Funções em PostgreSQL

Banco de Dados II
Professor Fabiano Baldo

Sumário

- Introdução
  - *Stored Procedures*
  - Linguagens
- PL/pgSQL
  - Características
  - Estruturas básicas
  - Estruturas de controle
  - Consultas SQL
  - Cache
  - Extra

Introdução | *Stored Procedures*

Justificativa e vantagens

- Bancos de Dados possuem uma estrutura cliente/servidor
- Para situações simples ou aplicações pequenas, SQL é suficiente
  - Cliente e servidor na mesma máquina
  - Consultas mais simples
  - Pequena quantidade de dados retornado
- Cenários maiores são mais complexos
  - Consultas mais complexas
    - Pode exigir processamento extra no cliente
  - Grandes quantidades de dados podem ser retornados
  - Diversidade de clientes
    - Capacidade de processamento diferente
    - Velocidade de conexão a Internet diferente
    - "Proximidade" aos dados diferente
    - Implementados em linguagens diferentes

3

Introdução | *Stored Procedures*

Justificativa e vantagens

- Solução: realizar processamento dos dados no servidor
  - Entregar aos clientes apenas o resultado do processamento
  - Ou seja: usar SGBD como ambiente de execução
- Vantagens:
  - Desempenho superior devido a "proximidade" aos dados
  - Menor tráfego de rede
  - Evita reimplementação de lógicas idênticas
    - Desperdício de tempo

4

Introdução | *Stored Procedures*

Significado e limitações do SQL

- Funções criadas e armazenadas no banco de dados são chamadas, historicamente, de *stored procedures*
  - Herança do SGBD Oracle
  - Procedimento: função que retorna vazio
  - No PostgreSQL não existem procedimentos, tudo é função
- SQL é muito limitada para ser usada na criação de funções
- Solução: programar funções usando outra linguagem

5

Introdução | Estruturas básicas

Diferencial do PostgreSQL

- Um dos grandes diferenciais do PostgreSQL em relação à maioria dos outros SGBDs é a presença de diversas linguagens para programação de funções diretamente no banco de dados.
  - Linguagens suportadas: PHP, Python, Java, Ruby, Perl, C, ...
    - Interpretador já vem embutido no SGBD ou pode ser adicionado
- Essa flexibilidade é obtida separando o mecanismo do banco de dados do executor de funções
  - PostgreSQL desconhece funcionamento das funções, apenas delega a execução para uma biblioteca dinâmica

6

Introdução | Estruturas básicas

Vantagens da PL/pgSQL

- Vantagens:
  - Manter uniformidade de linguagens no desenvolvimento de aplicativo
  - Não precisar aprender uma nova sintaxe apenas para trabalhar com os dados do servidor
- Apesar disso, a linguagem padrão do PostgreSQL é a PL/pgSQL
- A PL/pgSQL é mais prática na
  - utilização de consultas SQL dentro da função
  - criação de cache dessas consultas

7

PL/pgSQL | Características

- PL/pgSQL : **P**rocedure **L**anguage **P**ost**G**re**SQL**
- Permite executar comandos SQL diretamente no corpo das funções
- Não pode interagir diretamente com o sistema subjacente
  - Funções executam em um ambiente controlado

8

PL/pgSQL | Estruturas básicas

Instalação

- Instalar a linguagem no banco de dados
  - Até PostgreSQL 9.0

```
createlang plpgsql NOME_DO_BD
```

  - PostgreSQL 9.1+

```
CREATE EXTENSION plpgsql
```

9

PL/pgSQL | Estruturas básicas

Estrutura da linguagem

- Sintaxe da linguagem:

```
AÇÃO nome_da_funcao([ parametros ]) [ RETURNS tipo ] AS
$$
[ <<rotulo>> ]
[ DECLARE
        declarações ]
BEGIN
        comandos
END [ rotulo ];
$$
LANGUAGE plpgsql;
```

10

PL/pgSQL | Estruturas básicas
Ação a ser executada

- Sintaxe da linguagem:

```
AÇÃO nome_da_funcao([ parametros ]) [ RETURNS tipo ] AS
$$
[ <<rotulo>> ]
[ DECLARE
        declarações ]
BEGIN
        comandos
END [ rotulo ];
$$
LANGUAGE plpgsql;
```

CREATE FUNCTION
CREATE OR REPLACE FUNCTION

11

PL/pgSQL | Estruturas básicas
Corpo da função usando $$

- Sintaxe da linguagem:

```
AÇÃO nome_da_funcao([ parametros ]) [ RETURNS tipo ] AS
$$
[ <<rotulo>> ]
[ DECLARE
        declarações ]
BEGIN
        comandos
END [ rotulo ];
$$
LANGUAGE plpgsql;
```

12

PL/pgSQL | Estruturas básicas

Corpo da função usando tag

- Sintaxe da linguagem:

```
AÇÃO nome_da_funcao([ parametros ]) [ RETURNS tipo ] AS
$body$
[ <<rotulo>> ]
[ DECLARE
        declarações ]
BEGIN
        comandos
END [ rotulo ];
$body$
LANGUAGE plpgsql;
```

13

PL/pgSQL | Estruturas básicas

Corpo da função usando '

- Sintaxe da linguagem:

```
AÇÃO nome_da_funcao([ parametros ]) [ RETURNS tipo ] AS
'
[ <<rotulo>> ]
[ DECLARE
        declarações ]
BEGIN
        comandos
END [ rotulo ];
'
LANGUAGE plpgsql;
```

