

[MAC0313]
Introdução aos Sistemas de Bancos de Dados
Aula 20
Linguagem SQL (Parte 6):
Stored Procedures

Kelly Rosa Braghetto

DCC-IME-USP

5 de novembro de 2019

Persistent Stored Modules (SQL/PSM)

- ▶ Cada SGBD oferece um modo diferente para o usuário manter **junto ao esquema do BD** funções ou procedimentos que podem ser usados nas consultas SQL ou em outros comandos SQL
- ▶ Essas funções e procedimentos são escritos em linguagens simples, mas que nos permitem executar dentro do BD operações que não podem ser expressas em SQL
- ▶ O padrão *Persistent Stored Modules* (PSM), uma extensão da SQL, captura os principais conceitos existentes nessas linguagens simples implementadas nos SGBDs

Persistent Stored Modules (SQL/PSM)

O PSM padroniza a sintaxe e semântica dos comandos de:

- ▶ controle de fluxo
- ▶ tratamento de exceções (chamado de *condition handling* no PSM)
- ▶ variáveis locais
- ▶ atribuição de valores para variáveis e parâmetros
- ▶ uso de cursores (um tipo de estrutura de estrutura que possibilita a navegação em registros)

Funções e procedimentos em PSM

Estrutura geral da criação:

```
CREATE PROCEDURE <nome> (<parâmetros>)  
<declarações locais>  
<corpo do procedimento> ;
```

```
CREATE FUNCTION <nome> (<parâmetros>) RETURNS <tipo>  
<declarações locais>  
<corpo da função> ;
```

Funções e procedimentos em PSM

Parâmetros de um procedimento

- ▶ São triplas contendo modo–nome–tipo
- ▶ Modos:
 - ▶ IN – define um parâmetro de entrada
 - ▶ OUT – define um parâmetro de saída
 - ▶ INOUT – define um parâmetro de entrada e saída

Parâmetros de uma função

- ▶ São duplas contendo nome–tipo
- ▶ Um parâmetro de uma função é sempre do tipo IN
- ▶ A única forma de obter informações de uma função é por meio de seu valor de retorno

Funções e procedimentos em PSM

BD de exemplo – Filmes

```
CREATE TABLE ExecutivoDeCinema(  
    codigo INTEGER PRIMARY KEY,  
    nome VARCHAR(50),  
    patrimonio NUMERIC(10,2)  
);  
  
CREATE TABLE Estudio(  
    nome VARCHAR(30) PRIMARY KEY,  
    endereco VARCHAR(70),  
    codigo_presidente INTEGER  
        REFERENCES ExecutivoDeCinema(codigo)  
);
```

Funções e procedimentos em PSM

BD de exemplo – Filmes (Continuação)

```
CREATE TABLE Filme(  
    titulo          VARCHAR(50),  
    ano             INTEGER,  
    duracao         INTEGER,  
    colorido        BOOLEAN,  
    genero          VARCHAR(20),  
    nome_estudio    VARCHAR(30) REFERENCES Estudio(nome),  
    codigo_produto  INTEGER  
                REFERENCES ExecutivoDeCinema(codigo),  
    PRIMARY KEY (titulo,ano)  
);  
  
CREATE TABLE EstrelaDeCinema(  
    nome            VARCHAR(50) PRIMARY KEY,  
    endereco        VARCHAR(70),  
    sexo            CHAR,  
    dataNascimento  DATE  
);
```

Funções e procedimentos em PSM

BD de exemplo – Filmes (continuação)

```
CREATE TABLE Elenco(  
    titulo_filme    VARCHAR(50),  
    ano_filme       INTEGER,  
    nome_estrela    VARCHAR(50) REFERENCES EstrelaDeCinema(nome),  
    PRIMARY KEY (titulo_filme,ano_filme,nome_estrela),  
    FOREIGN KEY (titulo_filme,ano_filme) REFERENCES Filme(titulo,ano)  
);
```


Funções e procedimentos em PSM

Exemplo

```
CREATE PROCEDURE Mudanca(  
    IN nome_estrela VARCHAR(30)  
    IN endereco_novo VARCHAR(255) )  
UPDATE EstrelaDeCinema  
SET     endereco = endereco_novo  
WHERE  nome = nome_estrela;
```

