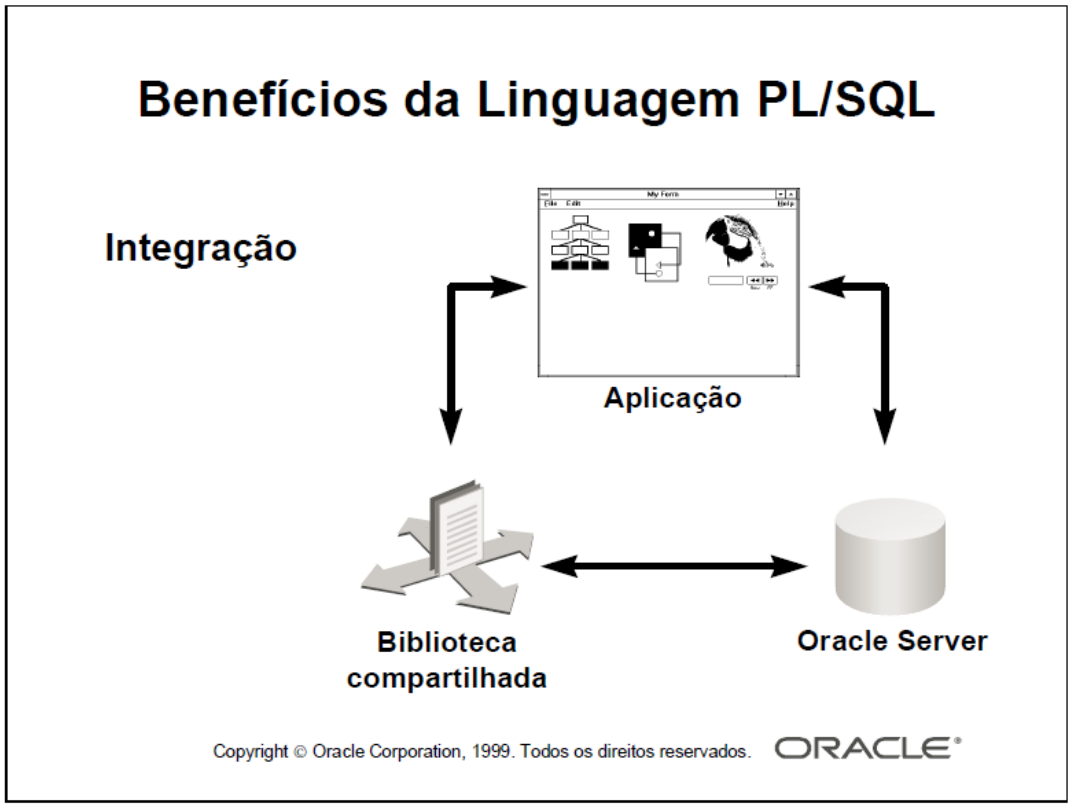
Curso Intensivo de PL/SQL

1. **Introdução ao Código PL/SQL**
2. **Blocos PL/SQL, Declaração de Identificadores e Tipos de Dados PL/SQL**
3. **Criação de Estruturas de Controle**
4. **Cursores Implícitos e Explícitos**
5. **Tratamento de Exceções**
6. **Criação de Procedures**
7. **Criação de Functions**
8. **Criação de Triggers**
9. **Criação de Packages**
10. **SQL Dinâmico**
11. **BULK COLLECT**
12. **Transações Autónomas.**

Material Elaborado por: Prof. Antonio Carlos da Silva (Toninho)
www.proftoninho.com

Curso Intensivo de PL/SQL

1. **Introdução ao Código PL/SQL**
   Sobre PL/SQL
   A linguagem PL/SQL (Procedural Language/SQL) é uma extensão de linguagem procedural da Oracle Corporation para SQL, a linguagem de acesso a dados padrão para bancos de dados relacionais. A linguagem PL/SQL oferece recursos de engenharia de software modernos, como, por exemplo, a encapsulação de dados, o tratamento de exceções, a ocultação de informações e a orientação a objeto, etc., trazendo os recursos de programação mais modernos para o Oracle Server e o Toolset.
   A linguagem PL/SQL incorpora muitos recursos avançados criados em linguagens de programação projetadas durante as décadas de 70 e 80. Além de aceitar a manipulação de dados, ele também permite que as instruções de consulta da linguagem SQL sejam incluídas em unidades procedurais de código e estruturadas em blocos, tornando a linguagem SQL uma linguagem avançada de processamento de transações. Com a linguagem PL/SQL, você pode usar as instruções SQL para refinar os dados do Oracle e as instruções de controle PL/SQL para processar os dados.



Integração
A linguagem PL/SQL desempenha um papel central tanto para o Oracle Server (através de procedimentos armazenados, funções armazenadas, gatilhos de banco de dados e pacotes) quanto para as ferramentas de desenvolvimento Oracle (através de gatilhos de componente do Oracle Developer).
As aplicações do Oracle Developer fazem uso das bibliotecas compartilhadas que armazenam código (procedimentos e funções) e que podem ser acessadas local ou remotamente. O Oracle Developer consiste no Oracle Forms, Oracle Reports e Oracle Graphics.

Os tipos de dados SQL também podem ser usados no código PL/SQL. Combinados com o acesso direto que a linguagem SQL fornece, esses tipos de dados compartilhados integram a linguagem PL/SQL com o dicionário de dados do Oracle Server. A linguagem PL/SQL une o acesso conveniente à tecnologia de banco de dados com a necessidade de capacidade de programação procedural.

PL/SQL nas Ferramentas Oracle

Várias ferramentas Oracle, inclusive o Oracle Developer, têm o seu próprio mecanismo PL/SQL, o qual é independente do mecanismo presente no Oracle Server.

O mecanismo filtra as instruções SQL e as envia individualmente ao executor da instrução SQL no Oracle Server. Ele processa as instruções procedurais restantes no executor da instrução procedural, que está no mecanismo PL/SQL.

O executor da instrução procedural processa os dados que são locais para a aplicação (que já estão no ambiente do cliente, em vez de estarem no banco de dados). Isso reduz o trabalho enviado ao Oracle Server e o número de cursores de memória necessários.

