Table of Contents

[CHAPTER 15: PACKAGE 2](#_Toc416724022)

[Theory 2](#_Toc416724023)

[AIM 5](#_Toc416724024)

[Lab Exercise 15: PACKAGE 6](#_Toc416724025)

[1. Package Specification and Body 7](#_Toc416724026)

[2. Package State 19](#_Toc416724027)

[3. SERIALLY\_REUSABLE package 26](#_Toc416724028)

[4. ACCESSABLE\_BY package 34](#_Toc416724029)

[SUMMARY 37](#_Toc416724030)

[REFERENCES 38](#_Toc416724031)

[INDEX 39](#_Toc416724032)

# CHAPTER 15: PACKAGE

## Theory

A Package is a logical group of functions, procedures, cursors, types, constants, and variables. You use Package to group these elements to do approximately one job. A Package usually consists of two parts: (1) Package Specification and (2) Package Body. A Package Specification serves as an Application Program Interface (API) that provides users with all information he needs to know about the package content while hiding the programs details. Thus, every item declared in a Package Specification is a public item. The Package Specification is a mandatory part. In contrast, a Package Body is optional part except if a Package Specification declares procedure, function, or cursor. Therefore, any function, procedure, or cursor declared in a Package Specification must be implemented in the Package Body. Package Body can also declare items but they will be considered as Private items; they can't be referred from outside the package. The basic package structure is shown below:

**Public subprograms**

Private subprograms

Private items

**Public items**

CREATE OR REPLACE PACKAGE BODY package\_name IS

TYPE n\_table IS TABLE OF INTEGER;

IP CONSTANT VARCHAR2(20):= '198.168.0.5';

v\_table n\_table;

v\_serial NUMBER;

*-- FUNCTION AND PROCEDURE IMPLEMENTATIONS*

FUNCTION func\_name(par1 NUMBER) RETURN CHAR  
 IS

BEGIN

....

RETURN ...;

END;

PROCEDURE proc\_name(par1 NUMBER) IS

BEGIN

....

END;

PROCEDURE private\_proc(par1 NUMBER) IS

BEGIN

....

END;

BEGIN

v\_serial := 1;

v\_max := 1;

END package\_name;

CREATE OR REPLACE PACKAGE package\_name IS

TYPE list IS TABLE OF NUMBER;

server CONSTANT VARCHAR2(20):= 'CN';

v\_list list;

v\_max NUMBER;

*-- PUBLIC FUNCTION AND PROCEDURE*

FUNCTION func\_name(par1 NUMBER) RETURN CHAR;

PROCEDURE proc\_name(par1 NUMBER);

END package\_name;

You may use a Package for the following reasons:

* Modularity: instead of defining tens or hundreds of standalone functions and procedures in a schema level and having difficulties finding and maintaining them, it's better to consider using a fewer set of packages that group similar elements together.
* Adding functionality: in some cases, you may be interested in defined a shared variables or cursor for a group of functions or procedures. These cases, a Package is considered the best solution.
* Better performance: The first time you invoke a package subprogram, Oracle Database loads the whole package into memory. Subsequent invocations of other subprograms in same the package require no disk I/O.

## AIM

The AIM of the following exercise is to demonstrate the basic usage of Package in PL/SQL.

The steps involved will include:

* Package Specification and Body
* Package State
* SERIALLY\_REUSABLE Packages
* ACCESSABLE BY package

In general, lab exercises are done in sequential order. Thus, it is assumed that you successfully completed the previous labs. However, not all previous labs are required. Please be sure to run the following lab before proceeding:

* Installing Oracle Database 12c.
* Stored Procedure
* Stored Function

Estimated Completion Time:

25 minutes

# Lab Exercise 15: PACKAGE

|  |
| --- |
|  |

## Package Specification and Body

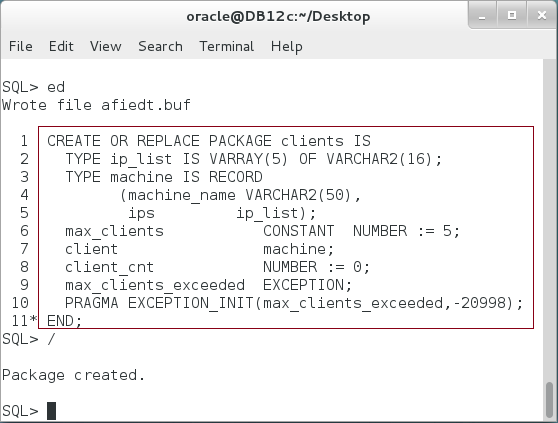
**Step 1:** Open the Terminal, open SQL\*Plus console and connect to hr schema.

|  |  |
| --- | --- |
| Command | Description |
| sqlplus | Open SQL\*Plus console. |
| hr/oracle | connect to **hr** schema. |

