7. LINGUAGEM PL/SQL

7.1. INTRODUÇÃO

O PL/SQL é uma linguagem da Oracle que tem por objetivo processar informações do banco de dados. Um bloco PL/SQL pode ser executado a partir do **SQL Developer**, SQL*Plus, ou de qualquer linguagem PRO*Oracle.

A linguagem PL/SQL é uma extensão da Linguagem SQL, vista na **disciplina de BDI**. 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 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.

7.2. BENEFÍCIOS DA LINGUAGEM PL/SQL

7.2.1. Integração

A Linguagem PL/SQL desempenha um papel central tanto para o Oracle Server, por meio de **blocos anônimos, procedimentos armazenados, funções armazenadas, gatilhos de banco de dados e pacotes**, quanto para as ferramentas de desenvolvimento Oracle, por meio de gatilhos de componentes da Oracle Develop (Forms, Reports e 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 capacidades de programação procedural.

7.2.2. Melhorar Desempenho

A Linguagem PL/SQL pode ser usada para agrupar as instruções SQL em um único bloco e enviar esse bloco inteiro para o servidor em uma única chamada, reduzindo assim o tráfego da rede. Sem o código PL/SQL, as instruções SQL seriam enviadas ao Oracle Server uma de cada vez. Cada instrução SQL resulta em outra chamada do Oracle Server e uma maior sobrecarga de desempenho. Em um ambiente de rede, a sobrecarga pode se tornar significativa. A Figura 1 a seguir ilustra a melhora do desempenho acima descrito:

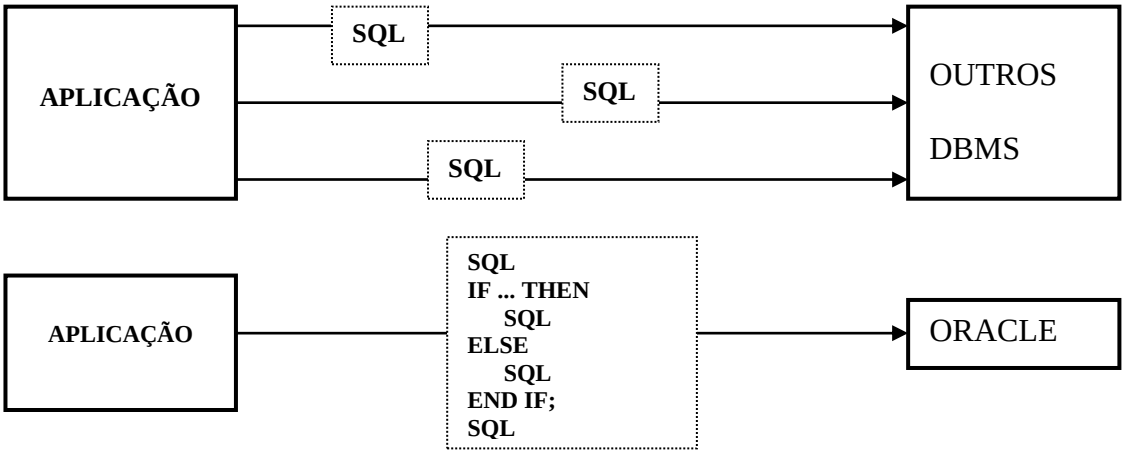


Figura 1. Exemplo de Melhora de Desempenho

A Linguagem PL/SQL também pode cooperar com as ferramentas de desenvolvimento de aplicação do Oracle Server como, por exemplo, Oracle Develop Forms e Reports. Ao adicionar recursos de processamento procedural a essas ferramentas, a Linguagem PL/SQL aumenta o desempenho.

7.2.3. Características

Operações Permitidas

- Manipulação de Dados: alteração, eliminação, inclusão e seleção;
- Criar variáveis e constantes herdando o tipo de dados e o tamanho de outras variáveis e constantes ou de colunas de tabelas;
- Criar cursores para tratar o resultado de uma *query* que retorna 0 ou mais linhas;
- Criar registros para guardar o resultado de um cursor ou campos de tabelas, herdando os tipos de dados e o tamanho das colunas;

- Tratar erros;
- Criar *labels* para controlar o fluxo de execução;
- Utilizar comando de repetição e comparação.
- Criar *triggers* (gatilhos) para garantia de integridade (restrições), segurança, etc.

7.3. ESTRUTURA DE UM BLOCO PL/SQL

A estrutura de um bloco PL/SQL é composta por uma área de declaração, uma área de comandos e uma área de exceções:

Ilustrando:

```
DECLARE – opcional
Declarações – variáveis, cursores, constantes, estruturas, tabelas, exceções definidas pelo usuário
BEGIN - obrigatório
        Estruturas executáveis (comandos)
        Instruções SQL (manipular dados do banco de dados)
        Instruções PL/SQL (manipular dados no bloco)
EXCEPTION - opcional
        Tratamento de exceções (pode conter outros blocos)
        Ações a serem desempenhadas qdo ocorrem erros ou condições anormais
END; - obrigatório
```

7.4. USO DE VARIÁVEIS

Com o código PL/SQL você poderá declarar variáveis para usá-las em instruções procedurais e SQL onde uma expressão possa ser usada.

- **Armazenamento temporário de dados**: os dados podem ser armazenados temporariamente em uma ou mais variáveis para uso quando na validação da entrada de dados para processamento posterior no processo de fluxo de dados;

- **Manipulação de valores armazenados**: as variáveis podem ser usadas para cálculo e manipulação de outros dados sem acessar o banco de dados;

- **Reutilização**: quando declaradas, as variáveis podem ser usadas repetidamente em uma aplicação simplesmente referenciando-as em outras instruções, incluindo outras instruções declarativas;

- **De fácil manutenção**: ao usar %TYPE e %ROWTYPE, você declara variáveis, baseando as declarações nas definições das colunas de banco de dados. As variáveis PL/SQL ou variáveis de cursor anteriormente declaradas no escopo atual, poderão usar também os atributos %TYPE e %ROWTYPE como especificadores de tipos de dados. Se uma definição subjacente for alterada, a declaração de variável se altera de acordo durante a execução. Isso permite a independência dos dados, reduz custos de manutenção e permite que os programas se adaptem, conforme o banco de dados for alterado, para atender às novas necessidades comerciais.

7.4.1. Tratando variáveis em PL/SQL

- Declarar e inicializar as variáveis na seção de declaração;
- Atribuir novos valores às variáveis na seção executável;
- Passar valores aos subprogramas PL/SQL por meio de parâmetros: IN passa valores, OUT retorna valores;
- Ver os resultados em um bloco PL/SQL por meio de variáveis de saída: usar variáveis de referência.

7.4.2. Tipos de variáveis PL/SQL

- **Escalar**: armazena um único valor.
- **Composta**: os registros permitem que os grupos de campos sejam definidos e manipulados nos blocos PL/SQL.
- **Referência**: armazenam valores chamados de indicadores, que designam outros itens de programa.
- **LOB**: armazenam valores chamados de endereços, que especificam a localização de objetos grandes (imagens gráficas) que são armazenados fora de linha.

7.4.3. Outros tipos de variáveis PL/SQL

- BOOLEANO: TRUE / FALSE;
- DATE: definição de data;
- BLOB: definição de imagens (fotografia);
- LONG RAW: definição de textos longos;
- BFILE: definição de imagens animadas.

7.4.4. Declarando variáveis PL/SQL

```
Identificador [CONSTANT] tipo de dados [NOT NULL]
[:= | DEFAULT expr];
```

- Identificador : é nome da variável.
- CONSTANT : restringe as variáveis para que o seu valor não possa ser alterado.
- Tipo de dados : escalares, compostos, referenciais ou LOB.
- NOT NULL: restringe a variável para que ela contenha um valor.
- Expr : é uma expressão PL/SQL que pode ser uma literal, outra variável ou uma expressão que envolve operadores e funções.

Exemplo de instruções:

```
Declare
   v_nascimento    DATE;
   v_codigo    NUMBER(5) NOT NULL := 10;
   v_cidade    VARCHAR2(35) :=  'Assis';
   v_numero    CONSTANT NUMBER := 1234;
```

7.4.5. Atribuindo valores a variáveis

```
Identificador := expr;
```

- Identificador : é nome da variável.
- expr : é uma expressão PL/SQL que pode ser uma literal, uma outra variável ou uma expressão que envolve operadores e funções.

Exemplo de instruções:

```
v_nascimento := '30-AGO-01'
```

```
v_nome := 'Alex Poletto'
```

```
SQL> SELECT salario*0.20
     2   INTO v_gratificacao
     3   FROM professor
     4   WHERE depto=2;
```

7.4.6. Tipos de dados escalares básicos

Um tipo de dados escalares armazena um valor único e não possui componentes internos. Os tipos de dados escalares podem ser classificados em quatro categorias: número, caractere, data e booleano.

Tipo de dado	Descrição
• VARCHAR2(maximum_length):	Tamanho variável com até 32.767 bytes. Não há tamanho default para as constantes e variáveis VARCHAR2.
• NUMBER[(precisão, escala)]	Tipo básico para números fixos e de ponto flutuante.
• DATE	Inclui hora do dia em segundos desde a meia-noite. Entre 4712 A.C. e 9999 D.C.
• CHAR[(maximum_length)]	Tamanho fixo com até 32.767 bytes. Se você não especificar um comprimento, o tamanho default será definido como 1.
• LONG	Tamanho variável com até 32.767 bytes. A largura máxima de uma coluna de banco de dados LONG é de 2.147.483.647 bytes.
• LONG RAW	Binaries e strings de byte de até 32.760 bytes. Não são interpretados pelo código PL/SQL.
• BOOLEAN	Lógicos: TRUE, FALSE ou NULL

Exemplo de instruções:

```
v_descricao   VARCHAR2(30);
v_contador    BINARY_INTEGER := 0;
v_soma        NUMBER(10,2) := 0;
v_reserva     DATE := SYSDATE + 7;
c_taxa        CONSTANT NUMBER(5,2) := 15.30;
v_filhos      BOOLEAN NOT NULL := TRUE;
v_preco1      NUMBER(10,2) := 1500
v_preco2      NUMBER(10,2) := 2500
v_aumento     BOOLEAN := (v_preco1 < v_preco2);
```

7.4.7. O atributo %TYPE

Utilizado para declarar uma variável de acordo com uma definição de coluna de banco de dados ou de acordo com outra variável anteriormente declarada.

