

Atualizado em 24/10/2019

PL/pgSQL – SQL Procedural Language: Procedimentos e Funções no PostgreSQL

Alex Sandro
alex@ifpb.edu.br

- *PL/pgSQL*
 - *Recursos*
- *Estruturas de controle*
 - *IF, WHILE, CASE*
- *Stored Procedure e Stored Function*
 - *Características*
 - *Sintaxe*
 - *Codificação*

■ Significado

- **PL/pgSQL** - *Procedural Language/postgreSQL*
- *Uma linguagem estruturada estendida o padrão SQL que tem por objetivo introduzir tarefas de programação no PostgreSQL.*

■ Características

- *Permite construir programas no PostgreSQL similares ao que é feito em outras linguagens*
- *A lógica de programação (relacionada ao acesso e escrita) pode ser direcionada para o próprio bando de dados*
- *Pode ser usada para definir a lógica em triggers*

- **Recursos**
 - *Declaração de variáveis*
 - *Tipos de dados*
 - *Operadores*
 - *Estruturas de Seleção*
 - *Estruturas de Repetição*
 - ***Funções de Procedimentos***
 - *Cursores*
 - *Tratamento de Exceções*

■ *Por que aprender a PL/pgSQL?*

“Com a PL/pgSQL, você pode agrupar um bloco de instruções e uma série de *queries* a rodar dentro do servidor de banco de dados, com considerável economia de sobrecarga de comunicação **cliente/servidor**”.

PostgreSQL Documentation

- *Evita-se a submissão de várias queries de análise*
- *Resultados intermediários podem ser omitidos e não precisam ser transferidos entre cliente/servidor*

■ Estrutura básica: definição de um bloco

```
DO $$  
[DECLARE  
    declarações de variáveis; ]  
BEGIN  
    instruções;  
END $$;
```

- Todas as palavras-chaves são *case-insensitive*
- Cada **declaração** ou **instrução** deve vir seguida de ;
- Blocos podem aparecer dentro de outros blocos
 - Os blocos internos devem ter um ; depois do **END**
 - O ultimo **END** não necessita de ;
- Comentários: -- (linha) e /* */ (bloco)

■ Variáveis

- *Um nome significativo para um local na memória, associado a um tipo, cujo valor pode ser lido/alterado.*
- *Recebem valores com **:=** ou **SELECT INTO***

```
nome_variável tipo [:= expressão]
```

```
DO $$  
DECLARE  
    pos integer := 1;  
    name varchar(40) := 'Diogo';  
    valor numeric(11,2) := 40.10;  
BEGIN  
    RAISE NOTICE '%- % pagou % Reais', pos,name,valor;  
END $$;
```

Estruturas de decisão (condicionais)

■ *Estrutura de decisão IF-ELSE*

- *Estrutura que executa uma sequência de instruções se a condição for **verdadeira***

```
IF <condição> THEN
    instruções pl/pgsql;
[ELSE
    instruções alternativas pl/pgsql;]
END IF;
```

- *O uso do ELSE é opcional*
- *As condições devem expresser um resultado lógico (**true** ou **false**)*
 - *Uso de operadores relacionais*
- *Aceita IF's aninhados*

- **Estrutura de decisão IF-ELSE**

- *Aceita estrutura IF-ELSE IF (em escada)*

```
IF <condição1> THEN
    instruções pl/pgsql-1;
ELSIF <condição-2> THEN
    instruções pl/pgsql-2 ;]
...
[ELSE
    instruções alternativas pl/pgsql;]
END IF;
```

- *A condição pode usar operadores tais como IN, IS NULL, EXISTS, etc.*

■ **Estrutura de decisão CASE**

- *Oferece execução condicional com base na igualdade de operandos*

```
CASE <condição>
  WHEN valor1 [,valor2,...] THEN
    resultado ou expressão
  [WHEN valor3 [,valor4,...] THEN
    resultado ou expressão]
  ...
  [ELSE
    resultado ou expressão]
END CASE;
```

- *Pode ser utilizado diretamente em instruções SELECT*

- **Exemplo 1:** Criar um bloco PL/pgSQL para exibir uma mensagem personalizada se um cliente estiver cadastrado e já tiver feito mais de 20 locações

```
DO $$
DECLARE
    nRentals integer := 20;
    customer_email varchar(40) := 'junior@gmail.com';
BEGIN
    IF EXISTS (SELECT email FROM customer
               WHERE email = customer_email) THEN
        RAISE NOTICE 'Bem-vindo %', customer_email;
        IF (SELECT count(*) FROM payment p, customer c
            WHERE c.customer_id = p.customer_id and
                  c.email = customer_email ) > nRentals THEN
            RAISE NOTICE 'Voce já realizou mais de %
                           pedidos', nRentals;
        END IF;
    END IF;
END $$;
```

