PROGRAMAÇÃO

PARA

BANCO DE DADOS

PL/ SQL

Prof. Marcos Alexandruk

SUMÁRIO

1. CONCEITOS DE PROGRAMAÇÃO PARA BANCO DE DADOS: INTRODUÇÃO AO PL/SQL

PL/SQL:

- Linguagem de programação: Procedural Language/Stuctured Query Language

Principais recursos oferecidos pela linguagem:

- Executar comandos SQL para manipular dados nas tabelas
- Criar constantes e variáveis
- Criar cursores para tratar o resultado de uma consulta
- Criar registros para guarda o resultado de um cursor ou campo de tabela
- Tratar erros
- Utilizar comandos de controle (if, if-then-else, case) e repetição (loop, for, while)

Vantagens:

- Versatilidade
- Portabilidade
- Integração com o SGBD (Sistema Gerenciador de Banco de Dados)
- Capacidade procedural (comandos de controle e repetição)
- Redução de tráfego de rede

Rotinas PL/SQL podem ser desenvolvidas com diversas ferramentas:

- SQL* Plus
- SQL* Plus Worksheet
- Oracle Forms
- etc.

Estrutura de um bloco PL/SQL:

```
DECLARE
    Inicializações, declaração de constantes, variáveis e cursores
BEGIN
    Comandos SQL, estruturas de programação e outros blocos PL/SQL
        BEGIN
        ...
        END;
EXCEPTION (opcional)
    Tratamento de exceções, emissão de mensagens
END;
```

IMPORTANTE:

Para visualizar a saída no SQL* Plus faça a seguinte alteração:

```
SET SERVEROUTPUT ON
```

2. DECLARAÇÕES

Na área **DECLARE** podemos declarar:

- constantes
- variáveis
- cursores
- estruturas
- tabelas

3. TIPOS DE DADOS

3.1. Os tipos de dados simples que podem ser utilizados em declarações PL/SQL são:

TIPOS	DESCRIÇÃO
CHAR	Alfanumérico, tamanho fixo, limite: 2000 caracteres
CHARACTER	Idêntico ao CHAR, mantém compatibilidade com outras versões SQL
VARCHAR2	Alfanumérico, tamanho variável, limite: 4000 caracteres
VARCHAR E STRING	Idêntico ao VARCHAR2, mantém compatibilidade com outras versões SQL
CLOB (Character Long Object)	Alfanumérico, tamanho variável, limite 4 Gb
LONG	Alfanumérico, limites: 2 GB, apenas um por tabela
ROWID	Armazena os valores dos ROWIDs das linhas das tabelas
BLOB (Binary Long Object)	Binário, tamanho variável, limite: 4 Gb
BFILE (Binary File)	Armazena uma referência a um arquivo externo (que deverá localizar-se na mesma máquina do banco de dados, não permite referência remota)
RAW	Hexadecimais, tamanho variável, limite: 2 Kb
LONG ROW	Hexadecimais, tamanho variável, limite: 2 Gb
NUMBER	Numérico, limite: 38 dígitos Exemplo: NUMBER (10,2) armazena 10 números (8 inteiros e 2 decimais) SUBTIPOS: • DECIMAL • DEC • DOUBLEPRECISION • INTEGER • INT • NUMERIC • REAL • SMALLINT • FLOAT • PLS_INTEGER
BINARY_INTEGER	Numérico, positivos e negativos, limites: -2147483647 e 2147483647 SUBTIPOS: • NATURAL (limites: 0 e 2147483647) • NATURALN (limites: 0 e 2147483647, não aceita valores nulos) • POSITIVE (limites: 1 e 2147483647) • POSITIVEN (limites: 0 e 2147483647, não aceita valores nulos)
DATE	Data e hora (formato padrão: DD-MMM-YY
TIMESTAMP	Data e hora (com milésimos de segundo)
BOOLEAN	Armazena os valores: TRUE, FALSE ou NULL

3.2. Tipos compostos:

- RECORD
- TABLE

3.3. Tipos referenciados:

- REF CURSOR

4. CONSTANTES E VARIÁVEIS

4.1. CONSTANTES

Para declarações de constantes, a palavra CONSTANT deve aparecer antes do tipo de dado e a seguir utiliza-se o operador **:=** para atribuir-lhe um valor.

Exemplo:

```
DECLARE
   pi     CONSTANT NUMBER(9,7) := 3.1415927;
   ...
BEGIN
   ...
END;
/
```

4.2. VARIÁVEIS

As variáveis são inicializadas de maneira similar às constantes: declarando-se o tipo e atribuindo-lhe um valor. Variáveis não inicializadas explicitamente recebem o valor NULL. Não é necessário inicializar uma variável com valor NULL. Pode-se aplicar a restrição NOT NULL a uma variável. Neste caso ela deverá ser inicializada.

Exemplos:

```
v1    NUMBER(4) := 1;
v2    NUMBER(4) := DEFAULT 1;
v3    NUMBER(4) NOT NULL := 1;
```

Podemos atribuir valores às variáveis de duas maneiras:

- Utilizando o operador de atribuição:

```
total := quant * valor;
```

- Utilizando um comando SELECT com a cláusula INTO:

```
SELECT ra_aluno, nome_aluno
INTO   ra, aluno
FROM   aluno;
```

4.2.1. HERANÇA DE TIPO E TAMANHO

As constantes e variáveis podem herdar o tipo de outras variáveis, de colunas ou até da linha inteira de uma tabela. Desta forma, diminuem-se as manutenções oriundas nas alterações realizadas nas colunas de tabelas (ex: tamanho da coluna).

- Herdando o tipo de uma variável previamente declarada:

```
nome_da_variavel_2    nome_da_variavel_1%Type;
```

- Herdando o tipo de uma coluna de uma tabela:

```
nome_da_variavel    nome_da_tabela.nome_da_coluna%Type;
```

- Herdando o tipo de uma linha inteira de uma tabela:

```
nome_da_variavel    nome_da_tabela%Rowtype;
```

4.2.2. ESCOPO DE VARIÁVEIS

Variáveis definidas dentro de um bloco serão locais para esse bloco e globais para os sub-blocos. Não serão reconhecidas em outros blocos isolados.

5. COMANDOS SQL DENTRO DE UM BLOCO PL/SQL

Comandos DML (SELECT, INSERT, UPDATE e DELETE) podem ser utilizados dentro de um bloco PL/SQL.
O comando SELECT deverá receber obrigatoriamente a cláusula INTO para que o resultado seja armazenado em variáveis e deverá retornar apenas uma linha. Caso mais de uma linha seja retornada, apresentará o erro: **too_many_rows** e se não retornar nenhuma linha, apresentará o erro: **no_data_found**. (veja: TRATAMENTO DE EXCEÇÕES)

Exemplo:

```
CREATE TABLE ALUNO (
RA NUMBER(5),
NOME VARCHAR2(40));

INSERT INTO ALUNO VALUES (1,'ANTONIO');
INSERT INTO ALUNO VALUES (2,'BEATRIZ');

DECLARE
    V_RA ALUNO.RA%TYPE;
    V_NOME ALUNO.NOME%TYPE;
BEGIN
    SELECT RA, NOME
    INTO V_RA, V_NOME
    FROM ALUNO
    WHERE RA=1;
    DBMS_OUTPUT.PUT_LINE(V_RA ||' - '|| V_NOME);
END;
/

1 - ANTONIO
```

Nota: Utilizamos **||** para concatenação.

