INSTITUTO FEDERAL
RIO GRANDE DO SUL

Introdução à PL/pgSQL

TIAGO G MORAES

Roteiro

- Introdução
- Sintaxe básica de uma função
- Sintaxe básica
  - Tipos
  - Parâmetros
  - Retorno
  - Saída de dados
  - Controle de fluxo
  - Estrutura de repetição
  - SQL embutida
- Triggers

Introdução

- *Procedural Language for PostgreSQL*
  - extensão da linguagem **SQL** para o **PostgreSQL**
- É uma linguagem de programação estruturada
  - une lógica de programação com comandos SQL
- Uma forma de otimizar processamento no banco de dados e diminuir tráfego Cliente -Servidor
- Produz trechos de códigos que são armazenados nos bancos de dados
  - Procedimentos armazenadas (*stored procedures*) → função que retorna void
  - Funções armazenadas (*stored functions*)
  - Gatilhos (*triggers*)

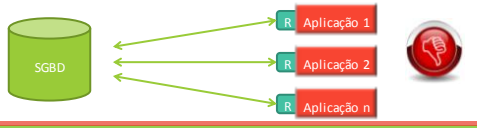
Introdução

- Por que utilizar?
  - O PL/pgSQL estende o poder da linguagem SQL
    - é uma linguagem de programação
- Considere um conjunto de regras (**R**) aplicadas a determinado banco de dados
  - Exemplo: regras para conceder um seguro de vida
- diversos softwares que utilizem essa base de dados

Introdução

- Por que utilizar?
  - O PL/pgSQL estende o poder da linguagem SQL
    - é uma linguagem de programação
- Considere um conjunto de regras (**R**) aplicadas a determinado banco de dados
  - Exemplo: regras para conceder um seguro de vida
- diversos softwares que utilizem essa base de dados

Opção 1: Regras **R** nas aplicações



Introdução

- Por que utilizar?
  - O PL/SQL estende o poder da linguagem SQL
    - é uma linguagem de programação
- Considere um conjunto de regras (**R**) aplicadas a determinado banco de dados
  - Exemplo: regras para conceder um seguro de vida
- diversos softwares que utilizem essa base de dados

Opção 2: Regras **R** no SGBD

Introdução

- Uma função/procedimento pode conter:
  - Nome da função
    - Parâmetros
    - retorno
  - Declaração das variáveis;
  - Código do programa;
  - Linguagem utilizada

Sintaxe básica

- Sintaxe básica para criação de uma função:

```
CREATE FUNCTION <nome> (lista de parâmetros) RETURNS
<tipo_retorno> AS

$$
DECLARE
<var's>
BEGIN
<algoritmo>
END;
$$ LANGUAGE 'plpgsql'
```


Sintaxe básica

- Sintaxe básica para criação de uma função:

```
CREATE FUNCTION <nome> (lista de parâmetros) RETURNS
<tipo_retorno> AS

$$
DECLARE
<var's>
BEGIN
<algoritmo>
END;
$$ LANGUAGE 'plpgsql'
```

Cria uma função CREATE OR REPLACE → salva por cima se nome já salvo

Sintaxe básica

- Sintaxe básica para criação de uma função:

```
CREATE FUNCTION <nome> (lista de parâmetros) RETURNS
<tipo_retorno> AS

$$
DECLARE
<var's>
BEGIN
<algoritmo>
END;
$$ LANGUAGE 'plpgsql'
```

Nome da função

Sintaxe básica

- Sintaxe básica para criação de uma função:

```
CREATE FUNCTION <nome> (lista de parâmetros) RETURNS
<tipo_retorno> AS

$$
DECLARE
<var's>
BEGIN
<algoritmo>
END;
$$ LANGUAGE 'plpgsql'
```

Parâmetros da função. Exemplo: idade int

Sintaxe básica

- Sintaxe básica para criação de uma função:

```
CREATE FUNCTION <nome> (lista de parâmetros) RETURNS
<tipo_retorno> AS

$$
DECLARE
<var's>
BEGIN
<algoritmo>
END;
$$ LANGUAGE 'plpgsql'
```

Tipo de retorno. Exemplo: int

Sintaxe básica

- Sintaxe básica para criação de uma função:

```
CREATE FUNCTION <nome> (lista de parâmetros) RETURNS
<tipo_retorno> AS

$$
DECLARE
<var's>
BEGIN
<algoritmo>
END;
$$ LANGUAGE 'plpgsql'
```

Início da função

Sintaxe básica

- Sintaxe básica para criação de uma função:

```
CREATE FUNCTION <nome> (lista de parâmetros) RETURNS
<tipo_retorno> AS

$$
DECLARE
<var's>
BEGIN
<algoritmo>
END;
$$ LANGUAGE 'plpgsql'
```

Declaração de Variáveis. Exemplo: aux int;

ATENÇÃO: Os parâmetros **não** são redeclarados!!