2. Blocos PL/SQL , Operadores, Declaração de Identificadores e Tipos de Dados PL/SQL

Blocos PL / SQL

Um programa consiste em pelo menos um bloco.
Os blocos de PL / SQL pode ser dos seguintes tipos:
Blocos anônimos
Subprogramas
Estrutura de um bloco

Os blocos PL / SQL tem uma estrutura específica composta por três partes distintas:

Seção declarativa, onde são declarados todas as constantes e variáveis que serão utilizadas na execução do bloco.
Seção de implementação, onde inclui instruções para a execução no bloco PL / SQL.
E seção de exceções, onde serão tratadas as falhas que ocorram no bloco PL / SQL.
Cada uma das partes acima é delimitada por uma palavra reservada, de modo que um bloco PL / SQL pode ser representada como se segue:

```
[ declare | is | as ]
    /*Parte declarativa*/
begin
    /*Parte de execução*/
[ exception ]
    /*Parte de exceções*/
end;
```

Na estrutura do bloco apresentado anteriormente, somente a seção de execução é obrigatório.
Ela é definida entre as cláusulas **BEGIN** e **END**.
A seguir temos um exemplo de bloco PL/SQL genérico. O bloco anônimo não possui um nome.
Podemos identificar um bloco anônimo pela sua parte declarativa com o uso da palavra-chave **DECLARE**.

```
DECLARE
    /*Parte declarativa*/
    nome_variavel DATE;
BEGIN
    /*Parte de execução
    * Este código atribui o valor da coluna "column_name"
    * Identificado pelo "nome_variavel" variável
    */
    SELECT SYSDATE
    INTO    nome_variavel
    FROM DUAL;
EXCEPTION
    /*Parte de exceções*/
    WHEN OTHERS THEN
    dbms_output.put_line('Erro!!!');
END;
```

Operadores

Operador de atribuição

```
:= (dois pontos + igual)
```

Operador de passagem de valor para variável

&

Operador de concatenação

||

Operadores aritméticos

Para criar expressões aritméticas em uma consulta SQL usamos os operadores abaixo:

```
+   somar
-   subtrair
*   multiplicar
/   dividir
```

Operadores de comparação

Os operadores de comparação são usados em condições que comparam uma expressão a outro valor ou expressão. Para isso utilizamos os operadores abaixo:

```
=   Igual a
>   Maior que
>=  Maior ou igual a que
<   Menor que
<=  Menor ou igual a que
<>  Diferente de
```

Operadores lógicos

O uso de um operador lógico faz com que duas condições tenham de produzir um resultado único.
Uma linha só poderá ser retornada se o resultado global da condição for verdadeiro.
A tabela abaixo mostra os operadores lógicos disponíveis em SQL:

```
AND Retorna TRUE se ambas as condições forem verdadeiras
OR  Retorna TRUE se uma das condições for verdadeira
NOT Retorna TRUE se a condição seguinte for falsa
```

Seção de Declaração Variáveis

Nesta parte são declaradas as variáveis necessárias para a execução do nosso programa. A variável é declarada através da atribuição de um nome ou "identificador", seguido do tipo de dado.
Nesta seção também são declarados os Cursores, muito utilizados em consultas que retornam um grande volume de dados. Podemos declarar também exceções definidas pelo usuário. Aqui você também pode especificar uma constante, de valor nulo ou com um valor inicial determinado.

A sintaxe genérica para a declaração de constantes e variáveis é:

```
nome_variavel [CONSTANT] <tipo_dado> [NOT NULL][:=valor_inicial]
```

Onde:

tipo_dado: define o tipo de dado da variável, ou seja, qualquer um dos tipos suportador pela Oracle, entre eles:
NUMBER, **DATE**, **CHAR, VARCHAR, VARCHAR2, BOOLEAN**, entre outros.
Existem tipos de de dados **(NUMBER e VARCHAR)** que podemos especificar o tamanho.
A cláusula **CONSTANT** indica a definição de uma constante cujo valor não pode ser modificado e deve ter um valor **(valor_inicial)** informado em sua declaração.
A cláusula **NOT NULL** impede uma variável seja atribuído com um valor nulo, e, portanto, deve ser inicializado com um valor.

Você também pode definir o tipo de uma variável ou constante, dependendo do tipo de outro identificador, usando cláusulas**% TYPE** e**% ROWTYPE**. Normalmente**% TYPE** é usada para definir a variável do mesmo tipo que foi definido um campo em uma tabela no banco de dados, enquanto**% ROWTYPE** é usada para declarar variáveis usando cursores.

Exemplos:

Estrutura de um bloco anônimo.

```
DECLARE
/*  Declaração de variável do tipo VARCHAR2(15) identificada por v_localizacao
    atribuido do valor "Brasil" */
v_location VARCHAR2(15) := 'Brasil';

/*  Declaração de constante do tipo NUMBER identificada por PI
    atribuido do valor 3.1416   */
PI CONSTANT NUMBER := 3.1416;

/*  Declaração de variável do mesmo tipo que coluna da tabela "emp" identificada
    por v_nome sem atribuição de valor  */
v_nome emp.name%TYPE;


/*  Declaração da variável do tipo de registro correspondente ao suposto cursor
    chamado de "meucursor", identificada por reg_dados  */
reg_dados meucursor%ROWTYPE;

BEGIN
    /*Parte de execução*/
EXCEPTION
    /*Parte de exceção*/
END;
```

Estrutura de um Subprograma (Procedure).

Curso Intensivo de PL/SQL

```
/*  Criação de uma Procedure */
CREATE OR REPLACE PRC_SIMPLES IS
/*  Declaração de variável do tipo VARCHAR2(15) identificada por v_localizacao
    atribuido do valor "Brasil" */
v_location VARCHAR2(15) := 'Brasil';

/*  Declaração de constante do tipo NUMBER identificada por PI
    atribuido do valor 3.1416   */
PI CONSTANT NUMBER := 3.1416;

/*  Declaração de variável do mesmo tipo que coluna da tabela "emp" identificada
    por v_nome sem atribuição de valor  */
v_nome emp.name%TYPE;


/*  Declaração da variável do tipo de registro correspondente ao suposto cursor
    chamado de "meucursor", identificada por reg_dados  */
reg_dados meucursor%ROWTYPE;

BEGIN
    /*Parte de execução*/
EXCEPTION
    /*Parte de exceção*/
END;
```

3. Criação de Estruturas de Controle

INSTRUÇÕES CONDICIONAIS:

IF – THEN – END IF

```
DECLARE
    V_NUM   NUMBER:= &NUM;
BEGIN
    IF  0 = V_NUM   THEN
        DBMS_OUTPUT.PUT_LINE('Zero Igual a Zero');
    END IF;
END;
```

IF – THEN – ELSE – END IF

```
DECLARE
    V_NUM   NUMBER:= &NUM;
BEGIN
    IF  0 <> V_NUM  THEN
        DBMS_OUTPUT.PUT_LINE('Zero Igual a Zero');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Zero Diferente de Zero');
    END IF;
END;
```