14

PL/pgSQL | Estruturas básicas
Estrutura dos blocos

- Sintaxe da linguagem:

```
AÇÃO nome_da_funcao([ parametros ]) [ RETURNS tipo ] AS
$$
[ <<rotulo>> ]
[ DECLARE
        declarações ]
BEGIN
        comandos
END [ rotulo ];
$$
LANGUAGE plpgsql;
```

Código é estruturado em blocos

15

PL/pgSQL | Estruturas básicas
Observação

- Sintaxe da linguagem:

```
AÇÃO nome_da_funcao([ parametros ]) [ RETURNS tipo ] AS
$$
[ <<rotulo>> ]
[ DECLARE
        declarações ]
BEGIN
        comandos
END [ rotulo ];
$$
LANGUAGE plpgsql;
```

Apesar de instalar a linguagem, é obrigatório informar isso!

16

PL/pgSQL | Estruturas básicas
Ação

```
AÇÃO ome_da_funcao([ parametros ]) [ RETURNS tipo ] AS
$$
[ <<rotulo>> ]
[ DECLARE
        declarações ]
BEGIN
        comandos
END [ rotulo ];
$$
LANGUAGE plpgsql;
```

17

PL/pgSQL | Estruturas básicas
Criar função

```
CREATE FUNCTION primeira_funcao() RETURNS void AS
$$
BEGIN
        RETURN;
END;
$$
LANGUAGE plpgsql;
```

18

PL/pgSQL | Estruturas básicas

Criar ou substituir função

```
CREATE OR REPLACE FUNCTION primeira_funcao() RETURNS void AS
$$
BEGIN
        RETURN;
END;
$$
LANGUAGE plpgsql;
```

19

PL/pgSQL | Estruturas básicas

Não é possível apenas substituir

```
CREATE OR REPLACE FUNCTION primeira_funcao() RETURNS void AS
$$
BEGIN
        RETURN;
END;
$$
LANGUAGE plpgsql;
```

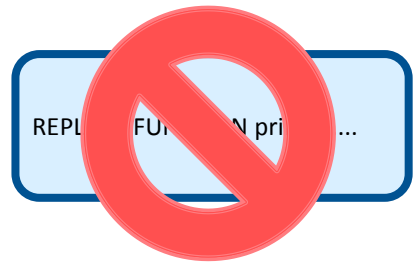
REPLACE FUNCTION primei.....

20

PL/pgSQL | Estruturas básicas

Não é possível apenas substituir

```
CREATE OR REPLACE FUNCTION primeira_funcao() RETURNS void AS
$$
BEGIN
        RETURN;
END;
$$
LANGUAGE plpgsql;
```



21

PL/pgSQL | Estruturas básicas

Observação quanto a sobrecarga

```
CREATE OR REPLACE FUNCTION primeira_funcao() RETURNS void AS
$$
BEGIN
        RETURN;
END;
$$
LANGUAGE plpgsql;
```

PostgreSQL suporta sobrecarga de funções, portanto parâmetros iguais = mesma função

22

PL/pgSQL | Estruturas básicas

Observação quanto ao retorno

```
CREATE OR REPLACE FUNCTION primeira_funcao() RETURNS void AS
$$
BEGIN
        RETURN;
END;
$$
LANGUAGE plpgsql;
```

Uma vez criada a função, NÃO é possível alterar o tipo do retorno

23

PL/pgSQL | Estruturas básicas

Apagar função

```
DROP FUNCTION primeira_funcao();
```

24

PL/pgSQL | Estruturas básicas
Observação

```
DROP FUNCTION primeira_funcao();
```

Não esquecer dos parâmetros!

25

PL/pgSQL | Estruturas básicas
Parâmetros

```
AÇÃO nome_da_funcao parametros [ RETURNS tipo ] AS
$$
[ <<rotulo>> ]
[ DECLARE
        declarações ]
BEGIN
        comandos
END [ rotulo ];
$$
LANGUAGE plpgsql;
```

26

PL/pgSQL | Estruturas básicas

Parâmetros

- Parâmetros podem ser de qualquer tipo dos dados SQL
- Tipos polimórficos

- São declarados como:

(parametro1 tipo, parametro2 tipo, parametro3 tipo, ...)

27

PL/pgSQL | Estruturas básicas

Parâmetro com nome

```
CREATE FUNCTION multiplica(valor_a real, valor_b real) RETURNS real AS
$$
BEGIN
        RETURN valor_a * valor_b;
END;
$$
LANGUAGE plpgsql;


DROP FUNCTION multiplica(valor_a real, valor_b real);
```

28

PL/pgSQL | Estruturas básicas

Parâmetro com nome

```
CREATE FUNCTION multiplica(valor_a real, valor_b real) RETURNS real AS
$$
BEGIN
        RETURN valor_a * valor_b;
END;
$$
LANGUAGE plpgsql;
```

```
DROP FUNCTION multiplica(valor_a real, valor_b real);
```

29

PL/pgSQL | Estruturas básicas

Parâmetro com nome

```
CREATE FUNCTION multiplica(valor_a real, valor_b real) RETURNS real AS
$$
BEGIN
        RETURN valor_a * valor_b;
END;
$$
LANGUAGE plpgsql;
```

```
DROP FUNCTION multiplica(real, real);
```

