PACKAGES

Packages are stored libraries in the database which allow us to group related PL/SQL objects under one name.

Packages are logical groups of related PL/SQL objects.

Packages are names PL/SQL block

Permanently stored into the database schema

Can be referenced or reused by your program

We can below things in packages

Stored procedure

PL/SQL functions

Database cursors

Type of Declarations

Variables

Package architecture

The package specification(header)

The package Body

Package specification

Is also known as package header. In this section we put the declaration of all the package elements

Whatever elements we declare in the specification are publically available and can be referenced outside of the package.

It is mandatory part of the package.

Below are few characteristics of the Package specification.

The elements which are all declared in the specification can be accessed from outside of the package. Such elements are known as a public element.

The package specification is a standalone element that means it can exist alone without package body.

Whenever a package has referred an instance of the package is created for that particular session.

After the instance is created for a session, all the package elements that are initiated in that instance are valid until the end of the session.

Syntax

CREATE OR REPLACE PACKAGE pkg\_name IS

Declare of all the package elements;

END pkg\_name;

Package Body

We provide the actual structure to all the package elements.

A package body contains the implementation of the elements listed in the package specification

A package body can contain both the declaration of the variables as well as the definition of all the package elements.

Any package elements which is not in the package specification but coded in the package body is called Private package element and thus cannot be referenced outside the package.

Below are characteristics of a package body.

It should contain definitions for all the subprograms/cursors that have been declared in the specification.

It can also have more subprograms or other elements that are not declared in specification. These are called private elements.

It is a dependable object, and it depends on package specification.

The state of the package body becomes 'Invalid' whenever the specification is compiled. Therefore, it needs to be recompiled each time after the compilation of specification.

The private elements should be defined first before they are used in the package body.

The first part of the package is the global declaration part. This includes variables, cursors and private elements (forward declaration) that is visible to the entire package.

The last part of the package is Package initialization part that executes one time whenever a package is referred first time in the session.

Syntax

CREATE OR REPLACE PACKAGE BODY pkg\_name IS

Variable\_declaration;

Type\_Declaration;

BEGIN

Implementation of the package elements;

END pkg\_name;

Example

create or replace PACKAGE pkg\_test IS

FUNCTION prnt\_Strng RETURN VARCHAR2;

PROCEDURE proc\_insert(regionid NUMBER,regionname VARCHAR2);

END pkg\_test;

/

create or replace PACKAGE BODY pkg\_test IS

FUNCTION prnt\_strng RETURN VARCHAR2 IS

BEGIN

RETURN 'Testing PAckage';

END prnt\_strng;

PROCEDURE proc\_insert(regionid NUMBER,regionname VARCHAR2) IS

BEGIN

INSERT INTO regions VALUES(regionid,regionname);

COMMIT;

END proc\_insert;

END pkg\_test;

/

Executing procedure

begin

pkg\_test.proc\_insert(5,'India');

end;

/

Executing function

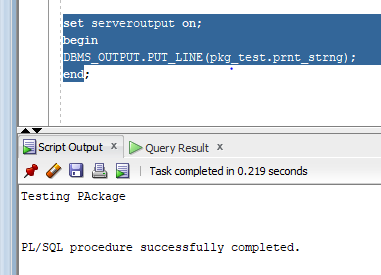
set serveroutput on;

begin

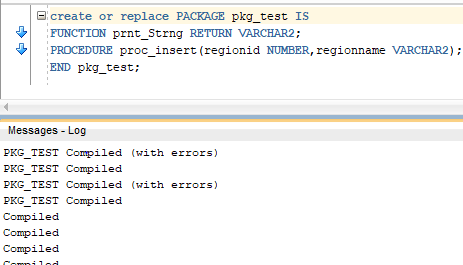
DBMS\_OUTPUT.PUT\_LINE(pkg\_test.prnt\_strng);

End

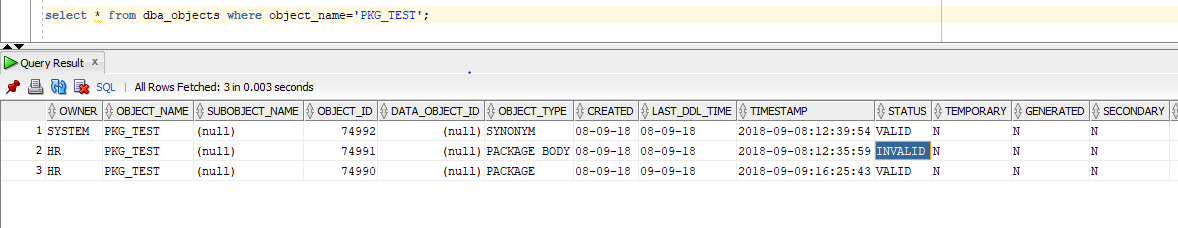
/



Whenever package specification is compiled , the package body will become invalid and hence body must be compiled.



Now package body has become invalid



Referring Package Elements

Once the elements are declared and defined in the package, we need to refer the elements to use them.

All the public elements of the package can be referred by calling the package name followed by the element name separated by period i.e. '<package\_name>.<element\_name>'.

The public variable of the package can also be used in the same way to assign and fetch values from them i.e. '<package\_name>.<variable\_name>'.

Create Package in PL/SQL

In PL/SQL whenever a package is referred/called in a session a new instance will be created for that package.

Oracle provides a facility to initialize package elements or to perform any activity at the time of this instance creation through 'Package Initialization'.

This is nothing but an execution block that is written in the package body after defining all the package elements. This block will be executed whenever a package is referred for the first time in the session.

Syntax

CREATE [OR REPLACE] PACKAGE BODY <package\_name> IS

<Private element definition> <sub\_program and public element definition> .