- **Exemplo 2:** Criar um bloco PL/pgSQL para informar em qual trimestre foi feita uma compra em uma data específica.

```
DO $$  
DECLARE  
    data date := CAST('2019-09-01' AS DATE);  
BEGIN  
    CASE date_part('month',data)  
        WHEN 1,2,3 THEN  
            RAISE NOTICE '1o Trimestre';  
        WHEN 4,5,6 THEN  
            RAISE NOTICE '2o Trimestre';  
        WHEN 7,8,9 THEN  
            RAISE NOTICE '3o Trimestre';  
        ELSE  
            RAISE NOTICE '4o Trimestre';  
    END CASE;  
END $$;
```

- **Exemplo 3:** Criar uma instrução SQL que exiba o destinatário e o status (se foi lida ou não, por extenso) de todas mensagens enviadas

```
SELECT m.assunto, r.destinatario,  
       CASE r.lida  
         WHEN 0 THEN 'Não lida'  
         WHEN 1 THEN 'LIDA'  
         ELSE 'desconhecido'  
       END as Situação, r.lida  
FROM si_mensagem m JOIN si_receptor r ON m.idMens =  
r.idMens;
```

No SELECT, feche
com **END**, e não com
END CASE

Estruturas de repetição (loops)

■ **Estrutura de Repetição WHILE**

- *Executa um bloco de comandos repetidamente enquanto uma determinada condição for avaliada como verdadeira.*

■ **Sintaxe**

```
WHILE <expressão-lógica> LOOP
    instruções pl/pgsql;
    [CONTINUE [WHEN <expressão-lógica>;]]
    [EXIT [WHEN <expressão-lógica>;]]
END LOOP;
```

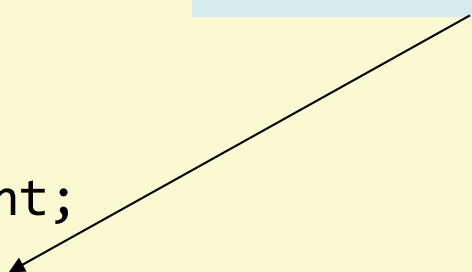
- *A instrução EXIT encerra a execução do laço e direciona o fluxo para a próxima instrução depois do END LOOP*
- *O CONTINUE força a execução da primeira instrução do LOOP*

■ Estrutura de Repetição WHILE

■ EXIT e CONTINUE em ação

```
DO $$  
DECLARE  
    count integer := 0;  
BEGIN  
    WHILE count >= 0 LOOP  
        count := count + 1;  
        RAISE NOTICE '%', count;  
        EXIT WHEN count > 70;  
        CONTINUE WHEN count < 50;  
        RAISE NOTICE 'Passou do CONTINUE';  
    END LOOP;  
END $$;
```

Mesmo que:
IF count > 70 THEN
 EXIT;
END IF;



■ **Estrutura de Repetição LOOP**

- *Estrutura de repetição incondicional (necessita de uma instrução condicional associada a um EXIT para quebrar o fluxo de repetição).*

LOOP

```
instruções pl/pgsql;  
[CONTINUE [WHEN <expressão-lógica>;]]  
[EXIT [WHEN <expressão-lógica>;]]  
END LOOP;
```

- *O fluxo pode ser interrompido, também, com uma instrução **RETURN** (encerra todo o bloco)*

■ **Estrutura de Repetição FOR**

- *Loop que itera sob uma faixa de valores inteiros*

```
FOR var IN [REVERSE] inicio..fim [BY step] LOOP
    instruções pl/pgsql;
    [CONTINUE [WHEN <expressão-lógica>;]]
    [EXIT [WHEN <expressão-lógica>;]]
END LOOP;
```

- **REVERSE:** *ordem descendente dos valores*

```
FOR i IN 1..10 LOOP
    -- i recebe os valores 1,2,...,10
END LOOP;
FOR i IN REVERSE 10..1 BY 2 LOOP
    -- i recebe os valores 10,8,6,4,2
END LOOP;
```

■ **Estrutura de Repetição FOR - QUERY**

- *Loop que permite iterar através do resultado de uma query*

```
FOR record_var IN query LOOP
    instruções pl/pgsql;
END LOOP;
```

```
DO $$
DECLARE
    linha RECORD;
BEGIN
    FOR linha IN SELECT film_id,title FROM film LOOP
        RAISE NOTICE 'id=?: %', linha.film_id,
            linha.title;
    END LOOP;
END $$;
```