- Um procedimento para mudar o endereço de uma estrela de cinema

Funções e procedimentos em PSM

Alguns comandos simples de PSM

- ▶ Declaração de variáveis locais:
`DECLARE <nome variável> <tipo>;`
- ▶ Atribuição de valores:
`SET <variável> = <expressão>;`
- ▶ Definição de blocos:
`BEGIN ... END;`
- ▶ Devolução do valor de retorno:
`RETURN <expressão>;`

Funções e procedimentos em PSM

Comandos para desvio condicional de fluxo

- ▶ Estrutura:

```
IF <condição> THEN
    <lista de comandos>
{[ ELSEIF <condição> THEN
    <lista de comandos>
  ]}
[ ELSE
    <lista de comandos>
]
END IF;
```

- ▶ Pode haver mais de um bloco de ELSIF
- ▶ Os blocos de ELSEIF e ELSE são opcionais

Funções e procedimentos em PSM

Exemplo

```
CREATE FUNCTION ProdComedia(ano_interesse INT, estudio_interesse VARCHAR(30))
RETURNS BOOLEAN
    IF NOT EXISTS (SELECT * FROM Filme WHERE ano = ano_interesse AND
                                                           nome_estudio = estudio_interesse)
    THEN RETURN TRUE;
    ELSEIF 1 <= (SELECT COUNT(*) FROM Filme WHERE ano = ano_interesse AND
                                                           nome_estudio = estudio_interesse
                                                           AND genero = 'comedia')
    THEN RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF;
```

- ▶ A função só devolve verdadeiro se o estúdio `estudio_interesse`, no ano `ano_interesse`, não produziu filme nenhum ou produziu ao menos uma comédia

Consultas em PSM

Há várias maneiras de se usar consultas
SELECT-FROM-WHERE em PSM:

- ▶ Subconsultas podem ser usadas em condições ou, de forma geral, em qualquer lugar onde são aceitas em SQL
- ▶ Consultas que devolvem um único valor podem ser usadas no lado direito de comandos de atribuição (SET ...)
- ▶ Um comando de seleção que retorna uma única tupla pode ser usado para atribuir valores a variáveis por meio da cláusula INTO
- ▶ Consultas podem ser usadas na declaração de cursores

Funções e procedimentos em PSM

Exemplo

```
CREATE FUNCTION ObtemPatrimonioPresidente(IN nome_estudio VARCHAR(30))
RETURNS NUMERIC(10,2)
DECLARE patrimonio_pres INTEGER;
  SELECT patrimonio INTO patrimonio_pres
  FROM Estudo, ExecutivoDeCinema
  WHERE codigo_presidente = codigo AND
        Estudo.nome = nome_estudio;
RETURN patrimonio_pres;
```

- ▶ A consulta dentro do procedimento acima devolve uma única linha de resultado
- ▶ A cláusula INTO no SELECT faz uma atribuição a uma variável
- ▶ O comando abaixo é equivalente ao SELECT ... INTO acima:

```
SET patrimonio_pres = (SELECT patrimonio
FROM Estudo, ExecutivoDeCinema
WHERE codigo_presidente = codigo AND
      Estudo.nome = nome_estudio);
```

Funções e procedimentos em PSM

Laços

- ▶ Estrutura geral:

```
<rótulo do laço>: LOOP  
    <lista de comandos>  
END LOOP;
```

- ▶ Para sair do laço:

```
LEAVE <rótulo do laço>;
```

Funções e procedimentos em PSM

LOOP - Exemplo: função que calcula somatório

```
CREATE FUNCTION Somatorio(IN n INTEGER) RETURNS INTEGER
DECLARE soma INTEGER;
DECLARE i INTEGER;
    SET soma = 0;
    SET i = 1;
    laço_soma: LOOP
        IF i > n THEN
            LEAVE laço_soma;
        ELSE
            SET soma = soma + i;
            SET i = i + 1;
        END IF;
    END LOOP;
    RETURN soma;
```


Funções e procedimentos em PSM

Outros tipos comuns de laços

```
WHILE <condição> DO  
    <lista de comandos>  
END WHILE;
```

```
REPEAT  
    <lista de comandos>  
UNTIL <condição>  
END REPEAT;
```

