[MAC0313]
Introdução aos Sistemas de Bancos de Dados
Aula 21
Stored Procedures

Kelly Rosa Braghetto

DCC–IME–USP

31 de outubro de 2014

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
- Esse tipo de funções ou procedimentos são chamados de ***Stored Procedures***
- 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 PSM captura os principais conceitos existentes nessas linguagens simples implementadas nos SGBDs

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 triplas 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

Exemplo – tabela `EstrelaDeCinema`

```
CREATE TABLE EstrelaDeCinema (
 nome CHAR(30) NOT NULL,
 endereco VARCHAR(255),
 sexo CHAR(1),
 data_nascimento TIMESTAMP WITH TIME ZONE
);
```

- Crie um procedimento que receba como parâmetros de entrada dois endereços – o endereço atual de uma estrela de cinema e o seu novo endereço – e atualize o BD com a mudança de endereço

Funções e procedimentos em PSM

Exemplo

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

```
1) CREATE PROCEDURE Mudanca(
2)     IN endereco_antigo VARCHAR(255),
3)     IN endereco_novo VARCHAR(255)
   )
4) UPDATE EstrelaDeCinema
5) SET endereco = endereco_novo
6) WHERE endereco = endereco_antigo;
```

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 condidional de fluxo

- Estrutura:

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

Funções e procedimentos em PSM

Exemplo – tabela `Filme`

```
CREATE TABLE Filme (
 titulo VARCHAR(255) NOT NULL,
 ano INTEGER NOT NULL,
 duracao INTEGER,
 colorido CHAR(1),
 nome_estudio CHAR(50),
 num_certificado_produtor INTEGER
);
```

- Crie uma função que recebe dois parâmetros – um nome de estúdio *e* e um ano *a* e devolve verdadeiro se o estúdio *e* no ano *a* não produziu filme nenhum ou produziu ao menos uma comédia

Funções e procedimentos em PSM

Exemplo

- A função só devolve verdadeiro se o estúdio *e*, no ano *a*, não produziu filme nenhum ou produziu ao menos uma comédia

```
1) CREATE FUNCTION ProdComedia(a INT, e CHAR(15)) RETURNS BOOLEAN
2) IF NOT EXISTS(
3) SELECT * FROM Filme WHERE ano = a AND nome_estudio = e)
4) THEN RETURN TRUE;
5) ELSEIF 1 <=
6) (SELECT COUNT(*) FROM Filme WHERE ano = a AND
                        nome_estudio = e AND genero = 'comedia')
7) THEN RETURN TRUE;
8) ELSE RETURN FALSE;
9) END IF;
```

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
- Um comando de seleção que retorna uma única tupla pode ser usado para atribuir valores a variáveis
- Consultas podem ser usadas na declaração de cursores

Funções e procedimentos em PSM

Exemplo – tabelas ExecutivoDeCinema e Estudio

```
CREATE TABLE ExecutivoDeCinema (
    num_certificado INTEGER NOT NULL,
    nome CHAR(30),
    endereco VARCHAR(255),
    patrimonio INTEGER
);

CREATE TABLE Estudio (
    nome CHAR(50) NOT NULL,
    endereco VARCHAR(255),
    num_certificado_presidente INTEGER
);
```

Funções e procedimentos em PSM

Exemplo – atribuindo a uma variável o resultado de uma consulta

- Uma consulta com uma única linha de resultado em PSM
- A cláusula INTO no SELECT faz uma atribuição a uma variável

```
CREATE PROCEDURE AlgumProc(IN nome_estudio CHAR(15))
DECLARE patrimonio_pres INTEGER;
SELECT patrimonio
INTO patrimonio_pres
FROM Estudio, ExecutivoDeCinema
WHERE num_certificado_presidente = num_certificado AND
      Estudio.nome = nome_estudio;
...
```

Funções e procedimentos em PSM

Exemplo – atribuindo a uma variável o resultado de uma consulta

- Procedimento equivalente ao mostrado no slide anterior, mas que usa o comando SET para fazer a atribuição de valor à variável:

```
CREATE PROCEDURE AlgumProc(IN nome_estudio CHAR(15))
DECLARE patrimonio_pres INTEGER;
SET patrimonio_pres = (SELECT patrimonio
FROM Estudio, ExecutivoDeCinema
WHERE num_certificado_presidente = num_certificado AND
      Estudio.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

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

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

