



# **Crescer 2016/1**

Banco de Dados II – Oracle

PL/SQL – Cursores, procedures e functions



## **2 - PL/SQL**

Cursores, procedures e functions.

André Luís Nunes

Junho/2016

# PL/SQL | tópicos

- Linguagem PL/SQL
- Blocos de comandos
  - Uso de cursores
- Procedimentos
  - Procedimentos armazenados.
- Funções
  - Funções armazenadas.

# PL/SQL: cursores

Consultas (select) que podem retornar vários registros. O Oracle alocará em uma área da memória para armazenar os dados. Existem dois tipos de cursores:

- ❑ **Cursores EXPLÍCITOS:** devem ser declarados, abertos (OPEN), executados (FETCH) e fechados (CLOSE);
- ❑ **Cursores IMPLÍCITOS:** são criados, abertos e fechados pelo Oracle. Um cursor deste tipo é implementado através da colocação da cláusula INTO no SELECT. Os comandos DML (update, insert e delete) também são cursores implícitos.

# PL/SQL: cursores

O cursor pode ser declarado na cláusula DECLARE e aberto, executado e fechado dentro da cláusula BEGIN.

```
DECLARE
  Cursor c_lista Is
    Select IDCidade, Nome
    From   Cidade;
BEGIN
  Open c_lista;
  ...
  Close c_lista;
END;
```

Constraint	Descrição
<b>OPEN</b>	Depois de declarado, o cursor deve ser aberto para ser usado. Se o cursor já estiver aberto, gera erro. OPEN <NOME_CURSOR>[parametro1, parametro2]
<b>FETCH</b>	Depois de aberto, se usa este comando para associar o conteúdo de UMA LINHA (linha atual) para dentro de uma variável
<b>CLOSE</b>	Fecha o cursor.

# PL/SQL: cursores

Atributos de cursores **EXPLÍCITOS**: os atributos servem para verificar o estado de um cursor ou o resultado de sua execução.

Constraint	Descrição
<b>%FOUND</b>	Retorna TRUE caso seja possível retornar alguma linha, FALSE se não conseguir e NULL se não houver sido executado o FETCH
<b>%NOTFOUND</b>	Inverso ao anterior e igualmente NULL se não houver sido dado FETCH
<b>%ROWCOUNT</b>	Retorna o número das linhas processadas pelo cursor
<b>%ISOPEN</b>	Retorna TRUE se o cursor foi aberto (OPEN) e FALSE se não foi aberto

# PL/SQL: cursores

Atributos de cursores **IMPLÍCITOS**: os atributos servem para verificar o estado de um cursor ou o resultado de sua execução.

Constraint	Descrição
<b>SQL%FOUND</b>	Retorna TRUE sempre que INSERT, UPDATE, DELETE, afetou uma ou mais linhas ou se um SELECT retornou uma linha
<b>SQL%NOTFOUND</b>	Retorna TRUE se um dos comandos acima não afetar nenhuma linha
<b>SQL%ROWCOUNT</b>	Retorna o número das linhas afetadas pelo comando
<b>SQL%ISOPEN</b>	Retorna sempre FALSE pois o comando é fechado automaticamente

# PL/SQL: cursores

## Exemplo de cursor EXPLÍCITO

```
DECLARE
    vNome    Cliente.Nome%type;
    vBairro  Cliente.Bairro%type;
    --
    CURSOR C_ListaCli IS
        Select Nome,
               Bairro
        From   Cliente
        Order  by 1;
BEGIN
    OPEN C_ListaCli;
    LOOP
        FETCH C_ListaCli INTO vNome, vBairro;
        EXIT WHEN C_ListaCli%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE( vNome || '-'|| vBairro);
    END LOOP;
    CLOSE C_ListaCli;
END;
```



