**Packages**

* Database objects that group logically related PL/SQL types, objects and subprograms
* They cannot be called, passed parameters to or nested
* There are two parts
  + Specification
  + Body

**Advantages of Packages**

* Modularity
  + allows encapsulation of related types, objects and subprograms in a named PL/SQL module
  + easy to understand with simple, clear and well defined interfaces
  + helps in application development
* Easier Application Design
  + when designing applications, only interface information in package specifications initially required
  + can code and compile specification without body
  + stored subprograms referencing the package can be compiled as well
  + need not define package bodies until ready to complete the application
* Information Hiding
  + can specify which types, items, and subprograms are public or private
  + definition of private subprograms is hidden so that only the package (not the application) is affected if the definition changes
  + simplifies maintenance and enhancement and protects integrity of the package
* Better performance
  + when a packaged subprogram is called for the first time, the whole package is loaded into memory
  + later calls to related subprograms in the package require no disk I/O

**Package Specification**

* Is an interface to the applications
* Declares the types, variables, constants, exceptions, cursors and subprograms available for use
* Holds public declarations, visible to the application
* Can be thought of as an operational interface
* Scope of the declarations are local to the database schema and global to the package
* Lists the package resources available to applications
* Created using CREATE PACKAGE command

**Syntax for Package Specification –**

CREATE [OR REPLACE] PACKAGE <packagename> AS

Global variables declaration;

Procedure specifications;

Function specifications;

Type Definitions;

Cursor Declarations

END [<packagename>];

**Package Body**

* Implements the package specification
* Fully defines cursors and subprograms
* Holds implementation details and private declarations, hidden from the application
* Can be thought of as a ‘black body’
* Can be replaced, enhanced or replaced without changing the interface
* Can be changed without recompiling calling programs
* Scope of the declarations are local to the package body
* Declared types and objects are inaccessible except from within the package body
* Initialization part of a package is run only once, the first time the package is referenced

**Syntax for Package Body –**

CREATE [OR REPLACE] PACKAGE BODY <packagename> AS

Private members (variables and procedure/functions/cursors/types)

Procedure Code;

Function Code;

Implementation of Types;

Use of Cursors;

Using Global variables in the members of the package.

END [<packagename>];

**Referencing Package Objects**

* Packaged objects and subprograms must be referenced using the dot notation

packagename.typename

packagename.objectname

packagename.subprogramname

E.g - **DBMS\_OUTPUT.*PUT\_LINE***

**Maintaining a Package**

* Can drop a package using the DROP command

DROP PACKAGE <packagename>

* Example

DROP PACKAGE airlines;

To drop just one construct, remove it from the package and then recompile the package

**Examples of Packages -**

**1) Creating a package of 3 procedures -**

**Package Specification –**

create or replace package pack1 is

procedure x(a number);

procedure y(b number);

procedure z(c number);

end;

**Package Body -**

create or replace package **body** pack1

is

procedure x(a number)

is

Begin

dbms\_output.put\_line('Procedure p1');

End x;

procedure y(b number)

is

Begin

dbms\_output.put\_line('Procedure p2');

End y;

**/\*Suppose in the package body if all the procedures are not written (defined0 then it will give error.\*/**

/\*procedure z(c number)

is

Begin

dbms\_output.put\_line('Procedure p3');

End z; \*/

End;

-------------------------------

**Using the Package pk1-**

SQL > Execute **PK1.X(4);**

**2) Use of global variable in a function and procedure -**

**Package Specification -**

create or replace package pack2

as

g number;

function m(a number) **return number**;

procedure n;

end pack2;

**Package Body -**

create or replace package body pack2

as

function m(a number) return number

is

Begin

g := a; **-- The function assigns its parameter's value to public variable g**

return g; **-- The function only returns the value of that public variable g**

End m;

procedure n

is

Begin

if g >= 100 then**-- Procedure should be called after the function is assigning value to g**

dbms\_output.put\_line('Discount is 20%');

else

dbms\_output.put\_line('Discount is 10%');

end if;

end n;

End pack2;

/

**Using the package in a Anonymous block -**

Declare

x number;

Begin

x := pack2.m(700);

pack2.n;

End;

Declare

x number;

Begin

x := pack2.m(40);

pack2.n;

End;

**3) Use of Cursor in Procedure -**

**Package Specification -**

create or replace package Manager\_Details

as

cursor cf is select \* from emp

where job = 'MANAGER'; **-- cf is Public cursor**

M cf%rowtype; **-- M is Public Variable**

procedure First\_Manager; **-- First\_Manager is Public Procedure**

End;

**Package Body -**

create or replace package body Manager\_Details

as

**y number; -- Private Variable**

procedure First\_Manager

is

**x number; -- Local Variable**

Begin

**Open cf; -- Directly the public cursor is getting opened!!!!**

Loop

fetch cf into **M**;