Exemplo de instruções:

```
…..
v_descricao   depto.descricao%TYPE
v_salario     v_soma%TYPE  := 0;
…..
```

7.4.8. Variáveis de tipo de dados LOB

Utilizado para armazenar blocos de dados não estruturados (texto, imagens gráficas, videoclipes e formatos de arquivo para armazenar sons) de até 4 gigabytes em tamanho. Os tipos de dados LOB fornecem acesso eficiente, aleatório e em intervalos aos dados, podendo ser atributos de um tipo de objeto.

Tipo de dado	Descrição
• CLOB	Armazenar blocos grandes de dados com caracteres de um único byte no banco de dados.
• BLOB	Armazenar objetos binários grandes no banco de dados em linha ou fora de linha.
• BFILE	Armazenar objetos grandes binários em arquivos do sistema operacional fora do banco de dados.
• NCLOB	Armazenar blocos grandes de dados NCHAR de byte único ou de bytes múltiplos de largura fixa no banco de dados, dentro e fora de linha.

7.4.9. Variáveis de ligação

Uma variável de ligação é uma variável que você declara em um ambiente de host e usa para passar valores de tempo de execução, número ou caractere, para ou de um ou mais programas PL/SQL, os quais podem usá-la como usariam qualquer outra variável.

Exemplo de instruções para declarar

```
SQL> VARIABLE retorna_codigo NUMBER
SQL> VARIABLE retorna_mensagem VARCHAR2(35)
```

Exemplo de instruções para exibir

```
SQL> PRINT retorna_codigo
```

7.4.10. DBMS_OUTPUT.PUT_LINE

- Um procedimento de pacote fornecido pela Oracle;
- Uma alternativa para exibir dados a partir de um bloco PL/SQL;
- Deverá ser ativado em SQL*Plus com SET SERVEROUTPUT ON

Exemplo de instruções:

```
SET SERVEROUTPUT ON
ACCEPT v_anual PROMPT 'Entre com o salário anual.:'
DECLARE
    v_salario NUMBER(10,2) := &v_anual;
BEGIN
    v_salario := v_salario/12;
    DBMS_OUTPUT.PUT_LINE ('O salário mensal é ' | | TO_CHAR(v_salario));
END:
```

7.5. CRIANDO INSTRUÇÕES EXECUTÁVEIS

7.5.1. Diretrizes e sintaxe de bloco PL/SQL

- As instruções podem continuar por várias linhas;
- As unidades lexicais podem ser separadas por: espaços, delimitadores, identificadores, literais e comentários.

Delimitadores

Símbolos Simples		**Símbolos Compostos**	
Símbolo	**Significado**	***Símbolo***	**Significado**
+	Adicionar	<>	Relacional
-	Subtrair	!=	Relacional
*	Multiplicar	||	Concatenação
/	Dividir	--	Comentário de 1 linha
=	Relacional	**/***	Inicia comentário
@	Acesso remoto	***/**	Finaliza comentário
:	Finalizador instruções	:=	Atribuição

Identificadores

- Podem conter até 30 caracteres;
- Não podem conter palavras reservadas, a não ser que estejam entre aspas duplas (“SELECT”);
- Devem ser iniciados por um caractere alfabético;
- Não devem ter o mesmo nome de uma coluna de tabela de banco de dados.

Literais

- É um valor booleano, um valor numérico explícito, um caractere ou uma *string*, não representados por um identificador.
  - Caracteres e literais de data devem estar entre aspas simples;
  - Os números poderão ser valores simples ou notações científicas;
- Um bloco PL/SQL é finalizado por uma barra (/) em uma linha sozinha.

7.5.2. Comentando código

- Crie prefixos de dois hífens (--) para comentários de uma única linha;
- Coloque os comentários de várias linhas entre os símbolos /* e */.

```
.....
  v_salario NUMBER(10,2);
BEGIN
   /* computer o salário annual baseado nos salários mensais
      para usá-lo em cálculos */
   v_salário := &v_anual * 12;
END;   -- Final do bloco
```

7.5.3. Funções SQL não disponíveis em PL/SQL

- Funções de grupo: AVG, MIN, MAX, COUNT, SUM, STDDEV e VARIANCE. Somente funcionam em instruções SQL dentro de um bloco PL/SQL

Obs: Assim não funciona: v_total := SUM (salario);

7.5.4. Funções PL/SQL

- Criar uma lista de correspondência:

Exemplo de instruções:

```
v_lista_endereco := v_nome | | CHR(10) | |
                    v_endereço | | CHR(10) | | v_estado | |
                    CHR(10) | | v_cep;
```

- CH(10) = ENTER

Exemplo de instruções:

```
v_nome  := LOWER(v_nome);
```

- Converte em letra minúscula

7.5.5. Conversão de tipos de dados

- As funções de conversão: TO_CHAR, TO_DATE e TO_NUMBER são utilizadas para conversão de dados compatíveis.

```
TO_CHAR (valor, fmt)
TO_DATE (valor, fmt)
TO_NUMBER (valor, fmt)
```

- fmt: é o modelo de formato usado para converter o valor

Exemplo de instruções:

```
v_data  := TO_DATE ('September 20, 2001', 'Month DD, YYYY');
```

7.5.6. Operadores em PL/SQL

Operador	**Operação**
**, NOT	Exponenciação, negação, lógica
+, -	Identidade, negação
*, /	Multiplicação, divisão
+, -, ||	Adição, subtração, concatenação
=, !=, <, >, <=, >=, IS NULL, LIKE, BETWEEN, IN	Comparação
AND	Conjunção
OR	Disjunção

7.6. INTERAGINDO COM O ORACLE SERVER

7.6.1. Instruções SQL em PL/SQL

- Extrair uma linha de dados de um banco de dados usando o comando SELECT;
- Fazer alterações nas linhas no banco de dados usando os comandos DML;
- Controlar uma transação com o comando COMMIT, ROLLBACK ou SAVEPOINT;
- Determinar o resultado do DML com cursores implícitos.

Comparando Os Tipos De Instruções SQL e PL/SQL

- Um bloco PL/SQL não é uma unidade de transação. Os comandos COMMIT, SAVEPOINT e ROLLBACK são independentes dos blocos, mas você pode emitir esses comandos em um bloco;
- O PL/SQL não suporta instruções em DDL como, por exemplo, CREATE TABLE, ALTER TABLE ou DROP TABLE;
- O PL/SQL não suporta instruções em DCL como, por exemplo, GRANT ou REVOKE.

7.6.2. Instrução SELECT em PL/SQL – A cláusula INTO é necessária.

Exemplo de instruções:

```
DECLARE
    v_codigo NUMBER(2);
    v_descricao  VARCHAR2(15);
BEGIN
    SELECT   codigo, descricao
    INTO       v_codigo, v_descricao
    FROM     depto
    WHERE    codigo=1;
END;
*/ devem retornar apenas um valor, caso contrário ocorrerá erros */
```

```
DECLARE
    v_codigo depto.codigo%TYPE;
    v_descricao  depto.descricao%TYPE;
BEGIN
    SELECT   codigo, descricao
    INTO       v_codigo, v_descricao
    FROM     depto
    WHERE    codigo=2;
END;
```

```
DECLARE
    v_soma_salario professor.salario%TYPE;
    v_codigo NUMBER NOT NULL := 2;
BEGIN
    SELECT   SUM (salario)
    INTO       v_soma_salario
    FROM     professor
    WHERE    codigo=v_codigo;
END;
*/ devem retornar apenas um valor, caso contrário ocorrerá erros */
```

7.6.3. Instrução INSERT em PL/SQL

Exemplo de instruções:

```
BEGIN
    INSERT INTO professor(codigo, nome, salario, nascimento)
    VALUES (codigo_sequence.NEXTVAL, 'Douglas', 1450, '10-DEC-00');
END;
```

7.6.4. Instrução UPDATE em PL/SQL

Exemplo de instruções:

```
DECLARE
    v_aumento  professor.salario%TYPE := 245;
BEGIN
    UPDATE   professor
    SET   salario = salario + v_aumento
    WHERE  codigo=10;
END;
```

7.6.5. Instrução DELETE em PL/SQL

Exemplo de instruções:

```
DECLARE
    v_codigo  professor.codigo%TYPE := 9;
BEGIN
    DELETE FROM   professor
    WHERE  codigo=v_codigo;
END;
```

7.7. CURSOR SQL

- Um cursor é uma área de trabalho SQL particular;
- Há dois tipos de cursores: implícitos e explícitos;
- O Oracle Server usa cursores implícitos para analisar e executar as instruções SQL;
- Os cursores explícitos são declarados especificamente pelo programador.
- Sempre que você emitir uma instrução SQL, o Oracle Server abrirá uma área de memória na qual o comando é analisado e executado. Essa área é chamada de cursor.

7.7.1 Atributos do cursor SQL

- Ao usar os atributos do cursor SQL, você poderá testar os resultados das instruções SQL.

SQL%ROWCOUNT	Número de linhas afetadas pela instrução SQL mais recente.
SQL%FOUND	Atributo booleano avaliado para TRUE se a instrução SQL mais recente afetar uma ou mais linhas.
SQL%NOTFOUND	Atributo booleano avaliado para TRUE se a instrução SQL mais recente não afetar uma ou mais linhas.
SQL%ISOPEN	Sempre é avaliado para FALSE porque o PL/SQL fecha os cursores implícitos imediatamente após a execução.

Excluir linhas que especificam um código de um professor a partir da tabela PROFESSOR. Imprimir o número de linhas excluídas.

Exemplo de instruções:

```
VARIABLE rows_deleted VARCHAR2(30)
DECLARE
   v_codigo  NUMBER := 4;
BEGIN
   DELETE FROM   professor
   WHERE  codigo=v_codigo;
   : rows_deleted := (SQL%ROWCOUNT | | ' rows deleted.');
END;
/
PRINT rows_deleted
```

7.8. CRIANDO ESTRUTURAS PARA CONTROLE

7.8.1 Controlando o fluxo de execução PL/SQL

Você pode alterar o fluxo lógico de instruções dentro do bloco PL/SQL com diversas estruturas para controle. Esta lição aborda os dois tipos de estruturas para controle do PL/SQL: construções condicionais com a instrução IF e estruturas para controle LOOP.