# PL/SQL: cursores

## Exemplo de cursor IMPLÍCITO

```
DECLARE
    V_QTD NUMBER;
BEGIN
    UPDATE Cliente
        SET nome      = 'Ziraldo Antunes'
        WHERE idcliente = 4;

    IF SQL%FOUND THEN
        V_QTD := SQL%ROWCOUNT;
        DBMS_OUTPUT.PUT_LINE('ATUALIZOU ' || V_QTD || ' REGISTROS');
        COMMIT;
    END IF;
END;
```

# PL/SQL: cursores

Cursores sendo executados com FOR-LOOP :

```
DECLARE
  CURSOR C_ListaCli IS
    Select Nome,
           Bairro
    From   Cliente
    Order  by 1;
BEGIN
  FOR reg IN C_ListaCli LOOP
    DBMS_OUTPUT.PUT_LINE( reg.nome || '-' || reg.Bairro);
  END LOOP;
END;
```

**SIMPLIFICANDO:** o FOR-LOOP abre implicitamente o cursor e fecha também, os campos do cursor podem ser referenciados a partir do índice do laço (neste exemplo “reg”).

# PL/SQL: cursores

Cursores com parâmetros:

```
DECLARE
  CURSOR C_ListaCli (pIDCidade in number) IS
    Select Nome,
           Bairro
    From   Cliente
    Where  IDCidade = pIDCidade
    Order by 1;
  vCidade Cliente.IDCidade%TYPE;
BEGIN
  vCidade := 20;
  FOR reg IN C_ListaCli(vCidade) LOOP
    DBMS_OUTPUT.PUT_LINE( reg.nome || '-' || reg.bairro );
  END LOOP;
END;
```

# PL/SQL: Exercícios - #1



- 1) Identifique qual o maior e menor ID da tabela Cidade. Imprima estes dois valores, e liste todos os IDs disponíveis (sem nenhum registro) neste intervalo.
- 2) Na base BANCO2 temos cidades duplicadas (com nomes repetidos). Identifique e liste, por estado (UF), todos os clientes que estão relacionados com cidades duplicadas.
- 3) [LÓGICA] faça um bloco PL/SQL que receba um palavra ou frase por parâmetro dinâmico (:p\_valor) e verifique se o conteúdo é um palíndromo, imprimir SIM ou NÃO.

# PL/SQL: cursores

## Exemplo de cursor IMPLÍCITO

```
DECLARE
    V_QTD NUMBER;
BEGIN
    UPDATE Cliente
        SET nome      = 'Ziraldo Antunes'
        WHERE idcliente = 4;

    IF SQL%FOUND THEN
        V_QTD := SQL%ROWCOUNT;
        DBMS_OUTPUT.PUT_LINE('ATUALIZOU ' || V_QTD || ' REGISTROS');
        COMMIT;
    END IF;
END;
```

# PL/SQL: procedure

❑ **Procedure:** é utilizado para executar determinado processo ou verificação. Seu código é armazenado no banco de dados. Abaixo um exemplo de procedure.

```
CREATE PROCEDURE PRIMEIRA_PROC AS  
BEGIN  
    DBMS_OUTPUT.PUT_LINE('Minha primeira PROCEDURE!!!');  
END;
```

Executando o procedimento criado:

```
BEGIN  
    Primeira_proc;  
END;
```

ou

```
Exec Primeira_proc;
```

# PL/SQL: procedure

Utilizando parâmetros de entrada (IN).

```
CREATE PROCEDURE IMPRIME_CLIENTE (pIDCliente in Number) AS  
  vNome Cliente.Nome%Type;  
BEGIN  
  
  Select Nome  
  Into   vNome  
  From   Cliente  
  Where  IDCliente = pIDCliente;  
  
  DBMS_OUTPUT.ENABLE(1000);  
  DBMS_OUTPUT.PUT_LINE('Nome: ' || vNome);  
  
EXCEPTION  
  When no_data_found Then  
    DBMS_OUTPUT.PUT_LINE('Cliente inexistente!');  
END;
```