Exercícios:

1. Alterar a cláusula WHERE acima para: **WHERE RA=1 OR RA=2** e verificar o resultado.

O comando SELECT retorna mais de uma linha, por isso é exibida a mensagem:

```
ORA-01422: a extração exata retorna mais do que o número solicitado de linhas
```

2. Alterar a cláusula WHERE acima para: **WHERE RA=3** e verificar o resultado.

O comando SELECT não retorna nenhuma linha, por isso é exibida a mensagem:

```
ORA-01403: dados não encontrados
```

3. Alterar a cláusula WHERE acima para: WHERE RA= &RA_DO_ALUNO.

```
Entre o valor para ra_do_aluno:
```

4. Inserir os seguintes valores na tabela aluno: ra= 5, nome= 'DANIEL'.

```
BEGIN
    INSERT INTO ALUNO (RA,NOME)
    VALUES (5,'DANIEL');
END;
/
```

5. Alterar o nome do aluno cujo RA= 5 para 'ERNESTO'.

```
BEGIN
    UPDATE ALUNO
    SET NOME=' ERNESTO'
    WHERE RA=5;
END;
/
```

6. Excluir da tabela o aluno cujo RA= 5.

```
BEGIN
    DELETE FROM ALUNO
    WHERE RA=5;
END;
/
```

6. INSTRUÇÕES IF-THEN-ELSE E CASE

6.1. IF-THEN-ELSE

Executa um conjunto de ações de acordo com uma ou mais condições.

```
IF condição_1
    THEN relação_de_comandos_1
[ELSIF condição_2
    THEN relação_de_comandos_2]
[ELSE    relação_de_comandos_3]
END IF;
```

Exemplo 1:

```
DECLARE
    V_1   NUMBER(2) := 4;
    V_2   VARCHAR2(5);
BEGIN
    IF MOD(V_1,2) = 0
        THEN V_2 := 'PAR';
    ELSE V_2 := 'IMPAR';
    END IF;
    DBMS_OUTPUT.PUT_LINE ('O número é: ' || V_2);
END;
/
```

NOTA: MOD(V1,2) divide o valor de V_1 por 2 e retorna o resto da divisão.

Exemplo 2:

```
CREATE TABLE ALUNO (
RA NUMBER(9),
NOTA NUMBER(3,1));

INSERT INTO ALUNO VALUES (1,4);

DECLARE
    V_RA    ALUNO.RA%TYPE := 1;
    V_NOTA ALUNO.NOTA%TYPE;
    V_CONCEITO VARCHAR2(12);
BEGIN
    SELECT NOTA
    INTO V_NOTA
    FROM ALUNO
    WHERE RA = V_RA;
    IF V_NOTA <= 5
        THEN V_CONCEITO := 'REGULAR';
    ELSIF V_NOTA < 7
        THEN V_CONCEITO := 'BOM';
    ELSE V_CONCEITO := 'EXCELENTE';
    END IF;
    DBMS_OUTPUT.PUT_LINE ('Conceito: ' || V_CONCEITO);
END;
/
```

6.2. CASE

Retorna determinado resultado de acordo com o valor da variável de comparação.

```
[variável :=]
    CASE
        WHEN expressão_1 THEN declaração_1
        WHEN expressão_2 THEN declaração_2
        ...
        ELSE declaração_n
    END;
```

Exemplo:

```
DECLARE
    V_RA    ALUNO.RA%TYPE := 1;
    V_NOTA ALUNO.NOTA%TYPE;
    V_CONCEITO VARCHAR2(12);
BEGIN
    SELECT NOTA
    INTO V_NOTA
    FROM ALUNO
    WHERE RA = V_RA;
    V_CONCEITO :=
        CASE
            WHEN V_NOTA <= 5 THEN 'REGULAR'
            WHEN V_NOTA < 7 THEN 'BOM'
            ELSE 'EXCELENTE'
        END;
    DBMS_OUTPUT.PUT_LINE ('Conceito: ' || V_CONCEITO);
END;
/
```

EXERCÍCIOS 01:

1. Criar uma tabela conforme segue:

```
CREATE TABLE ALUNO (
RA NUMBER(9),
DISCIPLINA VARCHAR2(30),
MEDIA NUMBER(3,1),
CARGA_HORA NUMBER(2),
FALTAS NUMBER(2),
RESULTADO VARCHAR2(10));
```

Inserir uma linha deixando a coluna RESULTADO em branco.

```
INSERT INTO ALUNO VALUES (1,'DISC 1',7.5,80,20,'');
```

Criar um bloco PL/SQL para preencher a coluna resultado conforme o seguinte:

Se o aluno obteve média igual ou maior que 7.0 e suas faltas não ultrapassarem 25% da carga horária da disciplina o resultado será: APROVADO.

Se o aluno obteve média inferior a 7.0 e suas faltas não ultrapassarem 25% da carga horária da disciplina o resultado será: EXAME.

Para demais casos o resultado será: REPROVADO.

2. Criar uma tabela, conforme segue:

```
CREATE TABLE PRODUTO (
CODIGO NUMBER(2),
DESCRICAO VARCHAR2(20));
```

Inserir sete produtos diferentes na tabela acima.

Criar um bloco PL/SQL para apresentar um produto diferente para cada dia da semana.

Nota: A mensagem acima deverá ser exibida dinamicamente, conforme a data do sistema (SYSDATE).

Apresentar a seguinte mensagem:

Hoje é TERÇA-FEIRA e o produto em oferta é PRODUTO 3.

7. INSTRUÇÕES LOOP, FOR E WHILE

FOR

Repete um bloco de comando n vezes, ou seja, até que a variável contadora atinja o seu valor final.
A variável contadora não deve ser declarada na seção DECLARE e deixará de existir após a execução do comando END LOOP.

```
FOR v_contador IN valor_inicial..valor_final
LOOP
    bloco_de_comandos
END LOOP;
```

Exemplo 1:

```
DECLARE
    V_AUX  NUMBER(2) := 0;
BEGIN
    FOR V_CONTADOR IN 1..10
    LOOP
        V_AUX := V_AUX +1;
    DBMS_OUTPUT.PUT_LINE (V_AUX);
    END LOOP;
END;
/
```

Exemplo 2:

```
DECLARE
    V_RA_INICIAL    ALUNO.RA%TYPE := 1;
    V_RA_FINAL      V_RA_INICIAL%TYPE;
    V_AUX           V_RA_INICIAL%TYPE := 0;
BEGIN
    SELECT COUNT(RA)
    INTO V_RA_FINAL
    FROM ALUNO;
    FOR V_CONTADOR IN V_RA_INICIAL..V_RA_FINAL
    LOOP
        V_AUX := V_AUX +1;
    DBMS_OUTPUT.PUT_LINE ('Total de alunos: ' || V_AUX);
    END LOOP;
END;
/
```

WHILE

Repete um bloco de comandos enquanto a condição que segue o comando WHILE for verdadeira.

Exemplo 1:

```
DECLARE
    V_AUX  NUMBER(2) := 0;
BEGIN
    WHILE V_AUX < 10
    LOOP
        V_AUX := V_AUX +1;
    DBMS_OUTPUT.PUT_LINE (V_AUX);
    END LOOP;
END;
/
```

Exemplo 2:

```
DECLARE
    V_RA_FINAL      ALUNO.RA%TYPE := 1;
    V_AUX           V_RA_FINAL%TYPE := 0;
BEGIN
    SELECT COUNT(RA)
    INTO V_RA_FINAL
    FROM ALUNO;
    WHILE V_AUX < V_RA_FINAL
    LOOP
        V_AUX := V_AUX +1;
    DBMS_OUTPUT.PUT_LINE ('Total de alunos: ' || V_AUX);
    END LOOP;
END;
/
```

EXIT

Interrompe a execução de um comando de repetição.

