PL/SQL – Parte 1

Curso de Introdução a Oracle 11g: SQL/Avançado

Prof.: Marlon Mendes Minussi

marlonminussi@gmail.br

Criando Rotinas PL/SQL

- Este tópico trata das funcionalidades de rotina de programação usando PL/SQL, como:
  - Procedimento, Funções, Pacotes e Gatilhos, que facilitam a criação de rotinas para a consulta e atualização de dados no banco de dados.

PROGRAMAÇÃO PL/SQL

- A programação PL/SQL (Procedural Language Extensions to SQL) acrescenta a construção de programas à linguagem SQL, resultando em uma linguagem estrutural mais poderosa do que o SQL.
- Todos os programas da PL/SQL são compostos por blocos, que podem ser aninhados em outros.
- O PL/SQL é uma linguagem padrão do Oracle desde a versão 6.0 lançada em 1991, quando surgiu o PL/SQL 1.0.
- A partir dai foi evoluindo conforme a evolução do próprio Oracle.
- A atual versão do PL/SQL é a 2.4

PROGRAMAÇÃO PL/SQL

ESTRUTURA DA LINGUAGEM CARACTERES SUPORTADOS

- Todo alfabeto maiúsculo e minúsculo
- Algarismos de 0 a 9
- Símbolos especiais: **( ) + - * / < > = ! ~ ; : ‘ @ % , “ # & $ _ | { } ? [ ] .**

OPERADORES ARITMÉTICOS

+ Adição **-** Subtração

* Multiplicação **/** Divisão

** Exponenciação

OPERAÇÕES RELACIONAIS

< > Diferente	**>** Maior
^ = Diferente	**<** Menor
! = Diferente	**> =** Maior Igual
= Igual	**< =** Menor Igual

PROGRAMAÇÃO PL/SQL

OUTROS SÍMBOLOS

( )	Separadores de lista	Ex.: AND MODELO IN('CARAZINHO','SÃO PAULO','SP')
;	Final de declaração	Ex.: COMMIT WORK;
.	Separador de item	Ex.: CLIENTES.CODIGO
'	Engloba uma string	Ex.: 'DIGIDATA'
: =	Atribuição	Ex.: NOME : = 'DIGIDATA'
| |	Concatenação	Ex.: 'Codigo Cliente: '| | CLIENTES.CODIGO
- -	Comentário na mesma linha	Ex.: Begin- - Inicio da execução

/* **e** ***/** Delimitadores de comentários abrangendo várias linhas (início e fim de comentário).

PROGRAMAÇÃO PL/SQL

TIPOS DE DADOS:

CHAR - Tipos de dados alfanuméricos

VARCHAR2 - Tipo alfanumérico com comprimento variável

NUMBER - Precisão de até 38 caracteres e ponto flutuante

DATE - Armazena valores de data de comprimento fixo

PROGRAMAÇÃO PL/SQL

- A linguagem SQL(Structure Query Language) define como os dados do Oracle são manipulados.
- As construções procedurais como PL/SQL tornam-se mais úteis quando combinadas com o poder de processamento da SQL, permitindo que os programas manipulem os dados no Oracle.
- As únicas instruções permitidas diretamente em um programa PL/SQL são DMLs (SELECT, INSERT, UPDATE, DELETE) e instruções controle de transações (COMMIT, ROLLBACK, SAVEPOINT).

PROGRAMAÇÃO PL/SQL

DECLARE

/*Declaração de variáveis, tipos, subprogramas*/ opcional

BEGIN

/*Seção das instruções SQL. Esta seção é única e indispensável*/ obrigatório

EXCEPTION

/*Seção para instruções de tratamento de erros*/ opcional

END;

- A ordem das partes é lógica, ou seja, primeiro deve-se efetuar as declarações, para depois utilizar (na lógica) as variáveis criadas. As exceções ocorridas durante a execução podem ser resolvidas na parte referente a erros.