****

**Step 2:** Create a Package Specification as shown below:

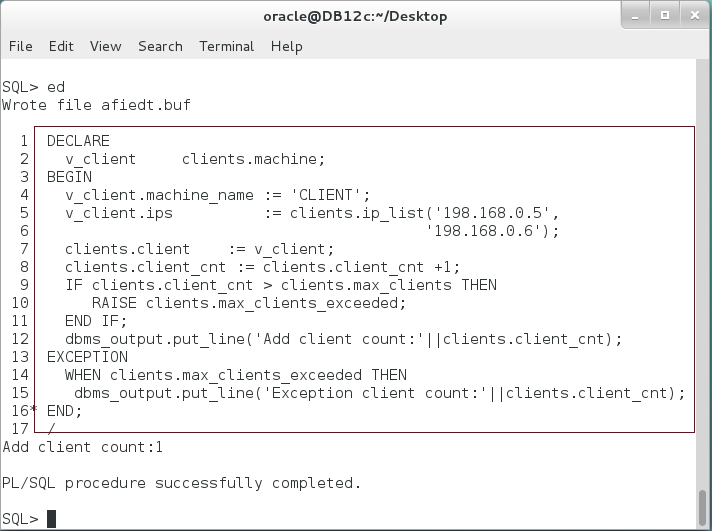
|  |  |
| --- | --- |
| Command | Description |
| CREATE OR REPLACE PACKAGE clients IS |  |
| **TYPE** ip\_list IS **VARRAY**(5) OF VARCHAR2(16); | Define Public types |
| **TYPE** machine **IS RECORD** |
| (machine\_name VARCHAR2(50), |  |
| ips ip\_list); |  |
| max\_clients CONSTANT NUMBER := 5; | Define Public variable |
| client machine; |
| client\_cnt NUMBER := 0; |
| max\_clients\_exceeded **EXCEPTION**; | Define Public Exception |
| PRAGMA EXCEPTION\_INIT(max\_clients\_exceeded,-20998); |
| END; |  |
| / |  |

****

**Please note:** The previous package does not have function or cursor. Thus, it does not need package body to work.

**Step 3:** Use the previous package as shown below:

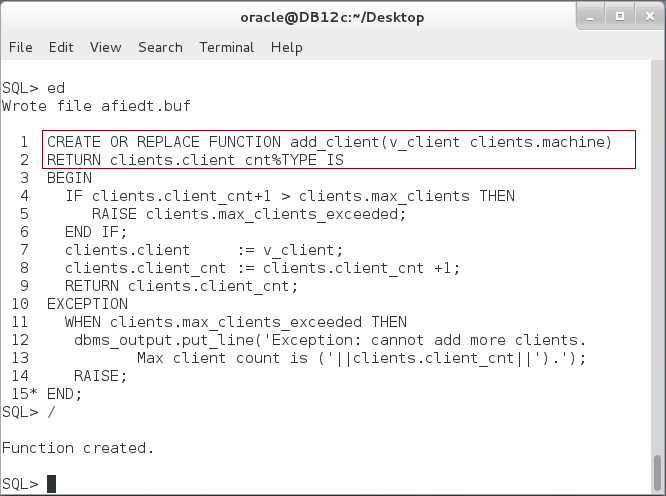
|  |  |
| --- | --- |
| Command | Description |
| DECLARE | Define a variable of type defined in package. |
| v\_client **clients.machine**; |
| BEGIN |
| **v\_client.machine\_name** := 'CLIENT'; | Set a Package's variables. |
| **v\_client.ips** := clients.ip\_list('198.168.0.5', |
| '198.168.0.6'); |
| **clients.client** := v\_client; |
| **clients.client\_cnt** := clients.client\_cnt +1; |
| IF **clients.client\_cnt** > **clients.max\_clients** THEN | Use an exception defined in a Package. |
| RAISE **clients.max\_clients\_exceeded**; |
| END IF; |
| dbms\_output.put\_line('Add client count:'||**clients.client\_cnt**); |  |
| EXCEPTION |  |
| WHEN **clients.max\_clients\_exceeded** THEN | Handle an exception defined in a package. |
| dbms\_output.put\_line('Exception client count:'||clients.client\_cnt); |
| END; |  |
| / |  |



**Please note:** to use any package's elements, you must refer a package name before the element's name like "*Package\_name.Element\_name*".

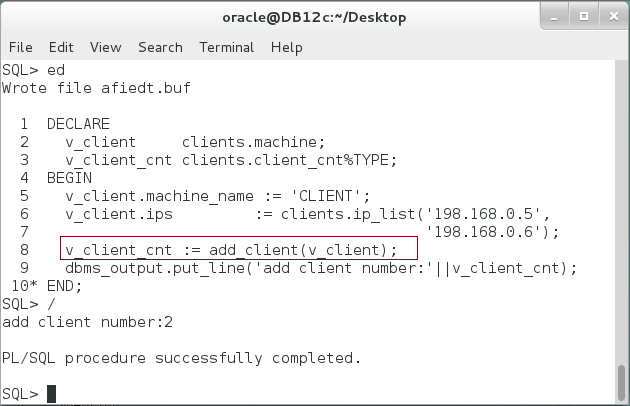
**Step 4:** You may also use a package variables in standalone subprograms. Execute the following block:

|  |  |
| --- | --- |
| Command | Description |
| **CREATE OR REPLACE FUNCTION** add\_client  (v\_client clients.machine) |  |
| **RETURN** clients.client\_cnt%TYPE **IS** |
| **BEGIN** |
| IF clients.client\_cnt+1 > clients.max\_clients THEN |
| RAISE clients.max\_clients\_exceeded; |
| END IF; |
| clients.client := v\_client; |  |
| clients.client\_cnt := clients.client\_cnt +1; |  |
| **RETURN** clients.client\_cnt; |  |
| **EXCEPTION** |  |
| WHEN clients.max\_clients\_exceeded THEN |  |
| dbms\_output.put\_line('Exception: cannot add more clients. |  |
| Max client count is ('||clients.client\_cnt||').'); |  |
| RAISE; |  |
| **END**; |  |
| / |  |