# PL/SQL: procedure

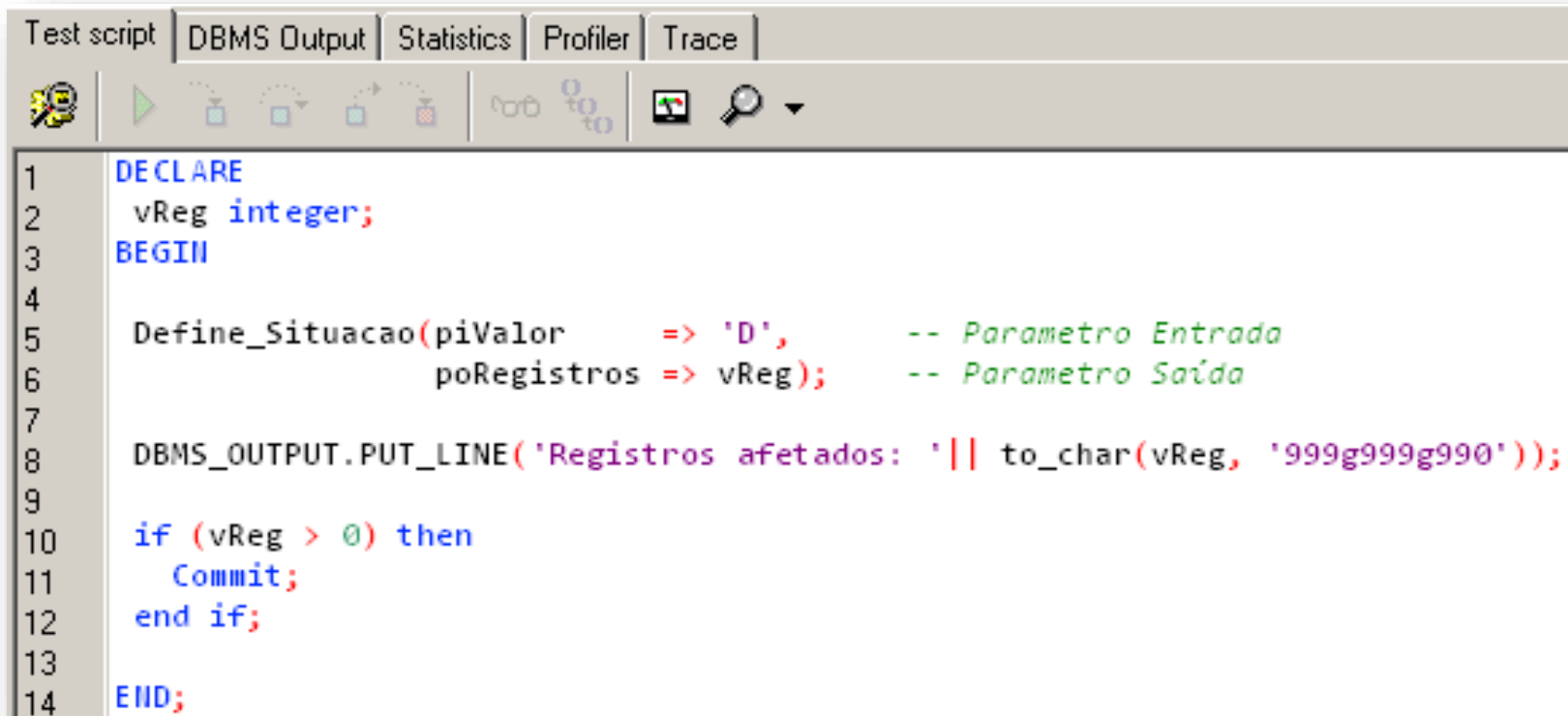
Utilizando parâmetros de entrada (IN).

```
CREATE OR REPLACE  
PROCEDURE INSERE_CIDADE( PNome IN VARCHAR2,  
                           PUF    IN VARCHAR2) AS  
BEGIN  
  
    Insert into Cidade (IDCidade, Nome, UF)  
        values ( seq_cidade.nextval, initcap(pNome), pUF);  
  
EXCEPTION  
    When DUP_VAL_ON_INDEX Then  
        Raise_Application_Error(-20001, 'Registro já cadastrado!');  
END;
```