IF – THEN – ELSIF – END IF

```
DECLARE
    V_NUM   NUMBER:= &NUM;
BEGIN
    IF  0 = V_NUM   THEN
        DBMS_OUTPUT.PUT_LINE('Zero Igual a Zero');
    ELSIF   1 = V_NUM THEN
        DBMS_OUTPUT.PUT_LINE('Um Igual a Um');
    ELSE
        DBMS_OUTPUT.PUT_LINE(v_num ||'<> 0 e 1');
    END IF;
END;
```

Curso Intensivo de PL/SQL

CICLOS:

LOOP

```
DECLARE
    V_CONT  NUMBER:= 0;
BEGIN
    LOOP
        V_CONT  :=  V_CONT + 1;
        Dbms_Output.Put_Line('LOOP: '||V_CONT);
        EXIT WHEN V_CONT = 5;
    END LOOP;
END;
--> Outros exemplos utilizando LOOP
--> Declaração CONTINUE
DECLARE
    V_CONT  NUMBER:= 0;
BEGIN
    LOOP
        V_CONT  :=  V_CONT + 1;
        if  v_cont < 3 then
            continue;
        end if;
        Dbms_Output.Put_Line('LOOP: '||V_CONT);
        EXIT WHEN V_CONT = 5;
    END LOOP;
END;

--> Declaração CONTINUE WHEN
DECLARE
    V_CONT  NUMBER:= 0;
BEGIN
    LOOP
        V_CONT  :=  V_CONT + 1;
        CONTINUE WHEN V_CONT < 3;
        Dbms_Output.Put_Line('LOOP: '||V_CONT);
        EXIT WHEN V_CONT = 5;
    END LOOP;
END;
```

WHILE

```
DECLARE
    V_CONT  NUMBER:= 0;
BEGIN
    WHILE   V_CONT  <   5   LOOP
        V_CONT  :=  V_CONT + 1;
        Dbms_Output.Put_Line('WHILE: '||V_CONT);
    END LOOP;
END;
```

FOR LOOP

```
DECLARE
    V_CONT  NUMBER:= 0;
BEGIN
    FOR X   IN  1..5    LOOP
        Dbms_Output.Put_Line('FOR LOOP: '||X);
    END LOOP;
END;
```

Curso Intensivo de PL/SQL

Material Elaborado por: Prof. Antonio Carlos da Silva (Toninho)
www.proftoninho.com

Curso Intensivo de PL/SQL

FOR LOOP + Exemplos

```
--> Outros exemplos utilizando FOR LOOP
--> Utilizando REVERSE
BEGIN
    FOR X   IN  REVERSE 1..3    LOOP
        Dbms_Output.Put_Line('REVERSE FOR LOOP: X = '||X);
    END LOOP;
END;

--> Utilizando variável para calcular o contador do FOR LOOP
DECLARE
    V_STEP  NUMBER := 5;
BEGIN
    FOR X   IN   1..3   LOOP
        Dbms_Output.Put_Line('STEP FOR LOOP: X = '||X * V_STEP);
    END LOOP;
END;


--> Imprimir "X" dentro e fora do FOR LOOP
BEGIN
    FOR X   IN   1..3   LOOP
        Dbms_Output.Put_Line('Dentro do FOR LOOP: X= '|| X);
    END LOOP;
        Dbms_Output.Put_Line('Fora FOR LOOP: X = '|| X);
END;
--> ERRO GERADO
/*
ORA-06550: linha 6, coluna 45:
PLS-00201: o identificador 'X' deve ser declarado
ORA-06550: linha 6, coluna 4:
PL/SQL: Statement ignored
*/

--> Imprimir "X" dentro e fora do FOR LOOP
--> SOLUÇÃO
--> Definir uma variável "X" para a impressão externa ao FOR LOOP
DECLARE
    X   NUMBER:=5;
BEGIN
    --> Imprimir o contador "X" fora do FOR LOOP
    FOR X   IN   1..3   LOOP
        Dbms_Output.Put_Line('Dentro do FOR LOOP: X = '|| X);
    END LOOP;
        Dbms_Output.Put_Line('Fora FOR LOOP: X= '|| X);
END;
```

```
BEGIN
    --> Imprimir o contador "X" fora do FOR LOOP
    FOR X   IN   1..3   LOOP
        Dbms_Output.Put_Line('Dentro do FOR LOOP: '|| X);
    END LOOP;
        Dbms_Output.Put_Line('Fora FOR LOOP: '|| X);
END;
--> ERRO GERADO
/*
ORA-06550: linha 6, coluna 45:
PLS-00201: o identificador 'X' deve ser declarado
ORA-06550: linha 6, coluna 4:
PL/SQL: Statement ignored
*/

--> SOLUÇÃO
--> Definir uma variável "X" para a impressão externa ao FOR LOOP
DECLARE
    X   NUMBER:=10;
BEGIN
    --> Imprimir o contador "X" fora do FOR LOOP
    FOR X   IN   1..3   LOOP
        Dbms_Output.Put_Line('Dentro do FOR LOOP: '|| X);
    END LOOP;
        Dbms_Output.Put_Line('Fora FOR LOOP: '|| X);
END;


--> Utilizando variável global
<<main>>  --> label do bloco
DECLARE
    X   NUMBER:=5;
BEGIN
    --> Imprimir o contador "X" fora do FOR LOOP
    FOR X   IN   1..3   LOOP
        Dbms_Output.Put_Line('local: '|| X ||', '||' global: '||main.x);
        --> "main.x" -  referência com a etiqueta do bloco
    END LOOP;
END main;
```

4. Cursores Implícitos e Explícitos

Cursores explícitos

Os comandos %FOUND, %NOTFOUND, e %ROWCOUNT são atributos do cursor que podem ser utilizados para determinar finalização do loop.