PROGRAMAÇÃO PL/SQL

- Ex.:

```
DECLARE
        v_valor     NUMBER(5);
BEGIN
        SELECT valor INTO v_valor FROM atendimento
        WHERE cod_atendimento = 1;
        IF v_valor < 70.5
        THEN
                UPDATE atendimento
                        SET valor = v_valor * 1.3
                WHERE cod_atendimento = 1;
        ELSE
                UPDATE atendimento
                        SET valor = v_valor * 1.15
                WHERE cod_atendimento = 1;
        END IF;
        ROLLBACK;
END;
```

PROGRAMAÇÃO PL/SQL

- Os tipos de blocos PL/SQL podem ser: blocos anônimos (rotulados ou não), subprogramas (procedimentos e functions) ou triggers.
- Um comando importante para começar a programação PL/SQL é o comando de configuração que permite a emissão de mensagens:
- SET SERVEROUTPUT ON

PROGRAMAÇÃO PL/SQL

- Se desejar desativar a emissão de mensagens pode-se trocar o ON por OFF.
- Com a configuração de saída ativado (ON) pode-se usar o procedimento,
  - **DBMS_OUTPUT.PUT_LINE** que recebe como parâmetro uma mensagem ou valor.
- Um exemplo de codificação de saída PL/SQL é mostrado a seguir, em que se deseja saber quanto é 5.000 multiplicado por 1,5%, simulando o cálculo de juros de uma poupança.
- A lista de código é mostrado a seguir:

PROGRAMAÇÃO PL/SQL

Ex.:

```
DECLARE
   DINHEIRO NUMBER;
   PERCENTUAL_RENDIMENTO NUMBER;
BEGIN
   DINHEIRO := 5000;
   PERCENTUAL_RENDIMENTO := 0.015;
   DBMS_OUTPUT.PUT_LINE('DINHEIRO * % RENDIMENTOS');
   DBMS_OUTPUT.PUT_LINE(DINHEIRO*PERCENTUAL_RENDIMENTO);
END;
/
```

PROGRAMAÇÃO PL/SQL

- Note que depois do END foi digitado uma barra (/), que indica ao SQL Plus do Oracle para executar o código PL/SQL.

- Na interface web Oracle Application Express permite-se usar a barra ou não, já o SQL Plus exige a barra para que o código seja executado em seguida.

PROGRAMAÇÃO PL/SQL

- Pode-se informar n SQL Plus em vez da barra (/), o ponto (.), porém o código é colocado no buffer de memória e não é executado até que seja digitado o comando RUN.

- Um exemplo do uso do ponto em vez da barra é o momento a seguir:

PROGRAMAÇÃO PL/SQL

Ex.:

```
DECLARE
   DINHEIRO NUMBER;
   PERCENTUAL_RENDIMENTO NUMBER;
BEGIN
   DINHEIRO := 5000;
   PERCENTUAL_RENDIMENTO := 0.015;
   DBMS_OUTPUT.PUT_LINE('DINHEIRO*%RENDIMENTOS');
    DBMS_OUTPUT.PUT_LINE(DINHEIRO*PERCENTUAL_RENDIMENTO);
END;
.
RUN
```

PROGRAMAÇÃO PL/SQL

- Vamos desenvolver uma condição em PL/SQL do tipo anônimo não-rotulado para demonstrar o uso do comando case.

- Neste exemplo será declarada uma variável chamada v_teste do tipo numérico e atribuído a ela o valor 1 na própria declaração.

PROGRAMAÇÃO PL/SQL

- No corpo do código PL/SQL aparece uma mensagem de "Implementando PL/SQL no Oracle!", e será utilizado o comando case para testa o valor de v_teste.
- Caso o valor de v_teste seja igual a 1, deve ser mostrado as mensagens correspondentes a cada número.
- O código do exemplo está na listagem a seguir e com ele já podemos concluir uma primeira codificação em PL/SQL.