Exemplo 1:

```
DECLARE
    V_AUX  NUMBER(2) := 0;
BEGIN
    FOR V_CONTADOR IN 1..15
    LOOP
        V_AUX := V_AUX +1;
    DBMS_OUTPUT.PUT_LINE (V_AUX);
    EXIT WHEN V_CONTADOR = 10;
    END LOOP;
END;
/
```

Exemplo 2:

```
DECLARE
    V_AUX  NUMBER(2) := 0;
BEGIN
    FOR V_CONTADOR IN 1..15
    LOOP
        V_AUX := V_AUX +1;
    DBMS_OUTPUT.PUT_LINE (V_AUX);
    IF V_CONTADOR = 10
        THEN EXIT;
    END IF;
    END LOOP;
END;
/
```

LOOP

Executa uma relação de comandos até que uma instrução de saída (EXIT) seja encontrada.

```
LOOP
  relação_de_comandos
  IF condição_de_saída
     THEN EXIT;
END LOOP;
```

Exemplo:

```
DECLARE
    V_AUX  NUMBER(2) := 0;
BEGIN
    LOOP
        V_AUX := V_AUX +1;
    DBMS_OUTPUT.PUT_LINE (V_AUX);
    IF V_AUX = 10
        THEN EXIT;
    END IF;
    END LOOP;
END;
/
```

EXERCÍCIO

Crie uma tabela chamada CIRCULO com as seguintes colunas:

RAIO NUMBER(2),
AREA NUMBER(8,2)

```
CREATE TABLE CIRCULO (
RAIO NUMBER(2),
AREA NUMBER(8,2));
```

Desenvolva um programa em PL/SQL para inserir os raios com valores 1 a 10 e as respectivas áreas na tabela acima.

SOLUÇÃO 1: WHILE

```
DECLARE
    PI      CONSTANT NUMBER(9,7) := 3.1415927;
    RAIO    NUMBER(2);
    AREA    NUMBER(8,2);
BEGIN
    RAIO := 1;
    WHILE RAIO <=10
    LOOP
        AREA := PI*POWER(RAIO,2);
        INSERT INTO CIRCULO VALUES (RAIO,AREA);
        RAIO := RAIO+1;
    END LOOP;
END;
/
```

SOLUÇÃO 2: FOR

```
DECLARE
    PI      CONSTANT NUMBER(9,7) := 3.1415927;
    RAIO    NUMBER(2) := 1;
    AREA    NUMBER(8,2);
BEGIN
FOR CONTADOR IN 1..10
LOOP
    AREA := PI*POWER(RAIO,2);
    INSERT INTO CIRCULO VALUES (RAIO,AREA);
    RAIO := RAIO +1;
END LOOP;
END;
/
```

LABELS

Utilizados para nomear blocos ou sub-blocos.
Devem localizar-se antes do início do bloco e preceder pelo menos um comando.
Se não houver necessidade de nenhum comando após o label, deve-se utilizar o comando NULL.

```
<<NOME_DO_LABEL>>
DECLARE
...
BEGIN
    RELAÇÃO_DE_COMANDOS
    <<NOME_DO_LABEL>>
    RELAÇÃO DE COMANDOS
END;
```

Um label também pode ser aplicado a um comando de repetição LOOP.

```
<<PRINCIPAL>>
LOOP
    ...
    LOOP
        ...
        -- SAIR DOS DOIS LOOPS
        EXIT PRINCIPAL WHEN ...
    END LOOP;
END LOOP PRINCIPAL;
```

GOTO

Utilizado para desviar um fluxo de um bloco PL/SQL para determinado label.

Não pode ser utilizado para:

- Desviar o fluxo para dentro de um IF;
- Desviar o fluxo de um IF para outro;
- Desviar o fluxo para dentro de um sub-bloco;
- Desviar o fluxo para um bloco externo ao bloco corrente;
- Desviar o fluxo de uma EXCEPTION para o bloco corrente e vice-versa.

GOTO nome_do_label

Exemplo 1:

```
<<PRINCIPAL>>
DECLARE
    V_NOME ALUNO.NOME%TYPE;
BEGIN
    SELECT COUNT(RA)
    INTO V_CONTA
    FROM ALUNO;
    IF V_CONTA = 10
        GOTO FIM;
    ELSE INSERT INTO ALUNO VALUES (20,'SILVA');
    END IF;
<<FIM>>
    DBMS_OUTPUT.PUT_LINE('Fim do programa');
END;
/
```

Exemplo 2:

```
<<PRINCIPAL>>
DECLARE
    V_NOME ALUNO.NOME%TYPE;
BEGIN
    SELECT NOME
    INTO V_NOME
    FROM ALUNO
    WHERE NOME LIKE '&NOME_ALUNO';
    FOR V_CONTADOR IN 1..5
    LOOP
    <<SECUNDARIO>>
    DECLARE
        V_NOME VARCHAR2(40);
    BEGIN
        SELECT NOME
        INTO V_NOME
        FROM ALUNO
        WHERE RA=V_CONTADOR;
        IF V_NOME = PRINCIPAL.V_NOME
            THEN DBMS_OUTPUT.PUT_LINE('Está entre os 5 primeiros');
        GOTO FIM;
        END IF;
    END;
    END LOOP;
<<FIM>>
    DBMS_OUTPUT.PUT_LINE('Fim do programa');
END;
/
```

8. TRATAMENTO DE EXCEÇÕES

Exceções são erros ou imprevistos que podem ocorrer durante a execução de um bloco PL/SQL.
Nesses casos, o gerenciador de banco de dados aborta a execução e procura uma área de exceções.
As exceções podem ser:

- Predefinidas
- Definidas pelo usuário

PREDEFINIDAS

Disparadas automaticamente quando, no bloco PL/SQL, uma regra Oracle for violada.
Podem ser identificadas por um nome e um número.

```
EXCEPTION
    WHEN nome_da_exceção THEN
        relação_de_comandos;
    WHEN nome_da_exceção THEN
        relação_de_comandos;
```

ERRO	NOME	DESCRIÇÃO
ORA-00001	DUP_VAL_ON_INDEX	Tentativa de armazenar valor duplicado em uma coluna que possui chave primária ou única.
ORA-01012	NOT_LOGGED_ON	Tentativa acessar o banco de dados sem estar conectado a ele.
ORA-01403	NO_DATA_FOUND	Ocorre quando um comando SELECT ... INTO não retorna nenhuma linha.
ORA-01422	TOO_MANY_ROWS	Ocorre quando um comando SELECT ... INTO retorna mais de uma linha.
ORA-01476	ZERO_DIVIDE	Tentativa de dividir qualquer número por zero.

Exemplo:

```
DECLARE
    V_RA ALUNO.RA%TYPE;
    V_NOME ALUNO.NOME%TYPE;
BEGIN
    SELECT RA, NOME
    INTO V_RA, V_NOME
    FROM ALUNO
    WHERE RA=30;
    DBMS_OUTPUT.PUT_LINE(V_RA ||' - '|| V_NOME);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE ('Não há nenhum aluno com este RA');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE ('Há mais de um aluno com este(s) RA(s)');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE ('Erro desconhecido');
END;
```

EXERCÍCIO

Elabore um programa em PL/SQL que faça o seguinte tratamento de exceção:

- Informe tentativa de inserir valor duplicado numa coluna que é chave primária.

```
DECLARE
BEGIN
    INSERT INTO ALUNO VALUES (1,'ANTONIO');
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        DBMS_OUTPUT.PUT_LINE ('Já existe um aluno com este RA');
END;
/
```

DEFINIDAS PELO USUÁRIO

Além dos erros tratados automaticamente pelo Oracle, regras de negócio específicas podem ser tratadas.
As exceções definidas pelo usuário devem ser declaradas e chamadas explicitamente pelo comando RAISE.

```
DECLARE
    nome_da_exceção EXCEPTION;
BEGIN
    ...
    IF ... THEN
        RAISE nome_da_exceção;
    END IF;
    ...
EXCEPTION
    WHEN nome_da_exceção THEN
        relação_de_comandos
END;
/
```

Exemplo 1:

```
DECLARE
    V_RA            ALUNO.RA%TYPE;
    V_NOTA          ALUNO.NOTA%TYPE;
    V_CONTA         NUMBER(2);
    CONTA_ALUNO     EXCEPTION;
BEGIN
    SELECT COUNT(RA)
    INTO V_CONTA
    FROM ALUNO;
    IF V_CONTA = 10 THEN
        RAISE CONTA_ALUNO;
    ELSE INSERT INTO ALUNO VALUES (20,'SILVA');
    END IF;
EXCEPTION
    WHEN CONTA_ALUNO THEN
        DBMS_OUTPUT.PUT_LINE('Não foi possível incluir: turma cheia');
END;
/
```

Exemplo 2:

```
DECLARE
    V_RA            ALUNO.RA%TYPE := &RA;
    V_NOME          ALUNO.NOMEA%TYPE := '&NOME';
BEGIN
    INSERT INTO ALUNO VALUES (V_RA,V_NOME);
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        DBMS_OUTPUT.PUT_LINE('Este RA já foi utilizado');
END;
/
```

9. CURSORES EXPLÍCITOS E IMPLÍCITOS

Cursores são áreas compostas de linhas e colunas em memória que servem para armazenar o resultado de uma seleção que retorna 0 (zero) ou mais linhas.

