Linguagem SQL de consulta - Operadores

Outros operadores interessantes para auxiliar nas consultas:

- Operador ILIKE ou ~~*
  Semelhante ao operador LIKE, sendo que sua pesquisa é "case-insensitive".
- Operador SIMILAR TO
  Semelhante ao operador LIKE, porém, com facilidades adicionais:

Caracter curinga	Significado	Exemplo
% ou _	Significa o mesmo que o caracter curinga do LIKE	... WHERE texto SIMILAR TO '%lu%';
[A-F]	Significa qualquer caracter entre A e F	... WHERE texto SIMILAR TO '[A-F]%';
[AEF]	Significa A, E ou F	... WHERE texto SIMILAR TO '[AEF]%';
[^AEF]	Significa qualquer coisa diferente de A, E ou F	... WHERE texto SIMILAR TO '[^AEF]%';
| e II	Significa respectivamente, o operador OR e concatenação	... WHERE texto SIMILAR TO '%(Cameron|Marie)%';

- POSIX
  Proporciona uma pesquisa mais poderosa que o operador de comparação LIKE e SIMILAR TO.
  - Operador (~) de pesquisa case-sensitive
    ... WHERE nome ~ '.*ANA.*'; - (onde o .* corresponde ao %)
  - Operador (~*) de pesquisa case-insensitive
    ... WHERE nome ~* '.*ANA.*';
- Operadores de conjunto
  - UNION e UNION ALL
  - INTERSECT e INTERSECT ALL
  - EXCEPT e EXCEPT ALL

- Expressão CASE
  Permite retornar um valor a partir das condições definidas. Sintaxe:

```
CASE WHEN condição THEN resultado
        WHEN …
        ELSE resultado
END
```

  Ex: SELECT nome,

```
(CASE sexo WHEN 'M' THEN 'Masculino' WHEN 'F' THEN 'Feminino' ELSE 'Não informado' END) as sexo,
FROM aluno;
```

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
DEPARTAMENTO DE COMPUTAÇÃO

CURSO ENGENHARIA DE COMPUTAÇÃO

PL/pgSQL

PL/pgSQL (Procedural Language / PostgreSQL) é uma linguagem que combina o poder expressivo da SQL com as características típicas de uma linguagem de programação, disponibilizando estruturas de controle tais como testes condicionais, loops e manipulação de exceções. Ao escrever uma função PL/pgSQL é possível incluir qualquer um dos comandos SQL, juntamente com os recursos procedurais.
Além disso, uma função escrita em PL/pgSQL pode ser executada por meio de um gatilho (trigger). Um gatilho é um procedimento que é acionado toda vez que um certo evento (inserção, deleção, alteração) ocorre em uma tabela. Por exemplo, pode-se desejar executar uma certa função toda vez que uma nova linha é adicionada a uma determinada tabela.

Instalando PL/pgSQL

Antes de se usar a PL/pgSQL em um dado banco, ela deve ser instalada no mesmo:

```
CREATE LANGUAGE plpgsql;
```

Estrutura da linguagem

A sintaxe básica da declaração de uma função PL/pgSQL é:

```
CREATE [OR REPLACE] FUNCTION nome ([tipo_param [, tipo_param, …]])
RETURNS tipo_retorno
AS
' DECLARE
variável tipo_variável;
...
BEGIN
instrução;
…
RETURN valor_retorno;
END;
'
LANGUAGE 'plpgsql';
```

Ex1:

```
CREATE OR REPLACE FUNCTION media (NUMERIC, NUMERIC)
RETURNS NUMERIC
AS
' DECLARE result NUMERIC;
BEGIN
result := ($1 + $2) / 2;
RETURN result;
END;
'
LANGUAGE 'plpgsql';
```

Ex2:

```
CREATE OR REPLACE FUNCTION media (NUMERIC, NUMERIC, NUMERIC)
RETURNS NUMERIC
AS
' DECLARE result NUMERIC;
BEGIN
result := ($1 + $2 + $3) / 3;
RETURN result;
END;
'
LANGUAGE 'plpgsql';
```

Obs: As duas implementações da função *média* apresentadas nos Exs 1 e 2 podem ser realizadas no mesmo banco, ou seja, é possível **sobrecarregar** funções, para que elas possam manipular diferentes tipos e quantidades de parâmetros.

Executando a função

```
> SELECT nome_função (parâmetros);
> resultado...
```

Excluindo uma função

```
DROP FUNCTION nome ([tipo_param [, tipo_param, …]])
```

Obs 1: é necessário especificar a assinatura completa: o nome e os parâmetros.
Obs 2: caso se deseje substituir a função existente por outra versão, não é necessário excluir a mesma, basta usar a cláusula REPLACE no comando CREATE FUNCTION.

