

# Stored Procedures

---



# O que é Stored Procedures?

---

- ❖ **Rotinas Armazenadas** são um conjunto de comandos SQL armazenados em SGBD (Sistema Gerenciador de Banco de Dados)
- ❖ **Stored Procedure** são rotinas armazenadas que devem ser executadas por intermédio de invocações explícitas do usuário



# Sintaxe Stored Procedure

---

**CREATE OR REPLACE FUNCTION** [Nome da  
Procedure]

**RETURN** [Tipo de Retorno]  
\$\$

**DECLARE**  
[Variáveis]

**BEGIN**  
[Comando]

**END**  
\$\$



# Funções -SQL

---

```
tpch=> CREATE OR REPLACE FUNCTION olamundo() RETURNS int4  
AS 'SELECT 1' LANGUAGE 'sql';
```

```
tpch=> select olamundo();
```



# Funções Procedure - SQL

---

```
tpch=> CREATE OR REPLACE FUNCTION add_numeros(nr1 int4, nr2 int4) RETURN  
S int4  
AS 'SELECT $1 + $2' LANGUAGE 'sql';
```

```
tpch=> SELECT add_numeros(300, 700) AS resposta ;
```



# Sintaxe Stored Procedure

---

**CREATE OR REPLACE FUNCTION** [Nome da  
Procedure]

**RETURN** [Tipo de Retorno]

**\$\$**

**DECLARE**

[Variáveis]

**BEGIN**

[Comando]

**END**

**\$\$**



**Comando para criação**



# Sintaxe Stored Procedure

---

**CREATE OR REPLACE FUNCTION** [Nome da  
Procedure]

**RETURN** [Tipo de Retorno]

\$\$

**DECLARE**

[Variáveis]

**BEGIN**

[Comando]