```
1) CREATE PROCEDURE MediaVariancia(
2)    IN e CHAR(15),
3)    OUT media REAL,
4)    OUT variancia REAL
   )
5) DECLARE Not_Found CONDITION FOR SQLSTATE '02000';
6) DECLARE CursorFilme CURSOR FOR
      SELECT duracao FROM Filme WHERE nome_estudio = e;
7) DECLARE novaDuracao INTEGER;
8) DECLARE contFilme INTEGER;
```

Funções e procedimentos em PSM

Exemplo de uso de LOOP e CURSOR – 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

Cursores

- Um cursor armazena um conjunto de tuplas (pode ser o conteúdo de uma tabela do BD e ou o resultado da execução de um comando SELECT)
- Um cursor nos permite percorrer suas tuplas uma a uma
- Os valores dos atributos de uma tupla de um cursor podem ser carregados em variáveis e "processados" dentro do procedimento ou função

Funções e procedimentos em PSM

Passos envolvidos no uso de um cursor:

1. **declaração do cursor**, que define relação (conjunto de tuplas) à qual o cursor está associado
2. **abertura do cursor**, que deixa o cursor pronto para fornecer a primeira tupla de sua relação
3. **um ou mais usos de um comando *fetch***, que obtém a próxima tupla da relação que o cursor percorre
4. **o fechamento do cursor**

Funções e procedimentos em PSM

Laços do tipo FOR

- São usados somente como iteradores para cursores
- Estrutura geral:

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

- O comando 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 CHAR(15), 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 sistema SQL indica condições de erro setando um código (de 5 caracteres) à variável SQLSTATE
- 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

```
1) CREATE FUNCTION ObtemAno(t VARCHAR(255)) RETURNS INTEGER
2) DECLARE Not_Found CONDITION FOR SQLSTATE '02000' ;
3) DECLARE Too_Many CONDITION FOR SQLSTATE '21000' ;
BEGIN
4)     DECLARE EXIT HANDLER FOR Not_Found, Too_Many
5)         RETURN NULL;
6)     RETURN (SELECT ano FROM Filme WHERE titulo = t ) ;
END;
```

Stored Procedures no PostgreSQL

- O PostgreSQL não diferencia *procedures* de *functions*
- Funções são executadas por meio de comandos SQL comuns:
  - Na cláusula SELECT:
    SELECT func1(atributo1) FROM tabela1;
  - Na cláusula FROM:
    `SELECT * FROM func1();`
  - Na cláusula WHERE:
    `SELECT * FROM tabela1 WHERE func1(atributo1) = 42;`

Funções no PostgreSQL – há 4 tipos:

- Funções escritas 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

PostgreSQL: Funções em SQL "puro"

```
CREATE FUNCTION ola_mundo() RETURNS int
AS $$ SELECT 1 $$ LANGUAGE sql;

-- executa a função
SELECT ola_mundo();

CREATE FUNCTION soma_numeros(IN nr1 int, IN nr2 int)
RETURNS int
AS $$ SELECT nr1 + nr2 $$ LANGUAGE sql;

-- executa a função e renomeia o resultado
SELECT soma_numeros(300, 700) AS resposta;
```

- Tipos dos parâmetros e do valor de retorno: tipos da SQL (int, varchar, char, time, etc.)
- Parâmetros podem ser de entrada (IN), saída (OUT) ou entrada e saída (INOUT)

PostgreSQL: Funções em SQL "puro"

- Atribuir nomes a parâmetros é opcional; é possível referenciar um parâmetro no corpo da função usando as **variáveis posicionais** $1, $2, ...

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

-- executa a função e renomeia o resultado
SELECT soma_numeros(300, 700) AS resposta;
```

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**

```
-- Cria uma tabela temporária com dados de empregados
CREATE TEMP TABLE empregados(nome TEXT, salario NUMERIC, idade INT);

INSERT INTO empregados VALUES('João',2200,21);
INSERT INTO empregados VALUES('José',4200,30);

-- Cria uma função que calcula o dobro do salário de um empregado
CREATE FUNCTION dobrar_salario(e empregados) RETURNS NUMERIC AS $$
        SELECT (e.salario * 2) AS salario;  $$ LANGUAGE SQL;

-- Executa a função para cada tupla na tabela Empregados
SELECT nome, dobrar_salario(empregados.*) AS sonho
FROM empregados WHERE nome = 'João';
-- ou
SELECT nome, dobrar_salario(ROW(nome, salario*1.1, idade))
  AS sonho FROM empregados;
```

PostgreSQL: Funções em SQL "puro"

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

```
CREATE OR REPLACE FUNCTION novo_empregado()
RETURNS empregados AS $$
   SELECT 'Nenhum' AS nome, 1000.0 AS salario, 25 AS idade;
$$ LANGUAGE SQL;
-- OU
CREATE OR REPLACE FUNCTION novo_empregado()
RETURNS empregados AS $$
   SELECT ROW('Nenhum', 1000.0, 25)::empregados;
$$ LANGUAGE SQL;

-- Para executar a função, temos duas opções:
SELECT novo_empregado();
-- ou
SELECT * FROM novo_empregado();
```

PostgreSQL: Funções em SQL "puro"

```
CREATE TEMP TABLE teste (cod INT, dept INT, nome text);
INSERT INTO teste VALUES (1, 1, 'João');
INSERT INTO teste VALUES (1, 2, 'José');
INSERT INTO teste VALUES (2, 1, 'Maria');

CREATE FUNCTION selecionaPorCodigo(INT) RETURNS teste AS $$
   SELECT * FROM teste WHERE cod = $1;
$$ LANGUAGE SQL;

SELECT nome FROM selecionaPorCodigo(1);
```

- Tabelas temporárias (TEMP) são removidas pelo SGBD quando a conexão com o BD é encerrada
- A chamada à função pode aparecer no FROM de uma consulta
- É importante ressaltar que a execução de `selecionaPorCodigo(1)` devolverá uma só tupla!

