CWI SOFTWARE

Crescer 2016/1

Banco de Dados II – Oracle

PL/SQL

Oracle | tópicos

- Linguagem PL/SQL
- Blocos de comandos
  - Blocos anônimos (estrutura e principais comandos)
- Procedimentos
  - Procedimentos armazenados.
- Funções
  - Funções armazenadas.

PL/SQL: linguagem

- É uma linguagem de programação sofisticada utilizada para acessar um banco de dados Oracle.

- PL/SQL (Procedural Language/SQL) combina o poder e flexibilidade da SQL com as construções procedurais de uma linguagem de 3ª geração.

Para imprimir o resultado é preciso utilizar o procedimento PUT_LINE do pacote DBMS_OUTPUT, veja o exemplo abaixo:

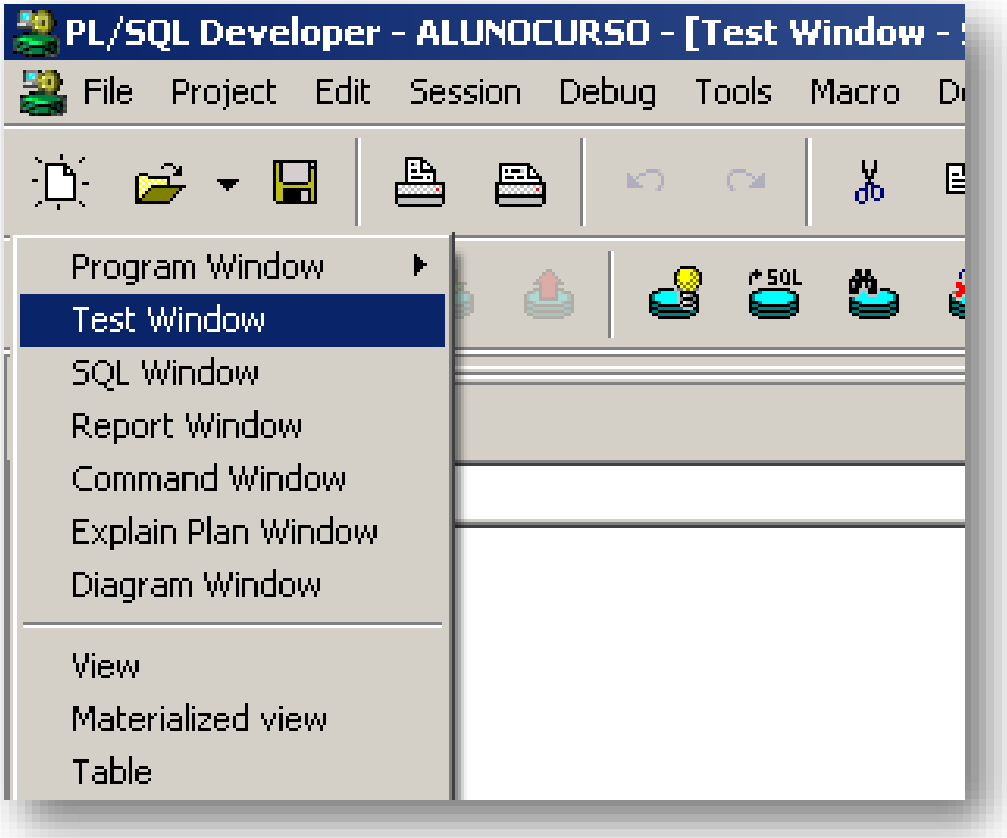
```
Begin
 DBMS_OUTPUT.PUT_LINE('Meu primeiro teste');
End;
```