**Step 5:** You may now use the stored function created in the previous step as shown below:

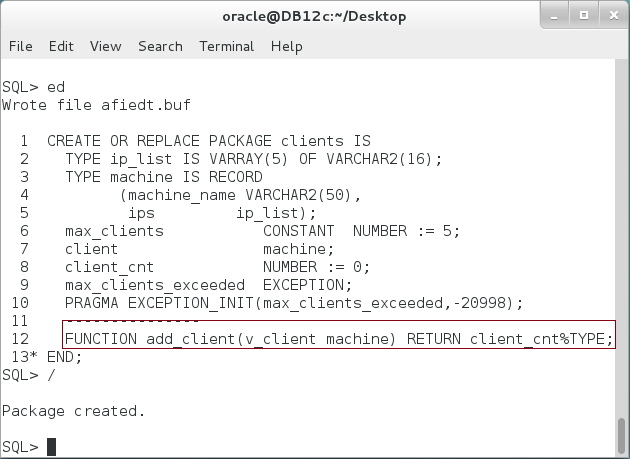
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| v\_client clients.machine; |
| v\_client\_cnt clients.client\_cnt%TYPE; | Call a stored function created previously. |
| BEGIN |
| v\_client.machine\_name := 'CLIENT'; |
| v\_client.ips := clients.ip\_list('198.168.0.5', |
| '198.168.0.6'); |
| **v\_client\_cnt := add\_client(v\_client);** |
| dbms\_output.put\_line('add client number:'||v\_client\_cnt); |
| END; |
| / |  |



**Please note:** Oracle recommends using package subprograms instead of a standalone subprograms. You may also notice that referring a package name before the name of the elements is a boring task.

**Step 6:** Alter a package specification with "**OR REPLACE**" clause and add subprogram "**add\_client**" function.

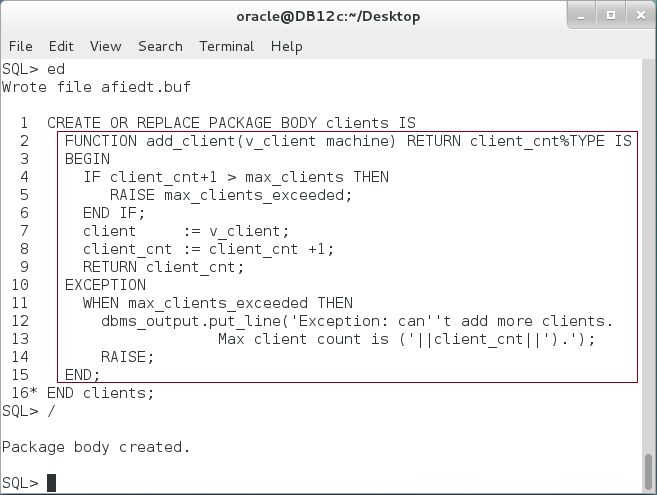
|  |  |
| --- | --- |
| Command | Description |
| **CREATE OR REPLACE PACKAGE** clients IS |  |
| TYPE ip\_list IS VARRAY(5) OF VARCHAR2(16); |  |
| TYPE machine IS RECORD |  |
| (machine\_name VARCHAR2(50), |  |
| ips ip\_list); |  |
| max\_clients CONSTANT NUMBER := 5; |  |
| client machine; |  |
| client\_cnt NUMBER := 0; |  |
| max\_clients\_exceeded EXCEPTION; |  |
| PRAGMA EXCEPTION\_INIT(max\_clients\_exceeded,-20998); |  |
| --------------- |  |
| **FUNCTION add\_client(v\_client machine) RETURN client\_cnt%TYPE;** |  |
| END; |  |
| / |  |

****

**Please note:** In specification part, you should only mention a function declaration.

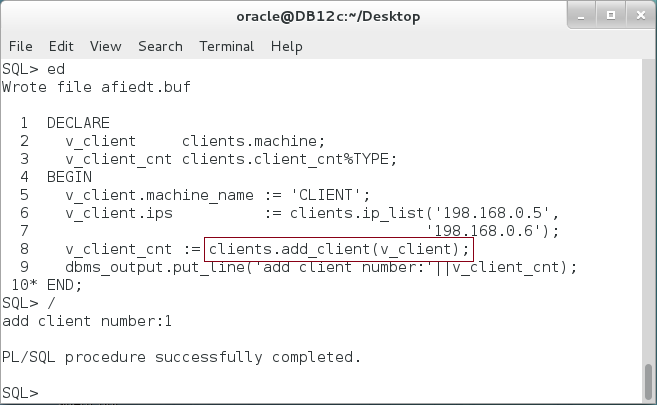
**Step 7:** After adding a subprogram in the specification part, you must add an implementation of the subprogram in the package body. Rebuild the package body as shown below:

|  |  |
| --- | --- |
| Command | Description |
| **CREATE OR REPLACE PACKAGE BODY** clients IS |  |
| FUNCTION add\_client(v\_client machine)   RETURN **client\_cnt**%TYPE IS | Inside the package's subprograms, you don't need to mention the package name. |
| BEGIN |
| IF **client\_cnt**+1 > max\_clients THEN |
| RAISE **max\_clients\_exceeded**; |
| END IF; |
| client := v\_client; |
| client\_cnt := client\_cnt +1; |
| RETURN client\_cnt; |
| EXCEPTION |
| WHEN **max\_clients\_exceeded** THEN |
| dbms\_output.put\_line('Exception: can''t add more clients. |  |
| Max client count is ('||client\_cnt||').'); |  |
| RAISE; |  |
| END; |  |
| END clients; |  |
| / |  |