- ***Estrutura de Repetição***

- *Veja também:*

FOR-IN-EXECUTE

Loop em arrays

- ***Fonte***

- *41.6 Controle de Estruturas (documentation)*

- <https://www.postgresql.org/docs/9.6/plpgsql-control-structures.html>

Procedimentos/Funções armazenados

■ **Definição**

- *Blocos de código nomeados definidos pelo usuário que utilizam vários elementos da PL/pgSQL*
 - *São pré-compilados e armazenados no servidor*

■ **Composição**

- *Composta por uma área de declaração de variáveis + área para comandos de fluxo lógico + uma área opcional para tratamento de erros + comandos SQL*

■ **Características**

- *Podem ser armazenados no banco de dados e acionados por qualquer programa aplicativo que tenha autorização para execução;*

Procedimentos/Funções armazenados

- **Características (continuação)**
 - Uma *Stored Procedure (sp/f)* é executada no *lado do servidor* e seu plano de execução fica na memória, *agilizando* as próximas chamadas
 - Podem receber um ou mais parâmetros formais
 - Podem devolver um valor como saída (*sf*)
 - Uma *sp/sf* pode chamar outra *sp/sf* dentro do seu corpo
 - *Sp's* não retornam valor nem podem ser utilizadas em expressões

■ **Desempenho**

- *Única compilação do subprograma*
- *Diminui o tráfego na rede por requisição de dados*

*SELECT * FROM tabela1*

■ **Manutenção**

- *Facilita o gerenciamento (reuso)*
- *Encapsulam rotinas de uso frequente no próprio servidor, estando disponível para várias aplicações*
- *Parte da lógica do sistema pode ser armazenada no próprio BD, em vez de ser codificada em várias aplicações*

Stored Procedures

■ **Segurança**

- *Uso de stored procedure/function para **limitar o acesso a alguns usuários** do banco*

■ **Justificativa:**

- *Você não deseja fornecer acesso às suas tabelas*
- *Não existe necessidade de se criar uma visão*

■ **Solução:**

- *Criar uma stored procedure com o **comando SQL desejado** em seu interior*
- *Conceder permissão ao usuário para que ele **apenas possa executar** o subprograma*

Stored Procedures

■ **Desvantagens**

- *Afeta a produtividade no desenvolvimento de software, pois a programação de **sp** requer habilidade especializada e muitos não possuem*
- *Difícil de gerenciar versões e realizar **debug***
- *A portabilidade não é garantida para outros SGBDs, como por exemplo, SQL Server e MySQL*

Stored Procedure

■ *Sintaxe*

```
CREATE [OR REPLACE] PROCEDURE name (  
    [arg1][,arg2][,argn] )  
LANGUAGE nome_da_linguagem  
AS $$  
    instruções;  
$$;
```

- O **nome** da procedure deve seguir as regras para criação de identificadores
- Argumentos: **identificador + tipo**, separado por ,
nome varchar(45), idade integer
- Seções **DECLARE**, **BEGIN** e **END** aparecem após o **AS \$\$**

Stored Procedure

■ *Sintaxe*

- A cláusula **LANGUAGE** indica o nome da linguagem utilizada no corpo do código: SQL, C, ou plpgsql
- O **\$\$** é uma substituição à aspas simples (‘)
- Dentro de uma **stored procedure**, é necessário passar o **body** sob a forma de uma string literal. Com o **\$\$** Evita-se a necessidade o problema de dupla ‘’
- Não pode conter chamadas do tipo **CREATE PROCEDURE, CREATE TRIGGER, CREATE VIEW**
- Não adicione **SELECT** solto dentro de uma sp com o intuito de visualizar o result set

Stored Procedure

■ Tipos de dados

■ Caracteres

- **char(n)**: armazena uma string de tamanho fixo
- **varchar(n)**: armazena uma string de tamanho variável, até no máximo **n**.
- **text**: string de tamanho variável, sem limite de tamanho

■ Inteiro

Name	Storage Size	Min	Max
<small>SMALLINT</small>	2 bytes	-32,768	+32,767
<small>INTEGER</small>	4 bytes	-2,147,483,648	+2,147,483,647
<small>BIGINT</small>	8 bytes	-9,223,372,036,854,775,808	+9,223,372,036,854,775,807

Criação de Stored Procedure

- **Tipos de dados**

- **Numérico**

- **double precision**: ponto flutuante com precisão de até 15 dígitos decimais
 - **real**: valores ponto flutuante com até 6 dígitos decimais dupla
 - **numeric** ou **numeric(p,s)**: para valores monetários e cálculos exatos