Existem três formatos de instruções IF:
- IF-THEN-END IF
- IF-THEN-ELSE-END IF
- IF-THEN-ELSIF-END IF

Instruções IF

```
IF condição THEN
   instruções
[ELSIF condição THEN
   instruções;]
[ELSE
    Instruções;]
END IF
```

- Condição : é uma expressão ou variável Booleana (TRUE, FALSE ou NULL).
- Esta associada a uma seqüência de instruções, que será executada somente se a expressão produzir TRUE.
- THEN : é uma cláusula que associa a expressão Booleana que a precede com a seqüência de instruções posteriores.
- instruções : pode ser uma ou mais instruções SQL ou PL/SQL.
- ELSIF : é uma palavra-chave que introduz uma expressão Booleana.
- ELSE : é uma palavra-chave que se for atingida pelo controle, executará a seqüência de instruções que segue a palavra-chave.

Instrução IF-THEN-END IF

Definir o código do gerente como 10 se o nome do funcionário for Alex.

Exemplo de instruções:

```
IF nome = 'Alex' THEN
   mensagem := 10;
END IF;
```

A estrutura da instrução IF do PL/SQL é semelhante à estrutura das instruções IF em outras linguagens procedurais. Ela permite que o PL/SQL execute ações de modo seletivo com base em condições.

Definir um aumento de 20% do salário atual se o nome for Alex.
Exemplo de instruções:

```
.....
IF nome = 'Alex' THEN
   v_novo_salario := salario * 1.20;
END IF;
.....
```

Instrução IF-THEN-ELSE-END IF

Definir um indicador aprovado quando a média for maior-igual a sete e reprovado caso contrário.

Exemplo de instruções:

```
.....
IF media >= 7 THEN
   v_mensagem := 'Aprovado';
ELSE
   v_mensagem := 'Reprovado';
END IF;
.....
```

Instrução IF-THEN-ELSIF-THEN-END IF

Definir aumento salarial de acordo com as faixas salariais: até 500.00 – 25%, de 500.01 até 1500.00 -15% e acima de 1500.01 – 10%.

Exemplo de instruções:

```
.....
IF salario < 500.00 THEN
    v_novo_salario := salario * 1.25;
ELSIF salario < 1000.00 THEN
   v_novo_salario := salario + (salario*15/100);
ELSE
   v_novo_salario := salario * 1.10;
END IF;
```

7.8.2. Controle iterativo: instruções LOOP

O PL/SQL oferece diversos recursos para estruturar loops para repetirem uma instrução ou seqüência de instruções várias vezes.

As construções em loop são o segundo tipo de estrutura para controle:

- Loop básico para fornecer ações repetitivas sem condições gerais;
- Loops FOR para fornecer controle iterativo para ações com base em uma contagem;
- Loops WHILE para fornecer controle iterativo para ações com base em uma condição;
- Instrução EXIT para terminar loops.

LOOP Básico

```
LOOP
   Instrução1;
.....
EXIT  [WHEN condição];
    Instruções;]
END LOOP;
```

Exemplo de instruções:

```
DECLARE
   v_codigo produto.codigo%TYPE := 21;
   v_contador NUMBER(2) := 1;
BEGIN
   LOOP
      INSERT INTO produto (codigo,numero)
      VALUES(v_codigo, v_contador);
      v_contador := v_contador + 1;
      EXIT WHEN v_contador > 10;
   END LOOP;
END;
```

FOR LOOP

```
FOR contador IN [REVERSE] lower_bound..upper_bound LOOP
   instrução1;
   instrução2;
   .....
END LOOP;
```

- Usar um FOR LOOP para desviar o teste para o número de iterações;
- Não declarar o contador; ele é declarado implicitamente como um inteiro. Os FOR LOOPs têm a mesma estrutura geral do loop básico. Além disso, eles têm uma instrução para controle no início da palavra-chave LOOP para determinar o número de iterações que o PL/SQL executa.
- contador : é um inteiro declarado implicitamente cujo valor aumenta ou diminui automaticamente em 1 cada iteração do loop até o limite superior ou inferior a ser alcançado;
- Instrução : pode ser uma ou mais instruções SQL ou PL/SQL;
- REVERSE : faz o contador decrescer a cada iteração a partir do limite superior até o limite inferior;
- lower_bound : especifica o limite inferior da faixa de valores do contador;
- upper_bound : especifica o limite superior da faixa de valores do contador.

Exemplo de instruções com inferior e superior fixos:

```
FOR i IN 1..3 LOOP
   instrução1;
END LOOP;
```

Exemplo de instruções com inferior e superior declarados como variáveis:

```
DECLARE
   v_inferior := 1;
   v_superior := 50;
BEGIN
   FOR i IN v_inferior..v_superior LOOP
      instrução1;
   END LOOP;
END;
```

Inserir os 5 novos produtos para o número do pedido 10.

Exemplo de instruções

```
DECLARE
   v_codigo pedido.codigo%TYPE := 10;
BEGIN
   FOR i IN 1..5 LOOP
      INSERT INTO pedido (pedido, produto)
      VALUES (v_codigo, i);
   END LOOP;
END;
```

WHILE LOOP

```
WHILE condição LOOP
   instrução1;
   instrução2;
   ....
```

- Usar um WHILE LOOP para repetir instruções enquanto um condição for TRUE;

Utiliza-se o WHILE LOOP para repetir uma seqüência de instruções até a condição para controle não ser mais TRUE. A condição é avaliada ao início de cada iteração. O LOOP terminará quando a condição for FALSE. Se a condição for FALSE no início do loop, nenhuma iteração futura será executada.

- condição : é uma expressão ou variável booleana;
- instrução : pode ser uma ou mais instruções SQL ou PL/SQL.

Exemplo de instruções:

```
ACCEPT v_pedido PROMPT 'Incluir o número do pedido:'
ACCEPT v_produto PROMPT 'Incluir o número de produtos no pedido:'
DECLARE
   v_contador  NUMBER(2)  := 1;
BEGIN
   WHILE v_contador <=  &v_produto LOOP
      INSERT INTO pedido (pedido, produto)
      VALUES (&v_pedido, v_contador);
      v_contador := v_contador + 1;
   END LOOP;
   COMMIT;
END;
```

7.9. CRIANDO CURSORES IMPLÍCITOS E EXPLÍCITOS

O Oracle Server usa áreas de trabalho chamadas áreas SQL particulares para executar instruções SQL e para armazenar informações de processamento. Você pode usar cursores do PL/SQL para nomear uma área SQL particular e acessar suas informações armazenadas. O cursor orienta todas as fases do processamento.

7.9.1. Cursores implícitos

Os cursores implícitos são declarados implicitamente pelo PL/SQL para todas as instruções DML e PL/SQL SELECT, incluindo consultas que retornam somente uma linha.

7.9.2. Cursores explícitos

São usados para consultas que retornam mais de uma linha. Os cursores explícitos são declarados e nomeados pelo programador e manipulados por meio de instruções específicas nas ações executáveis do bloco.

Funções do cursor explícito

Use cursores explícitos para processar individualmente cada linha retornada por uma instrução SELECT de várias linhas.

O conjunto de linhas retornado por uma consulta de várias linhas é chamado conjunto ativo. Seu tamanho é o número de linhas que atende aos critérios da pesquisa. O cursor explícito aponta para a linha atual do conjunto ativo. Isso permite que o programa processe as linhas uma de cada vez.

Um programa PL/SQL abre um cursor, processa linhas retornadas por uma consulta e, em seguida, fecha o cursor. O cursor marca a posição atual no conjunto ativo.

- Pode processar além da primeira linha retornada pela consulta, linha por linha;
- Controla que linha está sendo processada no momento;
- Permite que o programador controle as linhas manualmente no bloco PL/SQL.

Controlando cursores explícitos

1. Cria uma área SQL nomeada;
2. Identifica o conjunto ativo;
3. Carrega a linha atual para variáveis;
4. Testa para linhas existentes;
5. Retorna para FETCH se encontrar linhas;
6. Libera o conjunto ativo.

Controlando cursores explícitos usando quatro comandos

1. Declare o cursor nomeando-o e definindo a estrutura da consulta a ser executada dentro dele;
2. Abra o cursor. A instrução OPEN executa a consulta e vincula as variáveis que estiverem referenciadas. As linhas identificadas pela consulta são chamadas conjunto ativo e estão agora disponíveis para extração;
3. Extraia dados do cursor. Após cada extração você testa o cursor para qualquer linha existente. Se não existirem mais linhas para serem processadas, você precisará fechar o cursor;
4. Feche o cursor. A instrução CLOSE libera o conjunto ativo de linhas. Agora é possível reabrir o cursor e estabelecer um novo conjunto ativo.

Declarando o CURSOR

```
CURSOR cursor_name IS
        select  statement;
```

- Não inclua a cláusula INTO na declaração do cursor;
- Caso seja necessário o processamento de linhas em uma seqüência específica, use a cláusula ORDER BY na consulta.
- cursor_name : é um identificador do PL/SQL.
- select_statement : é uma instrução SELECT sem uma cláusula INTO.

Exemplos de instruções:

```
DECLARE
   v_matricula  professor.matricula%TYPE;
   v_nome  professor.nome%TYPE;
   CURSOR professor_cursor IS
SELECT matricula, nome
      FROM professor
   CURSOR depto_cursor IS
      SELECT * FROM depto
      WHERE codigo=10;
BEGIN
```

Abrir o cursor

```
OPEN cursor_name;
```

Extraindo dados do cursor

```
FETCH cursor_name INTO [[variavel1, variavel2, ...] | record_name];
```

Exemplos de instruções:

```
.....
OPEN  defined_cursor;
LOOP
   FETCH defined_cursor INTO defined_variables
   EXIT WHEN ...;
   .....
      -- Processo dos dados
   .....
END;
```

Recupere os primeiros 5 professores, um por um.

```
DECLARE
   v_codigo   professor.codigo%TYPE;
   v_nome  professor.nome%TYPE;
   CURSOR professor_cursor IS
SELECT codigo, nome
     FROM professor
BEGIN
    OPEN professor_cursor;
    FOR i IN 1..5 LOOP
        FETCH professor_cursor INTO v_codigo, v_nome;
        .....
    END LOOP;
    CLOSE  professor_cursor;
END;
```

Fechar o CURSOR

```
CLOSE cursor_name;
```

Obs: o parâmetro OPEN_CURSORS = número, define o número de cursores que pode ser abertos por um usuário.

