Packages in PL/SQL

PL/SQL packages are schema objects that groups logically related PL/SQL types, variables and subprograms.

A package will have two mandatory parts:

* Package specification
* Package body or definition

Package Specification

The specification is the interface to the package. It just DECLARES the types, variables, constants, exceptions, cursors, and subprograms that can be referenced from outside the package. In other words, it contains all information about the content of the package, but excludes the code for the subprograms.

All objects placed in the specification are called **public** objects. Any subprogram not in the package specification but coded in the package body is called a **private** object.

The following code snippet shows a package specification having a single procedure. You can have many global variables defined and multiple procedures or functions inside a package.

CREATE PACKAGE cust\_sal AS

PROCEDURE find\_sal(c\_id customers.id%type);

END cust\_sal;

/

When the above code is executed at SQL prompt, it produces the following result:

Package created.

Package Body

The package body has the codes for various methods declared in the package specification and other private declarations, which are hidden from code outside the package.

The CREATE PACKAGE BODY Statement is used for creating the package body. The following code snippet shows the package body declaration for the*cust\_sal* package created above. I assumed that we already have CUSTOMERS table created in our database as mentioned in [PL/SQL - Variables](http://www.tutorialspoint.com/plsql/plsql_variable_types.htm) chapter.

CREATE OR REPLACE PACKAGE BODY cust\_sal AS

PROCEDURE find\_sal(c\_id customers.id%TYPE) IS

c\_sal customers.salary%TYPE;

BEGIN

SELECT salary INTO c\_sal

FROM customers

WHERE id = c\_id;

dbms\_output.put\_line('Salary: '|| c\_sal);

END find\_sal;

END cust\_sal;

/

When the above code is executed at SQL prompt, it produces the following result:

Package body created.

Using the Package Elements

The package elements (variables, procedures or functions) are accessed with the following syntax:

package\_name.element\_name;

Consider, we already have created above package in our database schema, the following program uses the *find\_sal* method of the *cust\_sal* package:

DECLARE

code customers.id%type := &cc\_id;

BEGIN

cust\_sal.find\_sal(code);

END;

/

When the above code is executed at SQL prompt, it prompt to enter customer ID and when you enter an ID, it displays corresponding salary as follows:

Enter value for cc\_id: 1

Salary: 3000

PL/SQL procedure successfully completed.

Example:

The following program provides a more complete package. We will use the CUSTOMERS table stored in our database with the following records:

Select \* from customers;

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 3000.00 |

| 2 | Khilan | 25 | Delhi | 3000.00 |

| 3 | kaushik | 23 | Kota | 3000.00 |

| 4 | Chaitali | 25 | Mumbai | 7500.00 |

| 5 | Hardik | 27 | Bhopal | 9500.00 |

| 6 | Komal | 22 | MP | 5500.00 |

+----+----------+-----+-----------+----------+

THE PACKAGE SPECIFICATION:

CREATE OR REPLACE PACKAGE c\_package AS

-- Adds a customer

PROCEDURE addCustomer(c\_id customers.id%type,

c\_name customers.name%type,

c\_age customers.age%type,

c\_addr customers.address%type,

c\_sal customers.salary%type);

-- Removes a customer

PROCEDURE delCustomer(c\_id customers.id%TYPE);

--Lists all customers

PROCEDURE listCustomer;

END c\_package;

/

When the above code is executed at SQL prompt, it creates the above package and displays the following result:

Package created.

CREATING THE PACKAGE BODY:

CREATE OR REPLACE PACKAGE BODY c\_package AS

PROCEDURE addCustomer(c\_id customers.id%type,

c\_name customers.name%type,

c\_age customers.age%type,

c\_addr customers.address%type,

c\_sal customers.salary%type)

IS

BEGIN

INSERT INTO customers (id,name,age,address,salary)

VALUES(c\_id, c\_name, c\_age, c\_addr, c\_sal);

END addCustomer;

PROCEDURE delCustomer(c\_id customers.id%type) IS

BEGIN

DELETE FROM customers

WHERE id = c\_id;

END delCustomer;

PROCEDURE listCustomer IS

CURSOR c\_customers is

SELECT name FROM customers;

TYPE c\_list is TABLE OF customers.name%type;

name\_list c\_list := c\_list();

counter integer :=0;

BEGIN

FOR n IN c\_customers LOOP

counter := counter +1;

name\_list.extend;

name\_list(counter) := n.name;

dbms\_output.put\_line('Customer(' ||counter|| ')'||name\_list(counter));

END LOOP;

END listCustomer;

END c\_package;

/

Above example makes use of **nested table** which we will discuss in the next chapter. When the above code is executed at SQL prompt, it produces the following result:

Package body created.

USING THE PACKAGE:

The following program uses the methods declared and defined in the package*c\_package*.

DECLARE

code customers.id%type:= 8;

BEGIN

c\_package.addcustomer(7, 'Rajnish', 25, 'Chennai', 3500);

c\_package.addcustomer(8, 'Subham', 32, 'Delhi', 7500);

c\_package.listcustomer;

c\_package.delcustomer(code);

c\_package.listcustomer;

END;

/

When the above code is executed at SQL prompt, it produces the following result:

Customer(1): Ramesh

Customer(2): Khilan

Customer(3): kaushik

Customer(4): Chaitali

Customer(5): Hardik

Customer(6): Komal

Customer(7): Rajnish

Customer(8): Subham

Customer(1): Ramesh

Customer(2): Khilan

Customer(3): kaushik

Customer(4): Chaitali

Customer(5): Hardik

Customer(6): Komal

Customer(7): Rajnish

PL/SQL procedure successfully completed