Exemplo: %NOTFOUND

```
DECLARE
CURSOR C_EMP IS
        SELECT EMPNO, ENAME
        FROM EMP
        ORDER BY EMPNO;

V_ID      EMP.EMPNO%TYPE;
V_NOME    EMP.ENAME%TYPE;
BEGIN

    OPEN C_EMP;

    /*  Use o loop básico para buscar as linhas do cursor e
        imprimir cada uma das linhas até que todas sejam
        apresentadas    */

    LOOP
        FETCH C_EMP INTO V_ID, V_NOME;
        /* Use o atributo do cursor %NOTFOUND para determinar
        quando irá sair do loop básico */

        EXIT WHEN C_EMP%NOTFOUND;

        DBMS_OUTPUT.PUT_LINE(V_ID||' '||V_NOME);
    END LOOP;
    CLOSE C_EMP;
END;
```

Exemplo: %ROWCOUNT

```
DECLARE
    CURSOR C_ORG IS
        SELECT EMPNO, ENAME
        FROM   EMP;
    --
    NUM_TOTAL_ROWS NUMBER:=0;
BEGIN
    FOR X IN C_ORG LOOP
        /*  Verifica se é a primeira passagem do loop */
        IF C_ORG%ROWCOUNT = 1 THEN
            DBMS_OUTPUT.PUT_LINE('***************');
        END IF;
        /*  Atribui o valor 1 referente a primeira passagem do loop.
            Gerando assim um contador */
        NUM_TOTAL_ROWS := C_ORG%ROWCOUNT;
        DBMS_OUTPUT.PUT_LINE('EMPRESAS = '||TO_CHAR(NUM_TOTAL_ROWS));

    END LOOP;
    IF NUM_TOTAL_ROWS > 0 THEN
        DBMS_OUTPUT.NEW_LINE;
        DBMS_OUTPUT.PUT_LINE('TOTAL DE EMPRESAS = '||TO_CHAR(NUM_TOTAL_ROWS));
    END IF;
END;
```

Exemplo: Utilizando FOR sem o %ROWCOUNT para dar o mesmo resultado apresentado anteriormente

```
--> Utilizando FOR sem %ROWCOUNT
DECLARE
    CURSOR C_ORG IS
        SELECT EMPNO, ENAME
        FROM   EMP;
    --
    V_CONT  NUMBER := 0;
BEGIN
    FOR X IN C_ORG LOOP
        V_CONT  := V_CONT + 1;
        DBMS_OUTPUT.PUT_LINE('EMPRESAS: '||V_CONT);
    END LOOP;
        DBMS_OUTPUT.PUT_LINE('TOTAL DE EMPRESAS = '||TO_CHAR(V_CONT));
END;
```

Exemplo: %FOUND

```
DECLARE
    V_ORDER_NO              ITEM.ORDID%TYPE;
    V_TOT_ORDER_PRICE       ITEM.ACTUALPRICE%TYPE;
    CURSOR C1 IS
    SELECT ORDID, SUM(ACTUALPRICE * QTY + 1) TOTAL
    FROM ITEM
    GROUP BY ORDID;
BEGIN
    OPEN C1;
    FETCH C1 INTO V_ORDER_NO, V_TOT_ORDER_PRICE;
    /* Enquanto for encontrado registro no cursor mantenha o LOOP */
        WHILE C1%FOUND LOOP
            UPDATE ORD
            SET TOTAL = V_TOT_ORDER_PRICE
            WHERE ORDID = V_ORDER_NO;
            /*  A busca "FETCH" se repete para finalizar o LOOP do WHILE,
                senão ficará em LOOP infinito   */
            FETCH C1 INTO V_ORDER_NO, V_TOT_ORDER_PRICE;
        END LOOP;
    CLOSE C1;
END;
```

Cursores implícitos

SQL%NOTFOUND

```
BEGIN
UPDATE employee SET salary = 100 WHERE id = '20';
/*  Se o UPDATE não atualizar nenhuma linha,
        INSERT uma nova linha na tabela.    */
IF SQL%NOTFOUND THEN
    INSERT INTO employee (id, salary) VALUES ('99', 100);
    DBMS_OUTPUT.put_line('not found');
END IF;
END;
```

Exemplo: Cursor Implícito com Select SEM a cláusula "INTO"

```
BEGIN
    /*  Este tipo de estrutura de cursor implícito é criado quando
        queremos retornar várias linhas registros   */
    FOR CUR1 IN (SELECT * FROM EMPLOYEE) LOOP
        DBMS_OUTPUT.PUT_LINE(CUR1.ID || '= id funcionario');
    END LOOP;
END;
```

Exemplo: Cursor Implícito com Select COM a cláusula "INTO"

```
DECLARE
    V_DATA  DATE;
BEGIN
    /*  Este tipo de estrutura de cursor implícito é criado para
        retornar apenas uma linha registro  */
    SELECT  SYSDATE
    INTO    V_DATA
    FROM    DUAL;
    DBMS_OUTPUT.PUT_LINE('DATA DO SERVIDOR ORACLE: '||V_DATA);
END;
```

5. Tratamento de Exceções
Exceptions com RAISE

```
DECLARE
    V_TEMP      NUMBER:= &TEMP;
    V_QUENTE    EXCEPTION;
    V_FRIO      EXCEPTION;
BEGIN
    CASE
        WHEN V_TEMP < 90.00 THEN RAISE V_FRIO;
        WHEN V_TEMP > 140.00 THEN RAISE V_QUENTE;
        ELSE NULL;
    END CASE;
    DBMS_OUTPUT.PUT_LINE('A TEMPERATURA ESTÁ BOA.');
EXCEPTION
    WHEN VALUE_ERROR THEN
        DBMS_OUTPUT.PUT_LINE('POR FAVOR INSERIR UMA TEMPERATURA DE VALOR NUMÉRICO (COMO 100)');
    WHEN V_QUENTE THEN
        DBMS_OUTPUT.PUT_LINE('O TEMPO ESTÁ MUITO QUENTE...');
    WHEN V_FRIO THEN
        DBMS_OUTPUT.PUT_LINE('O TEMPO ESTÁ MUITO FRIO...');
END;
```

Exceptions padrões: NO_DATA_FOUND, TOO_MANY_ROWS e OTHERS

```
DECLARE
    V_SAL   NUMBER;
BEGIN
    /*  Para simular as EXCEPTIONS utilize o valores informados:
        Para NO_DATA_FOUND: utilize "WHERE  EMPNO = 1;"
        Para TOO_MANY_ROWS: utilize "WHERE  EMPNO = EMPNO;"
        Para OTHERS: utilize "WHERE EMPNO = 'A';"
        ---> Para rodar sem erros utilizar "WHERE   EMPNO = 7839;" <---
    */
    SELECT  SAL
    INTO    V_SAL
    FROM    EMP
    WHERE   EMPNO = 'A';

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('NAO ENCONTROU DADOS DO REGISTRO.');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('RETORNOU MAIS DE UM REGISTRO.');
    WHEN OTHERS THEN
        /*  SQLERRM: Retorna o erro ocorrido tratado pelo Banco Oracle
            SQLCODE: Retorna o número do erro ocorrido  */
        DBMS_OUTPUT.PUT_LINE('ERRO SIMULACAO : '||SQLERRM);
END;
```

Curso Intensivo de PL/SQL

Lista com as Principais Exceptions utilizados no PL/SQL.