30

PL/pgSQL | Estruturas básicas

Parâmetro sem nome

```
CREATE FUNCTION multiplica(real, real) RETURNS real AS
$$
DECLARE
        valor_a ALIAS FOR $1;
        valor_b ALIAS FOR $2;
BEGIN
        RETURN valor_a * valor_b;
END;
$$
LANGUAGE plpgsql;
```

31

PL/pgSQL | Estruturas básicas

Parâmetro sem nome

```
CREATE FUNCTION multiplica(real, real) RETURNS real AS
$$
DECLARE
        valor_a ALIAS FOR $1;
        valor_b ALIAS FOR $2;
BEGIN
        RETURN valor_a * valor_b;
END;
$$
LANGUAGE plpgsql;
```

Única forma de declarar parâmetros nas versões do PostgreSQL anteriores a 8.0

32

PL/pgSQL | Estruturas básicas

Observação quanto ao nome dos parâmetros

```
CREATE FUNCTION multiplica(valor_a real, valor_b real) RETURNS real AS
...
        RETURN multiplica.valor_a;    ....
```

```
CREATE FUNCTION multiplica(real, real) RETURNS real AS
...
DECLARE valor_a ALIAS FOR $1;
...
        RETURN multiplica.valor_a;    ....
```

Erro em tempo de execução

33

PL/pgSQL | Estruturas básicas

Observação quanto ao nome dos parâmetros

```
CREATE FUNCTION multiplica(valor_a real, valor_b real) RETURNS real AS
...
        RETURN multiplica.valor_a;    ....
```

```
CREATE FUNCTION multiplica(real, real) RETURNS real AS
...
        RETURN multiplica.$1;         ....
```

34

PL/pgSQL | Estruturas básicas
Parâmetros com valor DEFAULT

CREATE FUNCTION multiplica(valor_a real = 5, valor_b real = 7) RETURNS real AS

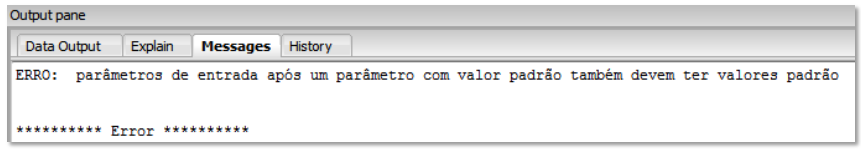
CREATE FUNCTION multiplica(real = 5, real = 7) RETURNS real AS

35

PL/pgSQL | Estruturas básicas
Parâmetros com valor DEFAULT

CREATE FUNCTION multiplica(valor_a real = 5, valor_b real) RETURNS real AS

CREATE FUNCTION multiplica(real = 5, real) RETURNS real AS



36

PL/pgSQL | Estruturas básicas
Parâmetros com valor DEFAULT

```
CREATE FUNCTION multiplica(a real, b real = 5, c integer = 8) RETURNS real AS
```

```
CREATE FUNCTION multiplica(real, real = 5, integer = 8) RETURNS real AS
```

Se colocar valor DEFAULT em um parâmetro, todos os parâmetros seguintes também deverão ter!

37

PL/pgSQL | Estruturas básicas
Observação quanto ao nome dos parâmetros

```
CREATE FUNCTION multiplica(valor_a real, real) RETURNS real AS
$$
DECLARE
        valor_b ALIAS FOR $2;
BEGIN
        RETURN valor_a * valor_b;
END;
$$
LANGUAGE plpgsql;
```

Pode declarar parâmetros dos dois modos ao mesmo tempo

38

PL/pgSQL | Estruturas básicas

Tipos polimórficos

- Tipos especiais de parâmetro:
  - anyelement
  - anyarray
- Ao declarar parâmetros desses tipos, é criado um parâmetro especial $0
  - representa o tipo de dado dos parâmetros polimórficos
  - PL/pgSQL deduz o tipo a partir dos dados recebidos
- Todos os parâmetros polimórficos DEVEM ter o mesmo tipo
  - Mesmo se declarar atributos anyelement e anyarray

39

PL/pgSQL | Estruturas básicas

Exemplo de tipos polimórficos

```
CREATE FUNCTION exibeDado(dado anyelement) RETURNS void AS
$$
BEGIN
        RAISE NOTICE 'O dado recebido é = %', dado;
        RETURN;
END;
$$
LANGUAGE plpgsql;
```

40

PL/pgSQL | Estruturas básicas
Retorno

```
AÇÃO nome_da_funcao([ parametros ] [ RETURNS tipo ] AS
$$
[ <<rotulo>> ]
[ DECLARE
        declarações ]
BEGIN
        comandos
END [ rotulo ];
$$
LANGUAGE plpgsql;
```

41

PL/pgSQL | Estruturas básicas
Retorno da função

- O retorno pode ser de qualquer tipo dos dados SQL
- Tipos especiais:
  - anyelement
  - anyarray
  - TABLE
  - SETOF

42

PL/pgSQL | Estruturas básicas
Exemplos de retornos

- Exemplos:

... RETURNS void ...

... RETURNS integer ...

... RETURNS real ...

... RETURNS varchar ...

