BANCO DE DADOS II

Stored Procedures

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE

Msc. Eliezio Soares
eliezio.soares@ifrn.edu.br | https://docente.ifrn.edu.br/elieziosoares

MÓDULOS ARMAZENADOS PERSISTENTES (PSM - PERSISTENT STORED MODULES)

Funções definidas por usuário e Stored Procedures são programas armazenados no servidor de banco de dados a fim de propciar situações como:

- Aumentar o reaproveitamento de código e melhorar a modularidade em contextos em que um banco de dados é utilizado por várias aplicações.
- Reduzir a transferência de dados e o custo de comunicação entre o cliente e o servidor de banco de dados.
- Contribuir para a manipulação de tipos complexos de dados montados pelos procedimentos.

MÓDULOS ARMAZENADOS PERSISTENTES (PSM - PERSISTENT STORED MODULES)

Uma stored procedure ou user-defined function (UDF) é um conjunto de comandos SQL e procedurais (declarações, atribuições, loops, condicionais etc) armazenados no servidor de banco de dados e que pode ser invocados por clientes desse servidor.

Normalmente as diferenças entre Stored Procedures e User Difined Functions são:

	Stored Procedure	Function
Usa-se em uma expressão	Não	Sim
Retorna um valor	Não	Sim
Retorna valor como parâmetro OUT	Sim	Não
Retorna um result set único	Sim	Sim
Retorna múltiplos result sets	Sim	Não

MÓDULOS ARMAZENADOS PERSISTENTES (PSM - PERSISTENT STORED MODULES)

É comum ainda que Funções:

- Não possam ser usadas para executar ações que modificam o estado do banco de dados.
- Não possam retornar vários conjuntos de resultados.
- Não dê suporte a TRY…CATCH etc.
- Não possam chamar um procedimento armazenado.
- Não possam fazer uso de SQL dinâmico ou tabelas temporárias.

Nesses casos seria indicado o uso de Stored Procedures.

MÓDULOS ARMAZENADOS PERSISTENTES (PSM - PERSISTENT STORED MODULES)

No entanto, O POSTGRESQL OPTA POR UMA SOLUÇÃO mais prática (e menos elegante), JUNTANDO OS DOIS CONCEITOS NO RECURSO FUNCTION.

```
CREATE [ OR REPLACE ] FUNCTION
    name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = }
default_expr ] [, ...] ] )
    [ RETURNS rettype
      | RETURNS TABLE ( column_name column_type [, ...] ) ]
  { LANGUAGE lang_name
    | TRANSFORM { FOR TYPE type_name } [, ... ]
    | WINDOW
    | IMMUTABLE | STABLE | VOLATILE | [ NOT ] LEAKPROOF
    | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
    | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
    | COST execution_cost
    | ROWS result_rows
    | SET configuration_parameter { TO value | = value | FROM CURRENT }
    | AS 'definition'
    | AS 'obj_file', 'link_symbol'
  } ...
    [ WITH ( attribute [, ...] ) ]
```

STORED PROCEDURES

As características, portanto, mais marcante de uma Stored Procedure no PostgreSQL são:

- SPs não retornam valores;
- SPs utilizam PLPGSQL;

```
CREATE OR REPLACE FUNCTION calculosMatematicos(x int, y int, OUT soma
int, OUT subtracao int, OUT multiplicacao int, OUT divisao int) AS $$
BEGIN
    soma = x + y;
    subtracao = x - y;
    multiplicacao = x * y;
    divisao = x / y;
END;
$$ LANGUAGE plpgsql;
```

```
SELECT * FROM CALCULOSmATEMATICOS(2,5);
```

STORED PROCEDURES

Para retornar um resultset de uma query utilizando PLPGSQL utiliza-se o comando RETURN QUERY:

```
CREATE OR REPLACE FUNCTION gerarboletim(mat integer)
RETURNS TABLE(mat_alu integer, nom_alu character varying, cod_disc integer,
semestre integer, media double precision, faltas double precision, situacao character)
AS
$$
BEGIN
   RETURN QUERY
            SELECT A.mat_alu, A.nom_alu, H.cod_disc, H.semestre, H.media, H.faltas,
H.situacao FROM Alunos A
            INNER JOIN Historicos_Escolares H ON A.mat_alu = H.mat_alu
            WHERE A.mat_alu = mat
            ORDER BY A.nom_alu, H.Semestre;
END;
$$ LANGUAGE plpgsql;
```

```
SELECT * FROM gerarboletim(912548);
```