Funções e procedimentos em PSM

WHILE - Exemplo: função que calcula somatório

```
CREATE FUNCTION Somatorio(IN n INTEGER) RETURNS INTEGER
DECLARE soma INTEGER;
DECLARE i INTEGER;
    SET soma = 0;
    SET i = 1;
    WHILE i <= n DO
        SET soma = soma + i;
        SET i = i + 1;
    END WHILE;
RETURN soma;
```

Funções e procedimentos em PSM

REPEAT - Exemplo: função que calcula somatório

```
CREATE FUNCTION Somatorio(IN n INTEGER) RETURNS INTEGER
DECLARE soma INTEGER;
DECLARE i INTEGER;
    SET soma = 0;
    SET i = 0;
    REPEAT
        SET soma = soma + i;
        SET i = i + 1;
    UNTIL i <= n
    END REPEAT;

    RETURN soma;
```

Funções e procedimentos em PSM

Exemplo – calcula a média e a variância das durações dos filmes de um estúdio

```
1) CREATE PROCEDURE MediaVariancia(  
2)   IN e VARCHAR(30),  
3)   OUT media REAL,  
4)   OUT variancia REAL  
5) )  
6) DECLARE Not_Found CONDITION FOR SQLSTATE '02000';  
7) DECLARE CursorFilme CURSOR FOR  
8)   SELECT duracao FROM Filme WHERE nome_estudio = e;  
9) DECLARE novaDuracao INTEGER;  
10) DECLARE contFilme INTEGER;
```

Funções e procedimentos em PSM

Exemplo – calcula a média e a variância das durações dos filmes de um estúdio (continuação)

```
BEGIN
```

```
9) SET media = 0.0;
```

```
10) SET variancia = 0.0;
```

```
11) SET contFilme = 0;
```

```
12) OPEN CursorFilme;
```

```
13) loopFilme: LOOP
```

```
14)     FETCH FROM CursorFilme INTO novaDuracao;
```

```
15)     IF Not_Found THEN LEAVE loopFilme END IF;
```

```
16)     SET contFilme = contFilme + 1;
```

```
17)     SET media = media + novaDuracao;
```

```
18)     SET variancia = variancia + novaDuracao * novaDuracao;
```

```
19) END LOOP;
```

```
20) SET media = media/contFilme;
```

```
21) SET variancia = variancia/contFilme - media * media;
```

```
22) CLOSE CursorFilme;
```

```
END;
```

Funções e procedimentos em PSM

Cursor

- ▶ É uma estrutura de controle que permite percorrer tuplas em um BD
- ▶ Um cursor é definido por meio de uma consulta, que determina as tuplas que ele controlará
- ▶ Com o cursor, apenas uma tupla é acessada por vez
- ▶ Assim, é possível realizar processamentos sobre linhas do BD de forma sequencial (algo que nem sempre é possível de se fazer em consultas SELECT-FROM-WHERE “puras”)

Funções e procedimentos em PSM

Para se usar um cursor em uma *stored procedure*, é preciso:

1. Declarar o cursor, associando a ele uma consulta
(linha 6 do exemplo)
2. Abrir o cursor, para que o resultado da consulta seja obtido
(linha 12 do exemplo)
3. Em um laço, usar o cursor para percorrer o resultado **uma linha por vez**, carregando os dados em variáveis locais
(conforme a necessidade)
(linhas 13 e 14 do exemplo)
4. Fechar o cursor quando terminar de usá-lo
(linha 22 do exemplo)

Funções e procedimentos em PSM

Laços do tipo FOR

- ▶ São usados somente para percorrer cursores
- ▶ Estrutura geral:

```
FOR <nome do laço> AS <nome do cursor> CURSOR FOR  
    <consulta>  
DO  
    <lista de comandos>  
END FOR;
```

- ▶ O FOR facilita a manipulação de cursores, pois elimina a necessidade de se abrir e fechar o cursor, recuperar tuplas e verificar o erro gerado quando o final do cursor é alcançado

Funções e procedimentos em PSM

Laços do tipo FOR – Exemplo

```
CREATE PROCEDURE MediaVariancia(  
    IN e VARCHAR(20), OUT media REAL, OUT variancia REAL)  
DECLARE contFilme INTEGER;  
BEGIN  
    SET media = 0.0; SET variancia = 0.0; SET contFilme = 0;  
    FOR loopFilme AS CursorFilme CURSOR FOR  
        SELECT duracao FROM Filme WHERE nome_estudio = e;  
    DO  
        SET contFilme = contFilme + 1;  
        SET media = media + duracao;  
        SET variancia = variancia + duracao * duracao;  
    END FOR;  
    SET media = media/contFilme;  
    SET variancia = variancia/contFilme - media * media  
END;
```