No PL/SQL os cursores podem ser de dois tipos:

- Explícitos
- Implícitos

9.1. CURSORES EXPLÍCITOS

São utilizados para execução de consultas que possam retornar nenhuma ou mais de uma linha.
Neste caso o cursor deve ser explicitamente declarado na área DECLARE.
Para nomear o resultado do cursor é necessário que ele e suas colunas possuam nomes (alias). Para isso algumas regras devem ser observadas:

- O nome do cursor não pode ser igual ao da tabela.
- Para dar um nome a uma coluna da seleção, basta colocar o nome do alias logo após a definição da coluna ou expressão.

```
CURSOR nome_do_cursor IS
  SELECT coluna1, coluna2 ...
  FROM nome_da_tabela;
```

9.1.1. UTILIZANDO: OPEN, FETCH E CLOSE

Após sua declaração, o cursor deverá ser manipulado com o uso de alguns comandos:

OPEN	abre o cursor
FETCH	disponibiliza a linha corrente e posiciona na próxima linha do cursor. As linhas armazenadas no cursor somente poderão ser processadas quando o seu conteúdo for transferido para variáveis que possam ser manipuladas no PL/SQL
CLOSE	fecha o cursor

OBSERVAÇÃO: Após declarar uma variável como sendo do tipo nome_do_cursor%rowtype, essa variável será um tipo de registro (variável composta de diversas subvariáveis) cujas subvariáveis terão os mesmos nomes, tipos e tamanhos e estarão na mesma ordem dos campos especificados no comando SELECT do cursor. O conteúdo da variável desse tipo é referenciado com nome_do_registro.nome da subvariável.

Para cada cursor, quatro atributos podem ser verificados, e seus valores podem ser alterados a cada execução de um comando FETCH. Esses atributos são:

nome_do_cursor%FOUND	retorna TRUE caso FETCH consiga retornar alguma linha e FALSE caso contrário. Se nenhum FETCH tiver sito executado, será retornado NULL
nome_do_cursor%NOTFOUND	retorna FALSE caso FETCH consiga retornar alguma linha e TRUE caso contrário. Se nenhum FETCH tiver sido executado, será retornado NULL
nome_do_cursor%ROWCOUNT	retorna o número de linhas já processadas pelo cursor. Se nenhum FETCH tiver sido executado, será retornado 0 (zero)
nome_do_cursor%ISOPEN	retona TRUE caso o cursor esteja aberto e FALSE caso contrário

Exemplo:

```
DECLARE
   CURSOR c_cliente IS
   SELECT codigo, nome
   FROM cliente;
   v_cliente  c_cliente%rowtype;
BEGIN
   OPEN c_cliente;
   LOOP
      FETCH c_cliente
      INTO  v_cliente;
      EXIT  when c_cliente%notfound;
      DBMS_OUTPUT.PUT_LINE('Cliente: '||v_cliente.nome);
   END LOOP;
   CLOSE c_cliente;
END;
```

9.1.2. UTILIZANDO: FOR

O comando FOR ... LOOP, quando aplicado a um cursor, executa automaticamente as seguintes ações:

- Cria a variável do tipo registro que receberá os dados;
- Abre (OPEN) o cursor;
- Copia as linhas uma a uma (FETCH), a cada interação do comando;
- Controla o final do cursor;
- Fecha (CLOSE) o cursor.

NOTA: Caso seja necessário sair do loop do comando FOR durante sua execução, o cursor deverá ser fechado explicitamente com o comando CLOSE.

Exemplo:

```
DECLARE
   CURSOR c_cliente IS
   SELECT codigo, nome
   FROM cliente;
BEGIN
   FOR v_cliente IN c_cliente
   LOOP
      DBMS_OUTPUT.PUT_LINE('Cliente: '|| v_cliente.nome);
   END LOOP;
END;
```

NOTA:

As variáveis devem ser visíveis no ponto da declaração do cursor:

```
DECLARE
   V_RA    ALUNO.RA%TYPE;
   C_ALUNO IS
      SELECT * FROM ALUNO
         WHERE RA = V_RA;
```

CORRETO

```
DECLARE
   C_ALUNO IS
      SELECT * FROM ALUNO
         WHERE RA = V_RA;
   V_RA    ALUNO.RA%TYPE;
```

ERRADO: V_RA não foi declarado antes de ser referenciado.

EXERCÍCIOS:

1. Criar a tabela PRODUTO:

```
CREATE TABLE PRODUTO (
CODIGO NUMBER(4),
VALOR NUMBER(7,2));
```

Inserir os valores:

```
INSERT INTO PRODUTO VALUES (1000, 300);
INSERT INTO PRODUTO VALUES (1001, 500);
INSERT INTO PRODUTO VALUES (2000, 300);
INSERT INTO PRODUTO VALUES (2001, 500);
```

Criar um bloco PL/SQL para atualizar os preços conforme segue:

- Produtos com CODIGO inferior a 2000: Acrescentar 10% ao VALOR atual.
- Produtos com CODIGO igual ou superior a 2000: Acrescentar 20% ao VALOR atual.

2. Criar a tabela ALUNO:

Observação: Similar ao exercício 1 da página 10. Porém, utiliza cursor.

```
CREATE TABLE ALUNO (
RA NUMBER(9),
DISCIPLINA VARCHAR2(30),
MEDIA NUMBER(3,1),
CARGA_HORA NUMBER(2),
FALTAS NUMBER(2),
RESULTADO VARCHAR2(10));
```

Inserir uma linha deixando a coluna RESULTADO em branco.

```
INSERT INTO ALUNO VALUES (1,'DISC 1',7.5,80,20,'');
INSERT INTO ALUNO VALUES (2,'DISC 1',5.5,80,20,'');
INSERT INTO ALUNO VALUES (3,'DISC 1',7.5,80,40,'');
```

Criar um bloco PL/SQL para preencher a coluna resultado conforme o seguinte:

- Se o aluno obteve média igual ou maior que 7.0 e suas faltas não ultrapassarem 25% da carga horária da disciplina o resultado será: APROVADO.
- Se o aluno obteve média inferior a 7.0 e suas faltas não ultrapassarem 25% da carga horária da disciplina o resultado será: EXAME.
- Para demais casos o resultado será: REPROVADO.

9.2. CURSORES IMPLÍCITOS

Como observamos na seção anterior, cursores explícitos são utilizados para processar instruções SELECT que retornam mais de uma linha.

Porém, todas as instruções SQL são executadas dentro de uma área de contexto e, por isso, têm um cursor (conhecido como cursor SQL) que aponta para esta área.

A PL/SQL implicitamente abre o cursor SQL, processa a instrução SQL nele e fecha o cursor.

O cursor implícito é utilizado para processar instruções (SELECT ... INTO, INSERT, UPDATE e DELETE).

Os comandos OPEN, FETCH e CLOSE não podem ser aplicados a este tipo de cursor.

Os cursores implícitos esperam que apenas uma linha seja retornada. Por isso exceções tais como NO_DATA_FOUND (nenhuma linha satisfaz os critérios de seleção) ou TOO_MANY_ROWS (mais de uma linha satisfaz o critério de seleção) devem ser observadas.

