PACKAGE

- Um package é uma área para armazenamento de subprogramas, tipos, constantes, cursores e variáveis de PL/SQL.
- Um package é definido em duas partes:
  - Especificação
  - Parte do Corpo

PACKAGE SPECIFICATION

- A especificação é a interface com as aplicações.
- É nela que declaramos os tipos, variáveis, constantes, exceções, cursores e subprogramas que desejamos que sejam visíveis e compartilháveis pelas aplicações.
- Tudo que ser definido na parte da especificação pode ser compartilhado.

PACKAGE SPECIFICATION

- A definição dos subprogramas não é completa.
- As aplicações só necessitam saber quais os parâmetros a serem passados para a rotina e quais os retornos fornecidos
- Não há necessidade de se saber como a rotina efetua a operação.

PACKAGE SPECIFICATION

- OBJETIVO:
  - O objetivo de empacotar é primeiramente organizacional.
  - O pacote oferece a possibilidade de juntarmos itens e subprogramas que realizem ações associadas.

PACKAGE SPECIFICATION

```
CREATE OR REPLACE PACKAGE PEMP IS
  TYPE REGEMP       IS RECORD (NOME     VARCHAR2(10),
                               NASC     DATE,
                               CARGO    VARCHAR2(09));
  CURSOR CEMP(MAT IN NUMBER) RETURN REGEMP;
  FUNCTION INCLUI(NOME  IN VARCHAR2, NASC IN DATE,
                  CARGO IN VARCHAR2) RETURN BOOLEAN;
  FUNCTION EXCLUI(MAT IN NUMBER) RETURN BOOLEAN;
END PEMP;
```

- Observe que as informações declaradas no pacote determinam quais os nomes das rotinas e itens, os parâmetros recebidos e o retorno esperado.
- Não definimos detalhes da implementação. Isto será feito no corpo.

PACKAGE BODY

- Na parte de corpo do pacote, definimos por completo todas as rotinas.
- Não só aquelas declaradas na parte de especificação como também todas aquelas rotinas que serão chamadas pelas outras, mas que não desejamos disponibilizar diretamente para as aplicações.

PACKAGE BODY

- Definimos também, agora os cursores por completo além de todas as variáveis que venham a ser usadas pelas rotinas e que não desejamos disponibilizar para as demais aplicações.
- O corpo implementa detalhes e declarações privadas que são invisíveis pelas aplicações.

PACKAGE BODY

```
CREATE SEQUENCE SEQ_MAT START WITH 400;
CREATE OR REPLACE PACKAGE BODY PEMP IS
  CURSOR CEMP(MAT IN NUMBER) RETURN REGEMP IS
    SELECT ENAME, HIREDATE, JOB
      FROM EMP
     WHERE EMPNO = MAT;
  FUNCTION INCLUI(NOME IN VARCHAR2, NASC  IN DATE,
                  CARGO IN VARCHAR2) RETURN BOOLEAN IS
  BEGIN
   INSERT INTO EMP (EMPNO, ENAME, HIREDATE, JOB, SAL, DEPTNO)
     VALUES (SEQ_MAT.NEXTVAL, NOME, NASC, CARGO,2000, 10);
    RETURN TRUE;
  END INCLUI;
  FUNCTION EXCLUI (MAT IN NUMBER) RETURN BOOLEAN IS
  BEGIN
    DELETE FROM EMP WHERE EMPNO = MAT;
    RETURN TRUE;
  END;
END PEMP;
/
```

PACKAGE BODY

- Resumindo, quando criamos um Package Body, complementamos a definição do cursor e das rotinas declaradas na especificação.
- Uma vez que apenas a parte da especificação é utilizada para compilação dos programas que façam uso dos pacotes, podemos alterar o corpo do pacote sem haver necessidade de recompilar os programas que o usam.

RESTRIÇÕES

- Como restrições à definição e acesso a pacotes, temos:
  - Um pacote não pode ser chamado diretamente.
    - Faremos referência aos objetos declarados dentro do pacote;
  - Um pacote não pode receber parâmetros
    - Somente as rotinas declaradas no pacote podem fazê-lo.
  - Um pacote não pode ser “aninhado”.
    - Não podemos declarar um pacote dentro de um outro pacote

RESTRIÇÕES

- Não podemos fazer referência remota a variáveis definidas em pacotes.
  - Se utilizarmos remotamente uma rotina que faça referência a uma variável de pacote, receberemos erro.
- Não podemos fazer referência a variáveis Bind dentro de pacotes.
  - Variáveis Bind são aquelas variáveis definidas no ambiente do aplicativo.

USANDO PACKAGES

- Para fazermos referência aos tipos, objetos e subprogramas definidos dentro da área de especificação de um pacote, devemos mencionar o nome do pacote como qualificador do item.
- A Oracle organiza as rotinas utilitárias (predefinidas) em pacotes.