****

**Step 8:** You can now use a package's subprogram as shown below:

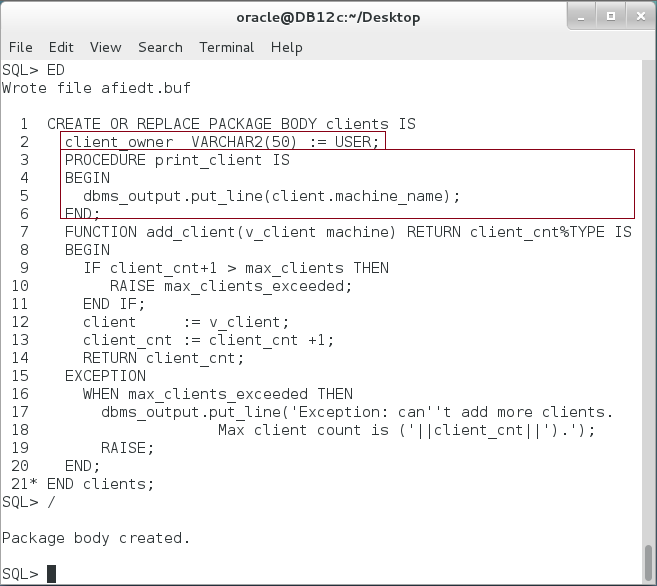
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| v\_client clients.machine; |  |
| v\_client\_cnt clients.client\_cnt%TYPE; |  |
| BEGIN | You can call a package subprograms be specify a package name before the subprogram name. |
| v\_client.machine\_name := 'CLIENT'; |
| v\_client.ips := clients.ip\_list('198.168.0.5', |
| '198.168.0.6'); |
| v\_client\_cnt := **clients.add\_client**(v\_client); |
| dbms\_output.put\_line('add client number:'||v\_client\_cnt); |
| END; |
| / |

****

**Please note:** All the types, variables, and subprograms defined previously are public items; you can access them from outside the package.

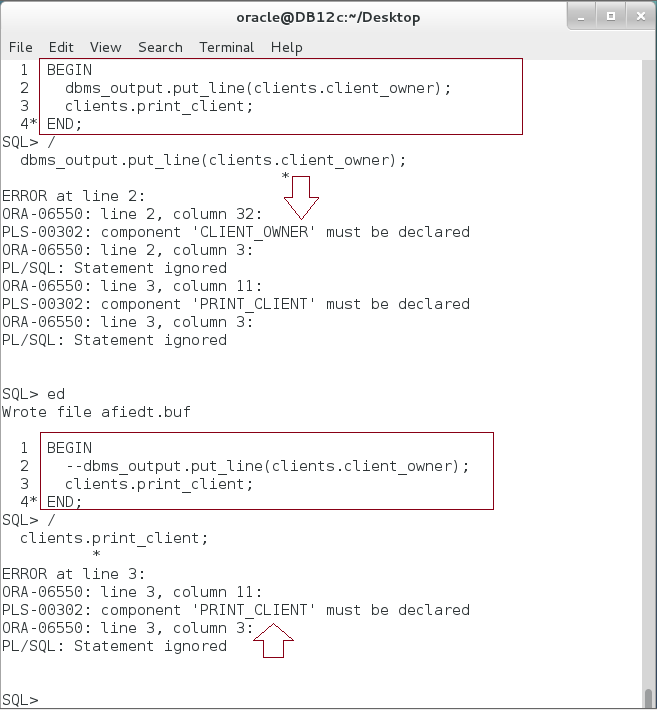
**Step 9:** Alter the package body to add private variables and subprogram as shown below:

|  |  |
| --- | --- |
| Command | Description |
| CREATE OR REPLACE PACKAGE BODY clients IS |  |
| **client\_owner VARCHAR2(50) := USER;** | Add private variable and procedure. |
| **PROCEDURE print\_client IS** |
| **BEGIN** |
| **dbms\_output.put\_line(client.machine\_name);** |
| **END;** |
| FUNCTION add\_client(v\_client machine)   RETURN client\_cnt%TYPE IS |  |
| BEGIN |  |
| IF client\_cnt+1 > max\_clients THEN |  |
| RAISE max\_clients\_exceeded; |  |
| END IF; |  |
| client := v\_client; |  |
| client\_cnt := client\_cnt +1; |  |
| RETURN client\_cnt; |  |
| EXCEPTION |  |
| WHEN max\_clients\_exceeded THEN |  |
| dbms\_output.put\_line('Exception: can''t add more clients. |  |
| Max client count is ('||client\_cnt||').'); |  |
| RAISE; |  |
| END; |  |
| END clients; |  |
| / |  |

****

**Step 10:** Try to access the private variable or function from an anonymous block as show below:

|  |  |
| --- | --- |
| Command | Description |
| BEGIN | You are not allowed to use the private items from outside the package. |
| **dbms\_output.put\_line(clients.client\_owner);** |
| **clients.print\_client;** |
| END; |
| / |

****

## Package State

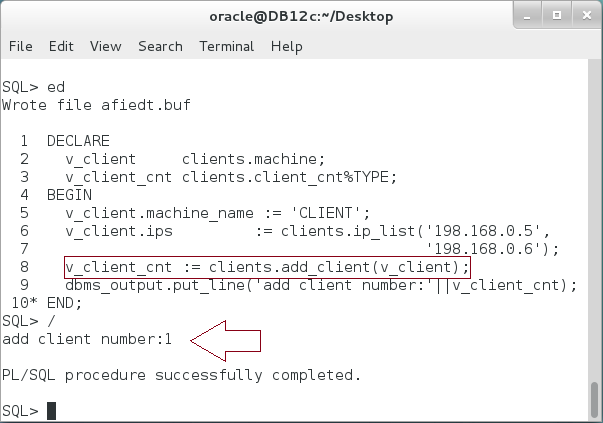
**Step 1:** We didn't explain the output of the previous steps in purpose. To explain the package state, you must start by re-connect a new session. Close the previous session and connection as shown:

|  |  |
| --- | --- |
| Command | Description |
| sqlplus | Open SQL\*Plus console. |
| hr/oracle | connect to **hr** schema. |