Atributos do cursor explícito

Atributo	Tipo	Descrição
%ISOPEN	Booleano	Será avaliado para TRUE se o cursor estiver aberto.
%NOTFOUND	Booleano	Será avaliado para TRUE se a extração mais recente não retornar uma linha.
%FOUND	Booleano	Será avaliado para TRUE se a extração mais recente não retornar uma linha; complemento de %NOTFOUND
%ROWCOUNT	Número	Será avaliado para o número total de linhas retornadas até o momento.

%ISOPEN

```
IF NOT professor_cursor%ISOPEN THEN
    OPEN professor_cursor;
END IF;
LOOP
    FETCH professor_cursor ...
```

%NOTFOUND

Recupere os primeiros 5 professores, um por um.

```
DECLARE
   v_codigo   professor.codigo%TYPE;
   v_nome  professor.nome%TYPE;
   CURSOR professor_cursor IS
SELECT codigo, nome
     FROM professor
BEGIN
    OPEN professor_cursor;
    LOOP
        FETCH professor_cursor INTO v_codigo, v_nome;
        EXIT WHEN professor_cursor%ROWCOUNT > 5
                        OR professor_cursor%NOTFOUND;
        .....
    END LOOP;
    CLOSE  professor_cursor;
END;
```

7.10. TRATAMENTO DE EXCEÇÕES

Exceção é um identificador em PL/SQL que é criado durante a execução. Ela é criada quando ocorre um erro do Oracle ou quando você a cria explicitamente. Ela é tratada capturando-a com um *handler* ou propagada para o ambiente de chamada.

Tipos de exceção

- Predefinida pelo Oracle Server (20 erros)
- Não predefinida pelo Oracle Server
- Definida pelo usuário

Exceções predefinidas

Nome da Exceção	Número	Descrição
ACCESS_INTO_NULL	ORA-06530	Tentativa de atribuir valores aos atributos de um objeto não inicializado
COLLECTION_IS_NULL	ORA-06531	Tentativa de aplicação de métodos de conjunto diferentes de EXISTS para um varray ou tabela aninhada não inicializada

CURSOR_ALREADY_OPEN	ORA-06511	Tentativa de abertura de um cursor já aberto
DUP_VAL_ON_INDEX	ORA-00001	Tentativa de inserção de um valor duplicado
INVALID_CURSOR	ORA-01001	Ocorreu operação ilegal de cursor
INVALID_NUMBER	ORA-01722	Falha da conversão de string de caracteres para número
LOGIN_DENIED	ORA-01017	Estabelecendo login com o Oracle com um nome de usuário ou senha inválida
NO_DATA_FOUND	ORA-01403	SELECT de linha única não retornou dados
NOT_LOGGED_ON	ORA-01012	O programa PL/SQL emite uma chamada de banco de dados sem estar conectado ao Oracle
PROGRAM_ERROR	ORA-06501	O código PL/SQL tem um problema interno
ROWTYPE_MISMATCH	ORA-06504	Variável de cursor de host e variável de cursor PL/SQL envolvidas em uma atribuição tem tipos de retorno incompatíveis
STORAGE_ERROR	ORA-06500	O PL/SQL esgotou a memória ou a memória está corrompida
SUBSCRIPT_BEYOND_COUNT	ORA-06533	Feita referência ao elemento de varray ou tabela aninhada usando um número de índice maior do que o número de elementos no conjunto
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	Feita referência a um elemento
TIMEOUT_ON_RESOURCE	ORA-00051	Ocorreu timeout enquanto o Oracle está aguardando por um recurso
TOO_MANY_ROWS	ORA-01422	SELECT de uma única linha retornou mais de uma linha
VALUE_ERROR	ORA-06502	Ocorreu erro aritmético, de conversão, truncamento ou restrição de tamanho
ZERO_DIVIDE	0RA-01476	Tentativa de divisão por zero

```
BEGIN
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        statement1;
        statement2;
    WHEN TOO_MANY_ROWS THEN
        statement1;
    WHEN DUP_VAL_ON_INDEX THEN
        statement1;
        statement2;
        statement3;
END;
```

Funções para capturar exceções

```
DECLARE
 v_error_code NUMBER;
  v_error_message VARCHAR(255);
BEGIN
  ....
EXCEPTION
  ....
    WHEN OTHERS THEN
      ROLLBACK;
      v_error_code := SQLCODE;
      v_error_message := SQLERRM;
      INSERT INTO errors VALUES(v_error_code, v_error_message);
END;
```

7.11. PROCEDIMENTO ARMAZENADO, FUNÇÃO, GATILHO e PACOTE

Procedimentos Armazenados, Funções, Gatilhos e Pacotes são quatro recursos de grande utilidade em Sistemas de Bancos de Dados (KROENKE, 1999). Eles também podem cooperar com as ferramentas de desenvolvimento de aplicação de usuários finais, adicionando recursos de processamento. Além disso, esses recursos também possibilitam declarar variáveis com base em atributos de conjuntos de entidades do próprio banco de dados, levando o código a se tornar independente de modificações físicas ocorridas com os atributos.

Com a utilização desses recursos, consegue-se uma redução no número de linhas de códigos a serem programadas por ferramentas utilizadas no desenvolvimento das aplicações dos usuários finais. Inúmeras das regras de negócio, de restrições e de integridade, podem ser programadas diretamente no Banco de Dados, permitindo, às aplicações finais, conter apenas instruções básicas, sem o objetivo de garantir a integridade, toda vez que uma rotina for programada. Isso possibilita que as aplicações finais tornem-se mais leves, além de disporem de um código mais simples.

7.11.1. Esquema do Banco de Dados

Para melhor andamento da disciplina, será utilizada a seguinte estrutura de dados:

Tabela de Funcionários (TABFUN)

CAMPO	TIPO	RESTRIÇÃO	REFERENCES
MATRICULA	Number(5)	Primary Key	
NOME	Varchar(35)	Not Null	
SALARIO	Number(9,2)	Check > 0	
ADMISSAO	Date	Not Null	
CARGO	Number(2)	Foreign Key	TABCAR

Tabela de Dependentes (TABDEP)

CAMPO	TIPO	RESTRIÇÃO	REFERENCES
MATRICULA	Number(5)	Primary Key/Foreign Key	TABFUN
SEQUENCIA	Number(2)	Primary Key	
NOME	Varchar(35)	Not Null	
NASCIMENTO	Date	Not Null	

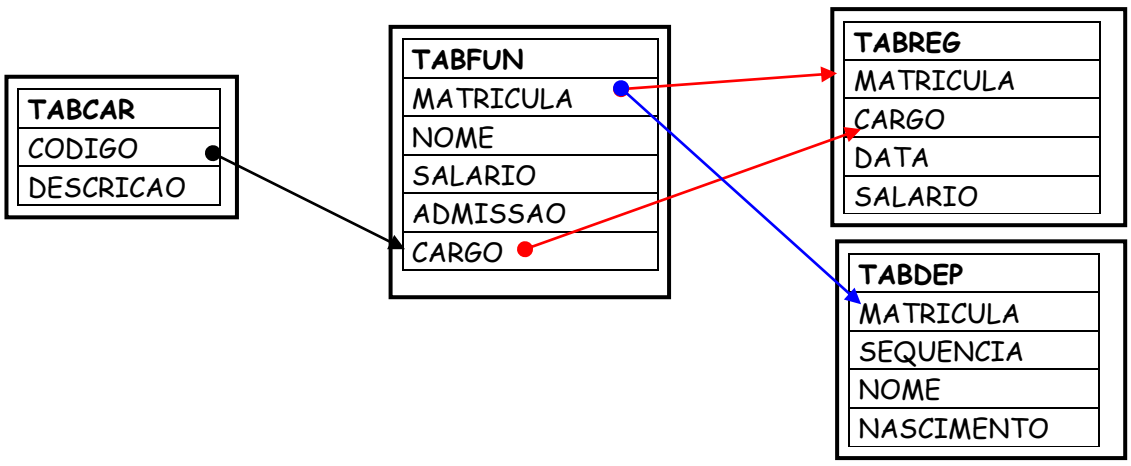
Tabela de Cargos (TABCAR)

CAMPO	TIPO	RESTRIÇÃO	REFERENCES
CODIGO	Number(5)	Primary Key	
DESCRICAO	Varchar(30)	Not Null	

Tabela de Registros de Alteração de Cargo (TABREG)

CAMPO	TIPO	RESTRIÇÃO	REFERENCES
MATRICULA	Number(5)	Primary Key/Foreign Key	TABFUN
CARGO	Number(2)	Primary Key/Foreign Key	TABCAR
DATA	Date	Primary Key	
SALARIO	Number(9,2)		

Movimento Relacional



Criação das tabelas

Tabela de Cargos

```
CREATE TABLE tabcar
(codigo NUMBER(2),
descricao VARCHAR2(35) NOT NULL,
CONSTRAINT tabcar_codigo_pk PRIMARY KEY (codigo));
```

Tabela de Funcionários

```
CREATE TABLE tabfun
(matricula NUMBER(5),
nome VARCHAR2(35) NOT NULL,
salario NUMBER(9,2) CHECK (salario > 0),
admissao DATE NOT NULL,
cargo NUMBER(2),
CONSTRAINT tabfun_matricula_pk PRIMARY KEY (matricula),
CONSTRAINT tabfun_cargo_fk FOREIGN KEY (cargo)
                                    REFERENCES tabcar(codigo));
```