# PL/SQL: procedure

Executando o procedimento "Define\_Situacao":



The screenshot shows a PL/SQL IDE window with a toolbar and a code editor. The toolbar includes tabs for 'Test script', 'DBMS Output', 'Statistics', 'Profiler', and 'Trace'. Below the tabs are icons for running, saving, undo, redo, and other standard IDE functions. The code editor displays a PL/SQL procedure named 'Define\_Situacao' with the following code:

```
1 DECLARE
2   vReg integer;
3 BEGIN
4
5   Define_Situacao(piValor    => 'D',      -- Parametro Entrada
6                  poRegistros => vReg);    -- Parametro Saída
7
8   DBMS_OUTPUT.PUT_LINE('Registros afetados: ' || to_char(vReg, '999g999g990'));
9
10  if (vReg > 0) then
11    Commit;
12  end if;
13
14 END;
```

# PL/SQL: function

❑ **Function:** é utilizado para executar determinado processo ou verificação e retornar (sempre) um valor. Abaixo um exemplo de procedure.

```
CREATE FUNCTION PRIMEIRA_FUNC RETURN VARCHAR2 AS  
BEGIN  
    IF TO_CHAR(SYSDATE, 'HH24') < 12 THEN  
        Return 'Bom dia!';  
    ELSIF TO_CHAR(SYSDATE, 'HH24') < 18 THEN  
        Return 'Boa tarde!';  
    ELSE  
        Return 'Boa noite!';  
    END IF;  
END;  
/
```

# PL/SQL: function

❑ **Function:** pode ser executada dentro de uma consulta (select).

Executando o procedimento criado:

```
DECLARE  
    vMsg varchar2(15) ;  
BEGIN  
    vMsg := Primeira_Func;  
    dbms_output.put_line(vMsg);  
END;
```