- **Boolean**

- **boolean**: (1 byte) true / false

Criação de Stored Procedure

- ***Tipos de dados***

- ***Data e hora***

- ***date***: (4 bytes) Data – sem hora

- ***timestamp***: (8 bytes) Data e hora

- ***Chamada***

```
CALL nome_procedure( argumentos );
```

Criação de Stored Procedure

- **Exemplo1**: chamada de um **bloco anônimo** da estrutura PL/pgSQL – Exibir a quantidade de registros da tabela **actor** (dvdrental)

```
DO $$  
DECLARE  
    count integer := 0;  
BEGIN  
    SELECT INTO count count(*) FROM actor;  
    RAISE NOTICE 'Resultado = %', count;  
END $$;
```

NOTICE: Resultado = 200

DO

Query returned successfully in 50 msec.

Criação de Stored Procedure

- **Exemplo2:** Criar uma procedure que exiba a quantidade de filmes em que um determinado ator participou.

```
CREATE OR REPLACE PROCEDURE showNumFilms( pri_nome varchar(45),
ult_nome varchar(45))
LANGUAGE plpgsql
AS $$
DECLARE
    contador integer := 0;
BEGIN
    SELECT INTO contador count(*) FROM actor a, film_actor fa, film f
    WHERE a.actor_id = fa.actor_id AND fa.film_id = f.film_id AND
    a.first_name = pri_nome AND a.last_name= ult_nome;
    RAISE NOTICE '% % Fez % filmes', pri_nome, ult_nome, contador;
END $$;
CALL showNumFilms('Ed','Chase')
```

Criação de Stored Procedure

- **Exemplo3:** Criar uma *sp* que transfira uma determinada quantia de uma conta para outra.

```
CREATE OR REPLACE PROCEDURE transferencia(integer, integer, numeric)
LANGUAGE plpgsql
AS $$
BEGIN
    -- subtraindo a quantia da conta de origem
    UPDATE contas
    SET saldo = saldo - $3
    WHERE idConta = $1;
    -- adicionando a quantia na conta de destino
    UPDATE contas
    SET saldo = saldo + $3
    WHERE idConta = $2;

    COMMIT;
END $$;
CALL transferencia(4528,3759,1500.00)
```

■ *Sintaxe*

```
CREATE [OR REPLACE] FUNCTION name (  
    [arg1][,arg2][,argn] )  
RETURNS tipo  
LANGUAGE nome_da_linguagem  
AS $$  
BEGIN  
    instruções-lógica-do-negócio;  
END; $$
```

- *Dentro da função deve haver o comando **RETURN** <valor>;*
- *Aceita **overloading** de funções*
- **Não aceita** a abertura de transações

- **Exemplo 4:** Crie uma *sf* que receba o código de um filme e informe quanto já foi arrecadado com aluguel do referido filme

```
CREATE OR REPLACE FUNCTION arrecadacao( idFilme integer)
RETURNS numeric
LANGUAGE plpgsql
AS $$
DECLARE
    total numeric;
BEGIN
    SELECT INTO total SUM(p.amount)
    FROM payment p, rental r, inventory i, film f
    WHERE p.rental_id = r.rental_id AND r.inventory_id = i.inventory_id
        AND i.film_id = f.film_id AND f.film_id = idFilme;
    RETURN total;
END $$;

--
SELECT arrecadacao(1);
```

Functions: Overloading in Action

- **Exemplo 5:** Crie uma *sf* que retorne a duração de dias (soma) em que um cliente específico locou DVD's

```
CREATE OR REPLACE FUNCTION get_rental_duration(p_customer_id INTEGER)
  RETURNS INTEGER AS $$
DECLARE
  rental_duration INTEGER;
BEGIN
  SELECT INTO rental_duration SUM( EXTRACT( DAY FROM return_date -
rental_date))
    FROM rental
   WHERE customer_id=p_customer_id;

  RETURN rental_duration;
END; $$
LANGUAGE plpgsql;
--
SELECT get_rental_duration(232);
```

Functions: Overloading in Action

- **Exemplo 6:** *Crie uma sf que retorne a duração de dias (soma) em que um cliente específico locou DVD's, a partir de uma data específica*

```
CREATE OR REPLACE FUNCTION get_rental_duration(p_customer_id INTEGER,  
p_from_date DATE)  
  RETURNS INTEGER AS $$  
DECLARE  
  rental_duration INTEGER;  
BEGIN  
  SELECT INTO rental_duration SUM( EXTRACT( DAY FROM return_date -  
rental_date))  
    FROM rental  
   WHERE customer_id=p_customer_id AND rental_date >= p_from_date;  
  
  RETURN rental_duration;  
END; $$  
LANGUAGE plpgsql;  
SELECT get_rental_duration(232, '2005-07-01');
```

