(int) RETURNS foo AS $$
SELECT * FROM foo WHERE fooid = $1;
$$ LANGUAGE SQL;

SELECT *, upper(fooname) FROM getfoo(1) AS t1;
```

```
 fooid | foosubid | fooname | upper
-------+----------+---------+-------
     1 |        1 | João    | JOÃO
(1 linha)
```

Definindo Funções

Retornando conjunto

```
CREATE FUNCTION getfoo(int) RETURNS SETOF foo AS $$
SELECT * FROM foo WHERE fooid = $1;
$$ LANGUAGE SQL;
SELECT * FROM getfoo(1) AS t1;
```

```
 fooid | foosubid | fooname
-------+----------+---------
     1 |        1 | João
     1 |        2 | José
(2 linhas)
```

Definindo Funções

Funções Polimórficas

```
CREATE FUNCTION constroi_matriz(anyelement, anyelement)
RETURNS anyarray AS $$
SELECT ARRAY[$1, $2];
$$ LANGUAGE SQL;
SELECT constroi_matriz(1, 2) AS intarray,
constroi_matriz('a'::text, 'b') AS textarray;
```

```
 intarray | textarray
----------+-----------
 {1,2}    | {a,b}
(1 linha)
```

```
CREATE FUNCTION eh_maior(anyelement, anyelement)
RETURNS boolean AS $$
SELECT $1 > $2;
$$ LANGUAGE SQL;
SELECT eh_maior(1, 2);
```

```
 eh_maior
----------
 f
(1 linha)
```

Definindo Funções

Funções Podem ser sobrecarregadas

```
CREATE FUNCTION teste(int) RETURNS ...

CREATE FUNCTION teste(int, int) RETURNS ...
```

Definindo Gatilhos

```
DROP TRIGGER phonebook on addressbook;

CREATE OR REPLACE FUNCTION add_to_phonebook() RETURNS
TRIGGER AS $phonebook$
DECLARE
new_name varchar;
new_phonenum varchar;
BEGIN
IF(TG_OP='INSERT') THEN
INSERT INTO phonebook(name,phonenum)
VALUES(NEW.name,NEW.phonenum);
END IF;
RETURN NEW;
END;
$phonebook$ LANGUAGE plpgsql;

CREATE TRIGGER phonebook AFTER INSERT ON addressbook FOR
EACH ROW EXECUTE PROCEDURE add_to_phonebook();
```

Exercício Prático 01

1) Criar uma função que passe como parâmetro dois números e retorne a soma;

2) Criar uma função que passe como parâmetro uma dada chave e retorne todas as tuplas associadas a esta chave;

3) Definir uma simples *trigger* (gatilho) para cada vez que uma tupla for incluída na tabela A, seja incluída também da tabela B.

(*ver página 583 do tutorial*)

Exercício Prático 02

1) Criar um banco cujo esquema representa a seguinte regra de negócio:
   - Autores escrevem artigos;
   - Artigos podem ser escritos por vários autores, assim como um autor para escrever mais de um artigo;
   - Uma submissão em um evento possui os seguintes atributos: código da submissão; código do evento; código do artigo; situação (aprovada/reprovada); data da submissão.
2) Defina as seguintes restrições:
   - O nome do autor deve ter no mínimo 10 caracteres;
   - As datas de submissão devem ser superiores ao ano de 2005;
3) Defina as seguintes funções:
   - Retornar a quantidade de autores de um dado artigo;
   - Quantas submissões aprovadas tem um dado autor;
   - Aprovar uma determinada submissão;
4) Defina uma *trigger* (gatilho) para :
   - Cada vez que uma submissão for aprovada, atualizar uma tabela que contem a quantidade de submissões aprovadas por autor.