PL-pgSQL - Módulo IV

- **PlPgSql – Funções em Consultas:**
- **Podemos utilizar como forma de projeção, funções nas consultas em SQL;**
- **Exemplo de consultas com função:**

```
SELECT
        NOME
        , FN_TEXTO_VALOR_MULTA(TOTALMULTAS)
        , TOTALMULTAS
FROM    EX_MOTORISTA;
```

PL-pgSQL – Funções em Consultas

- Continuação do Exemplo:

```
• CREATE OR REPLACE FUNCTION FN_TEXTO_VALOR_MULTA (_VALOR DECIMAL(8,2)) RETURNS VARCHAR(30) AS
$$
DECLARE
        _MSG VARCHAR(30);
BEGIN
        _MSG := 'DEVE 1000 EM MULTAS';
        IF _VALOR < 1000.00 THEN
                _MSG := 'MENOS DE 1000 EM MULTAS';
        ELSIF _VALOR > 1000.00 THEN
                _MSG := 'MAIS DE 1000 EM MULTAS';
        ELSIF _VALOR IS NULL THEN
                _MSG:= 'SEM MULTAS';
        END IF;
        -- SOMENTE A PARTIR DA VERSÃO 8.4
        /*CASE
                WHEN (_VALOR < 1000) THEN _MSG := 'MENOS DE 1000 EM MULTAS‘;
                WHEN (_VALOR > 1000.00) THEN _MSG := 'MAIS DE 1000 EM MULTAS‘;
                WHEN (_VALOR IS NULL) THEN _MSG := 'SEM MULTAS‘;
        END CASE;*/

        RETURN _MSG;
END;
$$ LANGUAGE PLPGSQL;
```

PL-pgSQL – Cursores e Tabelas Temporárias

- **Cursores:**
  - **Em outras ocasiões desejamos processar um resultSet linha a linha;**
  - **Nessas ocasiões utilizamos cursores. Atenção: cursores consomem muitos recursos do servidor de banco de dados, portanto, devem ser utilizados com parcimônia.**
- **Tabelas Temporárias:**
  - **Algumas vezes é necessário armazenar os dados temporariamente para a execução de um procedimento;**
  - **Um exemplo dessa necessidade é a obtenção de uma pequena parte da tabela para processamento;**
  - **Deixando a tabela livre para outros acessos;**
  - **Para isso usamos tabelas temporárias.**

PL-pgSQL – Cursores e Tabelas Temporárias

- **Exemplo de tabela temporária:**

```
CREATE OR REPLACE FUNCTION FN_EXEMPLO_TEMP_TABLE( ) RETURNS VARCHAR(10) AS
$$
BEGIN
-- CRIAÇÃO DE TABELA TEMPORÁRIA. EXISTE APENAS DURANTE A CONEXÃO DO USUÁRIO
        CREATE TEMP TABLE TMP_MATRICULA (CODTURMA VARCHAR(10), MATRICULAALUNO INTEGER, CODVENDEDOR INTEGER, DATAEFETIVACAO DATE, VALOR DECIMAL(9,2));

        INSERT INTO TMP_MATRICULA
        SELECT
                CODTURMA, MATRICULAALUNO, CODVENDEDOR, DATAEFETIVACAO, VALOR
        FROM
                ME_MATRICULA
        WHERE
                CODVENDEDOR = 1;

        --FAZ ALGO AQUI A PARTIR DAS MATRÍCULAS DO VENDEDOR 1

        RETURN 'FIM';

END;
$$
LANGUAGE PLPGSQL;
```

PL-pgSQL – Cursores e Tabelas Temporárias

- **Para rodar o exemplo de tabela temporária:**

  SELECT FN_EXEMPLO_TEMP_TABLE( ) ;

- **Tente executar novamente:**
- **SELECT FN_EXEMPLO_TEMP_TABLE( ) ;**

OBS: A temporária segue existindo até você dropar ou se desconectar.

