PL/SQL - PACKAGES

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 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 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 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 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 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 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
```