Tabela de Dependentes

```
CREATE TABLE tabdep
(matricula NUMBER(5),
sequencia NUMBER(2),
nome VARCHAR2(35) NOT NULL,
nascimento DATE NOT NULL,
CONSTRAINT tabdep_matricula_pk PRIMARY KEY (matricula,sequencia),
CONSTRAINT tabdep_matricula_fk FOREIGN KEY (matricula)
                                    REFERENCES tabfun(matricula));
```

Tabela de Registros

```
CREATE TABLE tabreg
(matricula NUMBER(5),
cargo NUMBER(2),
data DATE,
salario NUMBER(9,2),
CONSTRAINT tabreg_pk PRIMARY KEY (matricula, cargo, data));
```

Carga na base de dados

```
INSERT INTO tabcar VALUES (1,'Coordenador Técnico');
INSERT INTO tabcar VALUES (2,'Jogador de Futebol');
INSERT INTO tabcar VALUES (3,'Técnico de Futebol');
```

```
INSERT INTO tabfun VALUES (10,'Pareira',90000,'01/01/2002',3);
INSERT INTO tabfun VALUES (11,'Ricardinho',80000,'02/02/2000',2);
INSERT INTO tabfun VALUES (12,'Vampeta',75000,'10/10/2002',2);
INSERT INTO tabfun VALUES (13,'Zagalo',100000,'12/10/2002',1);
INSERT INTO tabfun VALUES (14,'Felipão',200000,'04/05/2001',3);
INSERT INTO tabfun VALUES (15,'Falcão',300000,'10/11/2002',1);
```

```
INSERT INTO tabdep VALUES (10,1,'Pareira Júnior','01/01/1998');
INSERT INTO tabdep VALUES (11,1,'Ricardinho Júnior','02/02/1999');
INSERT INTO tabdep VALUES (12,1,'Vampeta Júnior','10/10/1997');
INSERT INTO tabdep VALUES (13,1,'Zagalo Júnior','12/10/2000');
INSERT INTO tabdep VALUES (14,1,'Felipão Júnior','04/05/1996');
INSERT INTO tabdep VALUES (15,1,'Falcão Júnior','10/11/1995');
INSERT INTO tabdep VALUES (10,2,'Pareira Filha','01/01/2003');
```

```
INSERT INTO tabreg VALUES (10,3,'01/01/2002',90000);
INSERT INTO tabreg VALUES (11,2,'03/02/2000',80000);
INSERT INTO tabreg VALUES (12,2,'20/01/2002',75000);
INSERT INTO tabreg VALUES (13,1,'01/01/2002',100000);
INSERT INTO tabreg VALUES (14,3,'10/01/2000',200000);
INSERT INTO tabreg VALUES (15,1,'03/01/2002',300000);
```

Obs: se instalação em Inglês, usar o formato dd/mm/aaaa para os campos tipo date.

7.11.2. Procedimento Armazenado (Stored Procedure)

Segundo Price (2009, p.386), um procedimento contém um grupo de instruções SQL e PL/SQL. Os procedimentos permitem centralizar sua lógica do negócio no banco de dados e podem ser usados por qualquer programa que acesse o banco de dados.

Costa (2006, p.158) salienta que procedimentos armazenados são procedimentos análogos aos existentes em linguagens de programação tradicionais, mas que terão seu código fonte armazenado no servidor de banco de dados, o qual é capaz de compilá-lo e executá-lo.

Segundo Fanderuff (2000, p.151), “os procedimentos armazenados são ‘subprogramas’ que têm por objetivo executar uma ação específica, utilizando-se das instruções de processamento de dados da Linguagem SQL”.

Um procedimento armazenado consiste em um bloco de instruções nomeado que executa uma ação. Ele pode ser armazenado no banco de dados, como um objeto de banco de dados, para execução repetida de qualquer ambiente de chamada (KOCHHAR; KRAMER, 1998).

7.11.2.1. Objetivos

- Descrever a utilidade dos procedimentos;
- Criar procedimentos;
- Chamar um procedimento;
- Ver erros em um procedimento;
- Remover um procedimento;
- Obter informações sobre um procedimento.

7.11.2.2. Visão geral dos procedimentos

- Um procedimento é um bloco PL/SQL nomeado que executa uma ação;
- Um procedimento pode ser armazenado no banco de dados, como um objeto de banco de dados, para execução repetida.

7.11.2.3. Sintaxe para criação de procedimentos

```
CREATE [OR REPLACE] PROCEDURE nome_procedimento
  (parâmetro1 [modo1] tipodedado1,
  (parâmetro2 [modo2] tipodedado2,
   . . .)
IS | AS
PL/SQL Block;
```

Parâmetro: é o nome de uma variável PL/SQL passada para o procedimento;
Modo: identifica o tipo de parâmetro (IN/OUT/IN OUT)

IN: (entrada) passa o valor do ambiente de chamada para o procedimento.

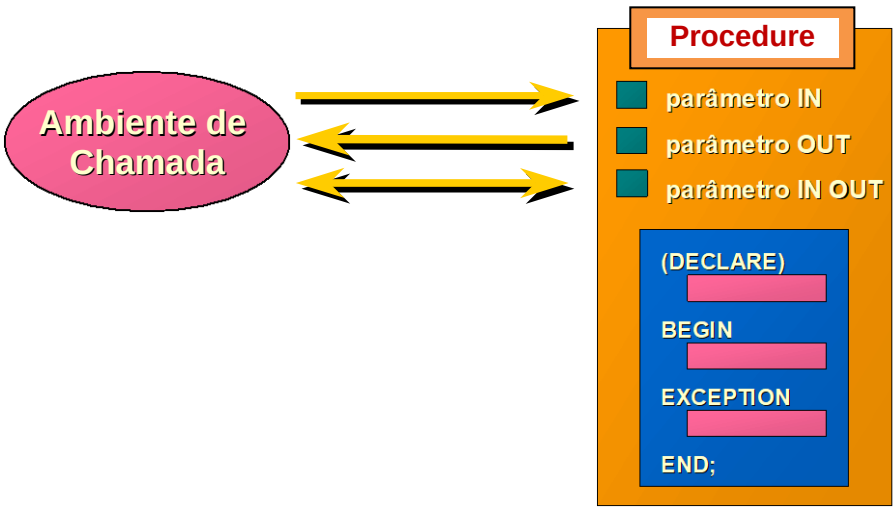
OUT: (saída) retorna um valor do procedimento para o ambiente de chamada.

IN OUT: (entrada saída) passa um valor do ambiente de chamada para o procedimento, e o procedimento retorna um valor para o ambiente de chamada.

Tipodedado: tipo do dado (datatype)

IS | *AS*: essas cláusulas são equivalentes, pode-se utilizar tanto uma quanto a outra.

PL/SQL Block: é o corpo do procedimento que define as ações que serão executadas quando o procedimento for executado.



7.11.2.4. Criando um procedimento usando o Oracle SQL Developer

Exemplo1 (IN): Criando um procedimento para armazenar informações de novos funcionários.

```
CREATE OR REPLACE PROCEDURE novos_funcionarios
(v_matricula IN tabfun.matricula%TYPE,
 v_nome IN tabfun.nome%TYPE,
 v_salario IN tabfun.salario%TYPE,
 v_admissao IN tabfun.admissao%TYPE,
 v_cargo IN tabfun.cargo%TYPE)
 IS
 BEGIN
   INSERT INTO tabfun VALUES (v_matricula, v_nome, v_salario, v_admissao,
    v_cargo);
   COMMIT;
END novos_funcionarios;
/
```

Executando o procedimento direto.

```
EXECUTE novos_funcionarios (20,'Luizão',100000, '01/01/2001',2);
```

Executando o procedimento novos_funcionarios via bloco PL/SQL

```
BEGIN
   novos_funcionarios (21,'Dida',85000,'01/01/2002',2);
END;
/
```

Consulta a tabela de funcionários

```
SELECT * FROM tabfun;
```

Caso você queira usar uma codificação automática para a matricula (auto-numeração), crie uma **seqüência** e troque a variável v_matricula por essa seqüência.

Exemplo: Criando uma seqüência

```
CREATE SEQUENCE num_fun START WITH 40;
```

Trocando a variável v_matricula pela seqüência.

1º - Não esqueça de tirar a linha que define a variável v_matricula.
2º - Faça a seguinte mudança.

```
.....
VALUES (num_fun.NEXTVAL, v_nome, v_salario, v_admissão, v_cargo);
.....
```

Exemplo2 (IN): Criando um procedimento para reajustar o salário dos funcionários.

```
CREATE OR REPLACE PROCEDURE reajuste_salario
 (v_matricula IN tabfun.matricula%TYPE,
  v_reajuste IN tabfun.salario%TYPE)
 IS
 BEGIN
   IF v_reajuste <= 100 THEN
      UPDATE tabfun
      SET salario = salario + (salario*v_reajuste/100)
      WHERE matricula = v_matricula;
   ELSE
      DBMS_OUTPUT.PUT_LINE('Valor Inválido');
   END IF;
 END reajuste_salario;
/
```

Executando o procedimento direto

```
EXECUTE reajuste_salario (10,10);
```

```
EXECUTE reajuste_salario (11,130);
```

Exemplo3 (OUT): Criando um procedimento para consultar funcionários.

```
CREATE OR REPLACE PROCEDURE consulta_fun
 (v_matricula IN tabfun.matricula%TYPE,
  v_nome OUT tabfun.nome%TYPE,
  v_salario OUT tabfun.salario%TYPE,
  v_admissao OUT tabfun.admissao%TYPE)
  IS
  BEGIN
    SELECT nome, salario, admissao
    INTO v_nome, v_salario, v_admissao
    FROM tabfun
    WHERE matricula = v_matricula;
  END consulta_fun;
/
```

Salvar o fonte (procedimento), e em seguida cria-lo.

```
SAVE <caminho> nome_arq.sql
START  <caminho> nome_arq.sql
```

Criando variáveis globais (de host) para utilizá-las na execução do procedimento

```
VARIABLE g_nome VARCHAR2(35)
VARIABLE g_salario NUMBER
VARIABLE g_admissao VARCHAR2(10)
```

Executando o procedimento consulta_fun e mostrando os valores das variáveis

```
EXECUTE consulta_fun (11, :g_nome, :g_salario, :g_admissao)
PRINT g_nome
PRINT g_salario
PRINT g_admissao     ou
PRINT g_nome  g_salario  g_admissao
```

Exemplo4 (IN OUT): Criando um procedimento para formatar telefone.

```
CREATE OR REPLACE PROCEDURE telefone
(v_fone IN OUT VARCHAR2)
IS
BEGIN
  v_fone :='('||substr(v_fone,1,2)||')'||substr(v_fone,3,4)||'-'||substr(v_fone,7,10);
  DBMS_OUTPUT.PUT_LINE(v_fone);
END telefone;
/
```

```
SET SERVEROUTPUT ON
```

Executando o procedimento telefone via bloco PL/SQL

```
DECLARE
   vfone VARCHAR2(17);
   numero VARCHAR(14);
BEGIN
   vfone := ('&numero');
   telefone(vfone);
   DBMS_OUTPUT.PUT_LINE(vfone);
END;
/
```

7.11.2.5. Removendo um procedimento usando o SQL*Plus

```
DROP PROCEDURE nome_procedimento;
```

Exemplo1: Eliminar o procedimento consulta de funcionários

```
DROP PROCEDURE consulta_fun;
```

7.11.2.6. Obtendo informações sobre um procedimento

A tabela USER_PROCEDURES é utilizada para armazenar os procedimentos criados. A seguir é apresentada sua estrutura.

