



Banco de Dados Aplicado

Cristiano Santos

cristiano.santos@amf.edu.br



The image features decorative blue lines in the corners. On the left, a vertical line descends from the top, with a small circle at its base. On the right, a horizontal line extends from the top, with a small circle at its end. At the bottom, there are two horizontal lines: one on the left with a small circle at its start, and one on the right with a small circle at its end. All lines are a vibrant blue color.

Function/Stored Procedure

Stored Procedure

- Stored procedures ou procedimentos armazenados
- São pequenos pedaços de código que ficam armazenados no lado do servidor de banco de dados

Stored Procedure

- O termo usado no padrão SQL para os procedimentos armazenados é **módulos armazenados persistentes** porque **esses programas são armazenados persistentemente pelo SGBD**, de modo semelhante aos dados persistentes armazenados pelo SGBD

Stored Procedure

- Para que serve?

A construção desses pedaços de código é **tida** como uma **boa prática** por programadores que **podem**, por exemplo, **deixar no lado do servidor códigos complexos** que são **utilizados por vários aplicativos**, evitando a necessidade de **replicá-los** em cada um deles

NOTA:

Isso reduz a duplicação de esforços e melhora a modularidade do software

Stored Procedure

- Definição formal:

“Programas de procedimentos de banco de dados ou funções — que são armazenados e executados pelo SGBD no servidor de banco de dados.”

Stored Procedure

- Principais **Vantagens** de usar uma stored procedure:
 - Melhoria na performance, já que terá uma menor quantidade de trânsito entre redes
 - Pode ser compartilhado entre as aplicações
 - Portabilidade

Stored Procedure

- Principal **Desvantagem** de usar uma stored procedure:
 - Se utilizarmos uma stored procedure podemos ficar bastante dependente da base de dados
 - Se for necessário mudar de base por algum motivo, precisa reescrever todas as stored procedures,

Nota: Existe ferramentas que consegue fazer uma migração como essa, entretanto nem sempre é funcional.

Stored Procedure

- Como funcionam?

São tratados como funções no PostgreSQL, assim como acontece com os gatilhos (trigger, que serão vistos nas próximas aulas).

O que diferencia uma função de gatilho das outras é o tipo de dado que ela retorna.

Stores Procedure / Function

- **São utilizadas extensões à SQL que são especificadas no padrão para incluir construções de programação de uso geral em SQL**
- **Essas extensões são conhecidas como SQL/PSM (SQL/Persistent Stored Modules) e podem ser usadas para escrever procedimentos armazenados**
- **A SQL/PSM também serve como exemplo de uma linguagem de programação de banco de dados que estende um modelo de banco de dados e linguagem — a saber, a SQL — a saber, a SQL — com algumas construções de programação, como instruções condicionais e loops.**

Stored Procedure

- Como funciona?
 - No PostgreSQL podemos utilizar uma série de linguagens diferentes para escrevermos nossas funções:
 - Essas linguagens podem ser divididas em três grupos ou categorias:
 - Não procedurais
 - Procedurais
 - Linguagem externa e mais complexa

Stored Procedure

- Não procedurais

Utilizam a **SQL como linguagem**. Essas são caracterizadas **por não possuírem estruturas comuns** às linguagens de programação

Por exemplo, **Não temos:**

condição (**if, else, case**) e

repetição (**for, while**)

Stored Procedure

- Procedurais

Temos PL/pgSQL (muito semelhante ao PL/SQL do Oracle), PL/Tcl, PL/Perl, PL/Python...

Em que **temos**, agora sim:

condição (**if, else, case**) e

repetição (**for, while**)

Stored Procedure

- Linguagem externa

Temos linguagens como **C e C++**. Devido ao “poder” de linguagens como o “c”, ela é tida como uma linguagem não confiável, o que indica que ela **só pode ser utilizada por** um usuário com permissões de **super usuário**.

Stored Procedure

- Linguagem externa

Temos linguagens como **C e C++**. **Devido ao “poder”** de linguagens como o “c”, ela é tida como uma linguagem não confiável, o que indica que ela **só pode ser utilizada por** um usuário com permissões de **super usuário**.



Stored Procedure

- NOTA:

Antes de ser utilizada para escrever uma função, **uma linguagem precisa ser instalada ou estar habilitada no banco** no qual queremos escrever nossa função

Stored Procedure

- **NOTA:**

Para verificar se uma linguagem já está disponível, basta, executar o comando:

```
"select * from pg_language;"
```

O resultado será a lista de linguagens disponíveis para serem utilizadas nesse banco

Stored Procedure/Function

Sintaxe:

A criação de uma nova função é feita através da execução do comando `CREATE FUNCTION`.