Funções e procedimentos em PSM

Exceções em PSM

- ▶ Um SGBD c/ SQL indica condições de erro atribuindo um código (de 5 caracteres) à variável SQLSTATE (da conexão)
- ▶ Exemplos de códigos de erros:
 - ▶ '02000' – nenhuma tupla encontrada (fim de cursor)
 - ▶ '21000' – um SELECT para o qual se esperava apenas uma linha de retorno devolveu mais de uma linha
- ▶ PSM permite a declaração de *exception handlers*, cada um deles associado a um bloco de código
- ▶ Componentes de um *handler*:
 - ▶ Uma lista de condições de exceção (que invocam o *handler* quando geradas)
 - ▶ O código a ser executado quando uma das exceções é gerada
 - ▶ Uma indicação do lugar aonde ir depois que o tratamento tiver sido concluído. Opções: CONTINUE, EXIT, UNDO

Funções e procedimentos em PSM

Exceções em PSM – Exemplo

```
CREATE FUNCTION ObtemAno(titulo_interesse VARCHAR(255))
RETURNS INTEGER
DECLARE Not_Found CONDITION FOR SQLSTATE '02000' ;
DECLARE Too_Many CONDITION FOR SQLSTATE '21000' ;
BEGIN
    DECLARE EXIT HANDLER FOR Not_Found, Too_Many
        RETURN NULL;
    RETURN (SELECT ano FROM Filme
            WHERE titulo = titulo_interesse ) ;
END;
```

A função acima recebe o nome de um filme e, caso haja apenas um filme como esse nome na tabela *Filme*, devolve o ano dele. Em caso de exceção (ou seja, o filme não está na tabela ou existe mais de um filme com esse nome), a função devolve NULL.

A implementação de *Stored Procedures* no PostgreSQL

Stored Procedures no PostgreSQL

- ▶ O PostgreSQL não diferencia *procedures* e *functions*
 - ▶ Só existe o comando `CREATE FUNCTION`
 - ▶ Para criar uma *procedure*, basta criar uma *function* com valor de retorno do tipo `VOID`
- ▶ Funções são executadas por meio de comandos SQL comuns. Exemplos envolvendo a chamada de uma função `myFunc` que tem um único parâmetro de entrada:
 - ▶ Na cláusula `SELECT`:
`SELECT minhaFuncao1(tabela1.atributo1) FROM tabela1;`
 - ▶ Na cláusula `FROM`:
`SELECT * FROM minhaFuncao1(valor1);`
 - ▶ Na cláusula `WHERE`:
`SELECT * FROM tabela1 WHERE
minhaFuncao1(tabela1.atributo1) = 42;`

Funções no PostgreSQL

Há 4 tipos de funções no PostgreSQL:

- ▶ Funções escritas puramente em SQL
- ▶ Funções em linguagens procedurais (PL/pgSQL, PL/Tcl, PL/php, PL/Java, etc)
- ▶ Funções internas (round(), now(), abs(), count(), etc).
- ▶ Funções na linguagem C

Veremos alguns exemplos de funções em SQL puro e PL/pgSQL.

PostgreSQL: Funções em SQL “puro”

```
CREATE OR REPLACE FUNCTION olamundo() RETURNS INT AS
$$
    SELECT 1;
$$ LANGUAGE sql;
```

```
SELECT olamundo();    -- exemplo de chamada
```

```
CREATE OR REPLACE FUNCTION soma_numeros(IN nr1 INT, IN nr2 INT)
RETURNS INT AS
$$
    SELECT $1 + $2;
$$ LANGUAGE sql;
```

```
SELECT soma_numeros(300, 700) AS resposta;    -- ex. de chamada
```

PostgreSQL: Funções em SQL “puro”

- ▶ Usa-se os tipos da SQL para os parâmetros e para o valor de retorno (int, varchar, char, time, etc.)
- ▶ Parâmetros podem ser de entrada (IN), saída (OUT) ou entrada e saída (INOUT)
- ▶ Atribuir nomes a parâmetros é opcional; é possível referenciar um parâmetro no corpo da função usando as variáveis posicionais \$1, \$2, ...