****

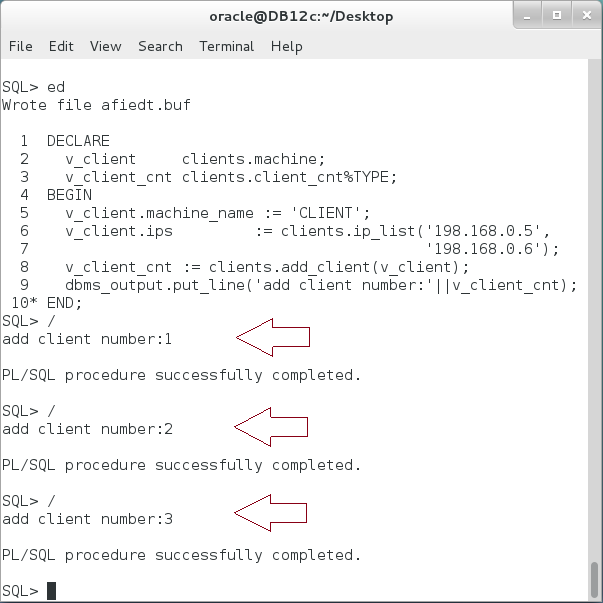
**Step 2:** Execute the following block:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| v\_client clients.machine; |  |
| v\_client\_cnt clients.client\_cnt%TYPE; |  |
| BEGIN |  |
| v\_client.machine\_name := 'CLIENT'; |  |
| v\_client.ips := clients.ip\_list('198.168.0.5', |  |
| '198.168.0.6'); |  |
| **v\_client\_cnt := clients.add\_client(v\_client);** | Execute "**add\_client**" subprogram. |
| dbms\_output.put\_line('add client number:'||v\_client\_cnt); |
| END; |
| / |

****

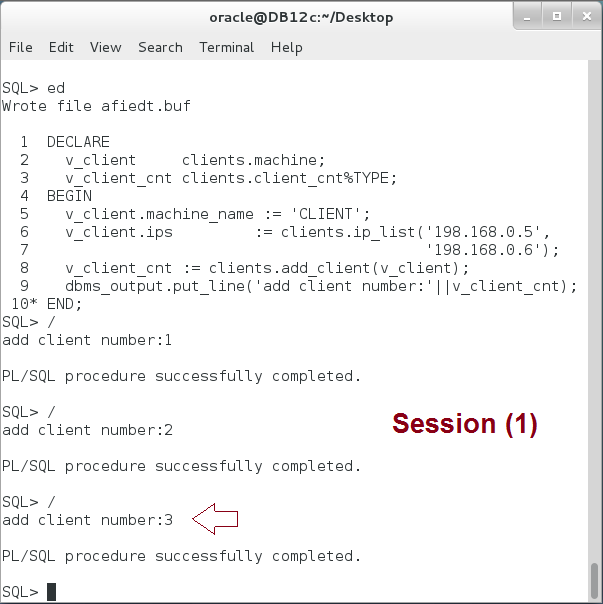
**Please note:** A function "add\_client" returns the value of "client\_cnt" public variable. Since the initial value is 0, adding one to it gives 1. After that, the function ends.

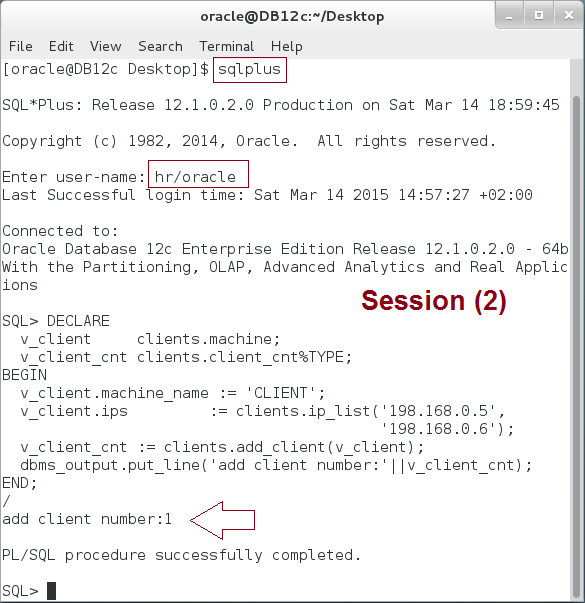
**Step 3:** Re-execute the previous block **twice**:



**Please note:** The value of "**client\_cnt**" public variable remains for the next execution.

**Step 4:** Keep the previous session opened and open new session. Execute the same block in the new session:

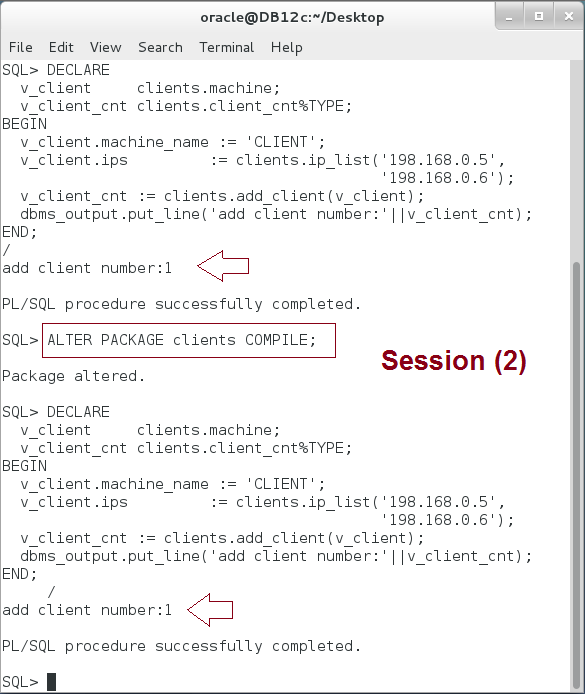




**Please note:** the second session restarts from the beginning! **Explain the output.**

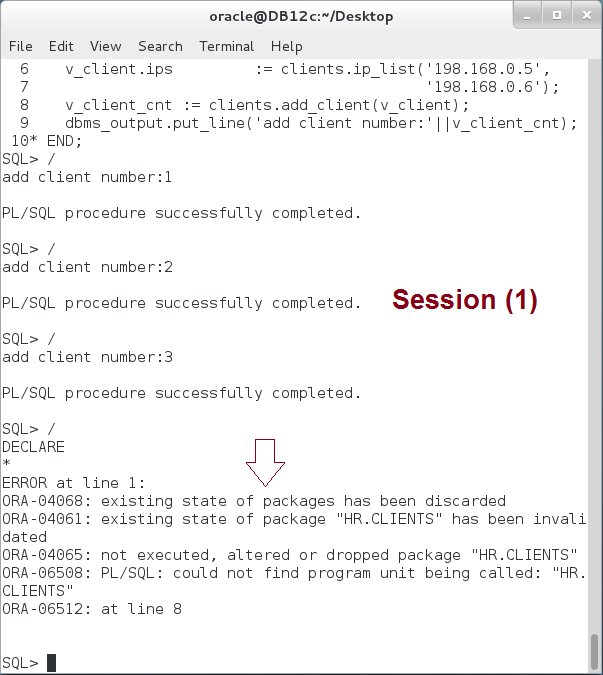
**Step 4:** Compile the package on the second session and re-execute the block:

|  |  |
| --- | --- |
| Line | Description |
| ALTER PACKAGE clients COMPILE; | Compile the package. |



**Please note:** After compiling the package, the package state resets on the same session.

**Step 5:** Return to the first session and re-execute the block:

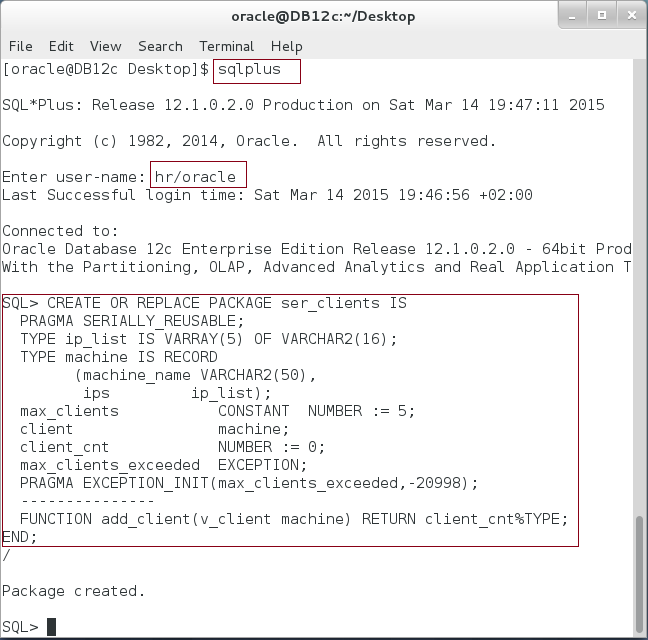


**Please note:** The first session raised exception: package state discarded. **What about the subsequent execution of the block?**

## SERIALLY\_REUSABLE package

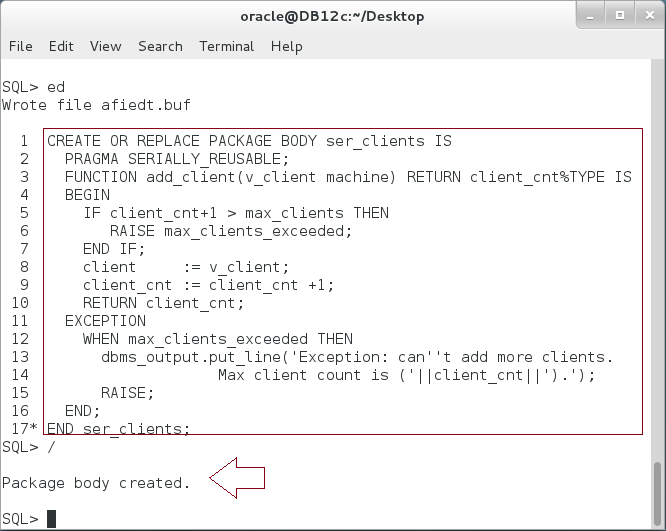
**Step 1:** Close all previous session and open new one (hr/oracle). Create a SERIALLY\_REUSABLE package "**ser\_clients**" similar to the "**clients**" package as shown below:

|  |  |
| --- | --- |
| Command | Description |
| CREATE OR REPLACE PACKAGE **ser\_clients** IS |  |
| **PRAGMA SERIALLY\_REUSABLE;** |  |
| TYPE ip\_list IS VARRAY(5) OF VARCHAR2(16); |  |
| TYPE machine IS RECORD |  |
| (machine\_name VARCHAR2(50), |  |
| ips ip\_list); |  |
| max\_clients CONSTANT NUMBER := 5; |  |
| client machine; |  |
| client\_cnt NUMBER := 0; |  |
| max\_clients\_exceeded EXCEPTION; |  |
| PRAGMA EXCEPTION\_INIT(max\_clients\_exceeded,-20998); |  |
| --------------- |  |
| FUNCTION add\_client(v\_client machine) RETURN client\_cnt%TYPE; |  |
| END; |  |
| / |  |