Para sabe o motivo dos erros acesse: **http://psoug.org/oraerror/**

Predefined (Named) PL/SQL Exceptions		
For a full list of all 18,000 PL/SQL Errors, visit the Oracle Error Code Library		
Exception Name	**Error**	**Description**
ACCESS_INTO_NULL	ORA-06530	Attempted to assign values to the attributes of an uninitialized (NULL) object.
CASE_NOT_FOUND	ORA-06592	None of the choices in the WHEN clauses of a CASE statement is selected and there is no ELSE clause.
COLLECTION_IS_NULL	ORA-06531	Attempt to apply collection methods other than EXISTS to an uninitialized (NULL) PL/SQL table or VARRAY.
CURSOR_ALREADY_OPEN	ORA-06511	Exactly what it seems to be. Tried to open a cursor that was already open
DUP_VAL_ON_INDEX	ORA-00001	An attempt to insert or update a record in violation of a primary key or unique constraint
INVALID_CURSOR	ORA-01001	The cursor is not open, or not valid in the context in which it is being called.
INVALID_NUMBER	ORA-01722	It isn't a number, even though you are treating it like one to trying to turn it into one.
LOGIN_DENIED	ORA-01017	Invalid name and/or password for the instance.
NO_DATA_FOUND	ORA-01403	The SELECT statement returned no rows or referenced a deleted element in a nested table or referenced an initialized element in an Index-By table.
NOT_LOGGED_ON	ORA-01012	Database connection lost.
PROGRAM_ERROR	ORA-06501	Internal PL/SQL error.
ROWTYPE_MISMATCH	ORA-06504	The rowtype does not match the values being fetched or assigned to it.
SELF_IS_fs	ORA-30625	Program attempted to call a MEMBER method, but the instance of the object type has not been intialized. The built-in parameter SELF points to the object, and is always the first parameter passed to a MEMBER method.
STORAGE_ERROR	ORA-06500	A hardware problem: Either RAM or disk drive.
SUBSCRIPT_BEYOND_COUNT	ORA-06533	Reference to a nested table or varray index higher than the number of elements in the collection.
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	Reference to a nested table or varray index outside the declared range (such as -1).
SYS_INVALID_ROWID	ORA-01410	The conversion of a character string into a universal rowid fails because the character string does not represent a valid rowid.
TIMEOUT_ON_RESOURCE	ORA-00051	The activity took too long and timed out.
TOO_MANY_ROWS	ORA-01422	The SQL INTO statement brought back more than one value or row (only one is allowed).
USERENV_COMMITSCN_ERROR	ORA-01725	Added for USERENV enhancement, bug 1622213.
VALUE_ERROR	ORA-06502	An arithmetic, conversion, truncation, or size-constraint error. Usually raised by trying to cram a 6 character string into a VARCHAR2(5) variable
ZERO_DIVIDE	ORA-01476	Not only would your math teacher not let you do it, computers won't either. Who said you didn't learn anything useful in primary school?

Dica:

```
/*  Simular forçando erros de excepitons
    utilizar as exceptions após a declaração RAISE
    NO_DATA_FOUND, TOO_MANY_ROWS, ZERO_DIVIDE, INVALID_CURSOR
*/

BEGIN
    NULL;
    RAISE INVALID_CURSOR ;
END;
```

Curso Intensivo de PL/SQL

6. Criação de Procedures

Criar tabela CRUD para simular a PROCEDURE

```
CREATE TABLE CRUD ( ID   NUMBER       NOT NULL,
                    NOME VARCHAR2(10) NOT NULL,
                    CONSTRAINT CRUD_PK PRIMARY KEY (ID));


CREATE OR REPLACE PROCEDURE PRC_CRUD
                  (   P_CRUD  IN      VARCHAR2,
                      P_ID    IN  OUT NUMBER,
                      P_NOME  IN  OUT VARCHAR2,
                      P_SAIDA     OUT VARCHAR2
                  )IS
BEGIN
    IF  P_CRUD = 'C' THEN --> CREATE: PARA REALIZAR INSERT
        INSERT INTO CRUD VALUES(P_ID, P_NOME);
        P_SAIDA:='INSERT REALIZADA COM SUCESSO!!!';
    ELSIF   P_CRUD = 'R' THEN --> READ: PARA REALIZAR SELECT
        SELECT  ID, NOME
        INTO    P_ID, P_NOME
        FROM    CRUD
        WHERE   ID = P_ID;
        P_SAIDA:= 'SELECT EXECUTADO COM SUCESSO!!!';
    ELSIF   P_CRUD = 'U' THEN --> UPDATE: PARA ATUALIZAR
        UPDATE  CRUD
        SET     NOME    =   P_NOME
        WHERE   ID      =   P_ID;
    ELSIF   P_CRUD = 'D' THEN --> DELETE: PARA EXCLUIR
        DELETE FROM CRUD WHERE ID = P_ID;
        P_SAIDA:= 'DELETE EXECUTADO COM SUCESSO!!!';
    ELSE
        P_SAIDA:= 'FAVOR DIGITAR UMA DAS OPÇÕES: C,R,U,D';
    END IF;
END;
```

Criar o bloco anônimo para executar a PROCEDURE

```
DECLARE
    /*  Utilizar um dos parametros abaixo para a variavel V_ACAO:
        C (CREATE) = INSERT, R (READ)= SELECT
        U (UPDATE) = UPDATE, D (DELETE) = DELETE    */
    V_ACAO  VARCHAR2(1):='C';
    V_ID    NUMBER:=1;
    V_NOME  VARCHAR2(10):='XXXX';
    V_SAIDA VARCHAR2(50);
BEGIN
    PRC_CRUD(V_ACAO,V_ID,V_NOME,V_SAIDA);
    DBMS_OUTPUT.PUT_LINE(V_SAIDA);
END;
```

7. Criação de Functions

Exemplo 01:

```
CREATE OR REPLACE FUNCTION PAR_IMPAR
                          (
                          P_VALOR IN NUMBER
                          ) RETURN VARCHAR2   IS
    V_SAIDA VARCHAR2(5);
BEGIN
    IF  P_VALOR MOD 2 = 0 THEN
        V_SAIDA :=  'PAR';
    ELSE
        V_SAIDA :=  'IMPAR';
    END IF;
    RETURN (V_SAIDA);
END;
```