BEGINE <Package Initialization>

END <package\_name>

Forward Declarations

Forward declaration/reference in the package is nothing but declaring the private elements separately and defining it in the later part of the package body.

Private elements can be referred only if it is already declared in the package body. For this reason, forward declaration is used. But it is rather unusual to use because in most of the time private elements are declared and defined in the first part of the package body.

Forward declaration is an option provided by Oracle, it is not mandatory and using and not using is up to programmer's requirement.

Syntax:

CREATE [OR REPLACE] PACKAGE BODY <package\_name>

IS

<Private element declaration>

.

.

.

<Public element definition that refer the above private element>

.

.

<Private element definition>

.

BEGIN

<package\_initialization code>;

END <package\_name>

Cursors Usage in Package

Unlike other Elements one needs to be careful in using cursors inside the package.

If the cursor is defined in the package specification or in global part of the package body, then the cursor once opened will persist till the end of the session.

So one should always use the cursor attributes '%ISOPEN' to verify the state of the cursor before referring it.

Cursors Usage in Package

Unlike other Elements one needs to be careful in using cursors inside the package.

If the cursor is defined in the package specification or in global part of the package body, then the cursor once opened will persist till the end of the session.

So one should always use the cursor attributes '%ISOPEN' to verify the state of the cursor before referring it.

Overloading

Overloading is the concept of having many subprograms with the same name. These subprograms will be differing from each other by a number of parameters or types of parameters or return type i.e. subprogram with the same name but with different number of parameters, different type of parameters or different retype are considered as overloading.

This is useful when many subprograms needs to do the same task, but the way of calling each of them should be different. In this case, the subprogram name will be kept same for all and the parameters will be changed as per calling statement.

create or replace PACKAGE guru99\_get\_set

IS

PROCEDURE set\_record (p\_emp\_rec IN employees%ROWTYPE);

FUNCTION get\_record (p\_emp\_no IN NUMBER) RETURN employees%ROWTYPE;

END guru99\_get\_set;

create or replace PACKAGE BODY guru99\_get\_set

IS

PROCEDURE set\_record(p\_emp\_rec IN employees%ROWTYPE)

IS

PRAGMA AUTONOMOUS\_TRANSACTION;

BEGIN

INSERT INTO employees

VALUES(p\_emp\_rec.employee\_id,p\_emp\_rec.first\_name,p\_emp\_rec.last\_name,p\_emp\_rec.email,p\_emp\_rec.phone\_number,p\_emp\_rec.hire\_Date,p\_emp\_rec.job\_id,p\_emp\_rec.salary,p\_emp\_rec.commission\_pct,p\_emp\_rec.manager\_id,p\_emp\_rec.department\_id);

COMMIT;

END set\_record;

FUNCTION get\_record(p\_emp\_no IN NUMBER)

RETURN employees%ROWTYPE

IS

l\_emp\_rec employees%ROWTYPE;

BEGIN

SELECT \* INTO l\_emp\_rec FROM employees where employee\_id=p\_emp\_no;

RETURN l\_emp\_rec;

END get\_record;

BEGiN

dbms\_output.put\_line('Control is now executing the package initialization part');

END guru99\_get\_set;

\

Below table gives the data definition table and the package information that is available in the table.

|  |  |  |
| --- | --- | --- |
| Table Name | Description | Query |
| ALL\_OBJECT | Gives the details of the package like object\_id, creation\_date, last\_ddl\_time, etc. It will contain the objects created by all users. | SELECT \* FROM all\_objects where object\_name ='<package\_name>' |
| USER\_OBJECT | Gives the details of the package like object\_id, creation\_date, last\_ddl\_time, etc. It will contain the objects created by the current user. | SELECT \* FROM user\_objects where object\_name ='<package\_name>' |
| ALL\_SOURCE | Gives the source of the objects created by all users. | SELECT \* FROM all\_source where name='<package\_name>' |
| USER\_SOURCE | Gives the source of the objects created by the current user. | SELECT \* FROM user\_source where name='<package\_name>' |
| ALL\_PROCEDURES | Gives the subprogram details like object\_id, overload details, etc created by all users. | SELECT \* FROM all\_procedures Where object\_name='<package\_name>' |
| USER\_PROCEDURES | Gives the subprogram details like object\_id, overload details, etc. created by the current user. | SELECT \* FROM user\_procedures Where object\_name='<package\_name>' |

DECLARE

l\_emp\_rec employees%ROWTYPE;

l\_get\_rec employees%ROWTYPE;

BEGIN

dbms\_output.put\_line('Insert new record for employee 1004');

l\_emp\_rec.employee\_id:=208;

l\_emp\_rec.first\_name:='Alert';

l\_emp\_rec.last\_name:='Einstein';

l\_emp\_rec.email:='ALBERT';

l\_emp\_rec.phone\_number:='515.123.44667';

l\_emp\_rec.hire\_date:=sysdate-100;

l\_emp\_rec.job\_id:='IT\_PROG';

l\_emp\_rec.salary:=10000;

l\_emp\_rec.commission\_pct:=10;

l\_emp\_rec.manager\_id:='100';

l\_emp\_rec.department\_id:=90;

guru99\_get\_set.set\_record(l\_emp\_rec);

dbms\_output.put\_line('Record inserted');

dbms\_output.put\_line('Calling get function to display the inserted record'):

l\_get\_rec:=guru99\_get\_set.get\_record(208);

dbms\_output.put\_line('Employee name: '||l\_get\_rec.first\_name);

dbms\_output.put\_line('Employee number:'||l\_get\_rec.employee\_id);

dbms\_output.put\_line('Employee salary:'||l\_get\_rec.salary);

dbms\_output.put\_line('Employee manager:'||1\_get\_rec.manager\_id);

END;

/