PL/SQL: tipos de blocos

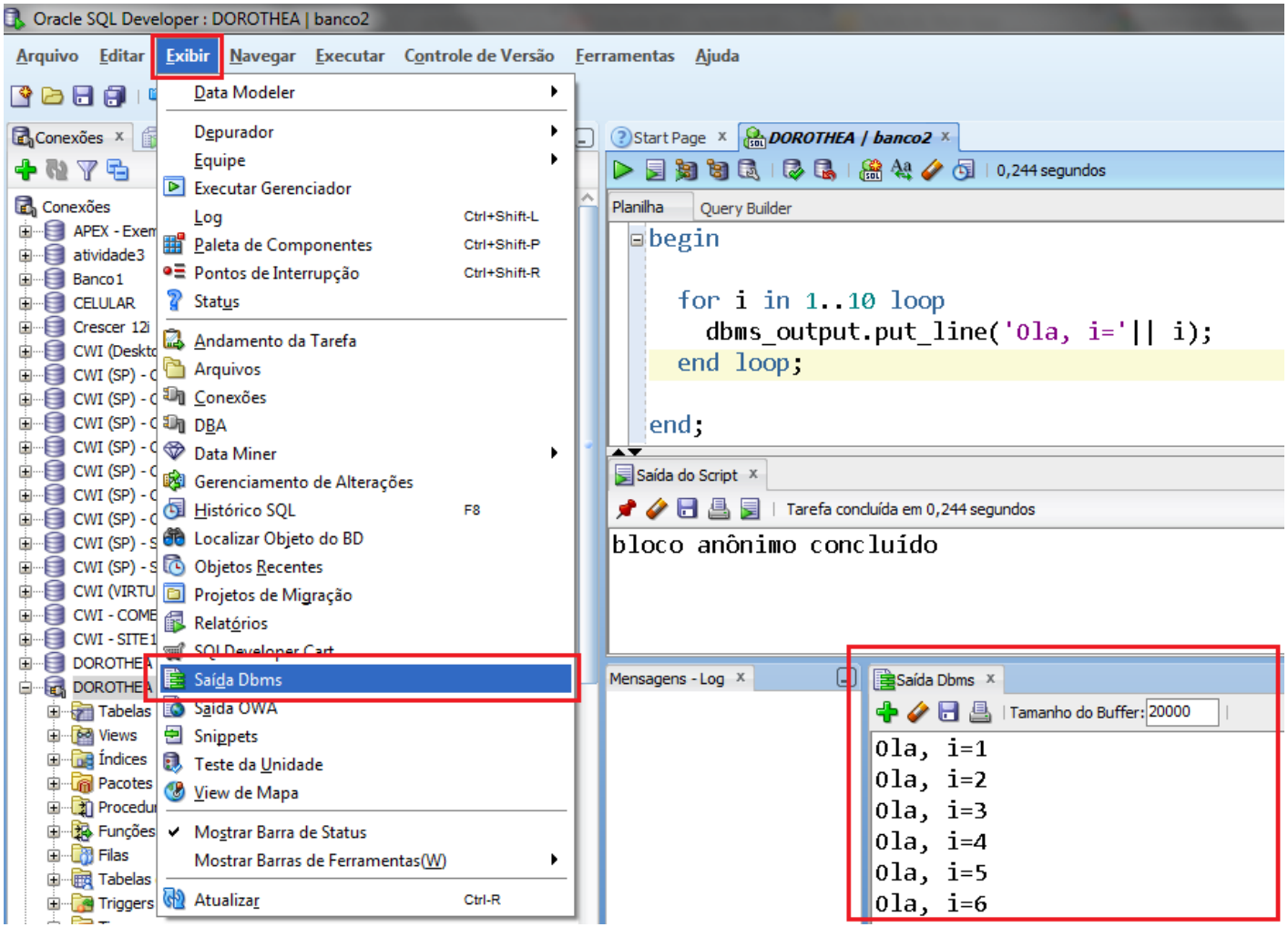
- **Blocos anônimos:** construídos dinamicamente e executados apenas uma vez.
- **Subprogramas:** consistem em procedures e funções. Podem ser armazenados no banco de dados como objetos independentes, como parte de um pacote ou como métodos de um tipo de objeto.
- **Triggers:** consistem em um bloco PL/SQL que está associado a um evento que ocorre no banco de dados.