Comentários

```
-- comenta até o final da linha
/* comenta todas as
linhas incluídas
*/
```

Parâmetros

O tipo de cada parâmetro é definido na lista de parâmetros na assinatura da função.
São nomeados automaticamente, de acordo com a ordem na lista: **$1, $2,** etc.
Obs: os parâmetros são **constantes**. Portanto, não podem receber valores no corpo da função.

Declaração de variáveis

As variáveis são declaradas na seção DECLARE.
O nome pode incluir letras, underscores e números (estes não no início). O nome é case **INsensitive**.
Variáveis podem ser inicializadas na declaração:

```
DECLARE salario NUMERIC := 1000;
```

Obs 1: variáveis não podem ser inicializadas na declaração usando-se parâmetros ou outras variáveis:

```
DECLARE salario NUMERIC := 1000;
desconto NUMERIC := salario * 0.15; _ ERRO!
```

Obs 2: CONSTANTES são declaradas assim:

```
DECLARE pi CONSTANT REAL := 3.141593;
```

Apelidos

Normalmente, é bastante útil criar apelidos para os parâmetros, para que possam ser usados no corpo da função.
Isto é feito através da cláusula ALIAS FOR:

```
DECLARE saldo ALIAS FOR $1;
saque ALIAS FOR $2;
```

Tipos Especiais

Há 3 tipos (pseudo-tipos) de dados que se adaptam aos tipos originais de um atributo ou de uma tupla de uma tabela, ou ainda ao conjunto de atributos de um result set.
O tipo **%TYPE** permite que se defina uma variável com o mesmo tipo de outra variável ou com o mesmo tipo de um atributo específico de uma tabela.
Ex:

```
DECLARE endereço1 CHAR(25);
endereço2 endereço1%TYPE;
desconto Produto.preço%TYPE;
```

O tipo **%ROWTYPE** permite que se defina uma variável do tipo registro, possuindo os mesmos campos de uma determinada tabela.
Ex:

```
DECLARE um_cliente Cliente%ROWTYPE;
```

O tipo **RECORD** permite que se defina uma variável do tipo registro, cuja estrutura será determinada em tempo de execução, adaptando-se aos dados que se deseja armazenar na mesma. Veremos exemplo na seção Estruturas de Repetição.

Atribuições

Atribuições possibilitam que se estabeleça um novo valor para uma variável. O operador utilizado é **:=** .

Ex: `saldo := saldo + saque;`

Se a expressão à direita do operador não resultar um valor do mesmo tipo da variável, o seu tipo será convertido para o mesmo tipo da variável. Se isto não for possível, será gerado um erro.

Armazenando em variáveis o resultado de uma consulta

Um outro tipo de atribuição possível é o armazenamento do resultado de uma consulta em uma variável. Utiliza-se o comando **SELECT INTO**.

Ex:

```
DECLARE data Cliente.dta_nasc%TYPE;
      um_cliente Cliente%ROWTYPE;
      sal Cliente.salario%TYPE;
BEGIN
      SELECT INTO data dta_nasc FROM Cliente WHERE cod_cli = 5;
      SELECT INTO um_cliente * FROM Cliente WHERE cod_cli = 10;
      sal := um_cliente.salario;
```

Obs: o resultado da consulta deverá ser um único valor ou uma única tupla. Caso contrário, será gerado um erro.

Mensagem

RAISE NOTICE gera mensagem na tela.

Ex:

```
RAISE NOTICE ' 'estoque abaixo do limite…' ';
RAISE NOTICE ' 'valor da média = %' ' , result;
```

Mensagem com interrupção

RAISE EXCEPTION gera mensagem na tela e interrompe a execução do respectivo bloco.

Ex:

```
RAISE EXCEPTION ' 'valor inválido!' ';
```

Exemplos

Ex3:

```
CREATE OR REPLACE FUNCTION altera_salario (Func.codigo%TYPE,
Func.salario%TYPE)
RETURNS BOOLEAN
AS
' DECLARE
      v_nome Func.nome%TYPE;
      v_salario Func.salario%TYPE;
      p_codigo ALIAS FOR $1;
      p_aumento ALIAS FOR $2;
BEGIN
      SELECT INTO v_nome, v_salario nome, salario FROM Func
            WHERE codigo = p_codigo;
      RAISE NOTICE ' 'nome: %' ', v_nome;
      RAISE NOTICE ' 'salario atual: %' ', v_salario;
      v_salario := v_salario + p_aumento;
      RAISE NOTICE ' 'novo salario: %' ', v_salario;
      UPDATE Func SET salario = v_salario
            WHERE codigo = p_codigo;
      RETURN ' 't' ';
END;
'
LANGUAGE 'plpgsql';
```

