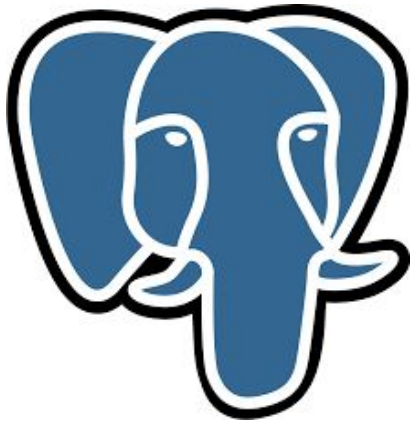


SQL Procedural

https://www.postgresql.org/docs/9.6/

SQL Procedural

- Vantagens
  - Agrupamento de instruções
  - Redução de rodadas entre cliente e servidor
  - Redução da geração de dados intermediários
  - Melhora no performance

SQL Procedural: Function vs Trigger

Triggers representam gatilhos acionados por um evento (um insert ou update)

Os gatilhos são implementados através de funções

Aspecto de função

```
CREATE FUNCTION name ( parameter_list)
RETURNS rettype
AS
$$
declare
-- variable declaration
begin
-- stored procedure body
end;
 $$
Language plpgsql ;
```

*Linguagem pode ser C, sql, plpgsql

Aspecto de função

```
CREATE FUNCTION name ( parameter_list)
RETURNS rettype
AS
$$
declare
-- variable declaration
begin
-- stored procedure body
end;
 $$
Language plpgsql ;
```

cont int=0;

*Linguagem pode ser C, sql, plpgsql

Aspecto de função

```
CREATE FUNCTION name ( parameter_list)
RETURNS rettype
AS
$$
declare
-- variable declaration
begin
-- stored procedure body
end;
 $$
Language plpgsql ;
```

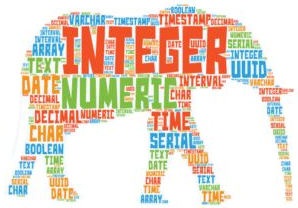
Corpo da função

*Linguagem pode ser C, sql, plpgsql

Exemplo

```
CREATE or REPLACE FUNCTION somefunc()
RETURNS void
AS
$$
DECLARE
    quantity int = 30;
BEGIN
    RAISE NOTICE 'Quantidade é  %', quantity;
END;
$$
LANGUAGE plpgsql;
```

Variáveis- Tipos

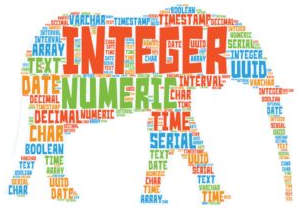


- int: -/+2147483648
- numeric(precisão,escala): 12.1245

  precisão:6 Escala:4
- varchar: string
- record: comportamento similar a uma struct

*https://www.postgresql.org/docs/9.2/datatype.html

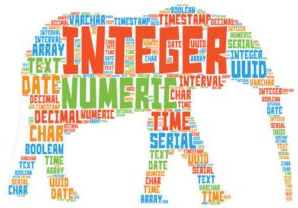
Variable RECORD



- Dinâmico
- NÃO é um tipo realmente
- Sintaxe:

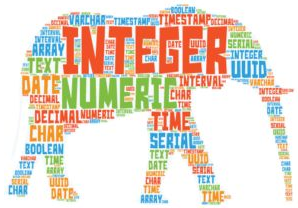
  Variável RECORD ;

Variável Record




```
CREATE or REPLACE FUNCTION merge_fields()
RETURNS text
AS $$
DECLARE
    r record;
BEGIN
    SELECT * INTO r FROM dados where id =1;
    RETURN r.nome;
END;
$$ LANGUAGE plpgsql;
```

Parâmetros



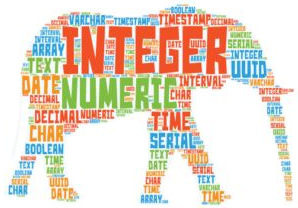

```
CREATE FUNCTION sales_tax(subtotal real)
RETURNS real
AS $$
BEGIN
    RETURN subtotal * 0.06;
END;
$$
LANGUAGE plpgsql;
```

Parâmetros-Exemplo


```
CREATE FUNCTION atualizaCurriculo2(varchar, int)
RETURNS boolean
AS$$
BEGIN
    UPDATE aluno SET curriculo = $1 WHERE num_matricula = $2;
    RETURN FOUND;
END;
$$
  LANGUAGE 'plpgsql' ;
```

- $1, $2 são parâmetros passados pela chamada da função
- FOUND : palavra reservada do sistema; booleano que retorna true se houve alteração