STORED PROCEDURES

Caso a Stored Procedure não utilize parâmetros OUT, deve então indicar que não há retorno.

```
CREATE OR REPLACE FUNCTION Renomear_aluno(matricula integer,nome text)
RETURNS void AS
$$
BEGIN
      UPDATE alunos
      set nom_alu = nome
      where mat_alu = matricula;
END;
$$
LANGUAGE PLPGSQL;
```

```
SELECT * FROM Renomear_aluno(912548, 'GARRINCHA')
```

PLPGSQL IF E CASE

Declarações IF e CASE permitem a execução de comandos baseada em condições.

```
IF boolean-expression THEN
    statements
END IF;
```

```
IF boolean-expression THEN
    statements
ELSE
    statements
END IF;
```

```
CASE search-expression
    WHEN expression [, expression [ ... ]] THEN
      statements
  [ WHEN expression [, expression [ ... ]] THEN
      statements
    ... ]
  [ ELSE
      statements ]
END CASE;
```

PLPGSQL IF E CASE

```
IF parentid IS NULL OR parentid = ''
THEN
    RETURN fullname;
ELSE
    RETURN hp_true_filename(parentid) || '/' || fullname;
END IF;
```

```
IF number = 0 THEN
    result := 'zero';
ELSIF number > 0 THEN
    result := 'positive';
ELSIF number < 0 THEN
    result := 'negative';
ELSE
    -- hmm, the only other possibility is that number is null
    result := 'NULL';
END IF;
```

```
CASE x
    WHEN 1, 2 THEN
        msg := 'one or two';
    ELSE
        msg := 'other value than one or two';
END CASE;
```

DECLARAÇÃO DE VARIÁVEIS

Para declarar variáveis a serem utilizadas dentro do procedimento, utiliza-se a seção **DECLARE**.

As variáveis são declaradas com um identificador e um tipo.

```
CREATE OR REPLACE FUNCTION buscar_aluno(nome text)
RETURNS SETOF Alunos
AS $$
DECLARE
        id integer;
BEGIN

    RETURN QUERY SELECT * FROM ALUNOS WHERE nom_alu like '%' || nome ||'%';

END;
$$
LANGUAGE PLPGSQL;
```

ATRIBUIÇÃO DE VALOR

Para atribuir valor a uma variável se utiliza a sintaxe:

```
id_usuario := 20;
taxa := subtotal * 0.06;
```

```
CREATE OR REPLACE FUNCTION buscar_aluno(nome text)
RETURNS SETOF Alunos
AS $$
DECLARE
        id integer;
BEGIN

    RETURN QUERY SELECT * FROM ALUNOS WHERE nom_alu like '%' || nome ||'%';

END;
$$
LANGUAGE PLPGSQL;
```

ATRIBUIÇÃO DE VALOR

O comando **SELECT INTO** permite fazer atribuição a uma variável escalar a partir de um select.

A variável especial FOUND pode ser verificada imediatamente após a instrução SELECT INTO para determinar se a atribuição foi bem-sucedida, ou seja, foi retornada pelo menos uma linha pela consulta.

```
CREATE OR REPLACE FUNCTION buscar_aluno(nome text) RETURNS integer AS $$
DECLARE
        id integer;
BEGIN

        SELECT INTO     id      mat_alu    from alunos
        where nom_alu like '%' || nome ||'%';

        IF NOT FOUND THEN
                RETURN -1;
        ELSE
                RETURN id;
        END IF;
END;
$$
LANGUAGE PLPGSQL;
```

COMUNICANDO ERROS

O comando **RAISE** pode ser utilizado para reportar mensagens e levantar exceções.

```
CREATE OR REPLACE FUNCTION buscar_aluno(nome text) RETURNS integer AS $$
DECLARE
        id integer;
BEGIN

        SELECT INTO     id      mat_alu    from alunos
        where nom_alu like '%' || nome ||'%';

        IF NOT FOUND THEN
                RAISE EXCEPTION 'não foi encontrado o(a) aluno %', nome;
        ELSE
                RETURN id;
        END IF;
END;
$$
LANGUAGE PLPGSQL;
```

DÚVIDAS?

REFERÊNCIAS BIBLIOGRÁFICAS

PostgreSQL 9.0.22 Documentation. Disponível em: <https://www.postgresql.org/files/documentation/pdf/9.0/postgresql-9.0-US.pdf>. Acesso em 28 Set. 2017.

Stored Procedures and Functions in PostgreSQL - Getting Started. Disponível em: <http://www.sqlines.com/postgresql/stored_procedures_functions>. Acesso em 28 Set. 2017.

Trabalhando com Stored Procedures no PostgreSQL. Disponível em: <http://www.devmedia.com.br/trabalhando-com-stored-procedures-no-postgresql/33354> Acesso em 28 Set. 2017.