Ex4:

```
CREATE OR REPLACE FUNCTION altera_salario (Func.codigo%TYPE,
Func.salario%TYPE)
RETURNS boolean
AS
' DECLARE
      v_func Func%ROWTYPE;
      p_codigo ALIAS FOR $1;
      p_aumento ALIAS FOR $2;
```

```
BEGIN
      SELECT INTO v_func * FROM Func
            WHERE codigo = p_codigo;
      RAISE NOTICE ' 'nome: %' ', v_func.nome;
      RAISE NOTICE ' 'salario atual: %' ', v_func.salario;
      RAISE NOTICE ' 'novo salario: %' ', v_func.salario+p_aumento;
      UPDATE Func SET salario = salario + p_aumento
            WHERE codigo = p_codigo;
      RETURN ' 't' ';
END;
'
LANGUAGE 'plpgsql';
```

Estruturas Condicionais

Sintaxe:

IF (condição)
THEN
    instruções;
    ...
ELSE
    instruções;
    ...
END IF;

Ex5:

```
CREATE FUNCTION fatorial(INTEGER)
RETURNS INTEGER AS
' DECLARE
      arg INTEGER;
BEGIN
      arg := $1;
      IF arg IS NULL OR arg < 0 THEN
            RAISE NOTICE ' 'Valor Inválido!' ';
            RETURN NULL;
      ELSE
            IF arg = 1 THEN
                  RETURN 1;
            ELSE
            DECLARE
                  next_value INTEGER;
            BEGIN
                  next_value := fatorial(arg - 1) * arg;
                  RETURN next_value;
            END;
            END IF;
      END IF;
END;
' LANGUAGE 'plpgsql';
```

Estruturas de Repetição

LOOP:

```
LOOP
...
EXIT WHEN (condição); -- ou: IF (condição) THEN EXIT;
-- END IF;
...
END LOOP;
```

WHILE:

```
WHILE (condição) LOOP
…
END LOOP;
```

FOR-IN:

```
FOR v_controle IN valor_inicio .. valor_fim LOOP
…
END LOOP;
```

FOR-IN-SELECT:

```
FOR v_controle IN SELECT… LOOP
…
END LOOP;
```

Obs: nesse último tipo de estrutura, o looping será executado uma vez para cada tupla retornada no result set da consulta.

Ex6:

```
CREATE OR REPLACE FUNCTION contador_cli ()
RETURNS INTEGER
AS
' DECLARE
      registro RECORD;
      qtde INTEGER;
BEGIN
      qtde := 0;
      FOR registro IN SELECT * FROM Cliente LOOP
            qtde := qtde + 1;
            RAISE NOTICE ' 'nome: %, cidade: %' ', registro.nome_cli,
            registro.cidade;
      END LOOP;
      RETURN qtde;
END;
'
LANGUAGE 'plpgsql';
```

PERFORM e EXECUTE

Um comando SQL pode ser executado diretamente dentro do corpo de uma função.

Ex:

```
BEGIN
CREATE TABLE xyz ( ... );
…
END;
```

Porém, em se tratando de uma **consulta**, o seu resultado, a princípio, deverá ser atribuído a alguma variável.

Ex:

```
BEGIN
SELECT INTO v_func * FROM Func WHERE id_func = 5;
…
END;
```

Se quisermos executar uma consulta ou uma função sem precisar guardar o seu resultado (o valor retornado, no caso da função), isto poderá ser feito por meio dos comandos PERFORM ou EXECUTE. A diferença entre eles é que com PERFORM o plano de execução da consulta é gerado e armazenado. Ou seja, PERFORM deve ser utilizado quando já se sabe a priori qual será a consulta, enquanto EXECUTE deve ser utilizado quando a consulta é montada em tempo de execução.

Ex7:

```
CREATE OR REPLACE FUNCTION conta_tempo (VARCHAR)
RETURNS INTERVAL
AS
' DECLARE
      tempo_ini TIMESTAMP;
      tempo_fim TIMESTAMP;
BEGIN
      tempo_ini := timeofday();
      EXECUTE $1;
      tempo_fim := timeofday();
      RETURN (tempo_fim - tempo_ini);
END;
'
LANGUAGE 'plpgsql';
```