PL/SQL: executando PL/SQL

- Para executar os exemplos de códigos PL/SQL utilize a janela "Test Window" do PL/SQL Developer.



PL/SQL: executando PL/SQL



- Para executar os exemplos de códigos PL/SQL no SQL Developer é preciso habilitar a impressão de DBMS.
- Menu Exibir > Saída Dbms
- Com isso exibirá outra janela com e deve ser habilitada, clicando em "+".

PL/SQL: estrutura básica

- Estrutura de um bloco PL/SQL:

```
DECLARE
  /* Seção declarativa: variaveis, tipos,
cursores e sobprogramas locais */
BEGIN
  /* Seção executável: instruções SQL e
procedurais entram aqui. É a principal seção do
bloco PL/SQL, e a única obrigatória. */
EXCEPTION
   /* Seção de tratamento de exceções: instruções
de tratamento de erros entram aqui. */
END;
```

- São aceitos comandos do tipo DML dentro de blocos PL/SQL, comandos DDL não.

PL/SQL: variáveis

- Utilizando variáveis:

```
DECLARE
  vNome_Completo  varchar2(30);
  vAno            number(4);
  vData           date;
BEGIN
  -- Atribuindo um valor para a variavel
  vNome_Completo := 'Joao da Silva';
  vData          := sysdate+1000;
  vAno           := to_char(vData, 'yyyy');
  -- Imprimindo as variaveis
  DBMS_OUTPUT.ENABLE(10000);
  DBMS_OUTPUT.PUT_LINE(vNome_Completo);
  DBMS_OUTPUT.PUT_LINE('Em 1000 dias estaremos em: ' ||
                       to_char(vAno));
END;
```

PL/SQL: tipos de dados

CARACTERE

- CHAR(<n>) armazena string de tamanho fixo. Tamanho default 1, máximo 32.767. Subtipo: CHARACTER
- VARCHAR2(<n>) armazena string de tamanho variável. É possível armazenar string de até 32.767 bytes. Subtipo: STRING
- VARCHAR(<n>) sinônimo para o tipo VARCHAR2.
- NCHAR(<n>) e NVARCHAR2(<n>) possuem as mesmas características dos tipos CHAR e VARCHAR2 e são usados para armazenar dados NLS (National Language Support). A arquitetura Oracle NLS permite armazenar, processar e recuperar informações em linguagens nativas.
- LONG é um tipo de dados que se tornou "obsoleto" com a chegada dos tipos LOB (Large Object). O tipo LONG armazena strings de tamanho variável de no máximo 32.760 bytes.

PL/SQL: tipos de dados

NUMÉRICO

- NUMBER(<x>, <y>) onde <X> corresponde ao número de dígitos e <Y> o número de casas decimais. Valores inseridos em colunas numéricas com número de casas decimais menor que o dado inserido serão arredondados. Subtipos: **DEC, DECIMAL, DOUBLE PRECISION, FLOAT, INTEGER, INT, NUMERIC, REAL, SMALLINT.**

- BINARY_INTEGER utilizado para armazenar inteiros com sinal, que variam de –2147483647 a 2147483647. Requerem menos memória que tipos NUMBER. Subtipos: NATURAL (n>=0), **NATURALN (n>=0 not null), POSITIVE (n>0), POSITIVEN (n>0 not null), SIGNTYPE (-1, 0, 1).**

- PLS_INTEGER Possui as mesmas características do tipo BINARY_INTEGER, entretanto possui melhor performance para cálculos.