PostgreSQL: Funções em SQL "puro"

```
CREATE FUNCTION selecionaPorCodigo(INT)
RETURNS SETOF teste AS $$
   SELECT * FROM teste WHERE cod = $1;
$$ LANGUAGE SQL;

SELECT nome FROM selecionaPorCodigo(1);
```

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

PostgreSQL: Funções em PL/pgSQL

- É possível criar blocos, que definem diferentes escopos para as variáveis

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

SELECT func_escopo();
```

Note que, na PL/pgSQL, a atribuição de valores p/ variáveis não possui SET.

PostgreSQL: Funções em PL/pgSQL

- Há diferentes formas de se referenciar parâmetros

```
CREATE FUNCTION func1(VARCHAR, INT) RETURNS INT AS $$
DECLARE
  param1 ALIAS FOR $1;     param2 ALIAS FOR $2;
BEGIN
  RETURN length(param1) + param2;
END; $$ LANGUAGE plpgsql;

CREATE FUNCTION func2(param1 VARCHAR, param2 INT) RETURNS INT AS $$
BEGIN
  RETURN length(param1) + param2;
END; $$ LANGUAGE plpgsql;

CREATE FUNCTION func3(VARCHAR, INT) RETURNS INT AS $$
BEGIN
  RETURN length($1) + $2;
END; $$ LANGUAGE plpgsql;
```

PostgreSQL: Funções em PL/pgSQL

```
CREATE TEMP TABLE teste2(a1 INT, a2 CHAR, a3 NUMERIC, a4 text);
INSERT INTO teste2 VALUES (1, 'A', 2.3, 'BCDEF');
INSERT INTO teste2 VALUES (4, 'G', 5.6, 'HIJKL');

CREATE FUNCTION concatena_valor_atributos(t teste2)
RETURNS text AS $$
BEGIN
   RETURN t.a1 || t.a2 || t.a3 || t.a4;
END;
$$ LANGUAGE plpgsql;

-- Chama a função para cada tupla de teste2
SELECT concatena_valor_atributos(*) FROM teste2;
```

- `t` é uma variável-tupla que contém os mesmos atributos que a tabela `teste2`
- Operador de concatenação de valores – `||`

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 $$
DECLARE
   t2_linha nome_tabela2%ROWTYPE;
BEGIN
  SELECT * INTO t2_linha FROM nome_tabela2 WHERE ... ;
  RETURN t_linha.a1 || t2_linha.b1 || t_linha.a2 || t2_linha.b2;
END;
$$ LANGUAGE plpgsql;

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

PostgreSQL: Funções em PL/pgSQL

- Funções às vezes são criadas para controlar a manipulação dos dados

```
CREATE OR REPLACE FUNCTION data_ctl(opcao CHAR, vdata DATE, vhora TIME)
RETURNS CHAR(10) AS $$
DECLARE
    retorno CHAR(10);
BEGIN
    IF opcao = 'I' THEN
        INSERT INTO datas (data, hora) VALUES (vdata, vhora);
        retorno = 'INSERT';
    ELSIF opcao = 'U' THEN
        UPDATE datas SET data = vdata, hora = vhora WHERE data='1995-11-01';
        retorno = 'UPDATE';
    ELSIF opcao = 'D' THEN
        DELETE FROM datas WHERE data = vdata;
        retorno = 'DELETE';
    ELSE
        retorno = 'NENHUMA';
    END IF;
    RETURN retorno;
END; $$ LANGUAGE plpgsql;
```

PostgreSQL: Funções em PL/pgSQL

- FOR ... LOOP – manipulação de cursores (semelhante à PSM)
- SETOF + RETURN NEXT – para retornar conjunto de valores

```
CREATE OR REPLACE FUNCTION nome_empregado(NUMERIC)
RETURNS SETOF TEXT AS $$
DECLARE
   registro RECORD;
   sal ALIAS FOR $1;
BEGIN
    FOR registro IN SELECT * FROM empregados
                    WHERE salario >= sal LOOP
            RETURN NEXT registro.nome;
    END LOOP;
    RETURN;
END;
$$ LANGUAGE plpgsql;
SELECT * FROM nome_empregado (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 obtem_salarios()
RETURNS SETOF salario_por_idade AS $$
DECLARE
   registro RECORD;
BEGIN
   FOR registro IN SELECT idade, AVG(salario)
                   FROM empregados GROUP BY idade LOOP
          RETURN NEXT registro;
   END LOOP;
   RETURN;
END  $$ LANGUAGE plpgsql;

select * from obtem_salarios();
```

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/9.3/static/plpgsql.html`
- Lista de códigos de erro para SQLSTATE no PotgreSQL: `http://www.postgresql.org/docs/9.3/static/errcodes-appendix.html`
- *Database Systems – A Complete Book* (2ª edição), Garcia-Molina, Ullman e Widom, 2002. – Capítulo 8