PL-pgSQL – Cursores e Tabelas Temporárias

- **Exemplo de cursor:**

```
CREATE OR REPLACE FUNCTION FN_ATUALIZAR_TOTAL_MULTA( ) RETURNS VOID AS
$$
DECLARE
        _CURSOR_MOTORISTA REFCURSOR;
        _CNH VARCHAR(5);
BEGIN

OPEN _CURSOR_MOTORISTA FOR
SELECT CNH FROM EX_MOTORISTA;

FETCH _CURSOR_MOTORISTA INTO _CNH;

WHILE FOUND LOOP
        UPDATE EX_MOTORISTA
        SET TOTALMULTAS = (SELECT FN_OBTER_TOTAL_MULTA(_CNH))
        WHERE CNH = _CNH;

        FETCH _CURSOR_MOTORISTA INTO _CNH;
END LOOP;
CLOSE _CURSOR_MOTORISTA;
RETURN;
END;
$$
LANGUAGE PLPGSQL;

SELECT FN_ATUALIZAR_TOTAL_MULTA( );
```

PL-pgSQL – Vetores

- **Exemplo de Vetor:**

```
create or replace function fn_exemploArray() returns void as
$$
declare
        _nome char(10) [5] := '{"Ana", "Antônia", "Amilton", "Antônio", "Armando"}'; -- criando um vetor e o populando ao mesmo tempo
        _cpf char(11) [5];
        _cont integer := 0; -- se não for inicializado será inicializado com null
        _cliente record;
Begin
        _cpf[1] := '123456';
        _cpf[2] := '789456';
        _cpf[3] := '654987';
        _cpf[4] := '321654';
        _cpf[5] := '852741';
        raise notice '%', _nome[0];
        raise notice '%', _cpf[0];
        for i in 1..5 loop
                raise notice '%', 'nome: ' || _nome[i] || ' cpf: ' || _cpf[i];
        end loop;
        while _cont < 5 LOOP
                _cont := _cont + 1;
                 select * into _cliente from cliente order by random() limit 1;
                raise notice '%', _cliente.nome;
        end LOOP;
        _cont := ceil(random() *5);
        raise notice '%', 'nome: ' || _nome[_cont] || ' cpf: ' || _cpf[_cont];
        return;
end;
$$ language 'plpgsql';
```

PL-pgSQL – SQL Dinâmico

- **Exemplo de SQL Dinâmico:**

```
CREATE OR REPLACE FUNCTION FN_SQLDINAMICO(_OP CHAR(1), _CAMPO VARCHAR(50), _VALOR DECIMAL(9,2), _CHAVE VARCHAR(10), _VLRCHAVE VARCHAR(5) ) RETURNS INTEGER AS $$
DECLARE
        _SQL TEXT;
        _TOTAL INTEGER :=0;
BEGIN
        IF _OP = 'U' THEN

                _SQL := 'UPDATE EX_MULTA SET ' || _CAMPO || ' = ' || _VALOR || ' WHERE ' || _CHAVE || ' = ' || QUOTE_LITERAL(_VLRCHAVE);
                RAISE NOTICE '%', _SQL;

                EXECUTE _SQL;

                GET DIAGNOSTICS _TOTAL = ROW_COUNT; -- QUANTIDADE DE LINHAS AFETADAS PELO ÚLTIMO COMANDO

        END IF;

        RETURN _TOTAL;

END;
$$ LANGUAGE PLPGSQL;


SELECT FN_SQLDINAMICO('U', 'PONTOS', '20', 'ID', '5');

-- OBSERVEM A ABA MENSAGENS !
```

PL-pgSQL – Retornando Conjuntos de Consultas

- **Exemplo de retorno do conjunto de consultas:**

```
create or replace function fn_exemploRetornoResultSet() returns setof varchar(30) as
$$
declare
        _nome varchar(30);
        _cliente record;
begin

FOR _cliente IN select * from cliente LOOP
        return next _cliente.nome;
END LOOP;

return;

end;
$$ language 'plpgsql';

select * from fn_exemploRetornoResultSet();
```

PL-pgSQL – Retornando Conjuntos de Consultas

- **Exemplo de retorno do conjunto de consultas:**

```
Create or replace function fn_lista_precoVlrMedio(PAR_VLR_MIN DECIMAL(5,2) ) returns setof mediaproduto as
$$
DECLARE
        _produtoprecoMedio mediaproduto%rowtype;
BEGIN

FOR _produtoprecoMedio IN
                                        SELECT
                                                cod_prod,  avg(preco)
                                        FROM
                                                produto
                                        GROUP BY
                                                cod_prod
                                        HAVING
                                                avg(preco) > PAR_VLR_MIN LOOP
        RETURN NEXT _produtoprecoMedio;
END LOOP;
RETURN;
END;
$$ language 'plpgsql';

create table mediaproduto (cod_prod integer, media decimal(8,2))

select * from fn_lista_precoVlrMedio(1);
```

PL-pgSQL – Exercício

Exercício:

Escreva uma função que receba um número inteiro entre 1 e 7 e retorne o nome do dia da semana baseado nesse número. Use vetor.

1 = domingo

...

7 = sábado

OBS: O procedimento deve retornar um erro caso seja passado por parâmetro um número que não esteja entre 1 e 7;