Exemplo 02:

```
CREATE OR REPLACE FUNCTION fnc_ra
                   (
                   P_RA    NUMBER
                   )   RETURN VARCHAR  IS
    V_COUNT NUMBER:=0;
    V_NUM   NUMBER:=0;
    V_SOMA  NUMBER:=0;
    V_SAIDA VARCHAR2(5);
BEGIN
    SELECT  LENGTH(P_RA)
    INTO    V_COUNT
    FROM    DUAL;

    FOR X   IN  1..V_COUNT  LOOP
    Dbms_Output.Put_Line(X);
        SELECT  SUBSTR(P_RA,X,1)
        INTO    V_NUM
        FROM    DUAL;
        V_SOMA  :=  V_SOMA + V_NUM;
    END LOOP;
        V_SAIDA :=  FNC_PAR_IMP(V_SOMA);
        RETURN( 'Nº do RA: '||P_RA||CHR(10)||
                'Valor da soma: '||V_SOMA||CHR(10)||
                'Resultado: '||V_SAIDA);
END;
```

8. Criação de Triggers

```
--> Criar tabela vigia
CREATE TABLE vigia  (marca VARCHAR2(100));

--> Criando uma trigger de acesso: Logon
CREATE OR REPLACE TRIGGER marca_logon
    AFTER LOGON ON SCHEMA
    --> Mudar para ON SCHEMA se houver problemas de permissão
BEGIN
    INSERT INTO vigia
    VALUES (USER || ' entrou no sistema em ' ||
    TO_CHAR(sysdate, 'DD-MM-YYYY HH24:MI:SS'));
    COMMIT;
END;
/
--> Criando uma trigger de saída: Logoff
CREATE OR REPLACE TRIGGER marca_logoff
    BEFORE LOGOFF ON SCHEMA
BEGIN
    INSERT INTO vigia
    VALUES (USER || ' saiu do sistema em ' ||
    TO_CHAR(sysdate, 'DD-MM-YYYY HH24:MI:SS'));
    COMMIT;
END;
```

Curso Intensivo de PL/SQL

9. Criação de Packages

ESPEC

```
CREATE OR REPLACE PACKAGE FUNCIONARIO AS
    /* GET NOME COMPLETO DO FUNCIONARIO */
    FUNCTION GET_NOMECOMPLETO(N_FUNC_ID NUMBER) RETURN VARCHAR2;
    /* GET SALARIO DO FUNCIONARIO  */
    FUNCTION GET_SALARIO(N_FUNC_ID NUMBER) RETURN NUMBER;
END FUNCIONARIO;
```

BODY

```
/* PACKAGE FUNCIONARIO BODY */
CREATE OR REPLACE PACKAGE BODY FUNCIONARIO AS
    /*GET FUNCIONÁRIO NOMECOMPLETO */
    FUNCTION GET_NOMECOMPLETO(N_FUNC_ID NUMBER) RETURN VARCHAR2 IS
        V_NOMECOMPLETO VARCHAR2(46);
    BEGIN
        SELECT FIRST_NAME || ',' || LAST_NAME
        INTO V_NOMECOMPLETO
        FROM EMPLOYEE
        WHERE ID = N_FUNC_ID;
        RETURN V_NOMECOMPLETO;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RETURN NULL;
        WHEN TOO_MANY_ROWS THEN
            RETURN NULL;
    END;
/*  GET SALARIO */
    FUNCTION GET_SALARIO(N_FUNC_ID NUMBER) RETURN NUMBER IS
        N_SALARIO NUMBER(8,2);
    BEGIN
        SELECT SALARY
        INTO N_SALARIO
        FROM EMPLOYEE
        WHERE ID = N_FUNC_ID;
        RETURN N_SALARIO;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RETURN NULL;
        WHEN TOO_MANY_ROWS THEN
            RETURN NULL;
    END;
END FUNCIONARIO;
```

Bloco anônimo para executar a PACKAGE

```
DECLARE
    N_SALARIO NUMBER(8,2);
    V_NOME VARCHAR2(46);
    N_FUNC_ID NUMBER := &EMPRESA_ID;
BEGIN
    V_NOME := FUNCIONARIO.GET_NOMECOMPLETO(N_FUNC_ID);
    N_SALARIO := FUNCIONARIO.GET_SALARIO(N_FUNC_ID);
    IF V_NOME IS NOT NULL AND N_SALARIO IS NOT NULL THEN
        DBMS_OUTPUT.PUT_LINE('FUNCIONÁRIO: ' || V_NOME);
        DBMS_OUTPUT.PUT_LINE('RECEBE SALÁRIO = ' || N_SALARIO);
    END IF;
END;
```

10. SQL Dinâmico

```
--> Exemplo de SQL Dinâmico
CREATE OR REPLACE PROCEDURE sql_dinamico IS
    V_SQL   VARCHAR2(4000); --> STRING QUE RECEBE O COMANDO SQL DINÂMICO
    V_ID        NUMBER(2);
    V_NOME  VARCHAR2(30) := 'JEDI';
BEGIN
--PASSO 1: executa um comando DDL
    EXECUTE IMMEDIATE 'CREATE TABLE T_PRODUTO (ID NUMBER, NOME VARCHAR2(30))';

--PASSO 2: executa um comando DML passando duas variáveis como parâmetro
    FOR X IN 1..10 LOOP
        V_SQL := 'INSERT INTO T_PRODUTO VALUES (:1, :2)';
        EXECUTE IMMEDIATE V_SQL USING V_ID||X, V_NOME||X;
    END LOOP;
--PASSO 3: executa um comando DCL
    EXECUTE IMMEDIATE 'GRANT SELECT ON T_PRODUTO TO SYSTEM';
END;
```

11. BULK COLLECT

```
--> criar table exer1
create table exer1(cod_number number(5),
descricao varchar2(40),
constraint exer1_pk primary key (cod_number));


--> inserir table exer1 através de dbms_random.string
begin
insert into exer1
select level, dbms_random.string('A',40) from dual connect by level <=100;
commit;
end;

--> select para verificar os registros gerados
select * from exer1

declare
cursor cur_exer is
    select cod_number, descricao from exer1;
    --> criando o tipo para coleção
    type trow_exer1 is table of cur_exer%rowtype index by pls_integer;
    --> declarando a coleção row_tab19x
    row_exer1 trow_exer1;
    begin
        open cur_exer;
            loop
                fetch cur_exer bulk collect into row_exer1 limit 70;
                exit when row_exer1.count = 0;
                for x in 1..row_exer1.count loop
                    Dbms_Output.Put_Line(   'cod: '||row_exer1(x).cod_number||
                                            ' descricao: '||row_exer1(x).descricao);
                end loop;
            end loop;
        close cur_exer;
end;
```