OU

```
Select Primeira_Func  
    From dual;
```

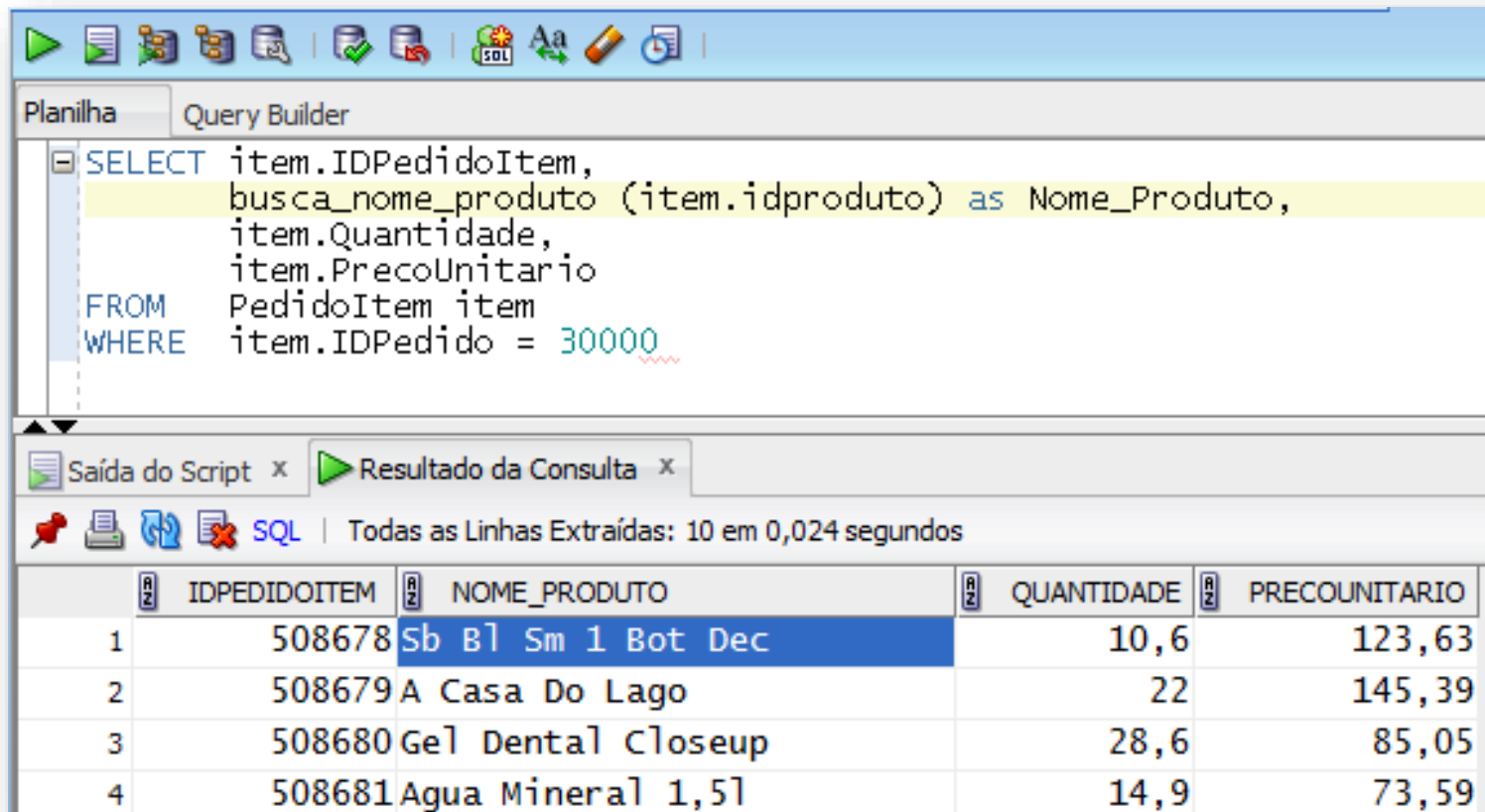
# PL/SQL: function

❑ **Function:** exemplo prático, função para retornar o nome de um produto:

```
CREATE OR REPLACE  
FUNCTION BUSCA_NOME_PRODUTO  
    (pIDProduto in Number) RETURN VARCHAR2 AS  
    vNome    Produto.Nome%type;  
BEGIN  
    Select Nome  
    Into    vNome  
    From    Produto  
    Where    IDProduto = pIDProduto;  
  
    Return vNome;  
EXCEPTION  
    When no_data_found Then  
        Return 'Produto não encontrado!';  
END;
```

# PL/SQL: function

❑ **Function:** utilizando a função dentro de uma consulta (select), pode ser utilizada em qualquer comando DML.



The screenshot shows a SQL Query Builder window. The query editor contains the following SQL statement:

```
SELECT item.IDPedidoItem,  
       busca_nome_produto (item.idproduto) as Nome_Produto,  
       item.Quantidade,  
       item.PrecoUnitario  
FROM PedidoItem item  
WHERE item.IDPedido = 30000
```

Below the query editor, the results are displayed in a table. The table has four columns: IDPEDIDOITEM, NOME\_PRODUTO, QUANTIDADE, and PRECUNITARIO. The results are as follows:

	IDPEDIDOITEM	NOME_PRODUTO	QUANTIDADE	PRECUNITARIO
1	508678	Sb B1 Sm 1 Bot Dec	10,6	123,63
2	508679	A Casa Do Lago	22	145,39
3	508680	Ge1 Dental Closeup	28,6	85,05
4	508681	Agua Mineral 1,5l	14,9	73,59

# PL/SQL: Exercícios - #2



exercícios