COLUNA	TIPO	DESCRIÇÃO
OBJECT_NAME	VARCHAR2(30)	Nome do objeto
PROCEDURE_NAME	VARCHAR2(30)	Nome do procedimento
AGGREGATE	VARCHAR2(3)	Se é um procedimento agregado
IMPLTYPEOWNER	VARCHAR2(30)	O proprietário do tipo
IMPLTYPENAME	VARCHAR2(30)	O nome do tipo
PARALLEL	VARCHAR2(3)	Se é ativada para consultas paralelas
INTERFACE	VARCHAR2(3)	

7.11.2.7. Sumário

- Um procedimento é um bloco PL/SQL nomeado que executa uma ação;
- Use parâmetros para especificar dados do ambiente de chamada para o procedimento;
- Os procedimentos podem ser chamados de qualquer ferramenta ou linguagem que suportem PL/SQL;
- Os procedimentos podem servir como blocos de construção de uma aplicação.

Exercícios:

1. Faça um procedimento para excluir funcionários.

2. Faça um procedimento para inserir dependentes.

7.11.3. Funções (FUNCTION)

Segundo Price (2009, p.391), uma função é semelhante a um procedimento, exceto que uma função deve retornar um valor, juntos, os procedimentos armazenados e as funções às vezes são referidos como subprogramas armazenados já que são pequenos programas.

7.11.3.1. Objetivos

- Descrever os usos das funções;
- Criar funções;
- Chamar uma função;
- Remover uma função;
- Diferenciar entre um procedimento e uma função.

7.11.3.2. Visão geral das funções

- Uma função é um bloco PL/SQL nomeado que retorna um valor;
- Uma função pode ser armazenada no banco de dados, como um objeto de banco de dados, para execução repetida;
- Uma função pode ser chamada como parte de uma expressão.

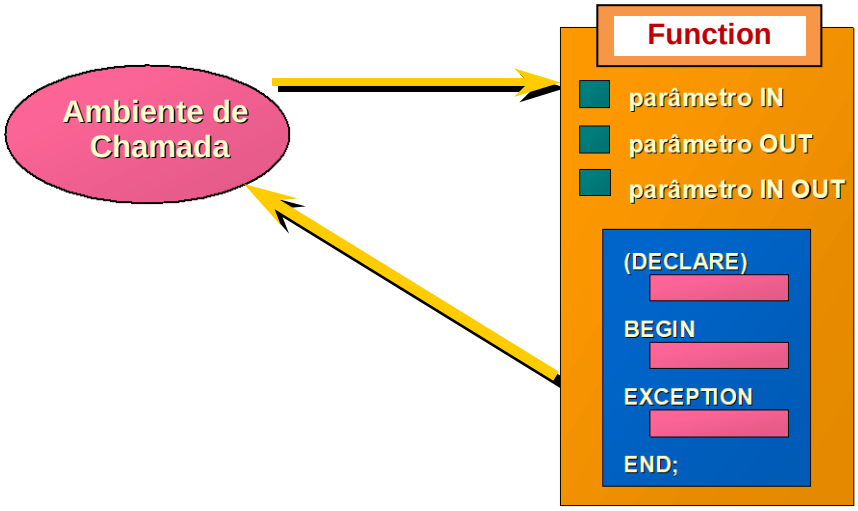
7.11.3.3. Sintaxe para criação de funções

```
CREATE [OR REPLACE] FUNCTION nome_função
 (parâmetro1 [modo1] tipodedados1,
  parâmetro2 [modo2] tipodedados2,
 . . .)
RETURN tipodedados
IS | AS
PL/SQL Block;
```

Parâmetro: é o nome de uma variável PL/SQL passada para a função;
Modo: identifica o tipo de parâmetro (IN)
IN: (entrada) passa o valor do ambiente de chamada para a função.
Tipodedado: tipo do dado (datatype)
IS | *AS*: essas cláusulas são equivalentes, pode-se utilizar tanto uma quanto a outra.

PL/SQL Block: é o corpo da função que define as ações que serão executadas quando a função for executada.
OBS: *o bloco PL/SQL deve ter pelo menos uma instrução RETURN.*

- RETURN: retorna um valor da função para o ambiente de chamada.



7.11.3.4. Gerenciar exceções em tempo de execução

A função RAISE_APPLICATION_ERROR nos possibilita gerenciar qualquer tipo de exceção em tempo de execução permitindo propagar para o ambiente de chamada ou receber ações quando essas ocorrem.

RAISE_APPLICATION_ERROR (numero_erro, texto_erro)

- numero_erro: é o número do erro definido pelo usuário. Deve estar entre –20999 a -20000
- texto_erro: é a mensagem definida pelo usuário.

7.11.3.5. Criando uma função

- Digite o texto para a instrução CREATE FUNCTION em um editor de texto;
- Use SHOW ERRORS para consultar os erros de compilação;
- Quando a compilação for bem-sucedida, a função estará pronta para execução.

Exemplo1: Criando uma função para consultar o salário dos funcionários.

```
CREATE OR REPLACE FUNCTION consulta_salario
  (v_matricula IN tabfun.matricula%TYPE)
  RETURN NUMBER
IS
   v_salario  tabfun.salario%TYPE := 0;
BEGIN
   SELECT salario
   INTO v_salario
   FROM tabfun
   WHERE tabfun.matricula = v_matricula;
   RETURN (v_salario);
END consulta_salario;
/
```

7.11.3.6. Executando funções

- Chame uma função como parte de uma expressão PL/SQL;
- Crie uma variável de host para armazenar o valor retornado;
- Execute a função. A variável de host será preenchida com o valor RETURN.

Criando variáveis de host

```
VARIABLE g_salario NUMBER
```

Executando a função

```
EXECUTE :g_salario := consulta_salario (11);   ou
ACCEPT num_matricula PROMPT 'Digite o número da matricula:'
EXECUTE :g_salario := consulta_salario (&num_matricula);
```

Consultando o valor

```
PRINT g_salario
```

Executando uma função de um bloco PL/SQL

Crie um bloco PL/SQL para a execução:

```
SET SERVEROUTPUT ON
```

```
DECLARE
     v_salario tabfun.salario%TYPE;
BEGIN
    v_salario := consulta_salario (12);
    DBMS_OUTPUT.PUT_LINE (v_salario);
END;
/
```

Exemplo2: Criar uma função para consultar quantas vezes o funcionário mudou de cargo ou de salário.

```
CREATE OR REPLACE FUNCTION mudanca_salario
  (v_matricula IN tabreg.matricula%TYPE)
  RETURN NUMBER
IS
   v_qtde  NUMBER(5) := 0;
BEGIN
   SELECT count(cargo)
   INTO v_qtde
   FROM tabreg
   WHERE tabreg.matricula = v_matricula
   GROUP BY cargo;
   RETURN (v_qtde);
END mudanca_salario;
/
```

Executando a função mudanca_salario via bloco PL/SQL

```
DECLARE
    v_quant NUMBER(5);
BEGIN
    v_quant := mudanca_salario (12);
    DBMS_OUTPUT.PUT_LINE (v_quant);
END;
/
```

7.11.3.7. Removendo funções

Sintaxe:

```
DROP FUNCTION nome_função;
```

Exemplo1:

```
DROP FUNCTION consulta_salario;
```

7.11.3.8. Obtendo informações sobre uma função

A tabela USER_PROCEDURES também armazena as funções criadas. A seguir é apresentada sua estrutura.

COLUNA	TIPO	DESCRIÇÃO
OBJECT_NAME	VARCHAR2(30)	Nome do objeto
PROCEDURE_NAME	VARCHAR2(30)	Nome da função
AGGREGATE	VARCHAR2(3)	Se é uma função agregada
IMPLTYPEOWNER	VARCHAR2(30)	O proprietário do tipo
IMPLTYPENAME	VARCHAR2(30)	O nome do tipo
PARALLEL	VARCHAR2(3)	Se é ativada para consultas paralelas
INTERFACE	VARCHAR2(3)	

7.11.3.9. Observações

- Uma função definida pelo usuário obtém apenas parâmetros IN, e não OUT ou IN OUT;
- Os tipos de dados devem ser tipos de dados SQL válidos, CHAR, DATE ou NUMBER;
- Os tipos de dados não podem ser tipos PL/SQL como BOOLEAN, RECORD ou TABLE;
- Comandos INSERT, UPDATE ou DELETE não são permitidos.

7.11.3.10. Sumário

- Uma função é um bloco PL/SQL nomeado que deve retornar um valor;
- Uma função é chamada como parte de uma expressão;
- Uma função armazenada pode ser chamada em instruções SQL.

Exercícios:

1) Faça uma função para consultar o nome do cargo.

2) Faça uma função para consultar os dependentes por funcionário.