Sintaxe básica

- Sintaxe básica para criação de uma função:

```
CREATE FUNCTION <nome> (lista de parâmetros) RETURNS
<tipo_retorno> AS

$$
DECLARE
<var's>
BEGIN
<algoritmo>
END;
$$ LANGUAGE 'plpgsql'
```

Começo do algoritmo

Sintaxe básica

- Sintaxe básica para criação de uma função:

```
CREATE FUNCTION <nome> (lista de parâmetros) RETURNS
<tipo_retorno> AS

$$
DECLARE
<var's>
BEGIN
<algoritmo>
END;
$$ LANGUAGE 'plpgsql'
```

Fim da função

Sintaxe básica

- Sintaxe básica para criação de uma função:

```
CREATE FUNCTION <nome> (lista de parâmetros) RETURNS
<tipo_retorno> AS

$$
DECLARE
<var's>
BEGIN
<algoritmo>
END;
$$ LANGUAGE 'plpgsql'
```

Linguagem utilizada: sempre utilizaremos 'plpgsql'

Sintaxe básica

- Exemplo:

```
CREATE FUNCTION soma(a int, b int) RETURNS int AS

$$
DECLARE
        res int;
BEGIN
        res:= a + b;
        return res;
END;
$$ LANGUAGE 'plpgsql';
```

Sintaxe básica

- Exemplo:

```
--Uso:
SELECT soma(5,9);
```

```
CREATE FUNCTION soma(a int, b int) RETURNS int AS

$$
DECLARE
       res int;
BEGIN
       res:= a + b;
       return res;
END;
$$ LANGUAGE 'plpgsql';
```


Sintaxe básica

- Comentários:

```
--comentário de linha
/*comentário
De linhas */
```

- Atribuição:

```
x := 10;
```

- Escrita de dados:

```
RAISE NOTICE 'string'
```

  - Caractere '%' → define uma variável a ser impressa
  - Exemplo:

```
RAISE NOTICE 'valor de x: %', x
```


Sintaxe básica

- Tipos de dados: mesmos do postgresSQL (tipos de colunas)
- Declaração de tipo baseado em tabela:
  - **%TYPE** → tipo baseado em outro tipo (coluna de uma tabela por exemplo)

```
v_cpf cliente.cpf%TYPE;
```

  - **%ROWTYPE**: o registro vai ter os mesmos campos (com seus tipos) de uma tabela

```
rec_usuario cliente%ROWTYPE;
--(cod number, nome varchar(100))
```


Sintaxe básica

- Exemplo:

```
CREATE OR REPLACE FUNCTION func1() RETURNS void AS
$$
DECLARE
       p pessoa%ROWTYPE;
       nom pessoa.nome%TYPE;
BEGIN
       p.nome:='tiago';
       nom := p.nome;
       p.idade:=30;
       RAISE  notice 'nome % idade: %', nom, p.idade;
END;
$$ LANGUAGE 'plpgsql';
```


Controle de fluxo - condicional

- IF/Then/Else
  - Sintaxe:
    - IF condição THEN comandos ELSE comandos END IF

    ou
    - IF condição THEN comandos ELSIF comandos … ELSE comandos END IF

```
IF x>30 THEN
   x:=x-1;
ELSE
   x:=x+1;
END IF;
```

```
IF x>30 THEN
   x:=x-1;
ELSIF x=10
   x:=0;
ELSE
   x:=x+1;
END IF;
```


Controle de fluxo - repetição

- Loop
  - Sintaxe:
    - LOOP comandos EXIT WHEN condição; END LOOP

```
LOOP
   x:=x+1;
EXIT WHEN x=18;
END LOOP;
```