Retorno tipo Table



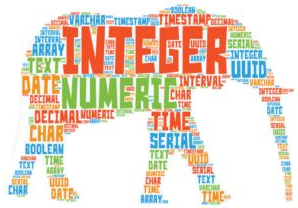
- Retornar uma nova tabela;
- Exemplo:
    - ```
      CREATE FUNCTION name()
      RETURNS TABLE(var1 int,  var2 float) ;
      ```
      ```

Retorno tipo Table


```
create table dados (name varchar(50),  salary float, id int);
```

```
CREATE FUNCTION selecionar(p_itemno int)
RETURNS TABLE(name varchar(50),  salary float)
AS $$
BEGIN
    RETURN QUERY SELECT s.name, s.salary FROM dados AS s  WHERE
s.id = p_itemno;
END;
$$ LANGUAGE plpgsql;
```

Exemplo




```
create table dados (name varchar(50),  salary float, id
int);
```

```
CREATE FUNCTION selecionar(p_itemno int)
RETURNS setof dados AS $$
BEGIN
    RETURN QUERY SELECT * FROM dados AS s
                 WHERE s.id = p_itemno;
END;
$$ LANGUAGE plpgsql;
```

Vamos criar uma primeira função

1-Crie uma função capaz de incrementar um dado número;

2-Crie uma função capaz de retornar um texto passado por argumento;

3- Crie uma tabela com a assinatura “users (id int, nome varchar(50))”. Após inserir 5 tuplas, faça uma função capaz de retornar os nomes com id maiores que a média;

Conectar: **sudo -u postgres psql postgres**

Exemplo 1

```
CREATE OR REPLACE FUNCTION incremento (valor int) returns
int AS $$
declare
  NovoValor int;
begin
    NovoValor=valor+1;
    return NovoValor;
end;
$$
LANGUAGE 'plpgsql';
```

Comando condicional

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

Operadores
>
<
>=
<=
!=

LOOPs

Formato:

```
LOOP
    statements
END LOOP [ label ];
```

Evitar seu uso!!!! → Usar comando sql (select)

Exemplo

```
create table users (id int, nome varchar(50), salario
float)
```

```
CREATE OR REPLACE FUNCTION test(p int, n int)
RETURNS setof users
AS $$
BEGIN
  IF p = 1 THEN
    return query SELECT * FROM users u where u.id=n;
  else
   UPDATE users  set salario=salario*1.1 where id=n;
   return query SELECT * FROM users u where u.id=n;
  END IF;
  RETURN;
END;
$$ LANGUAGE plpgsql;
```

Vamos criar uma função condicional

1- Crie uma função capaz de executar uma operação de incremento de 10% de um valor, se o parâmetro inicial for 1. No caso do parâmetro inicial ser 2, a função deve decrementar 10% do valor.

Assinatura: calcula_valor(operacao)

2. A partir da tabela a seguir, crie uma função capaz de atualizar o salário em 5% se o mesmo for menor que 10k e em 1% se o salário for maior que 10k.

```
create table users (id int, nome
varchar(50), salario float)
```

Exemplo 1

```
CREATE OR REPLACE FUNCTION calcula_valor(valor int) RETURNS setof users AS $$
BEGIN
  if valor = 1 then
     UPDATE users SET salario = salario*1.10;
  else
    if valor =2 then
       UPDATE users SET salario = salario*0.90;
    end if;
  end if;
  return query select * from users where salario>1000;
END;
$$
LANGUAGE 'plpgsql';


select * from calcula_valor(3);
```

Atividade A1

Crie uma tabela com a assinatura “employee (id int, name varchar(50), BirthYear int, salary float)”.

Insira 5 tuplas

A - Faça uma função capaz de aplicar um **aumento** de 10% em todos os funcionários;

B- Faça uma função capaz de aplicar um aumento de X% nos funcionários com **id maior que N**. Importante: **X e N** serão passados por argumento.

C- Faça uma função capaz de remover os funcionar com salário acima da média.

Atividade A1

D- Crie uma função que armazene o usuário corrente e a data atual ao adicionar uma nova tupla na tabela. Ex: insereDados(10,’joao’,2000’, 1000.00)

*current_user- retorna o usuário atual

*current_date - retorna a data atual

*ALTER TABLE table_name ADD column_name datatype;

```
CREATE OR REPLACE FUNCTION remove_acima() RETURNS void AS $$
DECLARE
    media numeric;
BEGIN
    select avg(salary) into media from employee;
    delete from employee where employee.salary > media;
END; $$
LANGUAGE 'plpgsql';

select * from remove_acima();
```

```
CREATE OR REPLACE FUNCTION inserir_dados(int id, varchar(50) name)
RETURNS void AS $$
BEGIN
   insert into employee values (id,name,1983,10000, current_user, current_date);
END; $$
LANGUAGE 'plpgsql';



select * from inserir_dados(10,'pedro');
```