Porém, os atributos %FOUND, %NOTFOUND, %ROWCOUNT e %ISOPEN podem ser verificados.

Exemplo:

```
DECLARE
   V_CODIGO   CLIENTE.CODIGO%TYPE;
   V_NOME     CLIENTE.NOME%TYPE;
BEGIN
   V_CODIGO := '&CODIGO';
   V_NOME   := '&NOME';
   UPDATE CLIENTE
   SET    NOME = V_NOME
   WHERE  CODIGO = V_CODIGO;
   IF SQL%NotFound THEN
      DBMS_OUTPUT.PUT_LINE ('Não houve alteração');
   END IF;
   COMMIT;
END;
```

EXERCÍCIO:

Criar a tabela CLIENTE com os campos CODIGO, VALOR e ESTADO.
Inserir os registros a seguir:

(1,1000,'SP')
(2,1500,'SP')
(3,1000,'MG')
(4,1500,'MG')

Utilizar PL/SQL com um cursor implícito para conceder desconto de 10% somente se ESTADO = 'SP' e VALOR > 1000.

Para efetuar a atualização o usuário deverá entrar com o código do cliente (CODIGO).

```
DECLARE
    v_codigo    cliente.codigo%TYPE;
    v_valor     cliente.valor%TYPE;
    v_estado    cliente.estado%TYPE;
BEGIN
    v_codigo :=&codigo;
    SELECT valor, estado
    INTO v_valor, v_estado
    FROM cliente
    WHERE codigo = v_codigo;
    IF valor > 1000 AND estado = 'SP' then
        UPDATE cliente
        SET valor = v_valor*0.9
        WHERE codigo = v_codigo;
    END IF;
END;
/
```

10. PROCEDURES

Subprogramas que executam uma determinada ação. Não retornam valores e, portanto, não são utilizadas para atribuir valores a variáveis ou como argumento em um comando SELECT.

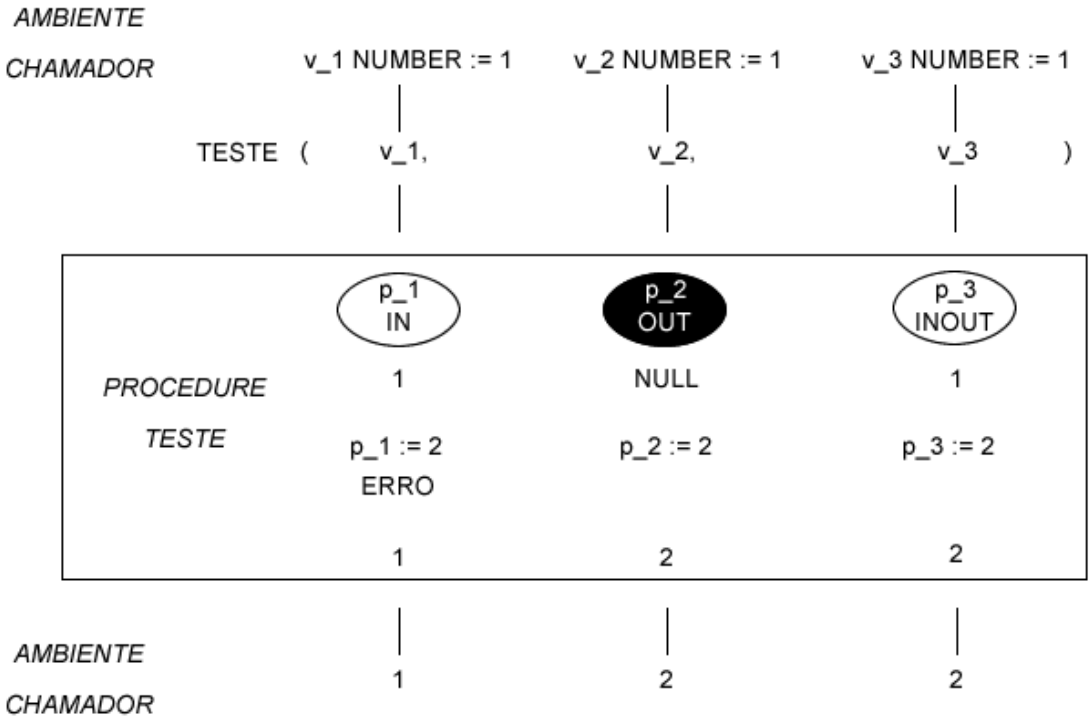
```
CREATE OR REPLACE PROCEDURE nome_procedure
    (argumento1   modo   tipo_de_dados,
     argumento2   modo   tipo_de_dados,
     ...
     argumentoN   modo   tipo_de_dados)
IS ou AS
     variáveis locais, constantes, ...
BEGIN
...
END nome_procedure;
```

ARGUMENTO

Nome da variável que será enviada ou retornada do ambiente chamador para a procedure. Pode ser passado em um dos três modos a seguir.

MODO

- IN (padrão): Passa um valor do ambiente chamador para procedure e este valor não pode ser alterado dentro da mesma. (Passagem de parâmetro por valor)
- OUT: Passa um valor da procedure para o ambiente chamador. (Passagem de parâmetro por referência)
- IN/OUT: Passa um valor do ambiente chamador para a procedure. Esse valor pode ser alterado dentro da mesma e retornar com o valor atualizado para o ambiente chamador. (Passagem de parâmetro por referência)



```
REM teste.sql
CREATE OR REPLACE PROCEDURE teste (
   p_1   IN  NUMBER,
   p_2   OUT NUMBER,
   p_3   IN OUT NUMBER) IS
   v_local    NUMBER;
BEGIN
   DBMS_OUTPUT.PUT_LINE('Recebendo p_1: ' || p_1);
   DBMS_OUTPUT.PUT_LINE('Recebendo p_2: ' || p_2); -- IS NULL
   DBMS_OUTPUT.PUT_LINE('Recebendo p_3: ' || p_3);
-- Utilizando p_1, p_2 e p_3 para atribuir valores a v_local:
   v_local := p_1;
   DBMS_OUTPUT.PUT_LINE('p_1 gravando em v_local: ' || v_local);
   v_local := p_2; -- Válido para Oracle 7, 7.3.4 e 8.0.4, 8i ou superior
   DBMS_OUTPUT.PUT_LINE('p_2 gravando em v_local: ' || v_local);
   v_local := p_3;
   DBMS_OUTPUT.PUT_LINE('p_3 gravando em v_local: ' || v_local);
-- Atribuindo valores para p_1, p_2, p_3:
-- p_1 := 2; -- Se retirar o comentário ocorrerá um erro
   p_2 := 2;
   p_3 := 2;
   DBMS_OUTPUT.PUT_LINE('Retornando p_1: ' || p_1);
   DBMS_OUTPUT.PUT_LINE('Retornando p_2: ' || p_2);
   DBMS_OUTPUT.PUT_LINE('Retornando p_3: ' || p_3);
END teste;
/

REM call_teste.sql
set serveroutput on
DECLARE
  v_1   NUMBER := 1;
  v_2   NUMBER := 1;
  v_3   NUMBER := 1;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Antes de chamar procedure TESTE v_1: ' || v_1);
  DBMS_OUTPUT.PUT_LINE('Antes de chamar procedure TESTE v_2: ' || v_2);
  DBMS_OUTPUT.PUT_LINE('Antes de chamar procedure TESTE v_3: ' || v_3);

  teste(v_1, v_2, v_3);

  DBMS_OUTPUT.PUT_LINE('Após chamar procedure TESTE v_1: ' || v_1);
  DBMS_OUTPUT.PUT_LINE('Após chamar procedure TESTE v_2: ' || v_2);
  DBMS_OUTPUT.PUT_LINE('Após chamar procedure TESTE v_3: ' || v_3);

END;
/
```