PL/SQL: tipos de dados

DATE

- TIMESTAMP semelhante ao tipo DATE, com a diferença de armazenar fração de segundos com precisão de até 9 digitos.
- TIMESTAMP WITH TIME ZONE armazena data/hora com informações de fuso horário.
- TIMESTAMP WITH LOCAL TIME ZONE armazena data/hora no fuso horário do servidor. Quando o usuário seleciona os dados, o valor é ajustado para as configurações da sua sessão.
- INTERVAL YEAR TO MONTH usado para armazenar espaço de tempo em anos e meses.
- INTERVAL DAY TO SECOND permite especificar intervalos em dias, horas, minutos e segundos.
- DATE usado para armazenar data e hora.

PL/SQL: tipos de dados

%TYPE: este tipo permite que a variável assuma o mesmo tipo e tamanho de uma coluna de uma tabela.

```
DECLARE
  vNome_Cidade Cidade.Nome%TYPE;
BEGIN
  -- Atribuindo um valor para a variavel %TYPE
  vNome_Cidade := 'Taquara';

  DBMS_OUTPUT.PUT_LINE(vNome_Cidade);
END;
```

Muito utilizado quando se deseja capturar o resultado de um consulta (SELECT), pois desta forma não é necessário alterar o código caso a coluna seja alterada na tabela.

Boolean: tipo boleano, aceita True ou False.

PL/SQL: executando consultas

- Para executar o comando SELECT dentro de um bloco SQL, assim qualquer outra linguagem é necessário atribuir o resultado da consulta a uma variável ou outra estrutura similar.

```
BEGIN

  -- Consultando o total de registros
  Select count(1) from Cliente;

END;
```

```
ERROR at line 4:
ORA-06550: line 4, column 3:
PLS-00428: an INTO clause is expected in this SELECT statement
```

PL/SQL: executando consultas

- Utilizando a cláusula INTO, veja neste exemplo que a variável vTotal está recebendo o resultado da consulta (1 linha).

```
DECLARE
  vTotal   Integer;
BEGIN

  -- Consultando o total de registros
  Select count(1)
  Into   vTotal
  From   Cliente;

  -- Imprimindo a variavel
  DBMS_OUTPUT.PUT_LINE('Total: ' || to_char(vTotal));
END;
```

PL/SQL: executando consultas

- Para executar o comando SELECT que retorne mais de 1 linha a cláusula INTO não pode ser utilizada.

```
DECLARE
  vNome   varchar2(60);
BEGIN
  -- Consultando o nome dos clientes
  Select Nome
  Into   vNome
  From   Cliente;
  -- Imprimindo a variavel
  DBMS_OUTPUT.PUT_LINE('Nome: ' || vNome);
END;
```

```
ERROR at line 1:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 5
```

PL/SQL: executando consultas

- Retornando apenas um registro a cada execução:

```
DECLARE
  vNome   varchar2(50);
BEGIN
  -- Consultando o nome do cliente
  Select Nome
  Into   vNome
  From   Cliente
  Where  IDCliente = :p_IDCliente;

  DBMS_OUTPUT.PUT_LINE('Nome: ' || vNome);
END;
```

- Neste exemplo foi utilizado um parâmetro (apenas para SQL Developer) que permite substituir o valor em tempo de execução.

PL/SQL: comparador SE

IF - THEN - ELSIF - ELSE: estrutura do comparador de expressões.

```
DECLARE
  vTotal number(5);
BEGIN
  Select count(*)
    into vTotal
    from Cliente;

  IF (vTotal = 0) THEN
    DBMS_OUTPUT.PUT_LINE('Nenhum registro foi encontrado!');
  ELSIF (vTotal = 1) THEN
    DBMS_OUTPUT.PUT_LINE('Um registro encontrado!');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Muitos clientes cadastrados!!!');
  END IF;
END;
```

PL/SQL: comparador CASE

CASE - WHEN - THEN - ELSE: comparação de condições, possui a mesma estrutura utilizada no SQL.

```
DECLARE
  vSituacao   Cliente.Situacao%type;
BEGIN
  Select Situacao
  into   vSituacao
  from   Cliente
  where  IDCliente = :p_IDCliente;

  CASE vSituacao
    WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('Ativo!');
    WHEN 'I' THEN DBMS_OUTPUT.PUT_LINE('Inativo!');
    ELSE          DBMS_OUTPUT.PUT_LINE('Outro!');
  END CASE;
END;
```