43

PL/pgSQL | Estruturas básicas
Retornos polimórficos

- Para retorno polimórfico, ao menos um parâmetro polimórfico deve ser declarado
  - Tipo do retorno será o mesmo tipo do parâmetro

```
CREATE FUNCTION add(v1 anyelement, v2 anyelement, v3 anyelement)
RETURNS anyelement AS
$$
DECLARE result ALIAS FOR $0;
BEGIN
        result := v1 + v2 + v3;
        RETURN result;
END;
$$ LANGUAGE plpgsql;
```

44

PL/pgSQL | Estruturas básicas

Retorno via parâmetro OUT

- É possível remover da declaração da função o “RETURNS tipo”
  - Função não pode ter nenhum “RETURN”
  - Ao menos um dos parâmetro ser configurado como OUT

```
CREATE FUNCTION soma_mult(x int, y int, OUT soma int, OUT mult int) AS
$$
BEGIN
        soma := x + y;
        mult := x * y;
END;
$$
LANGUAGE plpgsql;
```

45

PL/pgSQL | Estruturas básicas

Apagar função com parâmetro OUT

- Parâmetros de saída não são considerados parte da assinatura da função

```
DROP FUNCTION soma_mult (x int, y int, OUT soma int, OUT mult int);
```

OU

```
DROP FUNCTION soma_mult (int, int);
```

46

PL/pgSQL | Estruturas básicas
Variáveis

```
AÇÃO nome_da_funcao([ parametros ]) [ RETURNS tipo ] AS
$$
[ <<rotulo>> ]
[ DECLARE
        declarações ]
BEGIN
        comandos
END [ rotulo ];
$$
LANGUAGE plpgsql;
```

47

PL/pgSQL | Estruturas básicas
Variáveis

- Variáveis podem ser de qualquer tipo dos dados SQL
- Também podem ser dos tipos:
  - %ROWTYPE
  - %TYPE
  - RECORD
- São declarados como:

nome [CONSTANT] tipo [NOT NULL] [{ DEFAULT | := } expressão];

48

PL/pgSQL | Estruturas básicas
Cláusula CONSTANT

- Variáveis podem ser de qualquer tipo dos dados SQL
- Também podem ser dos tipos:
  - %ROWTYPE
  - %TYPE
  - RECORD
- São declarados como:

nome [CONSTANT] tipo [NOT NULL] [{ DEFAULT | := } expressão];

Proíbe que o valor da variável seja alterado.
Obrigatório definir valor na declaração da variável.

49

PL/pgSQL | Estruturas básicas
Cláusula NOT NULL

- Variáveis podem ser de qualquer tipo dos dados SQL
- Também podem ser dos tipos:
  - %ROWTYPE
  - %TYPE
  - RECORD
- São declarados como:

nome [CONSTANT] tipo [NOT NULL] { DEFAULT | := } expressão];

Se variável receber valor NULL ocorre erro em tempo de execução.
OBRIGATÓRIO definir valor DEFAULT.

50

PL/pgSQL | Estruturas básicas
Cláusula DEFAULT

- Variáveis podem ser de qualquer tipo dos dados SQL
- Também podem ser dos tipos:
  - %ROWTYPE
  - %TYPE
  - RECORD
- São declarados como:

nome [CONSTANT] tipo [NOT NULL] [DEFAULT :=] expressão ;

Define um valor padrão para a variável, válido toda vez que o bloco for executado

51

PL/pgSQL | Estruturas básicas
Exemplos de declaração de variável

- Exemplos:

```
qtdade real NOT NULL DEFAULT 32.3;
url varchar := 'http://mysite.com';
user_id CONSTANT integer := 10;
weight integer[] DEFAULT '{10,9,8,7,6,5,4,3,2,1}';
helpstr varchar(300);
mediaCompras numeric(9,2);
pom varchar DEFAULT $1;
```

52

PL/pgSQL | Estruturas básicas
Tipos compostos

- O desenvolvedor também pode criar seus próprios tipos

```
CREATE TYPE soma_produto AS (soma int, produto int);


CREATE FUNCTION soma_e_multiplica (int, int) RETURNS soma_produto AS
$$
BEGIN
        RETURN $1 + $2, $1 * $2
END;
LANGUAGE plpgsql;
```

53

PL/pgSQL | Estruturas básicas
Rótulo

```
AÇÃO nome_da_funcao([ parametros ]) [ RETURNS tipo ] AS
$$
[ <<rotulo>> ]
[ DECLARE
        declarações ]
BEGIN
        comandos
END [ rotulo ];
$$
LANGUAGE plpgsql;
```

54

PL/pgSQL | Estruturas básicas
Rótulos

```
[ <<rotulo>> ]
[ DECLARE
        declarações ]
BEGIN
        comandos
END [ rotulo ];
```

- Podem ser usados na identificação do bloco sendo finalizado pelo comando EXIT
- Qualificam nomes das variáveis declaradas no bloco

55

PL/pgSQL | Estruturas básicas
Uso do rótulo em sub-bloco

```
CREATE FUNCTION funcaoqtdade() RETURNS integer AS $$
<< externo >>
DECLARE qtdade integer := 30;
BEGIN
        RAISE NOTICE 'qtdade = %', qtdade; -- 30
        qtdade := 50;
        DECLARE qtdade integer := 80;
        BEGIN
                RAISE NOTICE 'qtdade = %', qtdade;  -- 80
                RAISE NOTICE 'qtdade externa =  %', externo.qtdade;  -- 50
        END;
        RAISE NOTICE 'qtdade = %', qtdade;  -- 50
        RETURN qtdade;
END; $$ LANGUAGE plpgsql;
```