Function

Sintaxe:

Nome - é o nome da função a ser criada

```
CREATE [ OR REPLACE ] FUNCTION nome ( tipo [, ...] ) [ RETURNS tipo_retorno ] as  
,  
    corpo da função  
,  
LANGUAGE nome_linguagem;
```

Function

Tipo - define o tipo do argumento que será recebido

Sintaxe:

```
CREATE [ OR REPLACE ] FUNCTION nome ( tipo [, ...] ) [ RETURNS tipo_retorno ] as  
  
    corpo da função  
  
LANGUAGE nome_linguagem;
```

Function

Sintaxe:

```
CREATE [ OR REPLACE ] FUNCTION nome ( tipo [, ...] ) [ RETURNS tipo_retorno ] as  
  
    corpo da função  
  
LANGUAGE nome_linguagem;
```

tipo_retorno - é o tipo de dado
que será retornado pela nossa
função.

Function

Sintaxe:

CREATE [OR REPLACE] FUNCTION nome (tipo [, ...]) [RETURNS tipo_retorno] as

,

corpo da função

,

LANGUAGE nome_linguagem;

Conteúdo - é o corpo da nossa
função

Function

Sintaxe:

CREATE [OR REPLACE] FUNCTION nome (tipo [, ...]) [RETURNS tipo_retorno] as

corpo da função

Conteúdo - é o corpo da nossa
função

LANGUAGE nome_linguagem;

Function

Sintaxe:

CREATE [OR REPLACE] FUNCTION nome (tipo [, ...]) [RETURNS tipo_retorno] as



Delimitador

corpo da função

LANGUAGE nome_linguagem;

Function

Sintaxe:

CREATE [OR REPLACE] FUNCTION nome (tipo [, ...]) [RETURNS tipo_retorno] as

corpo da função

Delimitador

LANGUAGE nome_linguagem;

Function

Sintaxe:

```
CREATE [ OR REPLACE ] FUNCTION [ schema_name . ] nome_funcao (arg1 tipo1 [, ...] ) [ RETURNS tipo_retorno ] as  
,  
    corpo da função  
,  
LANGUAGE nome_linguagem;
```

nome_linguagem - é o nome da linguagem que será utilizada para escrever a nossa função.

Function

Na prática: Digite o código abaixo e execute

create function soma(integer, integer) **returns** integer as

,

select \$1 + \$2;

,

language 'sql';

Function

Na prática: Execute o código abaixo

```
select soma(1, 2);
```

Nota: O \$1 e o \$2 fazem referência aos parâmetros passados para função. \$1 é o primeiro parâmetro e \$2 o segundo.

Function

- Na prática:

Para remover a função do banco, basta usar o comando

drop function nome_funcao(**lista de parâmetros**);

- Exemplo:

drop function soma(**integer, integer**)

Function

- **Na prática:** Exemplo 2

Criar uma função para trazer dados de funcionários de um determinado departamento

Function

- Na prática: Exemplo 2

Criar uma função para trazer dados de funcionários de um determinado departamento

Exemplo:

```
create function buscaFuncionarioIDDepto(integer) returns ??? as
```

```
,
```

```
???
```

```
'language 'sql';
```

Function

- Na prática: Exemplo 2

Criar uma função para tra

Exemplo:

```
create function buscaFuncionarioIDDepto(integer) returns ??? as
```

,

???

```
'language 'sql';
```

DICA:
returns setof funcionarios

minado departamento

Function

- Na prática: Exemplo 2

Criar uma função para trazer dados de funcionários de um determinado departamento


Exemplo:

```
create function buscaFuncionarioIDDepto(integer) returns setof funcionarios as  
'    select * from funcionarios where depto_id = $1  
  
language 'sql';
```

Function

- Na prática: **Exemplo 2 – Executando**

select buscaFuncionarioIDDepto(1);

	buscafuncionarioiddepto funcionarios 
1	(1,José,1,1)
2	(2,Pedro,1,4)

Function

- Na prática: **Exemplo 2 – Executando**

select id, nome, depto_id from buscaFuncionarioIDDepto(1);

	id integer		nome character varying (40)		depto_id integer
1		1	José		1
2		2	Pedro		1

