PL/SQL Packages

PL/SQL packages are schema objects that holds other objects like procedures, functions , variables, constants, cursors and exceptions within it. It is the way of creating generic, encapsulated and re-usable code.

ADVANTAGES OF USING PACKAGE :

- Enables the organization of commercial applications into efficient modules.
- Allows granting of privileges efficiently.
- All procedures and cursors that execute in this environment can share them.
- Enables the overloading of procedures and functions when required.
- Improve performance by loading multiple objects into memory at once.
- Promote code reuse through the use of libraries that contain stored Procedures and functions, thereby reducing redundant coding.

A package will have two mandatory parts:

- Package specification
- Package body or definition

❖ Package Specification

The specification is the interface to the package. It contains:

- Name of the package
- Name of the data type of any arguments
- This declaration is local to the database and global 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.**

Example :

```
CREATE PACKAGE cust_sal AS
        PROCEDURE find_sal(c_id customers.id%type);
END cust_sal;/
```

Output:

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.

Example :

The following code shows the package body declaration for the **cust_sal package** created above. I assumed that we already have CUSTOMERS table created in our database.

```
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;/
```

Output:

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;/
```

Output :

```
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 | Chaitali | 25 | Mumbai | 7500.00 |
| 4 | Hardik | 27 | Bhopal | 9500.00 |
+----+----------+-----+-----------+----------+
```

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;
/
```

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
        rcust customers%ROWTYPE;
                CURSOR c_cust is SELECT * FROM CUSTOMERS;
                BEGIN
                OPEN c_cust;
                        dbms_output.put_line('NAME ' ||'SALARY ' );
                LOOP
```

```
                FETCH c_cust into rcust;
                EXIT WHEN c_cust%notfound;
                dbms_output.put_line(rcust.name || ' ' || rcust.salary);
            END LOOP;
            CLOSE c_cust;
        END listCustomer;
END c_package;
        /
```

Package body created.

❖ **Calling the Package:**

The following program uses the methods declared and defined in the package c_package.

```
EXECUTE c_package.addcustomer(5,'komal',13,'baroda',5000);
OR
CALL c_package.addcustomer(5,'komal',13,'baroda',5000);
OR
DECLARE
        code customers.id%type:= 6;
BEGIN
        c_package.addcustomer(5, 'Rajnish', 25, 'Chennai', 3500);
        c_package.addcustomer(6, 'Subham', 32, 'Delhi', 7500);
        c_package.listcustomer;
        c_package.delcustomer(code);
        c_package.listcustomer;
END;
/
```

OUTPUT:

NAMESALARY

Ramesh 3000
Khilan 3000
Chaitali 7500
Hardik 9500
Rajnish 3500
Subham 7500

NAMESALARY

Ramesh 3000
Khilan 3000
Chaitali 7500
Hardik 9500

Rajnish 3500
PL/SQL procedure successfully completed

❖ Alterations To An Existing Package

ALTER PACKAGE command is used to recompile a package with compile keyword. It compiles a package explicitly after modifications.
It recompiles all objects defined within the package.
Recompiling does not change the definition of package or any its objects.

Statement recompiles the package specification

ALTER PACKAGE <PackageName> COMPILE PACKAGE;

Statements recompile just the package body.

ALTER PACKAGE <PackageName> COMPILE BODY;

Example

ALTER PACKAGE c_package COMPILE BODY

Package body altered.

❖ All packages can be recompiled by using Oracle utility called dbms_utility.

Syntax:

EXECUTE DBMS_UTILITY.COMPILE_ALL

❖ PL/SQL Overloading Procedures And Functions

- A package is an Oracle object that can hold a number of other objects like Procedures and functions.
- More than one procedure or function with the same name but with different parameters can be defined within a package or within a PL/SQL declaration block.
- Multiple procedures that are declared with same name are called **Overloaded Procedures.**
- Similarly, multiple functions that are declared with same name are called **Overloaded Functions.**

For example if we want to get a customer either by specifying the customer id or by specifying name, we could do it by writing following package:
Creating the package body:

```
CREATE OR REPLACE PACKAGE c_over_package AS
    --Lists customer whose id is passed
    PROCEDURE listCustomer(c_id cust.id%type);
    --Lists customer whose name is passed
    PROCEDURE listCustomer(c_name cust.name%type);
END c_over_package;
```

Package created.

```
CREATE OR REPLACE PACKAGE BODY c_over_package AS
   PROCEDURE listCustomer(c_id cust.id%type) IS
        c_name cust.name%type;
        c_sal cust.salary%type;
   BEGIN
                SELECT name,salary into c_name,c_sal
                from cust where id=c_id;
        dbms_output.put_line(c_id||' '|| c_name || ' ' ||c_sal);
   END listCustomer;
 PROCEDURE listCustomer(c_name cust.name%type) IS
c_id cust.id%type;
c_sal cust.salary%type;
   BEGIN
        SELECT id,salary into c_id,c_sal
        from cust where name=c_name;
        dbms_output.put_line(c_id||' '|| c_name || ' ' ||c_sal);
   END listCustomer;
END c_over_package;
```

Package body created.

Output :

EXEC c_over_package.listCustomer ('Komal');
6 Komal 5000
PL/SQL procedure successfully completed.

EXEC c_over_package.listCustomer (3);
3 Kaushik 2500
PL/SQL procedure successfully completed.

❖ **Where to overload functions and procedures.**

- Inside the declaration section of a PL/SQL block.
- Inside the package.

❖ **Restrictions On Overloading**

Overloading can be very useful technique when same operation can be done on arguments of different types. Overloading has several restrictions,

- **The data subtype of at least one of the parameters of the overloaded function or procedure must differ.**

```
PROCEDURE  overloadme(string1 IN char)
PROCEDURE overloadme (string1 IN varchar2)
```

- The datatype family of at least one of the parameters of overloaded programs must differ.

- INTEGER, REAL, DECIMAL, FLOAT, etc., are NUMBER subtypes.
- CHAR, VARCHAR2, and LONG are character subtypes.
- If the parameters differ only by datatype within the supertype or family of datatypes, PL/SQL does not have enough information with which to determine the appropriate program to execute/

• **The parameter list of overloaded function must be differ by more than name or parameter mode**

The following two procedures cannot be overloaded,

```
PROCEDURE OverloadMe (P_TheParameter IN NUMBER)
PROCEDURE OverloadMe (P_TheParameter OUT NUMBER)
```

• **Overloaded function must differ by more than their return type.**

For example the following functions cannot be overloaded :

```
FUNCTION OverloadMeToo RETURN DATE;
FUNCTION OverloadMeToo RETURN NUMBER;
```

- **All the overloaded modules must be defined within the same PL/SQL scope or block(PL/SQL block or package)**

Example

```
PROCEDURE DEVELOP_ANALYSIS (QUARTER_END_IN IN DATE, SALES_IN IN NUMBER)
IS
    PROCEDURE REVISE_ESTIMATE (DATE_IN IN DATE) IS
    BEGIN
        PROCEDURE REVISE_ESTIMATE (DOLLAR_IN IN NUMBER) IS
        BEGIN
            . . .
        END;

      BEGIN
            REVISE_ESTIMATE(QUARTER_END_IN);
            REVISE_ESTIMATE(DOLLARS_IN);
      END;
    END;
```

Pl/sql displays the error message because the scope and visibility of both the procedures is different.

https://docstore.mik.ua/orelly/oracle/prog2/ch15_08.htm