**/\*Showing the first entry of manager\*/**

if cf%rowcount = 1 then

dbms\_output.put\_line(M.empno || '-' || M.ename || '-' || M.sal);

else

**exit;**

end if;

End Loop;

Close cf;

End First\_Manager;

End Manager\_Details;

**Execute the procedure by following command:**

**Execute Manager\_Details.First\_Manager**

**4) Use of Type in a Procedure -**

**Package Specification -**

create or replace package pack5

as

Type t1 is RECORD

(e1 Emp.Empno %Type,

e2 Emp.Ename%Type,

e3 Emp.Sal%Type);

Procedure p1;

end;

**Package Body -**

create or replace package body pack5

as

procedure p1

is

**v t1;** **/\*Using the type of the package directly inside the procedure.\*/**

Begin

select empno,ename,sal into v

from emp

where ename = 'SMITH';

dbms\_output.put\_line(v.e1 || '-' || v.e2 || '-' || v.e3);

End p1;

End pack5;

/

Execute pack5.p1

**5) Example of private members in package body –**

create or replace package prvmembers

is

procedure m1;

end;

|  |
| --- |
| create or replace package body prvmembers  is  **/\*Creating Private Procedure in the body\*/**  **procedure m2**  **is**  **begin**  **dbms\_output.put\_line('This is the private member of the package body');**  **end m2;**  procedure m1  is  begin  m2;  end;  end; |

**Example of forward declaration of a private member**

|  |
| --- |
| create or replace package body prvmembers  is  **procedure m2;** /\* Forward declaration of m2 \*/  procedure m1  is  begin  m2;  end;  /\*Creating Private Procedure in the body\*/  procedure m2  is  begin  dbms\_output.put\_line('This is the private member of the package body');  end m2;  end; |

**Example for "Body less package" And also for "Persistent state of Package Public Variables":**

create or replace package bodyless

as

x number := 200;

y number := 100;

end;

Execute the following **Anonymous Block** code:

begin

bodyless.x := bodyless.x + 100;

dbms\_output.put\_line('Value of x is ' || bodyless.x);

bodyless.y := bodyless.y + 500;

dbms\_output.put\_line('Value of y is ' || bodyless.y);

end;

/\* Note-- Execute the block 3-4 times, you will see that the values of x and y are getting changed.

Start the new session. Give Set ServerOutput On command.

The original values of the x and y from the package gets initialized in that session.

Even if the session 1 and 2 are closed and again new session is started still it will take as per initial values of variables.

\*/

**An Bodyless package can ideally have the following things in a Live Project:**

**Constants are all below:**

Tax\_Value := 700;

Incentive := 2000;

Bonus := 7000;

start\_time:= 9:30:00 am;

end\_time := 5:30:00 pm;

transaction\_date := sysdate;

GST := 300;

**Public Variables are all below:**

amount number;

custid varchar(40);

mobile\_number number(10);

**One time only procedure (Actually it is an Procedure in package body)**

**Useful for setting remote connectivity for front ends.**

**In the session for the first time any of the members of the package is called then the procedure will be implicitly invoked.**

**Then for the future calls of the members of the same package in the same session the procedure will not get executed.**

create or replace package pack6

is

procedure p1;

procedure p2;

end;

**Package Body Code:**

create or replace package body pack6

is

procedure p1

is

begin

dbms\_output.put\_line('p1');

end p1;

procedure p2

is

begin

dbms\_output.put\_line('p2');

end p2;

**begin**

**dbms\_output.put\_line('Welcome to my package');**

end pack6;

**Example 1 of ideal code inside One Time Procedure:**

create or replace package pack7

as

type namelist is table of emp.ename%type index by binary\_integer;

v namelist;

cursor cf is select ename from emp;

M cf%rowtype;

procedure first\_five;

procedure next\_five;

end;

**create or replace package body pack7**

**is**

**i Number;**

**procedure first\_five**

**is**

**begin**

**for i in 1..5**

**loop**

**dbms\_output.put\_line(v(i));**

**end loop;**

**end first\_five;**

**procedure next\_five**

**is**

**begin**

**for i in 6..10**

**loop**

**dbms\_output.put\_line(v(i));**

**end loop;**

**end next\_five;**

**begin**

**i := 0;**

**open cf;**

**loop**

**i := i + 1;**

**fetch cf into M;**

**v(i) := M.ename;**

**exit when cf%notfound;**

**end loop;**

**end pack7;**

# Example 2 of ideal code inside One Time Procedure:

# To initialize the global variable by taking value from table (suitable for aggregate values)

create or replace package pack8

as

average\_sal number;

function calc\_sal\_deviation(s number) return number;

procedure sal\_status(s number);

end;

create or replace package body pack8

is

/\*function calc\_sal\_deviation(s number) return number;

procedure sal\_status(s number);\*/

function calc\_sal\_deviation(s number) return number

is

begin

if average\_sal > s then

return (average\_sal - s);

else

return 0;

end if;

end;

procedure sal\_status(s number)

is

begin

if average\_sal > s then

dbms\_output.put\_line('On Lower Side');

else

dbms\_output.put\_line('On Higher Side');

end if;

end;