Function

- Na prática: **Exemplo 3**

Consulta para verificar quantos funcionários em uma determinada cidade:

Function

- Na prática: **Exemplo 3**

Consulta para verificar quantos funcionários em uma determinada cidade:

```
select count(f.id) from funcionarios as f, cidades as c where  
f.cidade_id = c.id and c.nome like 'Restinga Seca'
```

Function

- Na prática: **Exemplo 3**

Solução: **Com função???**

Function

- Na prática: **Exemplo 3**

Solução: Com função

create function qtd_funcionarios_cidade(**varchar**) **returns integer** as

,

select count(f.id) from funcionarios as f, **idades** as c **where** f.cidade_id =
c.id and c.nome **like** \$1

'language 'sql';

Function

- Na prática: **Exemplo 3**

Solução: Com função

```
select qtd_funcionarios_cidade('Pelotas')
```


Function

- Na prática: **Exemplo 4**

Mostar nome da cidade e a quantidade de funcionários?

Function

- Na prática: **Exemplo 4**

Mostar nome da cidade e a quantidade de funcionários

```
select c.nome, count(f.id) from funcionarios as f, cidades as c  
where f.cidade_id = c.id group by c.nome
```

Function

- Na prática: **Exemplo 4**

E a função???



Function

- Na prática: **Exemplo 4**

Mostar nome da cidade e a quantidade de funcionários

create function qtd_funcionarios_porcidade() **returns table** (nome varchar, nro integer) as

,

select c.nome, count(f.id) from funcionarios as f, cidades as c where f.cidade_id = c.id group by c.nome

,



language 'sql';

Function

- Na prática: **Exemplo 4**

Mostar nome da cidade e a quantidade de funcionários

select nome, nro **from** qtd_funcionarios_porcidade()

	nome character varying 	nro integer 
1	Pelotas	2
2	Rio de Janeiro	1
3	São Paulo	1



PL/pgSQL

(Procedural Language/PostgreSQL)

PL/PGSQL

- PL/pgSQL (Procedural Language/PostgreSQL) é a linguagem de programação procedural do PostgreSQL.
- Com ela, é possível **inserir lógica em seu banco de dados.**

PL/PGSQL

- Por que precisaríamos de lógica?

PL/PGSQL

- Por que precisaríamos de lógica?
- Os objetivos do PL/pgSQL foram criar uma linguagem procedural carregável que pode ser usada para criar:
 - funções
 - procedimentos de gatilhos,
 - **acrescentar estruturas de controle** à linguagem SQL,
 - realizar cálculos complexos,
 - herdar tipos, funções e operadores definidos pelo usuário

Function - PL/PGSQL

```
create table funcionarios_testIF(  
    id                int not null primary key,  
    funcionario_codigo varchar(20),  
    funcionario_nome   varchar(100),  
    funcionario_situacao varchar(1) default 'A',  
    funcionario_comissao real,  
    funcionario_cargo    varchar(30),  
    data_criacao         timestamp,  
    data_atualizacao     timestamp);
```

Function - PL/PGSQL

```
insert into funcionarios_testIF(id,  
                                funcionario_nome,  
                                funcionario_situacao,  
                                funcionario_comissao,  
                                funcionario_cargo,  
                                data_criacao)  
values('0001',  
        'VINICIUS CARVALHO',  
        'B',  
        5,  
        'GERENTE',  
        '01/01/2016');
```

```
insert into funcionarios_testIF(id,  
                                funcionario_nome,  
                                funcionario_situacao,  
                                funcionario_comissao,  
                                funcionario_cargo,  
                                data_criacao)  
values('0002',  
        'SOUZA',  
        'A',  
        2,  
        'GARÇOM',  
        '01/01/2016');
```

Function - PL/PGSQL

create or replace function

retorna_nome_funcionario(func_id int)

returns text as

\$\$

declare

nome text;

situacao text;

begin

select funcionario_nome,
funcionario_situacao
into nome, situacao
from funcionarios_test1F
where id = func_id;

if situacao = 'A' then

return nome || ' Usuário Ativo';

else

return nome || ' Usuário Inativo';

end if;

end

\$\$

language plpgsql;

Function - PL/PGSQL

```
select retorna_nome_funcionario(1)
```

Function - PL/PGSQL

Cifrão duplo (\$\$)

São usados para limitar o corpo da função, e para o banco de dados entender que tudo o que está dentro dos limites do cifrão duplo é código de uma única função

Function - PL/PGSQL

Cifrão duplo (\$\$)

Cifrão não é parte do padrão SQL, mas muitas vezes é uma forma mais conveniente para escrever strings literais complicadas do que a sintaxe simples compatível com o padrão SQL .