PROGRAMAÇÃO PL/SQL

```
DECLARE
   v_teste NUMBER :=1 ;
BEGIN
DBMS_OUTPUT.PUT_LINE('Planos de Saúde!');
CASE v_teste
   WHEN 1 THEN DBMS_OUTPUT.PUT_LINE ('IPE');
   WHEN 2 THEN DBMS_OUTPUT.PUT_LINE ('UNIMED');
   WHEN 3 THEN DBMS_OUTPUT.PUT_LINE ('CASSI');
   WHEN 4 THEN DBMS_OUTPUT.PUT_LINE ('SEM PLANO DE SAÚDE');
END CASE;
END;
/
```

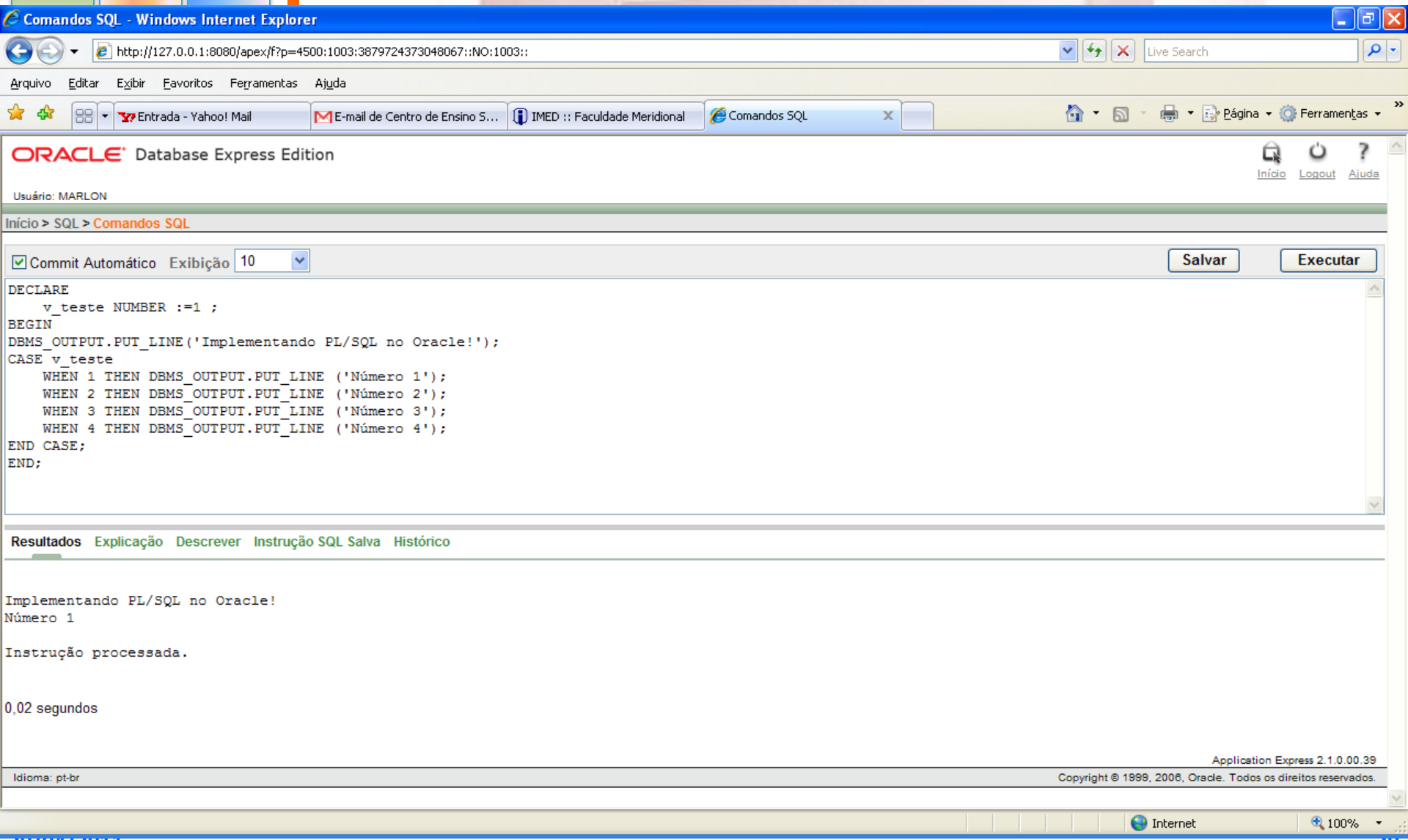
PROGRAMAÇÃO PL/SQL

- A execução do código PL/SQL na interface web do Oracle está na figura a seguir.

- Note que não há necessidade de digitar o comando /(barra) para que o código seja executado.

- Basta clicar no botão executar.

PROGRAMAÇÃO PL/SQL



PROGRAMAÇÃO PL/SQL

- Um outro exemplo de código PL/SQL é o da linguagem do próximo slide, em que é declarada uma variável chamada v_OutputStr, do tipo VARCHAR2 de tamanho 50, depois executa-se um comando SELECT que seleciona o nome do paciente na tabela paciente, cujo o código do paciente é igual a 8, e efetua-se a atribuição nome desse paciente à variável v_OutputStr. Por fim, emite-se uma mensagem que exibe o nome do paciente armazenado na variável v_OutputStr.

PROGRAMAÇÃO PL/SQL

```
DECLARE
   v_OutputStr VARCHAR2(50) ;
BEGIN
 SELECT nome
  INTO v_OutputStr
   FROM paciente
   WHERE cod_pac = 8;
 DBMS_OUTPUT.PUT_LINE (v_OutputStr);
END;
```

OBS:

- **Descrição**
  - O comando SELECT INTO cria uma nova tabela e a preenche com os dados produzidos por uma consulta.
  - Os dados não são retornados para o paciente, como acontece em um comando SELECT normal.
  - As colunas da nova tabela possuem os mesmos nomes e tipos de dado das colunas de saída do comando SELECT.