Controle de fluxo - repetição

- Loop
  - Sintaxe:
    - LOOP comandos EXIT WHEN condição; END LOOP;
- While
  - Sintaxe:
    - WHILE condição LOOP comandos; END LOOP;

```
WHILE x<18 LOOP
    x:=x+1;
END LOOP;
```


Controle de fluxo - repetição

- Loop
  - Sintaxe:
    - LOOP comandos EXIT WHEN condição; END LOOP;
- While
  - Sintaxe:
    - WHILE condição LOOP comandos; END LOOP;

```
FOR i IN 1..10 LOOP
    x:=x+1;
END LOOP;
```

- For
  - Sintaxe:
    - FOR contador IN início..fim LOOP comandos;END LOOP;

SQL embutida

- Blocos PL/pgSQL podem conter comandos SQL (DML)
- *insert*, *update* e *delete* → com valores de variáveis definidas no bloco
- As variáveis, também podem receber valores a partir de um *select*
  - **Cláusula into → para retorno de uma linha**

```
DECLARE
   x number;
BEGIN
   SELECT idade INTO x FROM pessoa WHERE codpessoa=3;
END
```

```
   Ou:
   SELECT idade FROM pessoa WHERE cod=3 INTO x;
```

```
   Ou:
   x:= idade FROM pessoa WHERE cod=3;
```

  - FOR → para ler linha a linha um retorno de select

SQL embutida

- Blocos PL/pgSQL podem conter comandos SQL (DML)
- *insert*, *update* e *delete* → com valores de variáveis definidas no bloco
- As variáveis, também podem receber valores a partir de um *select*
  - **Cláusula into → para retorno de uma linha**

Opção Mais utilizada

```
DECLARE
   x number;
BEGIN
   SELECT idade INTO x FROM pessoa WHERE codpessoa=3;
END
```

  - **Strict:** quando a select deve retornar apenas UMA linha

```
   SELECT idade INTO STRICT X FROM pessoa WHERE
   cod=3;
```


SQL embutida

- Blocos PL/pgSQL podem conter comandos SQL (DML)
- *insert*, *update* e *delete* → com valores de variáveis definidas no bloco
- As variáveis, também podem receber valores a partir de um *select*
  - Cláusula into → para retorno de apenas uma linha
  - **FOR → para ler linha a linha um retorno de select**
    - **Pode ser usado variável record genérica**

```
declareI
   x record;
begin
   for x in select * from pessoa loop
   …
   end loop;
end
```


SQL embutida

- Blocos PL/pgSQL podem conter comandos SQL (DML)
- *insert*, *update* e *delete* → com valores de variáveis definidas no bloco
- As variáveis, também podem receber valores a partir de um *select*
  - Cláusula into → para retorno de apenas uma linha
  - **FOR → para ler linha a linha um retorno de select**
    - **Pode ser usado variável record genérica**

```
declareI
   x record;
begin
   for x in select * from pessoa loop
   …
   end loop;
end
```

Equivale a:

```
-SELECT * INTO x FROM
pessoa WHERE cod =1
…
-SELECT * INTO x FROM
pessoa WHERE cod =2
…
```

E assim sucessivamente

SQL embutida

- Exemplo:

```
CREATE OR REPLACE FUNCTION func() RETURNS varchar AS
$$
DECLARE
        d disciplina%ROWTYPE;
        aux varchar:='';
BEGIN
        FOR d IN SELECT * FROM disciplina LOOP
                aux:= aux||d.cod||': '||d.nome||chr(10);
        END LOOP;
        RETURN aux;
END;
$$ LANGUAGE 'plpgsql';
```

Fim de linha

Controle de erros

- Um erro gera uma exceção que pode ser tratada em um bloco EXCEPTION

```
EXCEPTION
 WHEN <tipo exceção1> THEN <tratamento 1>
 WHEN <tipo exceção2> THEN <tratamento 2>...
 WHEN OTHERS THEN    <tratamento genérico >
```

```
declare
   x int;
begin
   X:= 2/0;
   EXCEPTION
        WHEN division_by_zero then
                RAISE NOTICE 'divisão por zero';
        WHEN others then
                RAISE NOTICE 'outra exceção';
end
```


Controle de erros

- Disparar um erro:

RAISE EXCEPTION <mensagem>

  - Equivalente ao RAISE MESSAGE mas gerando um erro com a mensagem especificada

```
CREATE OR REPLACE FUNCTION div (a int, b int) RETURNS
float AS
BEGIN
   IF(b=0) THEN
        RAISE EXCEPTION 'divisao por zero!';
   ELSE
        RETURN a/b;
end
```


Triggers - Gatilhos

- Um gatilho (ou *trigger*) é uma função que é chamada sempre que uma alteração no banco de dados é realizada:
  - Insert
  - Delete
  - Update
- Essa função pode ser chamada antes e depois do gatilho que a disparou
- Uma mesma função pode estar vinculada a mais de um gatilho
- Esse gatilho pode fazer um processamento para cada linha afetada no comando ou um processamento para a instrução
  - Trigger por instrução
  - Trigger por linha

Triggers - Gatilhos

- Decisões para um trigger:
  - Comando que chama a função:

INSERT	DELETE	UPDATE

  - Onde o comando é executado:

Tabela

  - Quando função é executada:

Antes da ação	Depois da ação

  - Quantas vezes a função é executada:

1 vez por comando executado	1 vez por linha afetada pelo comando

Triggers - Gatilhos

- Sintaxe:

```
CREATE TRIGGER <nome_gatilho>
{BEFORE|AFTER} (INSERT|DELETE|UPDATE) [OR
(INSERT|DELETE|UPDATE)...]
ON <tabela>
FOR EACH {ROW|STATEMENT}
EXECUTE PROCEDURE <nome_func>()
```

  - Uma procedure (função) já criada fica atrelada ao trigger...
    - RETURNS trigger

Triggers - Gatilhos

- Execução BEFORE:
  - Faz algo antes de a alteração na base de dados ter sido feita
- Execução AFTER:
  - Faz algo após a alteração ter sido confirmada (todas as restrições validadas e operação confirmada!)

Triggers - Gatilhos

- FOR EACH ROW:
  - Executa a função para cada linha afetada.
    - Caso um UPDATE tenha alterado 5 linhas, é executado 5 vezes
- FOR EACH STATEMENT:
  - Executa a função para cada comando.
    - A função é executada apenas UMA vez mesmo que o comando tenha afetado várias linhas

Triggers - Gatilhos

- Variáveis
  - São inicializadas Quando o trigger é disparado .
  - Podem ser utilizadas na função...
  - Principais:
    - New → tipo record com os dados de uma linha da tabela após a mudança (execução da operação no bd)
    - Old → tipo record com os dados de uma linha da tabela antes da mudança ter sido finalizada
      - Quando a trigger for STATEMENT, new e old são nulos

	new	old
insert	OK	X
delete	X	OK
update	OK	OK

Triggers - Gatilhos

- Outras Variáveis
  - TG_NAME → contém o nome do trigger disparado
  - TG_WHEN → contém "BEFORE" ou "AFTER"
  - TG_LEVEL → contém "ROW" ou "STATEMENT"
  - TG_OP → informa qual instrução disparou o gatilho:
    - "INSERT", "DELETE" ou "UPDATE"

Triggers - Gatilhos

- Retorno da trigger
  - NEW/OLD: alterações do comando que disparou são executadas
  - NULL: as alterações do comando que disparou não são executadas

Triggers - Gatilhos

- Outras questões
  - Triggers aninhadas
    - Quando a função de um trigger gera outra ação no BD que por sua vez dispara uma trigger
    - Mais de uma trigger atrelada ao mesmo evento
  - Triggers recursivas
- Desabilitando e habilitando a trigger:
  - `ALTER TABLE <nome_tabela > {DISABLE|ENABLE} TRIGGER <nome_trigger>`

Triggers - Gatilhos

- Exemplo

```
CREATE OR REPLACE FUNCTION "triggerFunc"() RETURNS
trigger AS
$$
BEGIN
       IF(NEW.salario<=0) THEN
       RAISE EXCEPTION 'salario negativo';
       END IF;
       RETURN NEW;
END;
$$LANGUAGE 'plpgsql';
```

```
CREATE TRIGGER "salPositivo" BEFORE INSERT ON
funcionario FOR EACH ROW EXECUTE PROCEDURE
"triggerFunc"();
```