**begin**

**select avg(sal) into average\_sal**

**from emp;**

end pack8;

exec pack8.SAL\_STATUS(800);

exec pack8.SAL\_STATUS(4000);

select sal, pack81.CALC\_SAL\_DEVIATION(sal) "Deviation" from emp;

**Example of Overloaded Members**

**Procedures Overloading:**

create or replace package pack9

is

procedure **p**(a number, b number);

procedure **p**(x varchar2, y varchar2);

procedure **p**(z number);

procedure **p**;

end pack9;

create or replace package body pack9

is

procedure p(a number, b number)

is

c number; **-- c is the Local Variable of current procedure p**

begin

c := a + b;

dbms\_output.put\_line(c);

end p;

procedure p(x varchar2, y varchar2)

is

begin

dbms\_output.put\_line(x || y);

end p;

procedure p(z number)

is

begin

if z >= 40 then

dbms\_output.put\_line('Passed');

else

dbms\_output.put\_line('Failed');

end if;

end p;

procedure p

is

begin

dbms\_output.put\_line('No parameter is passed');

end p;

end pack9;

**Function Overloading:**

Create Or Replace Package pack10

Is

Function **F** (A Number, B Number) **Return Number;**

Function **F** (A Number, C Number) **Return Varchar2**;

Function **F** (A Number, D Number) **Return Boolean**;

End;

Create Or Replace Package Body pack10

Is

Function F (A Number, B Number) Return Number

Is

Begin

If A > B Then

Return 100;

Else

Return 50;

End If;

End;

Function F (A Number, C Number) Return Varchar2

Is

Begin

If A > C Then

Return 'A Is Greater Than B';

Else

Return 'B Is Greater Than A';

End If;

End;

Function F(A Number, D Number) Return Boolean

Is

Begin

If A >D Then

Return True;

Else

Return False;

End If;

End;

End;

**Overloading in PL/SQL 🡪 Parameter data types can be same, but names should be different.**

create or replace package pack11

as

procedure **p**(salary number, commission number);

procedure **p**(salary number, tax number);

end pack11;

create or replace package body pack11

as

procedure p(salary number, commission number)

is

total number;

begin

total := salary **+** commission;

dbms\_output.put\_line(total);

end;

procedure p(salary number, tax number)

is

take\_home number;

begin

take\_home := salary **-** tax;

dbms\_output.put\_line(take\_home);

end;

end pack11;

**How to call the overloaded procedures or functions when their number of parameters are same and positional data types are also same????**

**While executing such procedure the parameters need to be called by named notation only.**

**Cannot call by Positional Notation.**

**Exec pack11.p(9000, 400) -- Error**

**Exec pack11.p(salary => 4000, commission => 500);**

**Exec pack11.p(salary => 4000, tax=> 200);**

**Persistent Cursors –**

**create or replace package pc**

**is**

**cursor cf is select \* from emp;**

**procedure first3;**

**procedure next3;**

**c cf%rowtype;**

**end;**

**create or replace package body pc is**

**procedure first3 is**

**begin**

**open cf;**

**loop**

**fetch cf into c;**

**dbms\_output.put\_line(c.empno||'-'||c.ename||'-'||c.sal);**

**exit when cf%rowcount >= 3;**

**end loop;**

**end first3;**

**procedure next3 is**

**begin**

**loop**

**fetch cf into c;**

**dbms\_output.put\_line(c.empno||'-'||c.ename||'-'||c.sal);**

**exit when cf%rowcount >= 6;**

**end loop;**

**close cf;**

**end next3;**

**end pc;**

**Altering (recompiling) and Dropping Packages Scenarios:**

1. **What will Happen if Package Specification is dropped by the command Drop Package PackageName;**

**Ans. When Package Specification is dropped then the Package Body gets implicitly dropped!!!**

1. **What will Happen if Package Body is dropped by the command Drop Package Body PackageName;**

**Ans. The Package Body gets dropped then Package Specification remains as it is.**

1. **Initially Package Specification and Package Body are there.**

**Now what will happen if Package Specification is Altered, means some more members are added?**

**Ans. The Package Body becomes invalid. You need to recreate the Package Body as per the new members or changes of the Package Specification.**

1. **Initially Package Specification and Package Body are ther.**

**Now what will happen if Package Body is Altered, means some more members are added?**

**Ans. Package Specification remains as it is. Only Package Body gets re-created.**