Function - PL/PGSQL

Declare

Precisamos fazer a declaração das variáveis, colocando o seu nome e o seu tipo

Em nosso código, nós fizemos a declaração da seguinte maneira:

```
declare  
nome text;  
situacao text;
```


PL/PGSQL

- Variáveis:

Todas as variáveis usadas em um bloco devem ser declaradas

Variáveis PL/pgSQL podem ter qualquer tipo de dado SQL, como inteiro, varchar e char.

PL/PGSQL

- Variáveis:

exemplos de declarações de variáveis:

```
user_id integer;
```

```
quantity numeric(5);
```

```
url varchar;
```

```
myrow tablename%ROWTYPE;
```

```
myfield tablename.columnname%TYPE;
```

```
arow RECORD;
```

PL/PGSQL

- Variáveis com alias

exemplos de declarações de variáveis:

```
CREATE FUNCTION sales_tax(real) RETURNS real AS $$  
DECLARE  
    subtotal ALIAS FOR $1;  
BEGIN  
    RETURN subtotal * 0.06;  
END;  
$$ LANGUAGE plpgsql;
```

PL/PGSQL

- Variáveis com alias

Quando uma função PL/pgSQL é declarada com parâmetros de saída, os parâmetros de saída recebem \$n nomes e aliases opcionalmente da mesma forma que os parâmetros de entrada normais.

Em outras palavras, não precisa de \$

PL/PGSQL

- Variáveis com alias

```
CREATE FUNCTION sales_tax(subtotal real, OUT tax real) AS $$  
BEGIN  
    tax := subtotal * 0.06;  
END;  
$$ LANGUAGE plpgsql;
```

PL/PGSQL

- Variáveis:

Os parâmetros de saída são mais úteis ao retornar valores múltiplos.
Um exemplo trivial é:

PL/PGSQL

- Variáveis:

Os parâmetros de saída são mais úteis ao retornar valores múltiplos. Um exemplo trivial é:

```
CREATE FUNCTION sum_n_product(x int, y int, OUT sum int, OUT prod int)
AS $$
BEGIN
    sum := x + y;
    prod := x * y;
END;
$$ LANGUAGE plpgsql;
```

PL/PGSQL

- Variáveis:

Os parâmetros de saída são mais úteis ao retornar valores múltiplos. Um exemplo trivial é:

Saída:

```
SELECT * FROM sum_n_product(2, 4);
```

sum		prod
-----	--	------

-----+		-----
--------	--	-------

6		8
---	--	---

PL/PGSQL

- Variáveis:

Isso também funciona com procedimentos, por exemplo:

```
CREATE PROCEDURE sum_n_product(x int, y int, OUT sum int, OUT prod int)
AS $$
BEGIN
    sum := x + y;
    prod := x * y;
END;
$$ LANGUAGE plpgsql;
```

PL/PGSQL

- Variáveis:

Isso também funciona com procedimentos, por exemplo:

Em uma chamada para um procedimento, todos os parâmetros devem ser especificados.

Para parâmetros de saída, NULL pode ser especificado ao chamar o procedimento a partir de SQL simples:

PL/PGSQL

- Variáveis:

Isso também funciona com procedimentos, por exemplo:

```
CALL sum_n_product(2, 4, NULL, NULL);
```

sum		prod
6		8

PL/PGSQL

- Variáveis:

Outra maneira de declarar uma função PL/pgSQL é com RETURNS TABLE, por exemplo:

```
CREATE FUNCTION extended_sales(p_itemno int)
RETURNS TABLE(quantity int, total numeric) AS $$
BEGIN
    RETURN QUERY SELECT s.quantity, s.quantity * s.price FROM sales AS s
        WHERE s.itemno = p_itemno;
END;
$$ LANGUAGE plpgsql;
```

PL/PGSQL

- Variáveis:

Outra maneira de declarar uma função PL/pgSQL é com RETURNS TABLE, por exemplo:

This is exactly equivalent to declaring one or more OUT parameters and specifying RETURNS SETOF sometype.