12. **Transações Autónomas**.

```
Exemplo 1
--> 1ª Parte - Sem PRAGMA
--> Criar Tabela
CREATE TABLE t (test_value VARCHAR2(25));
--> Criar Procedure Child_Block
CREATE OR REPLACE PROCEDURE child_block IS
BEGIN
INSERT INTO t
(test_value)
VALUES
('Child block insert');
COMMIT;
END child_block;
/
--> Criar Procedure Parent_Block
CREATE OR REPLACE PROCEDURE parent_block IS

BEGIN
INSERT INTO t
(test_value)
VALUES
('Parent block insert');

child_block;

ROLLBACK;
END parent_block;
/

-- run the parent procedure
exec parent_block

-- check the results
SELECT * FROM t;


--> 2ª Parte - Com PRAGMA
CREATE OR REPLACE PROCEDURE child_block IS

PRAGMA AUTONOMOUS_TRANSACTION;

BEGIN
INSERT INTO t
(test_value)
VALUES
('Child block insert');

COMMIT;
END child_block;
/

-- empty the test table
TRUNCATE TABLE t;

-- run the parent procedure
exec parent_block;

-- check the results
SELECT * FROM t;
```

```
Exemplo 2
--> 1ª Parte - Sem PRAGMA / irá retornar 0,3,6
--> Retorna: a=0, b=3 (valores inseridos mesmo sem commit), c=6 (valor final)

CREATE TABLE t (testcol NUMBER);
truncate table t
select * from t


CREATE OR REPLACE FUNCTION howmanyrows RETURN INTEGER IS
i INTEGER;
BEGIN
SELECT COUNT(*)
INTO i
FROM t;

RETURN i;
END howmanyrows;
/
CREATE OR REPLACE PROCEDURE testproc IS
a INTEGER;
b INTEGER;
c INTEGER;
BEGIN
SELECT COUNT(*)
INTO a
FROM t;

INSERT INTO t VALUES (1);
COMMIT;

INSERT INTO t VALUES (2);
INSERT INTO t VALUES (3);

b := howmanyrows;

INSERT INTO t VALUES (4);
INSERT INTO t VALUES (5);
INSERT INTO t VALUES (6);
COMMIT;

SELECT COUNT(*)
INTO c
FROM t;

dbms_output.put_line(a);
dbms_output.put_line(b);
dbms_output.put_line(c);
END testproc;
/
--set serveroutput on --> Configura o servidor de impressão, caso rode no SQLPlus

-- run the testproc procedure
exec testproc
-- check the results
SELECT * FROM t;
```

```
--> 2ª Parte - Com PRAGMA / irá retornar 0,1,6
--> Retorna: a=0, b=1 (único commit), c=6 (valor final)
CREATE OR REPLACE FUNCTION howmanyrows RETURN INTEGER IS
i INTEGER;

PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
SELECT COUNT(*)
INTO i
FROM t;

RETURN i;
END howmanyrows;
/

-- run the testproc procedure
exec testproc
-- check the results
SELECT * FROM t;
```

Curso Intensivo de PL/SQL

Referências:

Database PL/SQL Language Reference:
http://docs.oracle.com/database/121/LNPLS

Database PL/SQL User's Guide and Reference:
http://docs.oracle.com/cd/B19306_01/appdev.102/b14261/toc.htm

Database PL/SQL Packages and Types Reference:
http://docs.oracle.com/cd/B19306_01/appdev.102/b14258/toc.htm

Oracle Database Documentation Library (Biblioteca com informações e referências dos produtos Oracle):
http://www.oracle.com/pls/db102/homepage

Oracle Functions and Oracle Packages Libraries:
http://psoug.org/reference/library.html

Exemplos Diversos
http://www.proftoninho.com

Curso Intensivo de PL/SQL

EXERCÍCIOS – Parte 1

1) Criar um bloco pl-sql que imprima seu nome e sua idade.

2) Criar um bloco pl-sql para calcular o valor do novo salário mínimo que deverá ser de 25% em cima do atual, que e de R$ 724.00, no qual este valor deverá ser constante dentro do bloco, informando em seguida.

3) Criar um bloco pl-sql para calcular o valor em reais de 45 dólares, sendo que o valor do cambio a ser considerado e de R$ 2,98 reais, no qual estes valores deverão ser constantes dentro do bloco.

4) Criar um bloco pl-sql para calcular o valor do novo salário mínimo que deverá ser 25% em cima do atual, no qual o salário deverá ser informado em tempo de execução.

5) Criar um bloco pl-sql para converter em reais os dólares informados, sendo que o valor do cambio a ser considerado e de R$ 2,98 reais, no qual os valores dos dólares deverão ser informados em tempo de execução.

6) Leia idade (informado pelo usuário em tempo de execução). Se for maior ou igual a 18 exiba "Maior de idade", senão "Menor de idade".

7) Leia três notas (informado pelo usuário em tempo de execução). Calcule a média. Se media for maior ou igual a seis, exiba "Aprovado", senão "reprovado".

8) Leia salario (informado pelo usuário em tempo de execução). Se salario for maior que R$ 1.500,00 efetue um reajuste de 8% ¨, senão o reajuste será de 10%. Exiba: valor de reajuste e salario reajustado.

9) Leia dias letivos e faltas (informado pelo usuário em tempo de execução). Calcule o limite de faltas (25% dos dias letivos). Se as faltas forem maior do que o limite exiba "Reprovado por falta", senão "Ok".

10) Criar um bloco pl-sql para calcular o valor das parcelas de uma compra de um carro, nas seguintes condições:
Observação:

a) Parcelas para aquisição em 10 pagamentos.
b) O valor da compra deverá ser informado em tempo de execução.
c) O valor total dos juros e de 3% e deverá ser aplicado sobre o montante financiado
d) No final informar o valor de cada parcela.

Curso Intensivo de PL/SQL

EXERCÍCIOS – Parte 2

1) A empresa XPTO Veículos, deseja criar um sistema que efetue cálculos de financiamento