PostgreSQL: Funções em SQL “puro”

- ▶ Podemos passar como parâmetro uma tupla de uma tabela; nesse caso, o tipo do parâmetro é o próprio nome da tabela
- ▶ É possível gerar tuplas por meio da cláusula **ROW**

```
CREATE TABLE Empregado (  
    nome VARCHAR(20), salario NUMERIC, idade INTEGER);
```

```
INSERT INTO Empregado VALUES('João',2200,21);
```

```
INSERT INTO Empregado VALUES('José',4200,30);
```

```
CREATE FUNCTION ver_salario_dobrado(IN emp Empregado)  
RETURNS NUMERIC AS $$
```

```
    SELECT emp.salario * 2 AS salario;  
$$ LANGUAGE SQL;
```

```
SELECT nome, ver_salario_dobrado(Empregado.*) AS salario_sonhado  
FROM Empregado WHERE nome = 'João';
```

```
-- ou
```

```
SELECT nome, ver_salario_dobrado(ROW(nome, salario*1.1, idade))  
    AS salario_sonhado FROM Empregado;
```

PostgreSQL: Funções em SQL “puro”

- Uma função também pode retornar um tipo composto (tupla)

```
CREATE FUNCTION novo_empregado()  
RETURNS Empregado AS $$  
    SELECT VARCHAR(20) 'Fulano' AS nome, 1000.0 AS salario,  
           25 AS idade;  
$$ LANGUAGE SQL;  
  
-- ou  
CREATE FUNCTION novo_empregado()  
RETURNS Empregado AS $$  
    SELECT ROW('Fulano', 1000.0, 25)::Empregado;  
$$ LANGUAGE SQL;  
  
-- Exemplos de chamadas:  
SELECT novo_empregado();  
  
-- ou  
SELECT * FROM novo_empregado();
```

PostgreSQL: Funções em SQL “puro”

```
CREATE TABLE Pessoa (chave1 INT, chave2 INT, nome VARCHAR(20));  
INSERT INTO Pessoa VALUES (1, 1, 'João');  
INSERT INTO Pessoa VALUES (1, 2, 'José');  
INSERT INTO Pessoa VALUES (2, 1, 'Maria');
```

```
CREATE FUNCTION SeleccionaPessoaPelaChave(INT)  
RETURNS Pessoa AS  
$$  
    SELECT * FROM Pessoa WHERE chave1 = $1;  
$$ LANGUAGE SQL;
```

```
-- A chamada abaixo abaixo exibe apenas uma tupla como resposta,  
-- apesar de existirem duas tuplas com chave1 = 1 na tabela Pessoa  
SELECT * FROM SeleccionaPessoaPelaChave(1) AS t;
```

- ▶ A chamada à função pode aparecer na cláusula FROM de uma consulta
- ▶ SeleccionaPessoaPelaChave devolve uma só tupla por conta do tipo do seu valor de retorno (que é uma tupla de Empregado)

PostgreSQL: Funções em SQL “puro”

```
CREATE FUNCTION SeleccionaPessoaPelaChave2(INT)
RETURNS SETOF Pessoa AS
$$
    SELECT * FROM Pessoa WHERE chave1 = $1;
$$ LANGUAGE SQL;

SELECT * FROM SeleccionaPessoaPelaChave2(1) AS t;
```

- ▶ O construtor **SETOF** permite definir um tipo de dado que é um conjunto de tuplas
- ▶ Essa nova implementação de SeleccionaPessoaPelaChave pode devolver mais de uma tupla!