PL/PGSQL

- Variáveis: Row Types
- **Uma variável de um tipo composto é chamada de variável de linha (row-type variable)**
- **Exemplo:**

name table_name%ROWTYPE;

PL/PGSQL

- Variáveis: Row Types

Essa variável pode conter uma linha inteira de um resultado de consulta `SELECT` ou `FOR`, desde que o conjunto de colunas dessa consulta corresponda ao tipo declarado da variável.

PL/PGSQL

- Variáveis: Row Types

Os campos individuais do valor da linha são acessados usando a notação de ponto usual, por exemplo **rowvar.field**.

PL/PGSQL

- Variáveis: Row Types

Uma variável de linha pode ser declarada para ter o mesmo tipo que as linhas de uma tabela ou exibição existente, usando a notação:

```
table_name%ROWTYPE;
```

PL/PGSQL

- Variáveis: Row Types

Aqui está um exemplo de uso de tipos compostos. table1 e table2 são tabelas existentes que possuem pelo menos os campos mencionados:

```
CREATE FUNCTION merge_fields(t_row table1) RETURNS text AS $$  
DECLARE  
    t2_row table2%ROWTYPE;  
BEGIN  
    SELECT * INTO t2_row FROM table2 WHERE ... ;  
    RETURN t_row.f1 || t2_row.f3 || t_row.f5 || t2_row.f7;  
END;  
$$ LANGUAGE plpgsql;
```

PL/PGSQL

- Variáveis: **Record**

Variáveis de registro são semelhantes a variáveis do tipo linha, mas não têm estrutura predefinida. **Elas assumem a estrutura de linha de atribuída durante um comando SELECT ou FOR.**

PL/PGSQL

- Variáveis: Row Types

Aqui está um exemplo de uso de tipos compostos. table1 e table2 são tabelas existentes que possuem pelo menos os campos mencionados:

```
SELECT merge_fields(t.*) FROM table1 t WHERE ... ;
```

Function - PL/PGSQL

If e else

Usamos essas condições para testar uma condição e verificar se é verdadeira.

Function - PL/PGSQL

If e else

Há a possibilidade de testarmos quantas condições que quisermos.

```
$$  
begin  
  
    if situacao = 'A' then  
  
        'Usuário Ativo';  
  
    elsif situacao = 'I' then  
  
        'Usuário Inativo'  
  
    elsif situacao is null then  
  
        'Usuário Sem status'  
  
    else  
  
        'Usuário com status diferente de A e I'  
  
    end if;  
  
end  
$$
```

Function - PL/PGSQL

Uso de Loop \ While

Function - PL/PGSQL

Uso de Loop \ While para Funções que Retornam Conjuntos de Registros

Function - PL/PGSQL

Uso de Loop \ While – Criar as seguintes tabelas:

CREATE TABLE departamentos (id serial primary key, descricao varchar);

Function - PL/PGSQL

Uso de Loop \ While – Criar as seguintes tabelas:

```
CREATE TABLE empregados(  
codigo serial,  
nome_emp text,  
salario int,  
departamento_cod int,  
PRIMARY KEY (codigo),  
FOREIGN KEY (departamento_cod) REFERENCES departamentos (codigo));
```

Function - PL/PGSQL

Uso de Loop \ While

Prática:

Queremos criar uma função que retorne todos os funcionários que recebem acima de R\$ 2000

Como podemos fazer?

Function - PL/PGSQL

Uso de Loop \ While

Prática:

Queremos criar uma função que retorne todos os funcionários que recebem acima de R\$ 2000

Como podemos fazer?

```
SELECT * FROM empregados WHERE salario >= 2000
```

Function - PL/PGSQL

Uso de Loop \ While

Prática:

Queremos criar uma função que retorne todos os funcionários que recebem acima de R\$ 2000

E com Function, como podemos fazer?

Function - PL/PGSQL

Uso de Loop \ While

```
CREATE OR REPLACE FUNCTION codigo_empleado (codigo INTEGER)
RETURNS SETOF INTEGER AS $$
DECLARE
    registro RECORD;
    retval INTEGER;

BEGIN
    FOR registro IN SELECT * FROM empleados WHERE salario >= $1 LOOP
        RETURN NEXT registro.codigo;
    END LOOP;
    RETURN;

END;
$$ language 'plpgsql';
```

Function - PL/PGSQL

Uso de Loop \ While

```
CREATE OR REPLACE FUNCTION func_codigo
RETURNS SETOF INTEGER AS $$
DECLARE
    registro RECORD;
    retval INTEGER;

BEGIN
    FOR registro IN SELECT * FROM empregados WHERE salario >= $1 LOOP
        RETURN NEXT registro. codigo;
    END LOOP;
    RETURN;

END;
$$ language 'plpgsql';
```