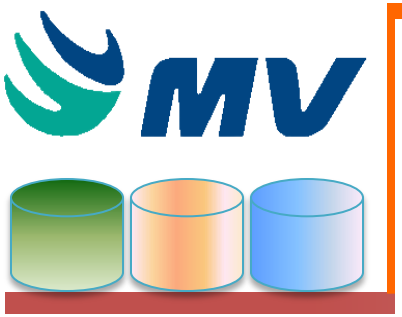
PROGRAMAÇÃO PL/SQL

- Veja outro exemplo de bloco anônimo não rotulado.
- Antes de executá-lo deve-se criar a tabela temporária que será utilizada no exemplo.

```
CREATE table temporaria(
Campo1 number,
Campo2 varchar2(50)
);
```

PROGRAMAÇÃO PL/SQL

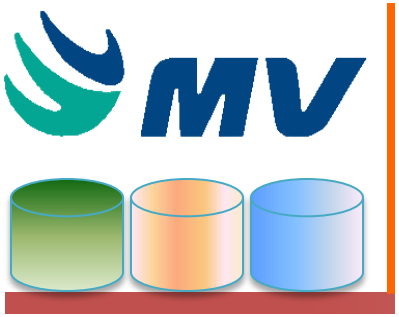
- O exemplo seguinte corresponde a um bloco anônimo não rotulado. Será usada uma tabela chamada “temporária” que possui dois campos, um denominado campo1 do tipo numérico e outro campo2 do tipo varchar2 (50).
- Serão efetuadas duas inserções nos dois campos e selecionados os dados que forem inseridos, mas antes do término será executado o comando ROLLBACK que desfaz as inserções.



```
BLOCO ANONIMO NÃO ROTULADO
DECLARE
 /*Declaração de variáveis para ser usada no bloco*/
 v_Num1    NUMBER := 1;
 v_Num2    NUMBER := 2;
 v_String1  VARCHAR2(50) := 'Mensagem1!';
 v_String2  VARCHAR2(50) := 'Mensagem2!';
 v_OutputStr  VARCHAR2(50);
BEGIN
       /*Inserção de duas linhas na tabela temporaria.*/
      INSERT INTO temporaria (campo1, campo2)
                VALUES (v_Num1, v_String1);
       INSERT INTO temporaria (campo1, campo2)
                VALUES (v_Num2, v_String2);
        /*Consulta à tabela temporaria.*/
      SELECT campo2
       INTO v_OutputStr
      FROM temporaria
       WHERE campo1 = v_Num1;
      DBMS_OUTPUT.PUT_LINE (v_OutputStr);
      SELECT campo2
       INTO v_OutputStr
       FROM temporaria
        WHERE campo1 = v_Num2;
      DBMS_OUTPUT.PUT_LINE (v_OutputStr);
 /* As inserções são desfeitas*/
 ROLLBACK;
END;
/
```