PostgreSQL: Funções em PL/pgSQL

- ▶ Em funções da PL/pgSQL, é possível criar variáveis e blocos que definem diferentes escopos para as variáveis

```
CREATE FUNCTION func_escopo() RETURNS INTEGER AS $$  
DECLARE  
    quantidade INTEGER := 30;  
BEGIN  
    RAISE NOTICE 'Aqui a quantidade é %', quantidade; -- qtde = 30  
    quantidade := 50;  
    -- Criando um sub-bloco  
    DECLARE  
        quantidade INTEGER := 80;  
    BEGIN  
        RAISE NOTICE 'Aqui a quantidade é %', quantidade; -- qtde = 80  
    END;  
    RAISE NOTICE 'Aqui a quantidade é %', quantidade; -- qtde = 50  
    RETURN quantidade;  
END; $$ LANGUAGE plpgsql;  
  
SELECT func_escopo();
```

PostgreSQL: Funções em PL/pgSQL

- ▶ Há diferentes formas de se fazer referência a parâmetros:

```
CREATE FUNCTION minhaFuncao1(VARCHAR, INT) RETURNS INT AS $$  
DECLARE  
    param1 ALIAS FOR $1;    param2 ALIAS FOR $2;  
BEGIN  
    RETURN length(param1) + param2;  
END; $$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION minhaFuncao2(param1 VARCHAR, param2 INT)  
RETURNS INT AS $$  
BEGIN  
    RETURN length(param1) + param2;  
END; $$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION minhaFuncao3(VARCHAR, INT) RETURNS INT AS $$  
BEGIN  
    RETURN length($1) + $2;  
END; $$ LANGUAGE plpgsql;
```

PostgreSQL: Funções em PL/pgSQL

- ▶ O operador de concatenação de valores – ||
 - ▶ O resultado da operação é uma string

```
CREATE FUNCTION concat_atrib_selecionados(tupla nome_da_tabela)
RETURNS text AS $$
BEGIN
    RETURN tupla.a1 || tupla.a3 || tupla.a5 || tupla.a7;
END;
$$ LANGUAGE plpgsql;
```

PostgreSQL: Funções em PL/pgSQL

O operador de concatenação (||) - Exemplo mais concreto

```
CREATE FUNCTION concat_atrib_selecionados(tupla Empregado)
    RETURNS text AS $$
BEGIN
    RETURN tupla.nome || '-' || tupla.salario || '-'
        || tupla.idade;
END;
$$ LANGUAGE plpgsql;

SELECT concat_atrib_selecionados(Empregado.*)
FROM Empregado;
-- Resposta
-- "João-2200-21"
-- "José-4200-30"
```


PostgreSQL: Funções em PL/pgSQL

- ▶ nome_tabela%ROWTYPE – permite criar uma variável do tipo tupla, com a estrutura de uma tupla de nome_tabela
- ▶ Também pode-se usar nome_tabela.nome_coluna.%TYPE, para pegar o tipo de uma coluna

```
CREATE FUNCTION mesclar_campos(t_linha nome_tabela)
RETURNS text AS $$
DECLARE
    t2_linha nome_tabela2%ROWTYPE;
BEGIN
    SELECT * INTO t2_linha FROM nome_tabela2 WHERE ... ;
    RETURN t_linha.a1 || t2_linha.f1 || t_linha.a2 || t2_linha.f2;
END;
$$ LANGUAGE plpgsql;

SELECT mesclar_campos(t.*) FROM nome_tabela t WHERE ... ;
```

PostgreSQL: Funções em PL/pgSQL

- ▶ FOR ... LOOP – manipulação de cursores (semelhante a PSM)
- ▶ SETOF – para retornar conjunto de valores

```
CREATE OR REPLACE FUNCTION obtemNomeEmpregados(NUMERIC)
RETURNS SETOF VARCHAR(20) AS $$
DECLARE
    registro RECORD;
    salario_interesse ALIAS FOR $1;
BEGIN
    FOR registro IN SELECT * FROM Empregado
                    WHERE salario >= salario_interesse LOOP
        RETURN NEXT registro.nome;
    END LOOP;
    RETURN;
END;
$$ LANGUAGE plpgsql;

-- obtém o nome de todos com salário >= 2200
SELECT * FROM obtemNomeEmpregados(2200);
```

PostgreSQL: Funções em PL/pgSQL

- ▶ SETOF de um tipo composto – para retornar conjuntos de registros

```
CREATE TYPE salario_por_idade AS (idade INT, salario NUMERIC);
```

```
CREATE FUNCTION obtemSalariosPorIdade()
```

```
RETURNS SETOF salario_por_idade AS $$
```

```
DECLARE
```

```
    registro RECORD;
```

```
BEGIN
```

```
    FOR registro IN SELECT idade, AVG(salario)
```

```
                    FROM Empregado GROUP BY idade LOOP
```

```
        RETURN NEXT registro;
```

```
    END LOOP;
```

```
    RETURN;
```

```
END $$ LANGUAGE plpgsql;
```

```
-- obtém pares (idade, média de salário)
```

```
select * from obtemSalariosPorIdade();
```