USANDO PACKAGES

```
SET SERVEROUT ON
BEGIN
  FOR REMP IN PEMP.CEMP(&MAT) LOOP
    DBMS_OUTPUT.PUT_LINE(REMP.NOME);
  END LOOP;
END;
/
```

USANDO PACKAGES

```
DECLARE
  REMP              PEMP.CEMP%ROWTYPE;
  MAT               EMP.EMPNO%TYPE := &MAT;
BEGIN
  OPEN PEMP.CEMP(MAT);
  FETCH PEMP.CEMP INTO REMP;
  IF PEMP.EXCLUI(MAT) THEN
     :MSG := TO_CHAR(REMP.NASC, 'DD/MM/YYYY');
  END IF;
  CLOSE PEMP.CEMP;
EXCEPTION
  WHEN OTHERS THEN
    CLOSE PEMP.CEMP;
END;
/
```

OVERLOADING

- Já sabemos que a PL/SQL permite a definição de dois subprogramas com o mesmo nome, desde que seus parâmetros formais sejam diferentes em número, ordem ou tipo familiar.
- Como restrição, temos que esta técnica só se aplica a subprogramas locais ou subprogramas definidos em pacotes.
- A característica de Overloading pode ser bastante explorada quando definimos as rotinas dentro de pacotes.

OVERLOADING

```
CREATE OR REPLACE PACKAGE POVER IS
  FUNCTION DISPLAY (DATA IN DATE) RETURN VARCHAR2;
  FUNCTION DISPLAY (TEXTO IN VARCHAR2) RETURN VARCHAR2;
  FUNCTION DISPLAY (NUMERO IN NUMBER) RETURN VARCHAR2;
END POVER;
/
CREATE OR REPLACE PACKAGE BODY POVER IS
  FUNCTION DISPLAY (DATA IN DATE) RETURN VARCHAR2 IS
  BEGIN
    RETURN 'DATA = '||TO_CHAR(DATA, 'DD/MM/YY HH24:MI');
  END;
  FUNCTION DISPLAY (TEXTO IN VARCHAR2) RETURN VARCHAR2 IS
  BEGIN
    RETURN 'TEXTO = ' || TEXTO;
  END;
  FUNCTION DISPLAY (NUMERO IN NUMBER) RETURN VARCHAR2 IS
  BEGIN
    RETURN 'NUMERO = '|| TO_CHAR(NUMERO, 'L999G999G999D99');
  END;
END POVER;
/
```

SQL DINÂMICO EM PL/SQL

- De um modo geral, construímos programas em PL/SQL nos quais incluímos comandos de SQL estáticos, isto é, comandos que são conhecidos e passíveis de ser compilados durante o desenvolvimento do programa.
- Eventualmente, temos aplicações que necessitam processar comandos de SQL que só completam a tempo de execução.

USANDO SQL DINÂMICO

- Comandos de SQL dinâmico são armazenados em strings construídas pelo programa a tempo de execução.
- Estas strings devem conter um comando SQL ou um bloco de PL/SQL válidos.
- Podem, ainda, conter argumentos (Bind) a serem supridos durante a execução.

USANDO SQL DINÂMICO

- Para processarmos comandos de SQL, tais como Insert, Update, Delete ou blocos de PL/SQL, usaremos o comando EXECUTE IMMEDIATE
- Para processarmos SQL Select, usaremos os comandos Open-For, Fetch e Close.

EXECUTE IMMEDIATE

- O comando EXECUTE IMMEDIATE está presente na package DBMS_SQL desde a versão 8i. Este comando permite executar imediatamente um SQL ou um bloco PL/SQL durante a sua criação.

  Muitas vezes a criação e execução de SQL's dinâmicos podem comprometer significativamente a performance do banco de dados. Porém o comando "execute immediate" reduz esse problema e ajuda a obter uma performance melhor, além de tornar mais amigável a programação de códigos PL/SQL, comparando-se com as versões anteriores (7.x e 8.x) onde se utilizava a package DBMS_SQL, diretamente.

EXECUTE IMMEDIATE

EXECUTE IMMEDIATE *string dinâmica*

[INTO {variável

[, variável]... | record}]

[USING [IN | OUT | IN OUT] bind

[, [IN | OUT | IN OUT] bind] ...];

EXECUTE IMMEDIATE

- String Dinâmica:
  - Representa um comando SQL qualquer, exceto queries que retornem múltiplas linhas (sem terminações) ou um bloco de PL/SQL (com terminação).
- Variável:
  - É uma variável que armazena o valor de uma coluna selecionada.

EXECUTE IMMEDIATE

- Record:
  - Corresponde a uma Record definido pelo usuário ou um %RowType que armazenará uma linha (row) selecionada.
- Bind:
  - É uma expressão cujo valor é passado dinamicamente para o comando SQL ou bloco de PL/SQL.