Exemplo 1:

```
REM parametros.sql
REM Esta procedure apresenta os diferentes modos dos parametros

CREATE OR REPLACE PROCEDURE parametros (
  p_In    IN NUMBER,
  p_Out   OUT NUMBER,
  p_InOut IN OUT NUMBER) IS

  v_Local  NUMBER := 0;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Dentro da procedure PARAMETROS:');
  IF (p_In IS NULL) THEN
    DBMS_OUTPUT.PUT('  p_In is NULL');
  ELSE
    DBMS_OUTPUT.PUT('  p_In = ' || p_In);
  END IF;

  IF (p_Out IS NULL) THEN
    DBMS_OUTPUT.PUT('  p_Out is NULL');
  ELSE
    DBMS_OUTPUT.PUT('  p_Out = ' || p_Out);
  END IF;

  IF (p_InOut IS NULL) THEN
    DBMS_OUTPUT.PUT_LINE('  p_InOut is NULL');
  ELSE
    DBMS_OUTPUT.PUT_LINE('  p_InOut = ' || p_InOut);
  END IF;

  v_Local := p_In;      -- Válido

  -- p_In := 7;         -- Ilegal

  p_Out := 7;           -- Válido

  v_Local := p_Out;     -- Válido para Oracle 7, 7.3.4 e 8.0.4, 8i ou superior

  v_Local := p_InOut;   -- Válido

  p_InOut := 8;         -- Válido

  DBMS_OUTPUT.PUT_LINE('Ao final da procedure PARAMETROS:');
  IF (p_In IS NULL) THEN
    DBMS_OUTPUT.PUT('  p_In is NULL');
  ELSE
    DBMS_OUTPUT.PUT('  p_In = ' || p_In);
  END IF;

  IF (p_Out IS NULL) THEN
    DBMS_OUTPUT.PUT('  p_Out is NULL');
  ELSE
    DBMS_OUTPUT.PUT('  p_Out = ' || p_Out);
  END IF;

  IF (p_InOut IS NULL) THEN
    DBMS_OUTPUT.PUT_LINE('  p_InOut is NULL');
  ELSE
    DBMS_OUTPUT.PUT_LINE('  p_InOut = ' || p_InOut);
  END IF;

END parametros;
/
```

```
REM call_parametros.sql

set serveroutput on

DECLARE
  v_In    NUMBER := 1;
  v_Out   NUMBER := 2;
  v_InOut NUMBER := 3;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Antes de chamar procedure PARAMETROS:');
  DBMS_OUTPUT.PUT_LINE('   v_In = '    || v_In ||
                       '   v_Out = '   || v_Out ||
                       '   v_InOut = ' || v_InOut);

  parametros(v_In, v_Out, v_InOut);

  DBMS_OUTPUT.PUT_LINE('Após chamar procedure PARAMETROS:');
  DBMS_OUTPUT.PUT_LINE('   v_In = '    || v_In ||
                       '   v_Out = '   || v_Out ||
                       '   v_InOut = ' || v_InOut);
END;
/

-- Chamar procedure PARAMETROS com um literal para p_In. Isso é valido.
DECLARE
  v_Out   NUMBER := 2;
  v_InOut NUMBER := 3;
BEGIN
  parametros(1, v_Out, v_InOut);
END;

-- Substituir v_Out or v_InOut por um literal, provocará um erro de compilação.
DECLARE
  v_Out NUMBER := 2;
BEGIN
  parametros(1, v_Out, 3);
END;

-- Substituir v_Out or v_InOut por um literal, provocará um erro de compilação.
DECLARE
  v_InOut NUMBER := 3;
BEGIN
  parametros(1, 2, v_InOut);
END;
```

Exemplo 2:

```
REM soma.sql
CREATE OR REPLACE PROCEDURE soma (
  p_1   IN NUMBER,
  p_2   IN NUMBER,
  p_t   OUT NUMBER) IS
BEGIN
  p_t := p_1 + p_2;
  DBMS_OUTPUT.PUT_LINE(p_1||'+'||p_2||'='||p_t);
END soma;
/

REM call_soma.sql
DECLARE
  v_1   NUMBER := 1;
  v_2   NUMBER := 2;
  v_t   NUMBER;
BEGIN
  soma(v_1, v_2, v_t);
END;
/
```

Exemplo 3:

```
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);

CREATE OR REPLACE PROCEDURE aumenta_valor
(v_categoria IN produto.categoria%TYPE,
 v_percentual  NUMBER)
IS
BEGIN
UPDATE produto
SET valor  = valor*(1+v_percentual/100)
WHERE categoria = v_categoria;
END aumenta_valor;
/

EXEC aumenta_valor  (10,25);

SELECT * FROM produto;

CD_PRODUTO NM_PRODUTO         VL_CUSTO   CD_CATEGORIA
---------- ----------------- ---------- ------------
         1 produto1                3,13           10
         2 produto2                 3,2           20
         3 produto3                 5,8           30
```

EXERCÍCIO 1:

Criar uma procedure para inclusão de dados na tabela produto.

```
CREATE OR REPLACE PROCEDURE insere_produto
(v_codigo     IN produto.codigo%TYPE,
 v_nome       IN produto.nome%TYPE,
 v_valor      IN produto.valor%TYPE,
 v_categoria IN produto.categoria%TYPE)
IS
BEGIN
INSERT INTO produto
(codigo, nome, valor, categoria)
VALUES
(v_codigo, v_nome, v_valor, v_categoria);
EXCEPTION
   WHEN DUP_VAL_ON_INDEX THEN
      DBMS_OUTPUT.PUT_LINE('Código de produto já cadastrado');
END insere_produto;
/

EXEC insere_produto (4,'Produto4',4.2,10)

SELECT * FROM produto;

CD_PRODUTO NM_PRODUTO         VL_CUSTO   CD_CATEGORIA
---------- ----------------- ---------- ------------
         1 produto1                 2,5           10
         2 produto2                 3,2           20
         3 produto3                 5,8           30
         4 Produto4                 4,2           10
```

EXERCÍCIO 2:

Criar uma procedure para consultar um produto informando o código:

```
CREATE OR REPLACE PROCEDURE CONSULTA_PRODUTO
   (P_CODIGO    IN  PRODUTO.CODIGO%TYPE)
IS
   V_CODIGO NUMBER(5);
   V_DESCRICAO VARCHAR2(20);
   V_VALOR      NUMBER(7,2);
   V_CATEGORIA NUMBER(5);
BEGIN
   SELECT CODIGO, DESCRICAO, VALOR, CATEGORIA
      INTO V_CODIGO, V_DESCRICAO, V_VALOR, V_CATEGORIA
      FROM PRODUTO
      WHERE CODIGO = P_CODIGO;
   DBMS_OUTPUT.PUT_LINE('CODIGO = '|| V_CODIGO);
   DBMS_OUTPUT.PUT_LINE('DESCRICAO = '|| V_DESCRICAO);
   DBMS_OUTPUT.PUT_LINE('VALOR = '|| V_VALOR);
   DBMS_OUTPUT.PUT_LINE('CATEGORIA = '|| V_CATEGORIA);
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      RAISE_APPLICATION_ERROR (-20500, 'PRODUTO NÃO ENCONTRADO');
END CONSULTA_PRODUTO;
/

EXEC consulta_produto (3);
```

EXERCÍCIO 3:

```
CREATE TABLE ALUNO (
RA NUMBER(5) PRIMARY KEY,
NOME        VARCHAR2(20),
NOTA1       NUMBER(3,1),
NOTA2       NUMBER(3,1),
MEDIA       NUMBER(3,1),
RESULTADO   VARCHAR2(10));

INSERT INTO ALUNO (RA, NOME, NOTA1, NOTA2) VALUES (1,'ANTONIO',9,7);
INSERT INTO ALUNO (RA, NOME, NOTA1, NOTA2) VALUES (2,'BEATRIZ',4,6);
INSERT INTO ALUNO (RA, NOME, NOTA1, NOTA2) VALUES (3,'CLAUDIO',8,6);
```

Criar a procedure NOTAS para inserir os outros valores, conforme as seguintes regras:

- MEDIA: (NOTA1+NOTA2)/2
- RESULTADO: MEDIA >= 7 – 'APROVADO'; < 7 - 'EXAME'

```
CREATE OR REPLACE PROCEDURE NOTAS IS
   CURSOR C_ALUNO IS
   SELECT * FROM ALUNO;
   V_MEDIA ALUNO.MEDIA%TYPE;
BEGIN
   FOR R_ALUNO IN C_ALUNO
   LOOP
   V_MEDIA := (R_ALUNO.NOTA1+R_ALUNO.NOTA2)/2;
      IF V_MEDIA >= 7 THEN
         UPDATE ALUNO
            SET MEDIA = V_MEDIA, RESULTADO = 'APROVADO'
            WHERE RA = R_ALUNO.RA;
      ELSE
         UPDATE ALUNO
            SET MEDIA = V_MEDIA, RESULTADO = 'EXAME'
            WHERE RA = R_ALUNO.RA;
      END IF;
   END LOOP;
END;
/
```

IMPORTANTE:

- O texto de origem (código-fonte) e a forma compilada (p-code) dos subprogramas (procedures e funções) são armazenados no dicionário de dados. Quando um subprograma é chamado, o p-code é lido a partir do disco e, se necessário, executado. Uma vez que tenha sido lido do disco, o p-code é armazenado é armazenado na parte compartilhada do pool da SGA (System Global Area), onde, quando necessário, pode ser acessado por vários usuários.
- Para verificar o status de uma procedure:

```
SELECT OBJECT_NAME, OBJECT_TYPE, STATUS
    FROM USER_OBJECTS WHERE OBJECT_NAME = 'nome_da_procedure';
```

- Para visualizar o código-fonte de uma procedure:

```
SELECT TEXT FROM USER_SOURCE
    WHERE NAME = 'nome_da_procedure'  ORDER BY LINE;
```

- Para eliminar uma procedure:

```
SHOW ERRORS PROCEDURE nome_da_procedure;
```

- Para eliminar uma procedure:

```
DROP PROCEDURE nome_da_procedure;
```

11. FUNCTIONS

Subprogramas que executam uma determinada ação e retornam valores. Portanto, podem ser invocadas por meio de um comando SELECT ou utilizadas em cálculos.

```
CREATE OR REPLACE FUNCTION nome_funcao
   (argumento1  modo  tipo_de_dados,
    argumento2  modo  tipo_de_dados,
    ...
    argumentoN  modo  tipo_de_dados)
RETURN tipo_de_dado
IS ou AS
    variáveis locais, constantes, ...
BEGIN
...
END nome_funcao;
```

Exemplo:

Criar uma função que retorne a quantidade de produtos de uma determinada categoria. Para isso, ela receberá o código da categoria que será totalizado.

```
CREATE OR REPLACE FUNCTION conta_produto
(p_categoria IN produto.categoria%TYPE)
RETURN number
IS
v_total  number;
BEGIN
SELECT COUNT(*) INTO v_total FROM produto
   WHERE categoria = p_categoria;
RETURN v_total;
END conta_produto;
/

SELECT conta_produto (10) FROM dual;
```

```
CONTA_PRODUTO(10)
-----------------
                2
```

EXERCÍCIOS:

1. Criar uma função para apresentar o fatorial de um número a ser informado no comando SELECT. Lembrete: x! = x * (x-1)!

```
CREATE OR REPLACE FUNCTION fatorial
(p_n IN NUMBER)
RETURN number
IS
BEGIN
   IF p_n = 1 THEN
      RETURN 1;
   ELSE
      RETURN p_n * fatorial (p_n-1);
   END IF;
END fatorial;

SELECT fatorial (3) FROM dual;
/
```

```
FATORIAL(3)
-----------
          6
```

2. Criar uma função que recebe um número de RA de aluno, como uma entrada e retorna o nome e o sobrenome concatenados.

```
CREATE TABLE ALUNO (
RA NUMBER,
NOME VARCHAR2(20),
SOBRENOME VARCHAR2(30));

INSERT INTO ALUNO VALUES (1,'ANTONIO','ALVES');
INSERT INTO ALUNO VALUES (2,'BEATRIZ','BERNARDES');

CREATE OR REPLACE FUNCTION NOME_ALUNO (
   P_RA ALUNO.RA%TYPE)
   RETURN VARCHAR2 IS
   V_NOMECOMPLETO VARCHAR2(60);
BEGIN
   SELECT NOME || ' ' || SOBRENOME
      INTO V_NOMECOMPLETO
      FROM ALUNO
      WHERE RA = P_RA;
   RETURN V_NOMECOMPLETO;
END NOME_ALUNO;
/

SELECT RA, NOME_ALUNO(RA) "NOME COMPLETO"
   FROM ALUNO;

        RA NOME COMPLETO
---------- --------------------
         1 ANTONIO ALVES
         2 BEATRIZ BERNARDES
```

IMPORTANTE:

- Para verificar o status de uma função:

```
SELECT OBJECT_NAME, OBJECT_TYPE, STATUS
   FROM USER_OBJECTS WHERE OBJECT_NAME = 'nome_da_funcao';
```

- Para visualizar o código-fonte de uma função:

```
SELECT TEXT FROM USER_SOURCE
   WHERE NAME = 'nome_da_funcao' ORDER BY LINE;
```

- Para eliminar uma função:

```
SHOW ERRORS FUNCTION nome_da_funcao;
```

- Para eliminar uma função:

```
DROP FUNCTION nome_da_funcao;
```

12. TRIGGERS

Blocos PL/SQL disparados automática e implicitamente sempre que ocorrer um evento associado a uma tabela (INSERT, UPDATE ou DELETE).

Utilizadas para:

- Manutenção de tabelas
- Implementação de níveis de segurança mais complexos
- Geração de valores de colunas (Exemplo: gerar o valor total do pedido a cada inclusão, alteração ou exclusão na tabela item_pedido)

```
CREATE OR REPLACE TRIGGER nome_trigger
{BEFORE/AFTER} {INSERT, UPDATE, DELETE} OF (nome_coluna1, nome_coluna2, ...)
ON nome_tabela
FOR EACH ROW
REFERENCING, OLD AS ANTIGO NEW AS NOVO
WHEN condição
DECLARE
...
BEGIN
...
END;
```

NOTA: A cláusula REFERENCING está substituindo as áreas de memória OLD e NEW por ANTIGO e NOVO.

TEMPO

Os tempos de uma trigger podem ser:

- BEFORE - antes do evento
- AFTER - depois do evento

EVENTO

Os eventos de uma trigger podem ser:

- INSERT
- UPDATE
- DELETE

TIPO

Indica quantas vezes a trigger poderá ser disparada. Os tipos podem ser:

- **Comando:** acionada antes ou depois de um comando, independentemente de este afetar uma ou mais linhas. Não permite acesso às linhas atualizadas por meio dos prefixos :OLD E :NEW. Não utiliza a cláusula FOR EACH ROW no cabeçalho de criação.
- **Linha:** acionada uma vez para cada linha afetada pelo comando ao qual a trigger estiver associada. Permite o uso dos prefixos :OLD e :NEW no corpo da trigger e das cláusulas REFERENCING e WHEN em seu cabeçalho. Deve-se incluir a cláusula FOR EACH ROW no cabeçalho.

Cláusula WHEN

Utilizada para restringir as linhas que irão disparar a trigger.

Regras para criação de triggers:

- Número máximo de triggers possíveis para uma tabela: doze (todas as combinações possíveis entre tempos, eventos e tipos).
- Não podem ser utilizados os comandos COMMIT e ROLLBACK, inclusive em procedures e functions chamadas pela trigger.
- Não podem ser alteradas chaves primárias, únicas ou estrangeiras.
- Não podem ser feitas referências a campos do tipo LONG E LONG RAW.