****

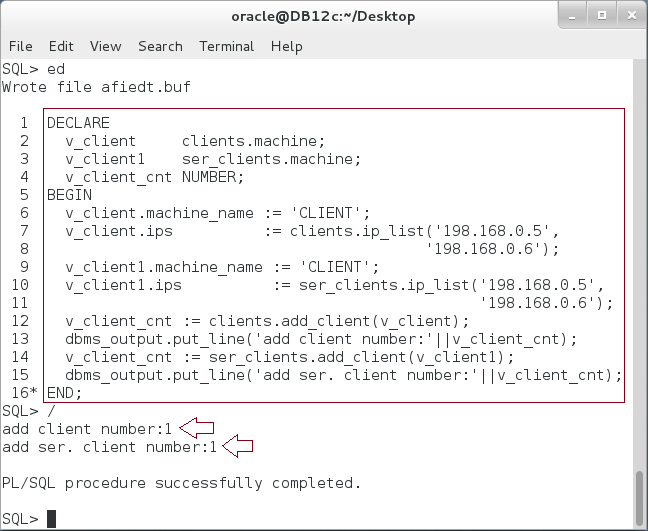
**Step 2:** Create the package body as shown below:

|  |  |
| --- | --- |
| Command | Description |
| CREATE OR REPLACE PACKAGE BODY **ser\_clients** IS | Create "ser\_clients" package body. |
| **PRAGMA SERIALLY\_REUSABLE;** |
| FUNCTION add\_client(v\_client machine) RETURN client\_cnt%TYPE IS |
| BEGIN |
| IF client\_cnt+1 > max\_clients THEN |
| RAISE max\_clients\_exceeded; |
| END IF; |
| client := v\_client; |
| client\_cnt := client\_cnt +1; |  |
| RETURN client\_cnt; |  |
| EXCEPTION |  |
| WHEN max\_clients\_exceeded THEN |  |
| dbms\_output.put\_line('Exception: can''t add more clients. |  |
| Max client count is ('||client\_cnt||').'); |  |
| RAISE; |  |
| END; |  |
| END **ser\_clients**; |  |
| / |  |

****

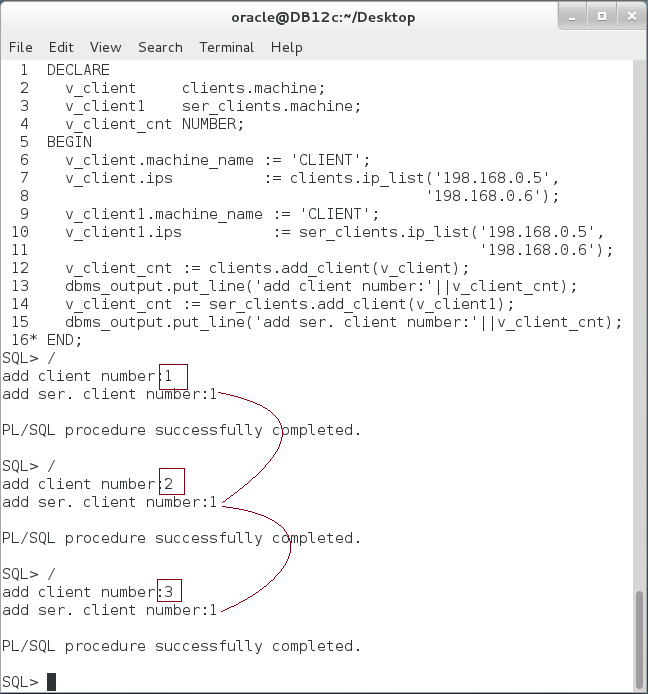
**Step 3:** Compare between "clients" and "ser\_clients" packages. Execute the following block:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| v\_client clients.machine; |  |
| v\_client1 ser\_clients.machine; |  |
| v\_client\_cnt NUMBER; |  |
| BEGIN |  |
| v\_client.machine\_name := 'CLIENT'; |  |
| v\_client.ips := clients.ip\_list('198.168.0.5', |  |
| '198.168.0.6'); |  |
| v\_client1.machine\_name := 'CLIENT'; |  |
| v\_client1.ips := ser\_clients.ip\_list('198.168.0.5', |  |
| '198.168.0.6'); |  |
| v\_client\_cnt := **clients.add\_client**(v\_client); |  |
| dbms\_output.put\_line('add client number:'||v\_client\_cnt); |  |
| v\_client\_cnt := **ser\_clients.add\_client**(v\_client1); |  |
| dbms\_output.put\_line('add ser. client number:'||v\_client\_cnt); |  |
| END; |  |
| / |  |