PL/SQL: laços de repetição

WHILE LOOP: repetição de um bloco de comandos até que determinada condição seja atendida.

```
DECLARE
  vContador number(2);
BEGIN
   vContador := 1;

   WHILE vContador <= 30 LOOP
     DBMS_OUTPUT.PUT_LINE('Executou: '|| to_char(vContador, '90'));
     vContador := vContador + 1;
   END LOOP;
END;
```

Executará o que estiver dentro do laço enquanto a condição for verdadeira (enquanto a variável vContador for menor ou igual a 30).

PL/SQL: laços de repetição

FOR LOOP: repetição de um bloco de comandos com número de execuções pré-definido.

```
BEGIN
   FOR vContador IN 1..30 LOOP
     DBMS_OUTPUT.PUT_LINE('Executou: '|| to_char(vContador, '90'));
   END LOOP;
END;
```

Executará o laço de 1 até 30 atribuindo o índice corrente para a variável vContador.

A variável vContador é declarada **implicitamente**, e pode ser utilizada dentro do escopo do FOR LOOP, porém não permite que seja manipulada durante a execução.

PL/SQL: escopo

- É possível declarar outros blocos dentro de um.

```
DECLARE
    vTotal   Integer;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Valor= ' || to_char(vTotal));

    DECLARE
      vTexto varchar2(30) := 'Brasil!';
    BEGIN
      DBMS_OUTPUT.PUT_LINE('Pais= '|| vTexto);
    END;

END;
```

PL/SQL: tratando exceções

- **EXCEPTION:** tratando a ocorrência de erros no bloco.

  - A seção EXCEPTION deve ser declarada no final do bloco, antes de encerrar o bloco PL/SQL (END;).
  - A linguagem PL/SQL define alguns erros pré-definidos, que podem ser tratados através do nome do erro (ver tabela).
  - Permite que vários erros sejam tratados;
  - Permite a criação de exceções a nível de aplicação, com código próprio;

PL/SQL: tratando exceções

- Tratando a ocorrência de erros em comandos SELECT.

```
DECLARE
  vNome Cliente.Nome%Type;
BEGIN

  Select Nome
  Into   vNome
  From   Cliente
  Where  IDCliente = :p_IDCliente;

  DBMS_OUTPUT.PUT_LINE('Nome: ' || vNome);

EXCEPTION
  When no_data_found Then
    DBMS_OUTPUT.PUT_LINE('Cliente inexistente!');
END;
```

PL/SQL: tratando exceções

- Tratando vários erros:

```
DECLARE
 vNome Cliente.Nome%Type;
BEGIN

  Select Nome
  Into   vNome
  From   Cliente
  Where  IDCliente = :p_IDCliente;

  DBMS_OUTPUT.PUT_LINE('Nome: ' || vNome);

EXCEPTION
  When no_data_found Then
    DBMS_OUTPUT.PUT_LINE('Cliente inexistente!');
 When too_many_rows Then
    DBMS_OUTPUT.PUT_LINE('Existe mais de 1 registro!');
END;
```

PL/SQL: tratando exceções

- Tratando qualquer erro:

```
DECLARE
 vNome Cliente.Nome%Type;
BEGIN

  Select Nome
  Into   vNome
  From   Cliente
  Where  IDCliente = :p_IDCliente;

  DBMS_OUTPUT.PUT_LINE('Nome: ' || vNome);

EXCEPTION
  When no_data_found Then
    DBMS_OUTPUT.PUT_LINE('Cliente inexistente!');
 When others Then
    DBMS_OUTPUT.PUT_LINE('Erro Código=' || sqlerr);
END;
```

Site com dicas: https://oracle-base.com/articles/misc/introduction-to-plsql