Para testar o evento de chamada da trigger são disponibilizados os seguintes predicados:

- Inserting: retorna TRUE se a trigger foi disparada por um comando INSERT.
- Updating: retorna TRUE se a trigger foi disparada por um comando UPDATE.
- Deleting: retorna TRUE se a trigger foi disparada por um comando DELETE.

Conteúdo das áreas OLD e NEW (apenas triggers de linha)		
EVENTO	OLD	NEW
INSERT	NULL	valor inserido
UPDATE	valor antes da alteração	valor após a alteração
DELETE	valor antes da exclusão	NULL

- Em triggers com cláusula de tempo BEFORE é possível consultar e alterar o valor de :NEW.
- Em triggers com cláusula de tempo AFTER é possível apenas consultar o valor de :NEW.

Comandos:

DROP TRIGGER nome_trigger	Elimina uma trigger
ALTER TRIGGER nome_trigger ENABLE	Habilita uma trigger
ALTER TRIGGER nome_trigger DISABLE	Desabilita uma trigger
ALTER TABLE nome_tabela ENABLE ALL_TRIGGERS	Habilita todas as triggers de uma tabela
ALTER TABLE nome_tabela DISABLE ALL_TRIGGERS	Desabilita todas as triggers de uma tabela

Exemplo:

```
CREATE TABLE produto
(codigo   NUMBER(4),
 valor    NUMBER(7,2));

CREATE TABLE valor_produto
(codigo          NUMBER(4),
valor_anterior   NUMBER(7,2),
valor_novo       NUMBER(7,2));

CREATE OR REPLACE TRIGGER verifica_valor
BEFORE UPDATE
OF valor
ON produto
FOR EACH ROW
BEGIN
INSERT INTO valor_produto
VALUES
(:OLD.codigo, :OLD.valor, :NEW.valor);
END;
/
```

```
INSERT INTO produto VALUES (1, 2.5);
INSERT INTO produto VALUES (2, 3.2);
INSERT INTO produto VALUES (3, 5.8);

UPDATE produto
SET valor = 5.4
WHERE codigo = 3;

SELECT * FROM valor_produto;

CODIGO     VALOR_ANTERIOR VALOR_NOVO
---------- -------------- ----------
         3            5,8        5,4
```

EXERCÍCIO:

Incluir na tabela valor_produto os campos:

```
usuario      VARCHAR2, (30)
data_atual   DATE
```

Alterar a trigger verifica_valor para que também sejam incluídos na tabela valor_produto a data do sistema no momento da atualização e o nome do usuário que realizou a alteração no campo valor.

```
ALTER TABLE valor_produto
ADD (usuario      VARCHAR2(30),
     data_atual   DATE);

CREATE OR REPLACE TRIGGER verifica_valor
BEFORE UPDATE
OF valor
ON produto
FOR EACH ROW
BEGIN
INSERT INTO valor_produto
VALUES
(:OLD.codigo, :OLD.valor, :NEW.valor, user, sysdate);
END;
/
```

13. PACKAGES

Objetos do Banco de Dados equivalentes a bibliotecas que armazenam:

- procedures
- functions
- definições de cursores
- variáveis e constantes
- definições de exceções

Um package é comporto de duas partes:

1. Especificação:

   Área onde são feitas as declarações públicas. As variáveis, constantes, cursores, exceções e subprogramas estarão disponíveis para uso externo ao package.

2. Corpo:

   Área onde são feitas as declarações privadas que estarão disponíveis apenas dentro do package e a definição de ações para os subprogramas públicos e privados.

O Corpo do package é um objeto de dicionário de dados separado da Especificação (cabeçalho). Ele não poderá ser compilado com sucesso a menos que a Especificação já tenha sido compilada.

```
-- especificacao

CREATE OR REPLACE PACKAGE nome_package IS
    PROCEDURE nome_procedure (lista_de_parametros);
    FUNCTION nome_function (lista_de_parametros);
    Declaração de variáveis, constantes, exceções e cursores públicos
END nome_package;



-- corpo

CREATE OR REPLACE PACKAGE BODY nome_package IS

    Declaração de variáveis, constantes, exceções e cursores privados

    PROCEDURE nome_procedure (lista_de_parametros)
    IS
       ...
    BEGIN
       ...
    END nome_procedure;

    FUNCTION nome_function (lista_de_parametros)
    RETURN tipo_de_dado
    IS
       ...
    BEGIN
       ...
    RETURN
    END nome_funcao;
END;
```

Exemplo 1:

```
CREATE OR REPLACE PACKAGE pack_1 IS
    PROCEDURE proc_1;
    FUNCTION  func_1 RETURN VARCHAR2;
END pack_1;
/


CREATE OR REPLACE PACKAGE BODY pack_1 IS
PROCEDURE proc_1
IS
BEGIN
    DBMS_OUTPUT.PUT_LINE('Mensagem da Procedure');
END proc_1;
FUNCTION func_1 RETURN VARCHAR2 IS
BEGIN
    RETURN('Mensagem da Function');
END func_1;
END pack_1;
/


EXEC pack_1.proc_1;

SELECT pack_1.func_1 FROM DUAL;
```

Exemplo 2:

```
CREATE OR REPLACE PACKAGE pack_aluno IS
    PROCEDURE adiciona_aluno
    ( v_ra          IN aluno.ra%TYPE,
      v_nome        IN aluno.nome%TYPE );
END pack_aluno;
/


CREATE OR REPLACE PACKAGE BODY pack_aluno IS
PROCEDURE adiciona_aluno
(v_ra           IN aluno.ra%TYPE,
 v_nome         IN aluno.nome%TYPE)
IS
BEGIN
INSERT INTO aluno
(ra, nome)
VALUES
(v_ra, v_nome);
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        DBMS_OUTPUT.PUT_LINE('RA já cadastrado');
END adiciona_aluno;
END pack_aluno;
/


EXEC pack_aluno.adiciona_aluno (1,'Antonio');
```

14. SQL DINÂMICO

Um bloco PL/SQL não executa comandos DDL (CREATE TABLE, DROP TABLE, TRUNCATE, etc.). Por isso, o Oracle oferece um recurso conhecido como NDS (Native Dynamic SQL) que permite, por meio da linguagem PL/SQL, executar dinamicamente comandos DDL.
SQL dinâmico é um comando válido, codificado dentro de uma string e executado através do comando EXECUTE IMMEDIATE.

A procedure abaixo utiliza um comando DDL (CREATE TABLE) e não poderá ser criada.

```
CREATE OR REPLACE PROCEDURE cria_tabela
IS
BEGIN
  create table teste (coluna1 number(5));
END cria_tabela;
/
```

Para criá-la devemos utilizar SQL dinâmico, conforme segue:

```
CREATE OR REPLACE PROCEDURE cria_tabela
(nome_tabela IN VARCHAR2)
IS
comando VARCHAR2(100);
BEGIN
  comando := 'create table '||nome_tabela||' ( coluna1 number(5) )';
  EXECUTE IMMEDIATE comando;
END cria_tabela;
/

EXEC cria_tabela ('teste');

INSERT INTO teste VALUES (1);
INSERT INTO teste VALUES (2);
INSERT INTO teste VALUES (3);
```

Para eliminar a tabela também devemos utilizar SQL dinâmico:

```
CREATE OR REPLACE PROCEDURE limpa_teste
IS
  comando VARCHAR2(100);
BEGIN
  comando := 'TRUNCATE TABLE teste';
  EXECUTE IMMEDIATE comando;
END limpa_teste;
/

exec limpa_teste;
```

BIBLIOGRAFIA

FANDERUFF, D. *Dominando o Oracle 9i – Modelagem e Desenvolvimento.* São Paulo: Makron Books, 2003.

MORELLI, E. T. *Oracle 9i – SQL, PL/SQL e Administração.* São Paulo: Érica, 2005.

RAMALHO, J. A. A. *Oracle 9i.* São Paulo: Berkeley, 2002.

URMAN, S. Oracle 9i – Programação PL/SQL. Rio de Janeiro: Campus, 2002.