56

PL/pgSQL | Estruturas básicas
Uso do rótulo em sub-bloco

```
CREATE FUNCTION funcaoqtdade() RETURNS integer AS $$
<< externo >>
DECLARE qtdade integer := 30;
BEGIN
        RAISE NOTICE 'qtdade = %', qtdade; -- 30
        qtdade := 50;
        DECLARE qtdade integer := 80;
        BEGIN
                RAISE NOTICE 'qtdade = %', qtdade;  -- 80
                RAISE NOTICE 'qtdade externa =  %', externo.qtdade;  -- 50
        END;
        RAISE NOTICE 'qtdade = %', qtdade;  -- 50
        RETURN qtdade;
END; $$ LANGUAGE plpgsql;
```

Sub-bloco

57

PL/pgSQL | Estruturas básicas
Comentários

- Comentário que ocupa apenas uma linha:

```
-- exemplo de comentário
```

- Comentário de mais de uma linha:

```
/* inicio do comentário
   continuação
   fim do comentário */
```

58

Sumário

- Introdução
  - *Stored Procedures*
  - Linguagens
- PL/pgSQL
  - Características
  - Estruturas básicas
  - **Estruturas de controle**
  - Consultas SQL
  - Cache
  - Extra

59

PL/pgSQL | Estruturas de controle

Estruturas de controle existentes

- PL/pgSQL oferece todos os métodos tradicionais presentes em outras linguagens de programação estruturada
- Controle condicional
  - IF ... THEN ... ELSIF ... THEN ... ELSE ... ... END IF;
- Laços de repetição
  - FOR ... IN ... LOOP ... ... END LOOP;
  - LOOP ... ... END LOOP;
  - WHILE ... LOOP ... ... END LOOP;

60

PL/pgSQL | Estruturas de controle
Exemplo de controle condicional

```
CREATE OR REPLACE FUNCTION maior(a int4, b int4) RETURNS int4 AS
$$
BEGIN
        IF a>b THEN
                RETURN a;
        ELSIF b>a THEN
                RETURN b;
        ELSE
                RETURN 0;
        END IF;
END;
$$
LANGUAGE plpgsql;
```

61

PL/pgSQL | Estruturas de controle
Informações sobre IF

```
CREATE OR REPLACE FUNCTION maior(a int4, b int4) RETURNS int4 AS
$$
BEGIN
        IF a>b THEN
                RETURN a;
        ELSIF b>a THEN
                RETURN b;
        ELSE
                RETURN 0;
        END IF;
END;
$$
LANGUAGE plpgsql;
```

Suporta operadores booleanos AND, OR, IS NOT NULL

62

PL/pgSQL | Estruturas de controle
ELSIF e ELSEIF

```
CREATE OR REPLACE FUNCTION maior(a int4, b int4) RETURNS int4 AS
$$
BEGIN
        IF a>b THEN
                RETURN a;
        ELSIF b>a THEN
                RETURN b;
        ELSE
                RETURN 0;
        END IF;
END;
$$
LANGUAGE plpgsql;
```

"ELSEIF" também funciona.

63

PL/pgSQL | Estruturas de controle
Atenção ao END IF

```
CREATE OR REPLACE FUNCTION maior(a int4, b int4) RETURNS int4 AS
$$
BEGIN
        IF a>b THEN
                RETURN a;
        ELSIF b>a THEN
                RETURN b;
        ELSE
                RETURN 0;
        END IF;
END;
$$
LANGUAGE plpgsql;
```

Não esquecer "END IF;"

64

PL/pgSQL | Estruturas de controle
Laço de repetição FOR

- Sintaxe:

```
[ <<rotulo>> ]
FOR nome IN [ REVERSE ] expressão .. expressão[BY expressão] LOOP
        operações
END LOOP [ rotulo ];
```

- A variável “nome” é automaticamente declarada
  - Existe apenas dentro do loop, enquanto ele executa

65

PL/pgSQL | Estruturas de controle
Laço de repetição FOR

```
FOR count IN 1..10 LOOP
        -- count assumirá os valores 1,2,3,4,5,6,7,8,9,10
END LOOP;


FOR count IN REVERSE 10..1 LOOP
        -- count assumirá os valores 10,9,8,7,6,5,4,3,2,1
END LOOP;


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

66

PL/pgSQL | Estruturas de controle
Observação no uso do FOR

```
FOR count IN 10..1 LOOP
        -- limite inferior maior que limite superior = não executa!
END LOOP;

FOR count IN REVERSE 1..10 LOOP
        -- limite superior maior que limite inferior = não executa!
END LOOP;
```

Não gera nenhum erro!

67

PL/pgSQL | Estruturas de controle
Observação no uso do FOR

```
CREATE OR REPLACE FUNCTION contador() RETURNS void AS
$$
DECLARE
        valor integer := 0;
BEGIN
        FOR count IN 10..1 LOOP
                valor := valor +count;
        END LOOP;
        RAISE NOTICE valor = %', valor; --0
END;
$$
LANGUAGE plpgsql;
```

O FOR com erro é simplesmente ignorado.

68

PL/pgSQL | Estruturas de controle
Laço de repetição WHILE

- Sintaxe:

```
[ <<rotulo>> ]
WHILE expressão_booleana LOOP
        operações
END LOOP [ rotulo ];
```

69

PL/pgSQL | Estruturas de controle
Laço de repetição LOOP

- Sintaxe:

```
[ <<rotulo>> ]
LOOP
        operações
END LOOP [ rotulo ];
```

- LOOP não possui condições de parada
- Sua execução termina apenas quando o comando RETURN (ou EXIT) é executado

70

PL/pgSQL | Estruturas de controle
EXIT e CONTINUE

- EXIT