**END**

\$\$



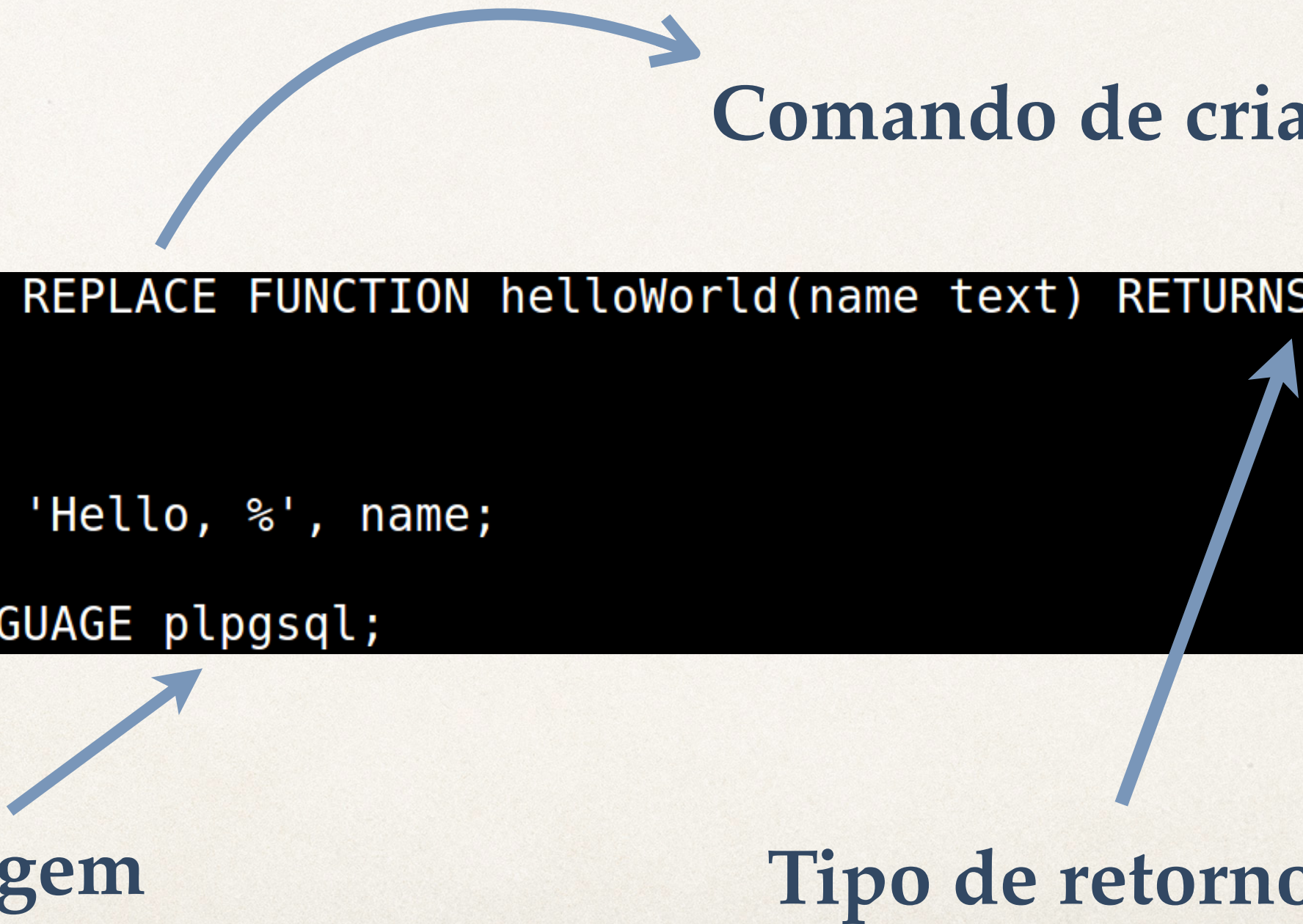
Corpo da função



# Stored Procedure

---

Comando de criação



```
tpch=> CREATE OR REPLACE FUNCTION helloWorld(name text) RETURNS void AS  
$helloWorld$  
DECLARE  
BEGIN  
    RAISE NOTICE 'Hello, %', name;  
END;  
$helloWorld$ LANGUAGE plpgsql;
```

Linguagem

Tipo de retorno



# Stored Procedure - Execução

---

```
tpch=> SELECT helloworld('Simone');
```



# Stored Procedure - Variáveis

---

- ❖ Declarando variáveis:

- ❖ **DECLARE**

- quantidade integer;**

- ❖ Atribuindo valores:

- ❖ **DECLARE**

- ❖ **quantidade integer :=10;**



# Stored Procedure

---

```
tpch=> CREATE FUNCTION func_escopo() RETURNS integer AS $$  
DECLARE  
quantidade integer := 30;  
BEGIN  
RAISE NOTICE 'Aqui a quantidade é %', quantidade;  
quantidade := 50;  
  
DECLARE  
quantidade integer := 80;  
BEGIN  
RAISE NOTICE 'Aqui a quantidade é %', quantidade;  
END;  
RAISE NOTICE 'Aqui a quantidade é %', quantidade;  
RETURN quantidade;  
END;  
$$ LANGUAGE plpgsql;■
```



# Stored Procedure

---

```
tpch=> select func_escopo();  
NOTA: Aqui a quantidade é 30  
NOTA: Aqui a quantidade é 80  
NOTA: Aqui a quantidade é 50  
func_escopo  
-----  
50  
(1 row)
```



# Stored Procedure

---

- ❖ Criada para ser utilizada mais de uma vez;



# Stored Procedure

---

- ❖ Criada para ser utilizada mais de uma vez;

**DROP FUNCTION nome\_da\_função();**



# Stored Procedure - Parâmetros

---

- ❖ A maioria das store procedures necessitam de parâmetros.
- ❖ Parâmetros de entrada **IN**.
  - ❖ Quem chama passa argumentos para a stored procedure;
  - ❖ Os parâmetros são passados por valor;
- ❖ Parâmetros de saída **OUT**.
  - ❖ **O valor da variável pode ser alterado dentro do stored procedure. Este valor é repassado para a variável de saída (OUT) quando o stored procedure termina**
  - ❖ **O valor inicial da variável OUT não pode ser acessada**



# Stored Procedure - IN e OUT

---

```
tpch=> CREATE FUNCTION calculosMatematicos(x int, y int, OUT  
soma int, OUT subtracao int, OUT multiplicacao int, OUT divis  
ao int ) AS $$  
BEGIN  
soma:= x+y;  
subtracao := x - y;  
    multiplicacao := x * y;  
    divisao := x / y;  
END;  
$$ LANGUAGE plpgsql;■
```



# Stored Procedure - IN e OUT

---

```
tpch=> SELECT calculosMatematicos(20,10);
```

```
calculosmatematicos
```

```
-----
```

```
(30,10,200,2)  
(1 row)
```



# Stored Procedure

---

```
tpch=> CREATE OR REPLACE FUNCTION somar_tres_valores(v1 anyelement,  
v2 anyelement, v3 anyelement)  
RETURNS anyelement as  
$$  
DECLARE  
resultado ALIAS FOR $0;  
BEGIN  
    resultado:=v1+v2+v3;  
    return resultado;  
END;  
$$ LANGUAGE plpgsql;
```



# Stored Procedure

---

```
tpch=> select somar_tres_valores(10,20,30);
```

```
somar_tres_valores
-----
                        60
(1 row)
```



# Stored Procedure - IF

---

- ❖ Sintaxe da declaração IF

IF if\_expression THEN commands

[ELSEIF el if i se \_expression THEN d ]

THEN commands]

[ELSE commands]

END IF;



# Stored Procedure - IF

```
tpch=> CREATE FUNCTION if_declare() RETURNS void AS $$  
DECLARE  
    a integer := 10;  
    b integer := 20;  
BEGIN  
    IF a > b THEN  
        RAISE NOTICE 'a is greater than b';  
    END IF;  
    IF a < b THEN  
        RAISE NOTICE 'a is less than b';  
    END IF;  
    IF a = b THEN  
        RAISE NOTICE 'a is equal to b';  
    END IF;  
END  
$$ LANGUAGE plpgsql;■
```