7.11.4. Procedimentos e funções

7.11.4.1. Comparando procedimentos e funções

Procedimentos	**Funções**
Executar como uma instrução PL/SQL	Chamar como parte de uma expressão
Tipo de dados sem RETURN	Deve conter um tipo de dados RETURN
Pode retornar nenhum, um ou muitos valores	Deve retornar um valor único

7.10.4.2. Benefícios de funções e procedimentos

- Desempenho melhorado;
    - Reduzir o número de chamadas ao banco de dados e diminuir o tráfico na rede;
    - Compartilhar execuções SQL por vários usuários.
- Manutenção melhorada;
    - Modificar rotinas on-line sem interferir com outros usuários;
    - Modificar uma rotina que afetará várias aplicações.
- Segurança de dados e Integridade melhorada.
    - Controle sobre acessos indiretos aos objetos de banco de dados executados por funcionários sem privilégios;
    - Assegurar que ações relacionadas sejam executadas conjuntamente.

7.1.4.3. Gerenciando funções e procedimentos

A tabela USER_OBJECTS armazena os objetos criados junto ao banco de dados, tais como procedimentos, funções, etc. A seguir, é apresentada a estrutura da tabela USER_OBJECTS.

COLUNA	TIPO	DESCRIÇÃO
OBJECT_NAME	VARCHAR2(128)	Nome do objeto
OBJECT_ID	NUMBER	Identificação do objeto
OBJECT_TYPE	VARCHAR2(19)	Tipo do objeto
CREATED	DATE	Data de criação do objeto
LAST_DDL_TIME	DATE	Última modificação do objeto
TIMESTAMP	VARCHAR2(19)	Data e hora do objeto
STATUS	VARCHAR2(7)	Se é Válido ou inválido
TEMPORARY	VARCHAR2(1)	Se é temporário
SECONDARY	VARCHAR2(1)	Se é secundário

```
SELECT DISTINCT object_type FROM user_objects;
```

```
SELECT object_name, object_type, created
FROM user_objects
WHERE object_type IN ('PROCEDURE','FUNCTION');
```

A tabela USER_SOURCE armazena o código fonte referente aos objetos criados. A seguir, é apresentada a estrutura da tabela USER_SOURCE.

COLUNA	TIPO	DESCRIÇÃO
NAME	VARCHAR2(30)	Nome do objeto
TYPE	VARCHAR2(12)	Tipo do objeto
LINE	NUMBER	Número da linha do código
TEXT	VARCHAR2(4000)	Código fonte

Consultando a tabela USER_SOURCE

```
SELECT * FROM user_source WHERE name='NOME_OBJETO'
```

Listar somente o código fonte

```
SELECT text FROM user_source
WHERE name='CONSULTA_SALARIO'
ORDER BY line;
```

Consultar tabelas DICT e USER_TABLES

```
SELECT table_name FROM dict;

SELECT table_name FROM user_tables;
```

Descrever as estruturas de armazenamento dos objetos

```
DESC user_objects
```

Descrever as estruturas das tabelas

```
DESC user_tables
```

Descrever as estruturas de funções, procedimentos e gatilhos

```
DESC user_source
```

Consultar erros de compilação

```
SHOW ERRORS

DESC user_errors
```

7.11.5. Gatilhos (TRIGGERS)

Segundo Price (2009, p.397) um gatilho é um procedimento executado (ou disparado) automaticamente pelo banco de dados, quando uma instrução DML (INSERT, UPDATE ou DELETE) especificada é executada em determinada tabela do banco de dados. Os gatilhos são úteis para fazer coisas como a auditoria avançada das alterações feitas nos valores de coluna em uma tabela.

Um gatilho consiste em um bloco de instruções ou subprograma acionado, de forma implícita, pela aplicação do usuário final, por meio de ferramentas de administração de bancos de dados ou pelo próprio banco de dados, sempre que ocorrer um determinado evento de inclusão, atualização ou exclusão de dados, associado a um conjunto de entidades, de acordo com a definição de um instante de tempo (KOCHHAR; KRAMER, 1998; RAMALHO, 2005).

Na caracterização de um gatilho, há três fatores a serem considerados (DATE, 2003):

a) especificar um evento que servirá como disparo do gatilho;
b) especificar as condições sob as quais o gatilho deve ser executado;
c) especificar as ações que serão tomadas quando um gatilho for disparado.

A especificação do evento consiste em delimitar o "evento" ou instrução a ocorrer no Banco de Dados Operacional para que a "ação", de acordo com as "condições" avaliadas, seja executada.

Analisando o segundo fator, para que um gatilho seja programado, é necessário levantar as regras de negócio, isto é, as necessidades e "condições" as quais o gatilho deverá atender a fim de respeitar as restrições de integridade, garantindo assim a consistência dos dados.

O terceiro fator contém a definição de quais e quantas "ações" de processamento deverão ser executadas com base nos conjuntos de entidades dos Bancos de Dados Operacionais em relação ao Banco de Dados Analítico-Temporal, quando um gatilho for acionado.

A combinação das três exigências (evento, condição e ação) leva os gatilhos a serem conhecidos como regras ECA, de evento-condição-ação (COSTA, 2006; DATE, 2003).

7.11.5.1. Objetivos

- Descrever gatilhos de banco de dados e suas utilizações;
- Criar gatilhos de banco de dados;
- Descrever regras para disparo de gatilhos de banco de dados;
- Remover gatilhos de banco de dados.

7.11.5.2. Visão geral dos gatilhos

- Um gatilho é um bloco PL/SQL executado de forma implícita sempre que ocorre um determinado evento;
- Ele pode ser tanto um gatilho de banco de dados ou um gatilho de aplicação (Form, Report, Navigator). Os gatilhos de aplicação não são totalmente iguais aos gatilhos de banco.

7.11.5.3. Projetando gatilhos: Diretrizes

- Projetar gatilhos para:
  - Executar ações relacionadas;
  - Centralizar operações globais.
- Não projetar gatilhos:
  - Onde a funcionalidade já exista;
  - Que dupliquem outros gatilhos.

7.11.5.4. Tempo do gatilho

- Para tabela: BEFORE, AFTER, INSTEAD OF e FOR
  - BEFORE: executa o corpo do gatilho antes do evento DML de acionamento em uma tabela;
  - AFTER: executa o corpo do gatilho após o evento DML de acionamento em uma tabela;
  - INSTEAD OF: executa o corpo do gatilho em vez do evento de disparo;
  - FOR: que é novidade do Oracle 11g, permite criar um gatilho composto, consistindo em até quatro seções no corpo do gatilho.

7.11.5.5. Disparo de outro gatilho

- FOLLOWS e PRECEDES (novo no Oracle 11g)
  - FOLLOWS: dispara o gatilho depois do disparo do gatilho especificado em *esquema.outro_gatilho*;
  - PRECEDES: antecede a execução do gatilho especificado em *esquema.outro_gatilho*.

7.11.5.6. Ativar e desativar um gatilho

- ENABLE e DISABLE (novo no Oracle 11g)
  - ENABLE: inicia o gatilho ativado;
  - DISABLE: inicia o gatilho desativado;
  - Usando a instrução ALTER TRIGGER nome_gatilho [ENABLE ou DISABLE] um gatilho pode ser ativado ou desativado a qualquer momento.

7.11.5.7. Evento de acionamento

- **INSERT**
- **UPDATE**
- **DELETE**

É por meio dessas instruções que os gatilhos são executados.

7.11.5.8. Tipo de gatilho

Os gatilhos são divididos em dois tipos, no qual defini quantas vezes o corpo do gatilho deverá ser executado quando ocorre um evento de acionamento.

- Instrução: o corpo do gatilho é executado uma vez para o evento de acionamento. Esse é o default.
- Linha: o corpo do gatilho é executado uma vez para cada linha afetada pelo evento de acionamento. Além disso, o gatilho de linha tem acesso aos valores de coluna antigos (OLD) e novos (NEW) quando é disparado como resultado de uma instrução UPDATE nessa coluna, e também o disparo pode ser limitado com uma condição de gatilho.

7.11.5.9. Corpo do gatilho

O corpo do gatilho é um bloco PL/SQL ou uma chamada para um procedimento.

7.11.5.10. Sintaxe para criação de gatilhos de instrução

```
CREATE [OR REPLACE] TRIGGER nome_gatilho
 {tempo} evento1 [OR evento2 OR evento3]
 ON nome_tabela
 . . .)
 corpo_gatilho
```

Exemplo1: Criando um gatilho de instrução para verificar alteração de dados fora de horário.

```
CREATE OR REPLACE TRIGGER fora_horario
BEFORE UPDATE ON tabfun
BEGIN
 IF (TO_NUMBER (TO_CHAR(SYSDATE,'HH24')) NOT BETWEEN 8 AND 12)
  THEN
     RAISE_APPLICATION_ERROR(-20001,'Atenção – Operação Fora de Horário');
  END IF;
END fora_horario;
/
```

Testando o gatilho:

```
UPDATE tabfun
SET salario = salario*1.10
WHERE matricula = 10;
```

7.11.5.11. Sintaxe para criação de gatilhos de linhas

```
CREATE [OR REPLACE] TRIGGER nome_gatilho
{BEFORE | AFTER | INSTEAD OF | FOR} evento1 [OR evento2 OR evento3] ON
nome_tabela  [REFERENCING OLD AS antigo | NEW AS novo]
FOR EACH ROW
[{FOLLOWS | PRECEDES} esquema.outro_gatilho]
[{ENABLE | DISABLE}]
[WHEN condição_gatilho]
BEGIN
  corpo_gatilho
END nome  gatilho;
```

- OLD: refere-se ao valor antigo da coluna (campo);
- NEW: refere-se ao novo valor da coluna;
- WHEN: condição que será testada em cada linha executada.