```
EXIT [ rotulo ] [ WHEN expressão_booleana ];
```

- CONTINUE

```
CONTINUE [ rotulo ] [ WHEN expressão_booleana ];
```

- Podem ser utilizados em qualquer tipo de laço de repetição
- Se executado dentro de um bloco BEGIN .. END, irá finalizar o bloco!

71

PL/pgSQL | Estruturas de controle
Exemplo de uso do comando EXIT

```
LOOP
        -- executa alguma coisa
        IF count > 0 THEN
                EXIT;
        END IF;
END LOOP;


LOOP
        -- executa alguma coisa
        EXIT WHEN count > 0; -- mesmo resultado
END LOOP;
```

72

PL/pgSQL | Estruturas de controle

Exemplo de uso do comando EXIT

```
BEGIN
        -- executa alguma coisa
        IF stocks > 100000 THEN
                EXIT;
        END IF;
END;
```

```
BEGIN
        -- executa alguma coisa
        EXIT WHEN stocks > 100000;
END;
```

73

PL/pgSQL | Estruturas de controle

Exemplo de uso do comando CONTINUE

```
LOOP
        -- executa alguma coisa
        EXIT WHEN count > 100;
        CONTINUE WHEN count > 50;
        -- executa esse trecho quando 50 <= count <= 100
END LOOP;
```

74

Sumário

- Introdução
  - *Stored Procedures*
  - Linguagens
- PL/pgSQL
  - Características
  - Estruturas básicas
  - Estruturas de controle
  - **Consultas SQL**
  - Cache
  - Extra

PL/pgSQL | Consultas SQL

Funções usando apenas SQL

- É possível criar funções usando SQL

```
CREATE FUNCTION clean_emp() RETURNS void AS '
   DELETE FROM emp WHERE salary < 0;
' LANGUAGE SQL;
```

```
CREATE FUNCTION um() RETURNS integer AS $$
   SELECT 1 AS result;
$$ LANGUAGE SQL;
```

PL/pgSQL | Consultas SQL

Funções usando apenas SQL

- É possível criar funções usando SQL

```
CREATE FUNCTION clean_emp() RETURNS void AS '
    DELETE FROM emp WHERE salary < 0;
LANGUAGE SQL;

CREATE FUNCTION um() RETURNS integer AS $$
    SELECT 1 AS result;
$$ LANGUAGE SQL;
```

77

PL/pgSQL | Consultas SQL

SQL em PL/pgSQL

- Comandos SQL podem ser embutidos nas funções desenvolvidas em PL/pgSQL
  - Válido para comandos de DML e DDL
    - DML: Data Manipulation Language
      - SELECT, INSERT, UPDATE, DELETE, ...
    - DDL: Data Definition Language
      - CREATE, ALTER, DROP, ...
- Clausulas SQL agem como se fossem clausulas da própria PL/pgSQL

78

PL/pgSQL | Consultas SQL

Exemplo de uso de SQL dentro de função

```
CREATE OR REPLACE FUNCTION cria_emp() RETURNS void AS $$
BEGIN
        CREATE TABLE emp (
                cod integer PRIMARY KEY,
                nome varchar,
                salario real);
        INSERT INTO emp VALUES (1,'André', 15000);
        INSERT INTO emp VALUES (2,'Bruno', -1);
        INSERT INTO emp VALUES (3,'Bruna', -1);
        INSERT INTO emp VALUES (4,'Carla', 17000);
        INSERT INTO emp VALUES (5,'José', -1);
        INSERT INTO emp VALUES (7,'Pedro', 25450);
        RETURN;
END;
$$ LANGUAGE plpgsql;
```


PL/pgSQL | Consultas SQL

Exemplo de uso de SQL dentro de função

```
CREATE FUNCTION limpa_emp() RETURNS integer AS $$
DECLARE
        linhas_afetadas integer DEFAULT 0;
BEGIN
        DELETE FROM emp WHERE salario < 0;
        GET DIAGNOSTICS linhas_afetadas = ROW_COUNT;
        RETURN linhas_afetadas;
END;
$$ LANGUAGE plpgsql;
```

PL/pgSQL | Consultas SQL
Destacando GET DIAGNOSTICS

```
CREATE FUNCTION limpa_emp() RETURNS integer AS $$
DECLARE
        linhas_afetadas integer DEFAULT 0;
BEGIN
        DELETE FROM emp WHERE salario < 0;
        GET DIAGNOSTICS linhas_afetadas = ROW_COUNT;
        RETURN linhas_afetadas;
END;
$$ LANGUAGE plpgsql;
```

81

PL/pgSQL | Consultas SQL
SQL dinâmica

- Outra possibilidade: consultas SQL dinâmicas
  - Unir clausulas SQL com parâmetros e variáveis da função

```
CREATE FUNCTION exclui_cliente(pid_cliente int4) RETURNS int4 AS $$
DECLARE
        vLinhas int4 DEFAULT 0;
BEGIN
        DELETE FROM clientes WHERE id_cliente = pid_cliente;
        GET DIAGNOSTICS vLinhas = ROW_COUNT;
        RETURN vLinhas;
END;
$$ LANGUAGE plpgsql;
```

82

PL/pgSQL | Consultas SQL

Observação quanto às SQL dinâmicas

- Outra possibilidade: consultas SQL dinâmicas
  - Unir clausulas SQL com parâmetros e variáveis da função