# Stored Procedure

---

```
tpch=> CREATE OR REPLACE FUNCTION totalRecords ()  
RETURNS integer AS $total$  
declare  
total integer;  
BEGIN  
    SELECT count(*) into total FROM customer;  
    RETURN total;  
END;  
$total$ LANGUAGE plpgsql;■
```



# Stored Procedure - Retorna Registros

---

```
tpch=> CREATE OR REPLACE FUNCTION get_customer() RETURNS SETOF RECORD AS $$  
BEGIN  
RETURN QUERY SELECT c_name FROM customer;  
RETURN;  
END;  
$$ LANGUAGE 'plpgsql';
```

```
tpch=> select * from get_customer() AS (name varchar);
```



# Stored Procedure - Retorna Registros

---

```
tpch=> CREATE OR REPLACE FUNCTION getcustomer() RETURNS SETOF  
customer AS $$  
BEGIN  
RETURN QUERY SELECT * FROM customer;  
RETURN;  
END;  
$$ LANGUAGE 'plpgsql';
```

```
tpch=> select * from getcustomer();
```



# Stored Procedure - FOR

---

```
tpch=> CREATE OR REPLACE FUNCTION factorial (i numeric)
RETURNS numeric AS $$
DECLARE
tmp numeric; result numeric;
BEGIN
    result :=1;
    FOR tmp IN 1 .. i LOOP
        result := result * tmp;
    END LOOP;
RETURN result;
END
$$ LANGUAGE plpgsql;■
```



# Stored Procedure - FOR

---

```
tpch=> select factorial(3::numeric);
```



# Stored Procedure - While

---

```
tpch=> CREATE OR REPLACE FUNCTION factorial (i numeric)
RETURNS numeric AS $$
DECLARE
tmp numeric; result numeric;
BEGIN
    result :=1;
    WHILE tmp <= i LOOP
        result := result * tmp;
    END LOOP;
RETURN result;
END
$$ LANGUAGE plpgsql;■
```



# Stored Procedure - Recursividade

---

```
tpch=> CREATE OR REPLACE FUNCTION factorial (i numeric)
RETURNS numeric AS $$
DECLARE
tmp numeric; result numeric;
BEGIN
    if i=0 THEN
        RETURN 1;
    elseif i=1 THEN
        RETURN 1;
    else
        RETURN i* factorial(i-1);
    end if;
END;
$$ LANGUAGE plpgsql;
```



# Stored Procedure

```
tpch=> CREATE OR REPLACE FUNCTION towerHanoi(ndisk INTEGER,src
c INTEGER,dst INTEGER, tmp INTEGER) RETURNS void AS '
Declare
BEGIN
    IF ndisk = 1 THEN
        RAISE NOTICE ' ' DISK % Move da haste % para haste % '
', ndisk,src,dst;
    ELSE
        perform towerHanoi(ndisk - 1,src,dst,tmp);
        RAISE NOTICE ' ' DISK % Move da haste % para haste % '
', ndisk,src,tmp;
        perform towerHanoi(ndisk - 1,src,tmp,src);
    END IF;
END;
' LANGUAGE 'plpgsql';
```

fonte: <http://www.dicas-l.com.br/arquivo/>

pl\_pgsql\_programando\_no\_postgresql\_usando\_recurividade.php#.W6xMQHb248p



# Stored Procedure - Vantagens

---

- ❖ Aumento da performance da aplicação;
- ❖ Uma vez criada, a stored procedure é compilada e armazenada no banco de dados.
- ❖ Reduz o tráfego de dados entre a aplicação e o banco de dados;
- ❖ A aplicação só precisa chamar a stored procedure;



# Stored Procedure - Vantagens

---

- ❖ Stored procedures são reusáveis por diversas linguagens de programação;
- ❖ Não é preciso reescrever a mesma função em cada aplicação diferente, ela já está implementada no banco
- ❖ Podemos administrar as permissões de cada aplicação a stored procedure



# Stored Procedure - Desvantagens

---

- ❖ Se há um número muito grande stored procedures, o tamanho de memória utilizada para cada conexão aumenta substancialmente;
- ❖ Não é fácil desenvolver e manter stored procedures e alinhá-los com as necessidades das aplicações



# Exercícios

---

1. Criar uma store procedure para somar valores de 1 até um valor de entrada, e retornar o resultado.
2. Criar uma store procedure para retornar todas os países da tabela nation através da entrada .
3. Criar uma store procedure para retornar as informações da tabela nation em que a região seja EUROPE.



# Exercícios

---

4. Criar um store procedure que dados dois inteiros indique se são iguais ou qual o maior.

5. Crie um store procedure que retorne a raiz quadrada da quantidade de clientes (customer)