- **Funções que retornam uma “tabela”**
 - *Definem como tipo de retorno uma **tabela**, que adiante receberá os dados de um SELECT*
 - *A cláusula **RETURNS** especifica a palavra chave **TABLE***

```
CREATE [OR REPLACE] FUNCTION name ( args )  
RETURNS TABLE ( colunas da tabela)  
LANGUAGE nome_da_linguagem  
AS $$  
BEGIN  
    instruções-lógica-do-negócio;  
    RETURN QUERY SELECT-FROM-WHERE;  
END; $$
```

- ***Funções que retornam uma “tabela”***
 - *Dentro da função, define-se um **return query** que é o resultado de uma instrução **SELECT***
 - *As colunas do **SELECT** devem ser compatíveis com as colunas da tabela a qual se deseja retornar.*
 - *Funções que retornam **tabela** devem ser posicionadas na seção **FROM** de um **SELECT***
 - ***SELECT** externo que aciona a função*

Functions

- **Exemplo 7:** Crie uma *sf* que recebe um padrão de strings aplicado ao operador LIKE e retorne uma tabela contendo todos os filmes (nome e data lançamento) compatíveis

```
CREATE OR REPLACE FUNCTION get_film (padrao VARCHAR)
  RETURNS TABLE (
    film_title VARCHAR,
    film_release_year INT
  )
AS $$
BEGIN
  RETURN QUERY
    SELECT title, cast( release_year as integer)
    FROM film
    WHERE title ILIKE padrao;
END; $$
LANGUAGE 'plpgsql';
SELECT * FROM get_film('A1%');
```

Tabela

Posicionamento: FROM

- **Exemplo 7:** Crie uma *sf* que recebe um padrão de strings aplicado ao operador LIKE e retorne uma tabela contendo todos os filmes (nome e data lançamento) compatíveis

	film_title character varying	film_release_year integer
1	Alabama Devil	2006
2	Aladdin Calendar	2006
3	Alamo Videotape	2006
4	Alaska Phantom	2006
5	Ali Forever	2006
6	Alice Fantasia	2006
7	Alien Center	2006
8	Alley Evolution	2006
9	Alone Trip	2006
10	Alter Victory	2006

EXERCÍCIOS PL/pgSQL

- **Exemplo 8:** Criar uma procedure para eliminar um usuário cujo código é passado por parâmetro, desde que ele não tenha recebido qualquer mensagem (usar o esquema **myMail**)

```
CREATE OR REPLACE PROCEDURE deleteUser (login VARCHAR)
AS $$
DECLARE
    ...
BEGIN
    ...

    ...
END; $$
LANGUAGE 'plpgsql';
```

EXERCÍCIOS PL/pgSQL

- **Exemplo 9:** Criar uma stored procedure que informe se o preço de venda de um determinado produto é maior, menor ou igual a média do preço de venda de todos os produtos da empresa (esquema SCP)

```
CREATE OR REPLACE PROCEDURE showMessage (codProd integer)
AS $$
DECLARE
    ...
BEGIN
    ...

    ...
END; $$
LANGUAGE 'plpgsql';
```

EXERCÍCIOS PL/pgSQL

- **Exemplo 10:** Criar uma stored procedure que reajuste o preço dos produtos de acordo com as seguintes regras:
 - O percentual de aumento dos produtos e a nova média de preço será fornecido como parâmetro
 - Os produtos só sofrerão reajustes desde que a média de preço calculada for inferior ao valor passado como parâmetro. O processo se repete até atingir a condição
 - Exiba na tela a quantidade de reajustes

- *Structure of PL/pgSQL:*
<https://www.postgresql.org/docs/9.0/plpgsql-structure.html>
- *PostgreSQL Stored Procedures:*
<http://www.postgresqltutorial.com/postgresql-stored-procedures/>
- *Create Procedure:*
<https://www.postgresql.org/docs/11/sql-createprocedure.html>
- *Create Function:* <https://www.postgresql.org/docs/9.0/sql-createfunction.html>
- *Trigger Procedures:*
<https://www.postgresql.org/docs/9.3/plpgsql-trigger.html>

- *PL/pgSQL function that returns a table:*
<http://www.postgresqltutorial.com/plpgsql-function-returns-a-table/>