```
CREATE FUNCTION exclui_cliente(pid_cliente int4) RETURNS int4 AS $$
DECLARE
        vLinhas int4 DEFAULT 0;
BEGIN
        DELETE FROM clientes WHERE id_cliente = pid_cliente;
        GET DIAGNOSTICS vLinhas = ROW_COUNT;
        RETURN vLinhas;
END;
$$ LANGUAGE plpgsql;
```

Variáveis não podem ter mesmo nome que objetos no banco de dados!

83

PL/pgSQL | Consultas SQL

Detalhe sobre usar variáveis em consultas

- Porém…

```
INSERT INTO tabela VALUES ($1);
```

```
INSERT INTO $1 VALUES (42);
```

84

PL/pgSQL | Consultas SQL
SQL dinâmicas mais complexas

- Também é possível construir SQLs em tempo de execução
  - Comando EXECUTE

```
CREATE FUNCTION apagaReg(tabela text, chave text, id int4)  RETURNS int4 AS
$$
DECLARE vLinhas int4 DEFAULT 0;
BEGIN
        EXECUTE 'DELETE FROM ' || tabela || ' WHERE ' || chave || ' = ' || id;
        GET DIAGNOSTICS vLinhas = ROW_COUNT;
        RETURN vLinhas;
END;
$$ LANGUAGE 'plpgsql';
```

85

PL/pgSQL | Consultas SQL
SQL dinâmicas mais complexas

- Também é possível construir SQLs em tempo de execução
  - Comando EXECUTE

```
CREATE FUNCTION apagaReg(tabela text, chave text, id int4)  RETURNS int4 AS
$$
DECLARE vLinhas int4 DEFAULT 0;
BEGIN
        EXECUTE 'DELETE FROM ' || tabela || ' WHERE ' || chave || ' = ' || id;
        GET DIAGNOSTICS vLinhas = ROW_COUNT;
        RETURN vLinhas;
END;
$$ LANGUAGE 'plpgsql';
```

Concatenação de strings

86

PL/pgSQL | Consultas SQL

SQL dinâmicas mais complexas

- Também é possível construir SQLs em tempo de execução
  - Comando EXECUTE

```
CREATE FUNCTION apagaReg(tabela text, chave text, id int4)  RETURNS int4 AS
$$
DECLARE vLinhas int4 DEFAULT 0;
BEGIN
        EXECUTE 'DELETE FROM ' | tabela | ' WHERE ' | chave | ' = ' | id;
        GET DIAGNOSTICS vLinhas = ROW_COUNT;
        RETURN vLinhas;
END;
$$ LANGUAGE 'plpgsql';
```


PL/pgSQL | Consultas SQL

Recuperar dados de consultas SQL

- Outra possibilidade: recuperar resultado de uma consulta SQL

```
SELECT ... INTO nome_da_variavel FROM ... ... ... ;
```

```
CREATE FUNCTION getEmpNome(pCod integer) RETURNS varchar AS
$$
DECLARE vNome varchar DEFAULT '';
BEGIN
        SELECT nome INTO vNome FROM emp WHERE cod = pCod;
        RETURN vNome;
END;
$$ LANGUAGE plpgsql;
```

PL/pgSQL | Consultas SQL

Tipos especiais para consultas SQL

- Tipos especiais para recuperar resultado de consultas SQL mais complexas
  - %TYPE
  - %ROWTYPE
  - RECORD

89

PL/pgSQL | Consultas SQL

Exemplo de uso do %TYPE

- %TYPE
  - Tipo da variável será o mesmo que o atributo da tabela

```
...
DECLARE
        dado cliente.nome%TYPE;
BEGIN
        SELECT nome INTO dado FROM cliente WHERE id_cliente = pid;
...
```

90

PL/pgSQL | Consultas SQL
Exemplo de uso do %ROWTYPE

- %ROWTYPE
  - Armazena uma linha inteira de uma dada tabela

```
...
DECLARE
        linhaCliente cliente%ROWTYPE;
BEGIN
        SELECT * INTO linhaCliente FROM cliente WHERE id_cliente = pid;
...
```

91

PL/pgSQL | Consultas SQL
Exemplo de uso do RECORD

- RECORD
  - Semelhante ao %ROWTYPE, mas aceita uma linha de qualquer tabela

```
...
DECLARE
        linhaCliente RECORD;
BEGIN
        SELECT * INTO linhaCliente FROM cliente WHERE id_cliente = pid;
...
```

92

PL/pgSQL | Consultas SQL
Cursores

- Cursores são áreas de memória utilizadas pelo PostgreSQL para armazenar registros enquanto eles são processados
- Diretamente manipulados para processar consultas que geram mais de um resultado
- Comando de iteração: FOR ... IN

```
[ <<rotulo>> ]
FOR alvo IN consulta LOOP
        operações
END LOOP [ rotulo ];
```

93

PL/pgSQL | Consultas SQL
Exemplo de uso de cursor

```
CREATE FUNCTION processaClientes() RETURNS void AS $$
DECLARE linha RECORD;
BEGIN
        FOR linha IN SELECT * FROM cliente LOOP
                -- processa linha
        END LOOP;
        RETURN;
END;
$$ LANGUAGE plpgsql;
```

94

PL/pgSQL | Consultas SQL
Tipos especiais de retorno

- RETURNING
  - Substitui o RETURNS