PROGRAMAÇÃO PL/SQL

- O próximo exemplo corresponde a um bloco anônimo rotulado.
- Será usada a mesma tabela temporária criada anteriormente, e o mesmo tipo de inserção e visualização do resultados, porém neste caso as informações são confirmadas com o comando COMMIT.



```
BLOCO ANONIMO ROTULADO
<<bloco_inserir>>
DECLARE
 /*Declaração de variáveis para ser usada no bloco*/
 v_Num1    NUMBER := 1;
 v_Num2    NUMBER := 2;
 v_sTRING1              VARCHAR2(50) := 'Mensagem1!';
 v_sTRING2              VARCHAR2(50) := 'Mensagem2!';
 v_OutputStr            VARCHAR2(50);
BEGIN
      /*Inserção de dus linhas na tabela temporaria.*/
     INSERT INTO temporaria (campo1, campo2)
               VALUES (v_Num1, v_String1);
     INSERT INTO temporaria (campo1, campo2)
              VALUES (v_Num2, v_String2);
      /*Consulta à tabela temporaria.*/
      SELECT campo2
       INTO v_OutputStr
     FROM temporaria
      WHERE campo1 = v_Num1;
     DBMS_OUTPUT.PUT_LINE (v_OutputStr);
     SELECT campo2
      INTO v_OutputStr
     FROM temporaria
     WHERE campo1 = v_Num2;
     DBMS_OUTPUT.PUT_LINE (v_OutputStr);
 /* As inserções são confirmadas*/
 COMMIT;
END bloco_inserir;
/
```

Caso queiram testar:

```
DECLARE
  v_Num1   NUMBER := 1;
  v_Num2   NUMBER := 2;
  v_OutputStr varchar2(50);
 begin
  SELECT campo2
    INTO v_OutputStr
    FROM temporaria
    WHERE campo1 = v_Num1;
  DBMS_OUTPUT.PUT_LINE (v_OutputStr);
END;
```

Exceções

- As exceções ocorrem quando no processamento de uma rotina acontece um erro. Os erros no Oracle são nomeados como ORA+ um traço e um número.
- O seguinte site possui o significado das mensagens de erro do Oracle.
  - http://www.ora-error.com/
  - http://www.ora-code.com/
  - http://www.techonthenet.com/oracle/erros/index.php
- Um exemplo de erro no Oracle é o ORA-01403 (no data found) que significa que nenhum dado foi encontrado.
- Para simular a ocorrência dessa execução, leva-se em consideração que não há nenhum (atendimento) cadastrado com código igual a 35.