PostgreSQL: Funções em PL/pgSQL

- ▶ É possível usar RETURN QUERY para devolver conjuntos de registros sem usar cursores

```
CREATE TYPE salario_por_idade AS (idade INT, salario NUMERIC);
```

```
CREATE FUNCTION obtemSalariosPorIdade2()
```

```
RETURNS SETOF salario_por_idade AS $$
```

```
BEGIN
```

```
    RETURN QUERY
```

```
        SELECT idade, AVG(salario)
```

```
        FROM Empregado GROUP BY idade;
```

```
END
```

```
$$ LANGUAGE plpgsql;
```

```
-- obtém pares (idade, média de salário)
```

```
select * from obtemSalariosPorIdade2();
```

PostgreSQL: Funções em PL/pgSQL

Exemplo 1 de uso de laços sem cursores - com LOOP

```
CREATE OR REPLACE FUNCTION ContaAteDez() RETURNS SETOF INT AS $$  
DECLARE  
    contador INT := 0;  
BEGIN  
    LOOP  
        IF contador < 10 THEN  
            contador := contador + 1;  
            RETURN NEXT contador;  
        ELSE  
            EXIT; -- sai do laço  
        END IF;  
    END LOOP;  
    RETURN;  
END;  
$$ LANGUAGE plpgsql;
```

PostgreSQL: Funções em PL/pgSQL

Exemplo 2 de uso de laços sem cursores - com LOOP

```
CREATE OR REPLACE FUNCTION ContaAteDez2()
RETURNS SETOF INT AS $$
DECLARE
    contador INT := 0;
BEGIN
    LOOP
        contador := contador + 1;
        RETURN NEXT contador;

        EXIT WHEN contador = 10; -- sai do laço
    END LOOP;
    RETURN;
END;
$$ LANGUAGE plpgsql;
```

PostgreSQL: Funções em PL/pgSQL

Exemplo 3 de uso de laços sem cursores – com WHILE

```
CREATE OR REPLACE FUNCTION ContaAteDez3()  
RETURNS SETOF INT AS $$  
DECLARE  
    contador INT := 0;  
BEGIN  
    WHILE contador < 10 LOOP  
        contador := contador + 1;  
        RETURN NEXT contador;  
    END LOOP;  
    RETURN;  
END;  
$$ LANGUAGE plpgsql;
```

PostgreSQL: Funções em PL/pgSQL

Exemplo de tratamento de exceções

```
CREATE TABLE Departamento(cod INT PRIMARY KEY, nome VARCHAR(20));
CREATE TABLE Funcionario(nome VARCHAR(30) PRIMARY KEY,
                           cod_dept INT REFERENCES Departamento(cod));

CREATE OR REPLACE FUNCTION insereFunc(nome_func VARCHAR(30), cod_dept INT)
RETURNS VOID AS $$
BEGIN
    BEGIN    -- Começa bloco de tratamento de exceções
        INSERT INTO Funcionario VALUES (nome_func, cod_dept);

        EXCEPTION WHEN SQLSTATE '23503' THEN -- violação de foreign key
            BEGIN
                RAISE NOTICE 'O departamento informado para o func. nao existe!';
                RETURN;    -- encerra a execução da função
            END;
        END;
    RAISE NOTICE 'O func. foi inserido com sucesso!';
END;
$$ LANGUAGE plpgsql;
```


Referências Bibliográficas

- ▶ Exemplos de funções em PostgreSQL extraídos de:
PostgreSQL Prático –
http://pt.wikibooks.org/wiki/PostgreSQL_Prático
- ▶ Documentação do PL/PgSQL – <http://www.postgresql.org/docs/9.5/static/plpgsql.html>
- ▶ Lista de códigos de erro para SQLSTATE no PostgreSQL:
<http://www.postgresql.org/docs/9.5/static/errcodes-appendix.html>
- ▶ *Database Systems – A Complete Book* (2ª edição), Garcia-Molina, Ullman e Widom, 2002. – Capítulo 8