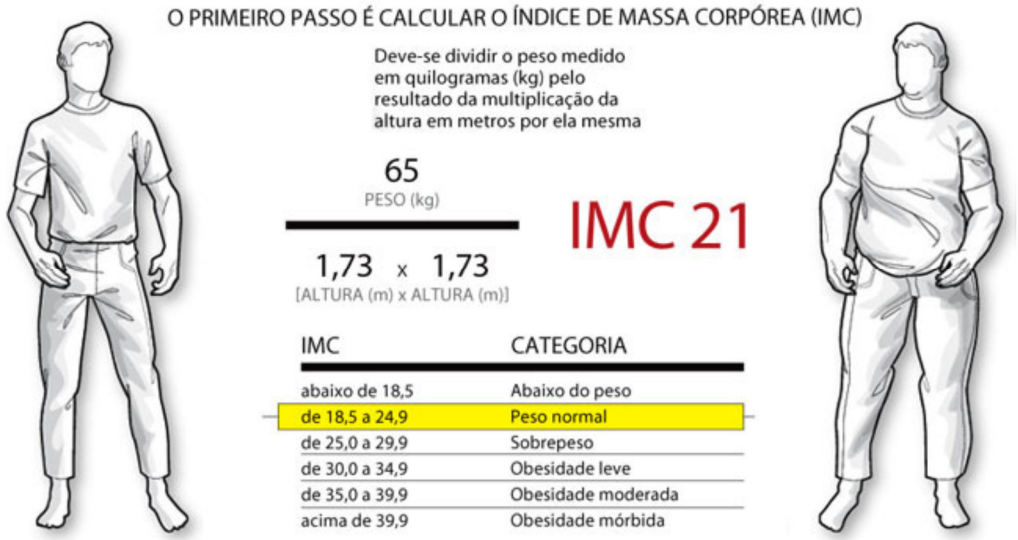
Por isso contratou seus serviços para desenvolver um programa simples que automatize os cálculos das parcelas de venda de carros.
O programa deverá atender as seguintes condições de parcelamento:

a) Parcelamento em 06 pagamentos
b) Parcelamento em 12 pagamentos
c) Parcelamento em 18 pagamentos

Observação:

1. O valor da compra deverá ser informado em tempo de execução
2. O nº de parcelas que se deseja pagar deverá ser informado em tempo de execução
3. Deverá ser dada uma entrada de 20% do valor da compra
4. Deverá ser aplicada uma taxa de juros sobre o saldo restante, nas seguintes condições:
   - ✓ Pagamento em 06 parcelas: 10%
   - ✓ Pagamento em 12 parcelas: 15%
   - ✓ Pagamento em 18 parcelas: 20%
5. No final, informar o valor da compra, o valor da entrada, o nº de parcelas escolhido para pagamento, bem como seu valor.

2) O IMC – Índice de Massa Corporal é um critério da Organização Mundial de Saúde para dar uma indicação sobre a condição de peso de uma pessoa adulta.



Elabore um bloco PL/SQL que leia o peso e a altura de um adulto e identifique em que condição ele se encontra, de acordo com a tabela acima.
No final, apresente seu peso, sua altura e sua condição.

3) Montar um programa PLSQL que retorne uma das 03 tipos de triângulos abaixo, após o usuário entrar com 03 valores em tempo de execução.

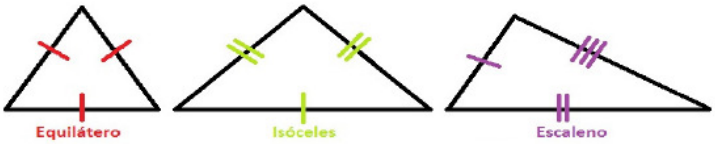


Fig. Tipos de Triângulos

As formas a serem retornadas são:

- ✓ Triângulo Equilátero: Todos os lados iguais
- ✓ Triângulo Isósceles: Dois lados iguais e um diferente
- ✓ Triângulo Escaleno: todos os lados diferentes
- ✓ Forma indefinida, caso algum dos valores informados seja zero.

Curso Intensivo de PL/SQL

EXERCÍCIOS – Parte 3

Dica:
Leia atentamente os exercícios abaixo antes de executá-los.
No exercício 03 pede-se para desenvolver toda a solução numa única procedure, porém caso tenha muita dificuldade, execute-a em três partes (PRC_PROD_INS, PRC_PROD_DEL, PRC_SEL).

1) Crie a Procedure PRC_NOME_IDADE para imprimir o Nome e a IDADE que está executando esta atividade. (Nível: Super Fácil)

2) Crie a Procedure PRC_TEMPO para somar os campos dia, mês e ano com um valor qualquer passado como parâmetro no campo data da tabela tempo. (Nível: Fácil)

```
CREATE TABLE TEMPO (
ID INT PRIMARY KEY,
DIA INT,
MES INT,
ANO INT,
DATA INT
);
INSERT INTO TEMPO (ID, DIA, MES, ANO) VALUES (1, 15, 1, 1999);
INSERT INTO TEMPO (ID, DIA, MES, ANO) VALUES (2, 15, 2, 1999);
INSERT INTO TEMPO (ID, DIA, MES, ANO) VALUES (3, 15, 3, 1999);
INSERT INTO TEMPO (ID, DIA, MES, ANO) VALUES (4, 15, 4, 1999);
INSERT INTO TEMPO (ID, DIA, MES, ANO) VALUES (5, 15, 5, 1999);
INSERT INTO TEMPO (ID, DIA, MES, ANO) VALUES (6, 15, 6, 1999);
INSERT INTO TEMPO (ID, DIA, MES, ANO) VALUES (7, 15, 7, 1999);
INSERT INTO TEMPO (ID, DIA, MES, ANO) VALUES (8, 15, 8, 1999);
INSERT INTO TEMPO (ID, DIA, MES, ANO) VALUES (9, 15, 9, 1999);
INSERT INTO TEMPO (ID, DIA, MES, ANO) VALUES (10, 15, 10, 1999);
```

Exemplo: Caso seja passado o valor 10 como parâmetro de entrada

Resultado Esperado:

```
EXECUTE PRC_TEMPO(10);
```

ID	DIA	MES	ANO	DATA
1	15	1	1999	2025
2	15	2	1999	2026
3	15	3	1999	2027
4	15	4	1999	2028
5	15	5	1999	2029
6	15	6	1999	2030
7	15	7	1999	2031
8	15	8	1999	2032
9	15	9	1999	2033
10	15	10	1999	2034

3) Rode o script abaixo para criar a tabela produto e em seguida desenvolva 01 única procedure PRC_PROD que: (Nível: Médio)
   a) Inclua novos dados na tabela produto;
   Obs.: Para inclusão utilize a exceção (exception) com a mensagem definido abaixo
   WHEN DUP_VAL_ON_INDEX THEN
   DBMS_OUTPUT.PUT_LINE ('Código de produto já cadastrado');

   b) Atualize (aumentando) o valor do produto;
   Obs.: Informando o código e o percentual a ser aplicado no aumento do produto.

   c) Consulte um produto ao informar um código;

```
CREATE TABLE PRODUTO (
CODIGO NUMBER(4) PRIMARY KEY,
NOME VARCHAR2(20),
VALOR NUMBER(7,2),
CATEGORIA NUMBER(4)
);
INSERT INTO PRODUTO VALUES (1, 'PRODUTO1', 2.5, 10);
INSERT INTO PRODUTO VALUES (2, 'PRODUTO2', 3.2,20);
INSERT INTO PRODUTO VALUES (3, 'PRODUTO3', 5.8, 30);
```