Exceções

- Vamos criar uma rotina e tentar selecionar o código de um atendimento igual a 35, desta forma força-se a ocorrência de um erro.
- Ex.:

```
DECLARE
v_OutputStr VARCHAR2(50);
BEGIN
        SELECT cod_atendimento
        INTO v_OutputStr
        FROM atendimento
        WHERE cod_atendimento = 35;
        DBMS_OUTPUT.PUT_LINE(V_OutputStr);
 END;
```

Exceções

- Pode-se tratar a exceção.
- Nesta rotina usa-se a seção EXCEPTION para tratar a exceção gerada, que exibe uma mensagem personalizada.
- A listagem do código é mostrada a seguir:

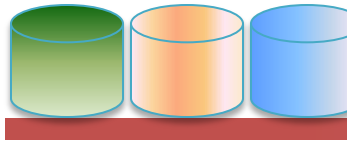
Tratamento de Exceção

Ex.:

```
DECLARE
v_OutputStr VARCHAR2(50);
BEGIN
        SELECT cod_atendimento
        INTO v_OutputStr
        FROM atendimento
        WHERE cod_atendimento = 35;
        DBMS_OUTPUT.PUT_LINE(V_OutputStr);
  EXCEPTION
      WHEN NO_DATA_FOUND THEN
      BEGIN
      --Caso não seja selecionado o nenhum registro para este atendimento emite-se a mensagem
      DBMS_OUTPUT.PUT_LINE('Não há atendimentos com este código!');
     END;
  END;
```

Escopo de Variável e Visibilidade

- O escopo de uma variável é a parte do programa em que a variável pode ser acessada antes de ser liberada da memória.
- A visibilidade de uma variável é parte do programa em que a variável pode ser acessada sem ter de qualificar a referência.



```
<<bloco1>>
DECLARE
v_nome CHAR(100);
v_idade INT;
BEGIN
/*Aqui v_nome e v_idade estão visíveis*/
v_nome := 'Andreia';
v_idade := 20;
    DECLARE
    v_endereco CHAR(100);
    v_idade CHAR(10);
    BEGIN
    v_endereco := ‘Av. Brasil';
    v_idade := '30';
    /*Aqui v_endereco CHAR(100), v_idade CHAR(10) estão visíveis. Para acessar v_idade
      INT , é nescessário informar o  bloco a que pertence: bloco.v_idade*/
     DBMS_OUTPUT.PUT_LINE(v_nome);
      DBMS_OUTPUT.PUT_LINE(bloco1.v_idade);
      DBMS_OUTPUT.PUT_LINE(v_idade);
      DBMS_OUTPUT.PUT_LINE(v_endereco);
      END;
      /*Aqui  v_endereco não , mais visível*/
--DBMS_OUTPUT.PUT_LINE(v_endereco);
END bloco1;
```