Exemplo2: Criando um gatilho de linha para não deixar reajustar o salário dos funcionários em mais de 20%.

```
CREATE OR REPLACE TRIGGER reajuste_fun
BEFORE UPDATE OF salario ON tabfun
FOR EACH ROW
BEGIN
 IF :new.salario > :old.salario*1.20 THEN
    RAISE_APPLICATION_ERROR(-20002,'Atenção – Aumento não Permitido');
 END IF;
END reajuste_fun;
/
```

Testando o gatilho:

```
UPDATE tabfun
SET salario = salario * 1.30
WHERE matricula = 10;
```

Exemplo3: Criando um gatilho de linha para fixar um teto máximo de salário

```
CREATE OR REPLACE TRIGGER salario_alto
BEFORE INSERT OR UPDATE OF salario ON tabfun
FOR EACH ROW
BEGIN
  IF :new.salario > 300000  THEN
     RAISE_APPLICATION_ERROR(-20003,'Atenção – Salário Muito Alto');
  END IF;
END salario_alto;
/
```

Testando o gatilho:

```
INSERT INTO tabfun VALUES(33,'Maradona',400000,'01/01/2002',3);
```

Exemplo4: Criando um gatilho de linha para fixar salário de 200000 para os funcionários novos do cargo 3 (jogadores).

```
CREATE OR REPLACE TRIGGER salario_fixo
BEFORE INSERT ON tabfun
FOR EACH ROW
WHEN (new.cargo = 3)
BEGIN
  IF INSERTING THEN
     :new.salario := 200000;
  END IF;
END salario_fixo;
/
```

Testando o gatilho:

```
INSERT INTO tabfun VALUES(33,'Zico',240000,'01/05/2002',3);
```

Exemplo5: Criando um gatilho de linha para inserir registros na tabela de registros (tabreg) cada vez que houver alteração no salário dos funcionários.

```
CREATE OR REPLACE TRIGGER atualiza_reg
BEFORE INSERT OR UPDATE OF salario ON tabfun
FOR EACH ROW
BEGIN
 IF INSERTING THEN
    INSERT INTO tabreg
        VALUES(:new.matricula,:new.cargo,:new.admissao,:new.salario);
 ELSE
    INSERT INTO tabreg
        VALUES(:old.matricula,:old.cargo,sysdate,:new.salario);
 END IF;
END atualiza_reg;
/
```

7.11.5.12. Obtendo informações sobre um gatilho

A tabela USER_TRIGGERS armazena informações sobre os gatilhos criados. A seguir, é apresentada a estrutura da tabela USER_TRIGGERS.

COLUNA	TIPO	DESCRIÇÃO
TRIGGER_NAME	VARCHAR2(30)	Nome do gatilho
TRIGGER_TYPE	VARCHAR2(16)	Tipo do gatilho (linha ou instrução)
TRIGGERING_EVENT	VARCHAR2(227)	Evento de acionamento do gatilho
TABLE_OWNER	VARCHAR2(30)	Proprietário da tabela
BASE_OBJECT_TYPE	VARCHAR2(16)	Tipo do objeto da base de dados
TABLE_NAME	VARCHAR2(30)	Tabela de acionamento do gatilho
COLUMN_NAME	VARCHAR2(4000)	Coluna da tabela de acionamento gatilho
REFERENCING_NAMES	VARCHAR2(128)	Nome especificado do OLD e do NEW
WHEN_CLAUSE	VARCHAR2(4000)	Condição de acionamento
STATUS	VARCHAR2(8)	Ativado ou Desativado
DESCRIPTION	VARCHAR2(4000)	Descrição do cabeçalho do código
ACTION_TYPE	VARCHAR2(11)	Tipo de ação (PL/SQL)
TRIGGER_BODY	LONG	Corpo do gatilho

Na tabela USER_SOURCE também pode ser encontrado o código fonte referente aos gatilhos criados.

Consultando a tabela USER_SOURCE

```
SELECT * FROM line, text WHERE name='NOME_OBJETO' order by line;
```

7.11.5.13. Gerenciando gatilhos

Habilitar um gatilho de uma tabela

```
ALTER TRIGGER nome_trigger ENABLE
```

Desabilitar um gatilho

```
ALTER TRIGGER nome_trigger DISABLE
```

Habilitar todos os gatilhos de uma tabela

```
ALTER TABLE nome_tabela ENABLE ALL_TRIGGERS
```

Desabilitar todos os gatilhos de uma tabela

```
ALTER TABLE nome_tabela DISABLE ALL_TRIGGERS
```

Carregar um gatilho

```
START caminho nome_gatilho.SQL
```

Compilar um gatilho

```
ALTER TRIGGER nome_gatilho COMPILE
```

Sintaxe:

```
DROP TRIGGER nome_gatilho;
```

Exemplo1:

```
DROP TRIGGER salario_fixo;
```

Exercícios:

1) Faça um gatilho que não permita inserir um funcionário com data de admissão maior que a data do servidor (SYSDATE).
2) Faça um gatilho que não permita alterar ou incluir um funcionário com salário igual a zero.

7.11.6. Pacotes (PACKAGES)

Segundo Price (2009, p.393) os pacotes permitem encapsular funcionalidade relacionada em uma unidade independente. Modularizando seu código PL/SQL por meio de pacotes, é possível construir suas próprias bibliotecas de código que outros programadores podem reutilizar.

Normalmente, os pacotes são constituídos de dois componentes: uma *especificação* e um *corpo*. A especificação do pacote lista os procedimentos, funções, tipos e objetos disponíveis. A especificação não contém o código que constitui os procedimentos e funções; o código está contido no corpo do pacote.

Todos os itens do corpo que não estão listados na especificação são *privados* do pacote. Os itens privados só podem ser usados dentro do corpo do pacote. Usando uma combinação de itens públicos e privados, é possível construir um pacote cuja complexidade fica oculta do mundo exterior. Esse é um dos principais objetivos de toda programação: ocultar a complexidade de seus usuários.

7.11.6.1. Objetivos

- Criar uma especificação de um pacote;
- Criar o corpo de um pacote;
- Chamar procedimentos e funções em um pacote;
- Obtendo informações sobre procedimentos e funções em um pacote;
- Remover pacotes.

7.11.6.2. Sintaxe para criação de uma especificação de pacote

```
CREATE [OR REPLACE] PACKAGE nome_pacote
{IS | AS}
  especificação_pacote
END nome_pacote;
```

- *especificação_pacote*: lista os procedimentos, funções, tipos e objetos públicos disponíveis para os usuários de seu pacote;

O exemplo a seguir cria uma especificação para um pacote chamado tabfun_pacote:

```
CREATE OR REPLACE PACKAGE tabfun_pacote AS
  TYPE t_ref_cursor IS REF CURSOR;
  FUNCTION get_tabfun_ref_cursor RETURN t_ref_cursor;
  PROCEDURE update_tabfun_salario (
    n_matricula IN tabfun.matricula%TYPE,
    n_fator IN NUMBER
  );
END tabfun_pacote;
/
```

O tipo *t_ref_cursor* é um tipo REF CURSOR da PL/SQL. Um REF CURSOR é semelhante a um ponteiro na linguagem de programação C++ e aponta para um cursor. A função *get_tabfun_ref_cursor()* retorna um tipo *t_ref_cursor* e, aponta para um cursor que contém as linhas recuperadas da tabela **tabfun**. O procedimento *update_tabfun_salario()* multiplica o salário de um funcionário e confirma a alteração.

7.11.6.3. Sintaxe para criação do corpo de um pacote

```
CREATE [OR REPLACE] PACKAGE BODY nome_pacote
{IS | AS}
   corpo_pacote
END nome_pacote;
```

- *corpo_pacote*: contém o código dos procedimentos e funções.

O exemplo a seguir cria o corpo do pacote tabfun_pacote:

```
CREATE OR REPLACE PACKAGE BODY tabfun_pacote AS
  FUNCTION get_tabfun_ref_cursor
  RETURN t_ref_cursor IS
     v_tabfun_ref_cursor t_ref_cursor;
  BEGIN
    --- obtém o REF CURSOR
    OPEN v_tabfun_ref_cursor FOR
      SELECT matricula, nome, salario
      FROM tabfun;
    --- retorna o REF CURSOR
    RETURN v_tabfun_ref_cursor;
  END get_tabfun_ref_cursor;

  PROCEDURE update_tabfun_salario (
    n_matricula IN tabfun.matricula%TYPE,
    n_fator IN NUMBER
  ) AS
    v_tabfun_count INTEGER;
 BEGIN
   --- conta o número de produtos com o valor de n_matricula fornecido
   --- será 1 se o funcionário existir
   SELECT COUNT(*)
   INTO v_tabfun_count
   FROM tabfun
   WHERE matricula = n_matricula;

   --- se o produto existe (n_tabfun_count=1) então atualiza salário
   IF v_tabfun_count = 1 THEN
       UPDATE tabfun
       SET salario = salario * n_fator
       WHERE matricula = n_matricula;
       COMMIT;
    END IF;
  EXCEPTION
    WHEN OTHERS THEN
       ROLLBACK;
  END update_tabfun_salario;
END tabfun_pacote;
/
```

7.11.6.4. Chamando procedimentos/funções em um pacote

Para chamar procedimentos e funções em um pacote, deve-se incluir o nome do pacote na chamada. O exemplo a seguir chama *tabfun_pacote.get_tabfun_ref_cursor()*, que retorna uma referência para um cursor contendo os valores de matricula, nome e salário dos funcionários.

```
SELECT tabfun_pacote.get_tabfun_ref_cursor
FROM dual;
```

O próximo exemplo chama tabfun_pacote.update_tabfun_salario() para multiplicar o salário do funcionário 10 por 1.15. Caso o aumento ultrapasse 20% o gatilho *reajuste_fun* será acionado não permitindo o aumento.

```
CALL tabfun_pacote.update_tabfun_salario(10, 1.15);
```

7.11.6.5. Obtendo informações sobre procedimentos e funções de um pacote

A tabela USER_PROCEDURES também armazena informações referentes aos pacotes criados. A estrutura da tabela USER_PROCEDURES já foi apresentada nas seções anteriores.

```
SELECT object_name, procedure_name
FROM user_procedures
WHERE object_name = 'TABFUN_PACOTE';

OBJECT_NAME                    PROCEDURE_NAME
------------------------------ ------------------------------
TABFUN_PACOTE                  GET_TABFUN_REF_CURSOR
TABFUN_PACOTE                  UPDATE_TABFUN_SALARIO
```

7.11.6.6. Removendo um pacote

```
DROP PACKAGE tabfun_pacote;
```