EXECUTE IMMEDIATE

- O comando “execute immediate” NÃO grava uma transação de DML, para gravá-lo uma instrução de “commit” deverá ser executada. Já um comando DDL, quando processado via “execute immediate”, será gravado diretamente.

  Para consultas que retornam mais de uma linha este comando não é suportado, como alternativa deve-se criar uma tabela temporária para gravar os registros, ou usar cursores REF.

  Para os desenvolvedores de Forms, está opção NÃO funciona em Forms 6i e em PL/SQL 8.0.6.3.

EXECUTE IMMEDIATE

```
 DECLARE
   TEXTO         VARCHAR2(1000) := '&VALOR';
 BEGIN
   EXECUTE IMMEDIATE TEXTO USING SYSDATE;
 END;
/

INSERT INTO EMP(EMPNO,HIREDATE,DEPTNO)
    VALUES (169,:DT,20)
```

EXECUTE IMMEDIATE

- Passando valores para um SQL dinâmico (utilizando a cláusula – USING).
- declare
  l_depnome varchar2(20) := 'teste';
  l_local varchar2(10) := 'Sorocaba';
  begin
  execute immediate 'insert into dept values (:1, :2, :3)'
  using 50, l_depnome, l_local;
  commit;
  end;

EXECUTE IMMEDIATE

- Retornando valores de um SQL dinâmico (utilizando a cláusula – INTO).

```
declare
   l_cnt varchar2(20);
   begin
   execute immediate 'select count(1) from emp'
   into l_cnt;
   dbms_output.put_line(l_cnt);
   end;
```

EXECUTE IMMEDIATE

- Retornando um valor para dentro de um registro em um código PL/SQL.
  - declare
    type empdtlrec is record (empno number(4),
    ename varchar2(20),
    deptno number(2));
    empdtl empdtlrec;
    begin
    execute immediate 'select empno, ename, deptno ' ||
    'from emp where empno = 7934'
    into empdtl;
    end;

EXECUTE IMMEDIATE

- Passando e retornando valores. Na cláusula INTO deve-se utilizar a cláusula USING.

```
declare
l_dept pls_integer := 20;
l_nam varchar2(20);
l_loc varchar2(20);

begin
execute immediate 'select dname, loc from dept where
deptno = :1'
into l_nam, l_loc
using l_dept ;
end;
```

EXECUTE IMMEDIATE

- Para consultas que retornem mais de uma linha. Usar o comando “insert” para popular uma tabela temporária.

```
declare
l_sal pls_integer := 2000;
begin
execute immediate 'insert into temp(empno, ename) ' ||
' select empno, ename from emp ' ||
' where sal > :1'
using l_sal;
commit;
end;
```

EXECUTE IMMEDIATE

- DICA FINAL

  O comando “execute immediate” possui um método muito mais fácil e eficiente para processar SQL’s dinâmicos. Porém não se esqueça que quando a intenção é executar SQL’s dinâmicos, o tratamento de exceções torna-se muito importante.

OPEN-FOR, FETCH e CLOSE

- Para efetuarmos o processamento de uma consulta, que retorne diversas linhas, de forma dinâmica, deveremos utilizar três comandos:
  - Open uma variável cursor
  - For uma consulta de múltiplas linhas
  - Fetch as linhas para variáveis locais da aplicação
  - Close a varíavel cursor, quando todas as linhas tiverem sido processadas.

OPEN-FOR, FETCH e CLOSE

- Para cada valor retornado pela consulta (associado à variável cursor), deve existir uma variável do tipo correspondente compatível com o valor a ser atribuído.
- O comando Close desabilita a variável cursor.
- Após esta ação, o conjunto resultado associado a ela fica indefinido.

OPEN-FOR, FETCH e CLOSE

```
DECLARE
  TEXTO    VARCHAR2(1000) := '&VALOR';
  TYPE RC  IS REF CURSOR;
  C1       RC;
BEGIN
  OPEN C1 FOR TEXTO USING SYSDATE;
  :MSG := 'LINHAS OBTIDAS: ';
  LOOP
     FETCH C1 INTO TEXTO;
     EXIT WHEN C1%NOTFOUND;
     :MSG := :MSG || TEXTO || '; ';
  END LOOP;
  CLOSE C1;
END;
/
SELECT ENAME FROM EMP WHERE HIREDATE > (:DT - 30000)
```

PASSANDO UM ARGUMENTO NULL

- A forma abaixo recebe um erro porque o literal Null não é permitido na cláusula Using.

```
BEGIN
        EXECUTE IMMEDIATE 'UPDATE EMP SET COMM = :x'
         USING NULL;
END;
/
```

- Uma forma de resolvermos o problema seria a concatenação do texto e retirada da variável Bind

```
DECLARE
        A_NULL NUMBER;
BEGIN
        EXECUTE IMMEDIATE 'UPDATE EMP SET COMM = :x'
                USING A_NULL;
END;
/
```