Escopo de Variável e Visibilidade

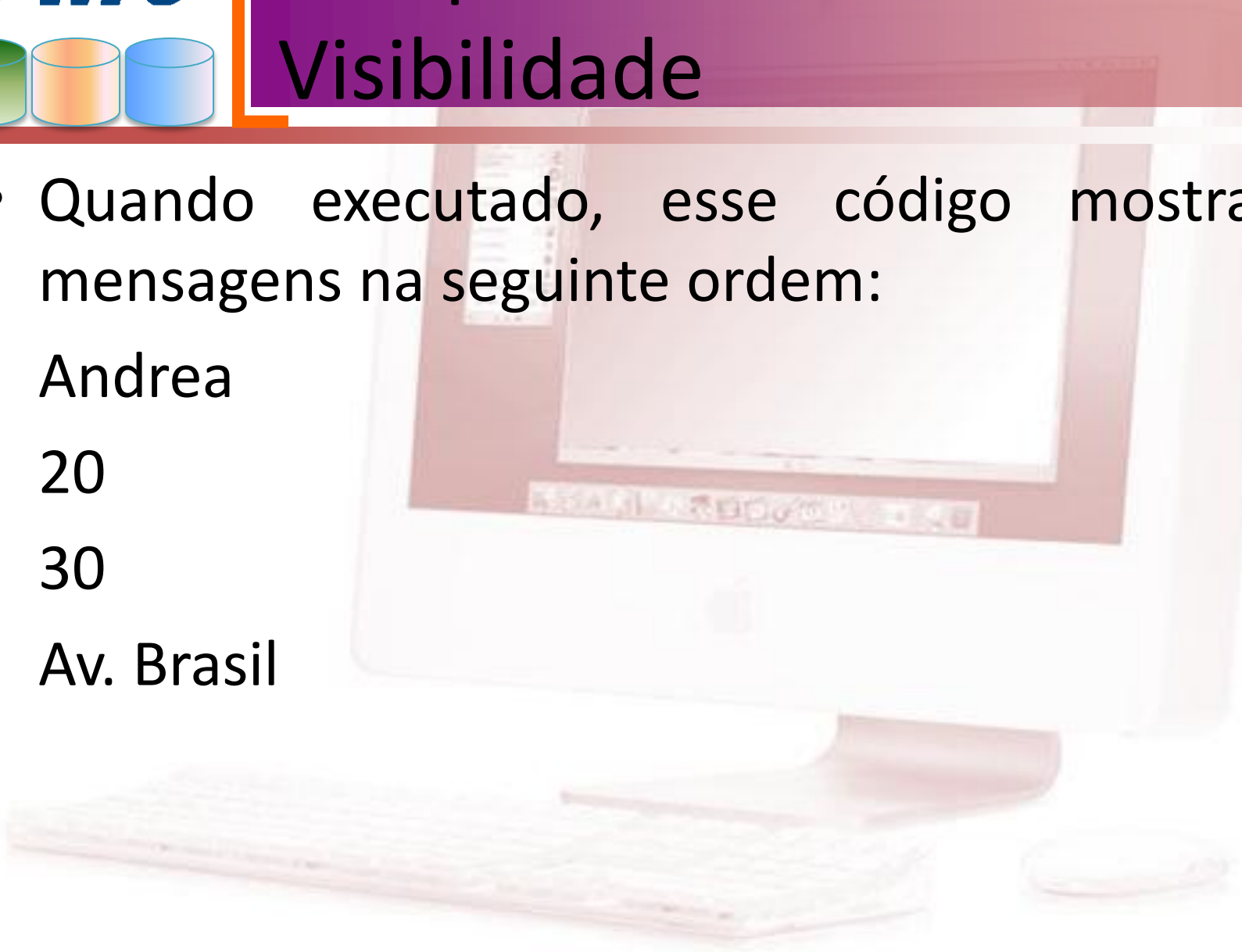
- Quando executado, esse código mostra as mensagens na seguinte ordem:

  Andrea

  20

  30

  Av. Brasil



IF-THEN-ELSE

Sintaxe:

1.
```
IF <condição> THEN
   <comandos>
END IF;
```

2.
```
IF <condição> THEN
   <comandos>
ELSE
   <comandos>
END IF;
```

3.
```
IF <condição> THEN
      <comandos>
ELSIF <condição> THEN
      <comandos>
END IF;
```

4.
```
IF <condição> THEN
      <comandos>
ELSIF <condição> THEN
      <comandos>
   ELSE
      <comandos>
END IF;
```

4.
```
IF <condição> THEN
      IF <condição> THEN
         <comandos>
      END IF;
END IF;
```

IF-THEN-ELSE

- No exemplo seleciona-se a soma dos valores das consultas:
  - e utiliza-se o IF para verificar se a soma dos valores é menor ou igual a 400, emitindo a seguinte mensagem:
    - ‘A Clínica não vai bem!’
  - depois, testa-se a soma dos valores é maior que 400 e menor que 1200 e exibindo a seguinte mensagem:
    - 'A Clínica está mais ou menos!‘
  - caso contrário exiba:
    - 'A Clínica está indo bem!';

```
DECLARE
v_valor atendimento.valor%type;
v_mensagem VARCHAR2(100);
BEGIN
  SELECT SUM(valor)
        INTO v_valor
        FROM  atendimento;
  IF v_valor <= 400 THEN
        v_mensagem:= 'A Clínica não está  indo bem.';
  ELSIF v_valor > 400 and v_valor < 1200 THEN
         v_mensagem:='A Clínica está mais ou menos!';
         ELSE
             v_mensagem:= 'A Clínica está indo bem!';
  END IF;
  DBMS_OUTPUT.PUT_LINE('Valor : ' || to_char (v_valor));
  DBMS_OUTPUT.PUT_LINE(v_mensagem);
END;
/
```