```
CREATE FUNCTION tf1 (integer, numeric) RETURNS numeric AS $$
UPDATE bank
        SET balance = balance - $2
        WHERE accountno = $1
        RETURNING balance;
$$ LANGUAGE SQL;
```

95

PL/pgSQL | Consultas SQL
Tipos especiais de retorno

- Retornar uma linha de uma tabela específica

```
CREATE FUNCTION getfoo(int) RETURNS foo AS
$$
SELECT * FROM foo WHERE fooid = $1;
$$
LANGUAGE SQL;
```

96

PL/pgSQL | Consultas SQL
Tipos especiais de retorno

- Retornar várias linhas de uma tabela específica

```
CREATE FUNCTION getfoo() RETURNS SETOF foo AS
$$
SELECT * FROM foo;
$$
LANGUAGE SQL;
```

97

PL/pgSQL | Consultas SQL
Tipos especiais de retorno

- Construir a tabela de retorno

```
CREATE FUNCTION vendas_ext (id int)
RETURNS TABLE(qtdade int, total numeric) AS
$$
BEGIN
        RETURN QUERY SELECT qtd, qtd*preco FROM vendas WHERE itemnr=id;
END;
$$ LANGUAGE plpgsql;
```

98

Sumário

- Introdução
  - *Stored Procedures*
  - Linguagens
- PL/pgSQL
  - Características
  - Estruturas básicas
  - Estruturas de controle
  - Consultas SQL
  - **Cache**
  - Extra

99

PL/pgSQL | Cache

Introdução

- Ao indicar para o PostgreSQL como a função se comporta, ele pode configurar o uso de cache para obtenção de respostas mais rapidamente

- Três tipos:
  - IMMUTABLE
  - STABLE
  - VOLATILE

100

PL/pgSQL | Cache

Cache em funções IMMUTABLE

- IMMUTABLE
  - Sempre que os mesmos parâmetros forem passados, o retorno da função será o mesmo
    - Ou seja, independente dos dados do banco de dados
  - Não pode alterar o banco de dados
  - O Otimizador pode pré-avaliar a função para otimizar argumentos constantes
    - Por exemplo: trocar “valor := 10+7;” por “valor := 17”

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

101

PL/pgSQL | Cache

Cache em funções STABLE

- STABLE
  - Mesmo que os mesmos parâmetros sejam passados, o retorno da função poderá ser diferente
  - Se mantém estável na mesma varredura de tabela
  - Não pode alterar o banco de dados
  - O Otimizador pode considerar múltiplas chamadas como uma só, e assim não precisa reavaliar a função para cada chamada

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

102

PL/pgSQL | Cache

Cache em funções VOLATILE

- VOLATILE
  - Função sempre retorna resultados diferentes, portanto não é possível fazer nenhuma otimização
  - Permite alterar o banco de dados
  - Se nada for informado, esta é a configuração padrão

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


Sumário

- Introdução
  - *Stored Procedures*
  - Linguagens
- PL/pgSQL
  - Características
  - Estruturas básicas
  - Estruturas de controle
  - Consultas SQL
  - Cache
  - **Extra**

PL/pgSQL | Extra

Aumentar controle de acesso ao banco de dados

- Um banco de dados pode possuir vários usuários
- Pode ser interessante configurar as permissões de cada um
- Em determinadas situações, seria interessante que usuários pudessem extrapolar permissões
  - Exemplo: é possível remover permissão de exclusão de um certo usuário, mas é possível criar uma função que o permite apagar dados de tabelas específicas
- É possível fazer isso com funções
  - A cláusula SECURITY DEFINER indica que a execução da função será feita com os privilégios de quem a criou, e não com os de quem a invocou

105

PL/pgSQL | Extra

Exemplo da cláusula SECURITY DEFINER

```
CREATE FUNCTION apagaReg(campo text, val int4, pid int4) RETURNS int4 AS
$body$
DECLARE vLinhas int4 DEFAULT 0;
BEGIN
        EXECUTE 'DELETE FROM tabela
                WHERE ' || campo || ' = ' || val ||
                ' AND usr_id = ' || pid;
        GET DIAGNOSTICS vLinhas = ROW_COUNT;
        IF vLinhas > 1 THEN
                RAISE EXCEPTION 'Exclusão de várias linhas não permitida.';
        END IF;
RETURN vLinhas;
END;
$body$
LANGUAGE plpgsql SECURITY DEFINER;
```

106

PL/pgSQL | Extra

Exemplo da cláusula SECURITY DEFINER

```
CREATE FUNCTION apagaReg(campo text, val int4, pid int4) RETURNS int4 AS
$body$
DECLARE vLinhas int4 DEFAULT 0;
BEGIN
        EXECUTE 'DELETE FROM tabela
                 WHERE ' || campo || ' = ' || val ||
                 ' AND usr_id = ' || pid;
        GET DIAGNOSTICS vLinhas = ROW_COUNT;
        IF vLinhas > 1 THEN
                RAISE EXCEPTION 'Exclusão de várias linhas não permitida.';
        END IF;
RETURN vLinhas;
END;
$body$
LANGUAGE plpgsql SECURITY DEFINER;
```

Em caso de erro, a transação é anulada.

PL/pgSQL | Extra

Como executar funções

- Mas... como executar as funções?
  - SQL

```
SELECT nome_da_funcao();

SELECT * FROM aluno WHERE media >= mediaGeralDaTurma();
```

  - Triggers (gatilhos)
    - Próxima aula ☺

Dúvidas?