Precisamos de um conjunto de dados, pois podemos ter mais de um ID de funcionário

Function - PL/PGSQL

Uso de Loop \ While

```
CREATE OR REPLACE FUNCTION codigo_er  
RETURNS SETOF INTEGER AS $$  
DECLARE  
    registro RECORD;  
    retval INTEGER;  
  
BEGIN  
    FOR registro IN SELECT * FROM empregados WHERE salario >= $1 LOOP  
        RETURN NEXT registro.codigo;  
    END LOOP;  
    RETURN;  
  
END;  
$$ language 'plpgsql';
```

Ao utilizarmos laço de repetição vamos precisar percorrer um conjunto de dados!

Function - PL/PGSQL

Uso de Loop \ While

```
CREATE OR REPLACE FUNCTION codigo_er  
RETURNS SETOF INTEGER AS $$  
DECLARE  
    registro RECORD;  
    retval INTEGER;  
  
BEGIN  
    FOR registro IN SELECT * FROM empregados WHERE salario >= $1 LOOP  
        RETURN NEXT registro.codigo;  
    END LOOP;  
    RETURN;  
  
END;  
$$ language 'plpgsql';
```

Record é um conjunto de dados!

Function - PL/PGSQL

Uso de Loop \ While

```
CREATE OR REPLACE FUNCTION codigo_empregado (codigo INTEGER)
RETURNS SETOF INTEGER AS $$
DECLARE
    registro RECORD;
    retval INTEGER;

BEGIN
    FOR registro IN SELECT * FROM empregados WHERE salario >= $1 LOOP
        RETURN NEXT registro.codigo;
    END LOOP;
    RETURN;

END;
$$ language 'plpgsql';
```

Sintaxe:
For Loop
....
End Loop

Function - PL/PGSQL

Uso de Loop \ While

```
CREATE OR REPLACE FUNCTION codigo_empregado (  
    RETURNS SETOF INTEGER AS $$  
    DECLARE  
        registro RECORD;  
        retval INTEGER;  
  
    BEGIN  
        FOR registro IN SELECT * FROM empregados WHERE salario >= $1 LOOP  
            RETURN NEXT registro.codigo;  
        END LOOP;  
        RETURN;  
    END;  
    $$ language 'plpgsql';
```

Precisamos guardar o dado na variável
registro

Function - PL/PGSQL

Uso de Loop \ While

A média por departamento calculada pelo **For**, como faz?

Function - PL/PGSQL

Uso de Loop \ While

Podemos criar um tipo de dado para retorno:

```
CREATE TYPE media_sal AS (deptcod int, medsal int);
```

Function - PL/PGSQL

Uso de Loop \ While

```
CREATE OR REPLACE FUNCTION medsal() RETURNS SETOF
media_sal AS
$$
DECLARE
    s media_sal%ROWTYPE;
    salrec RECORD;
    sum_sal int;
    count int;

BEGIN
    sum_sal :=0;
    count :=0;
    s.deptcod :=0;
    FOR salrec IN SELECT salario AS salario, d.id AS
departamento FROM empleados e, departamentos d WHERE
e.departamento_cod = d.id ORDER BY d.id
    LOOP
```

```
        IF s.deptcod = 0 THEN
            s.deptcod := salrec.departamento;
            count := count + 1;
            sum_sal := sum_sal + salrec.salario;

        ELSE
            IF s.deptcod = salrec.departamento THEN
                sum_sal := sum_sal + salrec.salario;
                count := count +1;

            ELSE
                s.medsal := sum_sal/count;
                RETURN NEXT s;
                s.deptcod := salrec.departamento;
                count := 1;
                sum_sal := salrec.salario;

            END IF;

        END IF;
    END LOOP;

    s.medsal := sum_sal/count;
    RETURN NEXT s;
    RETURN;

END
$$
LANGUAGE 'plpgsql';
```

Function - PL/PGSQL

Uso de Loop \ While

select * from medsal()

Function - PL/PGSQL

Uso de Loop \ While

Mais elaborado:

```
select d.descricao, a.medsal from medsal() as a,  
departamentos as d where d.id = a.deptcod
```




Procedures

(Procedural Language/PostgreSQL)



Banco de Dados Aplicado

Cristiano Santos

cristiano.santos@amf.edu.br