LOOP

Sintaxe:

1. LOOP
   <seqüência de comandos>
   END LOOP

2. LOOP
   <seqüência de comandos>
   IF ....
   THEN
   EXIT; -- encerra o loop
   END IF;
   END LOOP;

LOOP

Ex.:

```
DECLARE
X NUMBER := 0;
COUNTER NUMBER := 0;
BEGIN
  LOOP
    X := X + 1000;
    COUNTER := COUNTER + 1;
    IF COUNTER > 4 THEN EXIT;
    END IF;
    DBMS_OUTPUT.PUT_LINE (X ||' '|| COUNTER || 'LOOP');
  END LOOP;
END;
/
```

WHILE

- É possível criar loops com while, cuja condição é avaliada antes de cada iteração do loop.
- Sintaxe:

```
WHILE condição LOOP
    Sequência_da_intrução;
END LOOP;
```

» Na listagem seguinte demonstra-se o uso do while em que são inseridos registros na tabela temporária que foi criada no início do capítulo. Antes de executar o exemplo seguinte pode-se excluir todos os registros da tabela temporaria.

```
DELETE FROM  temporaria
```

WHILE

Ex.:

```
DECLARE
   v_Counter BINARY_INTEGER := 1;
 BEGIN
   -- Testa o contador do loop antes de cada interação
   WHILE v_Counter <= 50 LOOP
         INSERT INTO temporaria
               VALUES (v_Counter, 'Loop index');
         v_Counter := v_counter + 1;
   END LOOP;
 END;
```

FOR

- Pode-se criar também um loop usando FOR, que possui um número definido de iterações.
- Sintaxe:

```
FOR contador IN [REVERSE] limite_inferior .. limite_superior LOOP
END LOOP
```

FOR

- Ex.:

```
DECLARE
A NUMBER(3):= 0;
BEGIN
 FOR A IN 1..25 LOOP
   DBMS_OUTPUT.PUT_LINE('LOOP1 - '||A);
  END LOOP;
END;
/
```

FOR

Ex.:

```
BEGIN
   --Não há necessidade de declarar v_contador
   --Ela é declarada automaticamente como BINARY_INTEGER
   FOR v_contador IN 1..50 LOOP
         INSERT INTO temporaria
           VALUES (v_contador, 'Programa com estrutura FOR
           IN');
  END LOOP;
END;
/
```

FOR

- Se a palavra REVERSE estivar presente no loop FOR, o índice de LOOP irá iterar a partir do valor maior para o valor menor.

Ex.:

```
BEGIN
    --Não há  necessidade de declarar v_contador
    --Ela é declarada automaticamente como BINARY_INTEGER
    FOR v_contador IN REVERSE 1..50 LOOP
INSERT INTO temporaria
    VALUES (v_contador, 'Programa com estrutura FOR IN REVERSE');
     END LOOP;
END;
```

GOTO

- A linguagem PL/SQL também possui uma instrução GOTO para passar o controle para uma série específica do bloco.
- A PL/SQL não permite utilizar GOTO para desviar o controle para um bloco interno, para dentro de uma condição IF.

GOTO

```
DECLARE
    v_contador BINARY_INTEGER := 1;
    BEGIN
        -- Testa o contador do loop antes de cada interação
      LOOP
          INSERT INTO temporaria
              VALUES (v_contador, 'Usando Loop GOTO');
          v_contador : = v_contador + 1;
          IF v_contador = 50 THEN
              GOTO Fim_do_Loop;
          END IF;
      END LOOP;
      <<Fim_do_Loop>>
      INSERT INTO temporaria
        VALUES (v_contador, 'Fim!!!');
    END;
    /
```