****

**Please note:** So far, there is no difference. Both package states retain the public variable value "1".

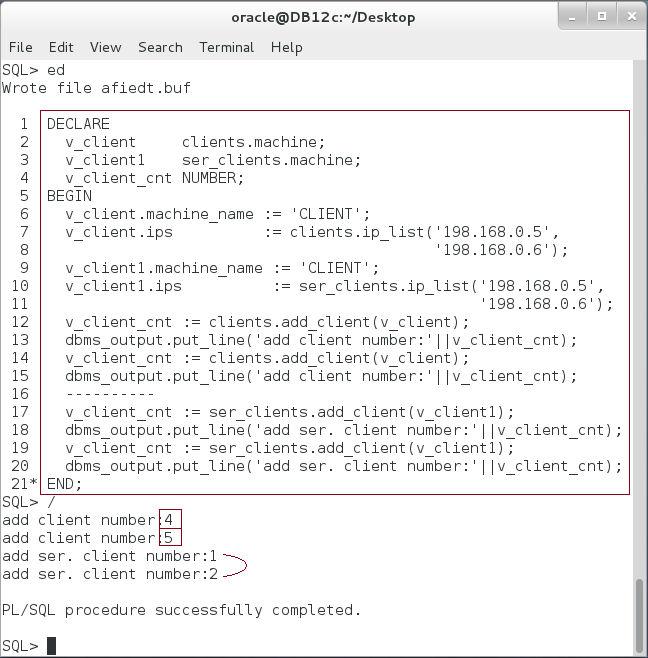
**Step 4:** Re-execute the previous block several times:

****

**Please note:** A SERIALLY\_REUSABLE package does not retain the package state for the session life. It always resets the package state after each block execution. **Does it mean that a SERIALLY\_REUSABLE package is stateless?**

**Step 5:** Second, you may compare pipelined function with bulk function:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| v\_client clients.machine; |  |
| v\_client1 ser\_clients.machine; |  |
| v\_client\_cnt NUMBER; |  |
| BEGIN |  |
| v\_client.machine\_name := 'CLIENT'; |  |
| v\_client.ips := clients.ip\_list('198.168.0.5', |  |
| '198.168.0.6'); |  |
| v\_client1.machine\_name := 'CLIENT'; |  |
| v\_client1.ips := ser\_clients.ip\_list('198.168.0.5', |  |
| '198.168.0.6'); |  |
| v\_client\_cnt := **clients.add\_client**(v\_client); | Call a **regular** package's function twice. |
| dbms\_output.put\_line('add client number:'||v\_client\_cnt); |
| v\_client\_cnt := **clients.add\_client**(v\_client); |
| dbms\_output.put\_line('add client number:'||v\_client\_cnt); |
| ---------- |  |
| v\_client\_cnt := **ser\_clients.add\_client**(v\_client1); | Call a **Serially\_Reusable** package's function twice. |
| dbms\_output.put\_line('add ser. client number:'||v\_client\_cnt); |
| v\_client\_cnt := **ser\_clients.add\_client**(v\_client1); |
| dbms\_output.put\_line('add ser. client number:'||v\_client\_cnt); |
| END; |  |
| / |  |

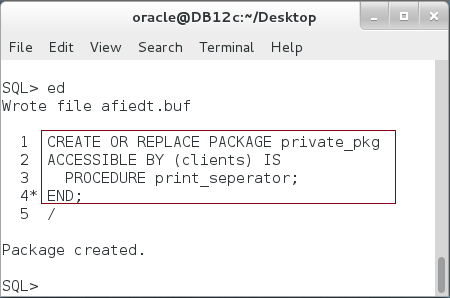
****

**Please note:**  The **SERIALLY\_REUSABLE** package retains the state for a **work unit** not of the session.

## ACCESSABLE\_BY package

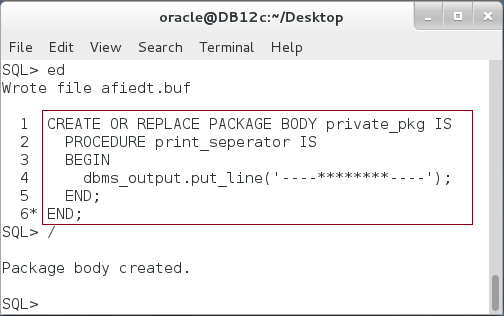
**Step 1:** Use ACCESSABLE\_BY clause to define a white list of other packages that allowed to access the package. Create a private package that can't be accessed by any users or packages but "clients" package as show below:

|  |  |
| --- | --- |
| Command | Description |
| CREATE OR REPLACE PACKAGE private\_pkg |  |
| **ACCESSIBLE BY (clients) IS** |  |
| PROCEDURE print\_seperator; |  |
| END; |  |
| / |

****

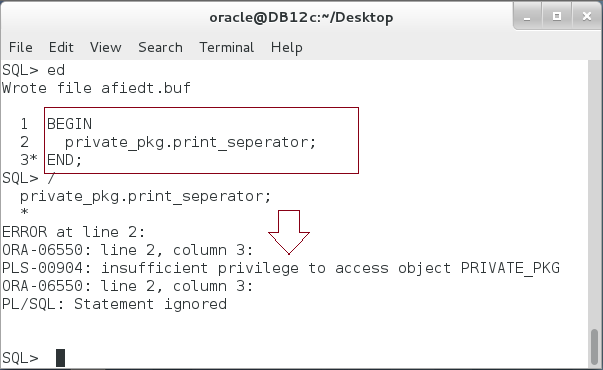
**Step 2:** Create package body as shown below:

|  |  |
| --- | --- |
| Command | Description |
| CREATE OR REPLACE PACKAGE BODY private\_pkg IS |  |
| **PROCEDURE print\_seperator IS** |  |
| **BEGIN** |  |
| dbms\_output.put\_line('----\*\*\*\*\*\*\*\*----'); |  |
| **END;** |
| END; |  |
| / |  |



**Step 3:** Try to access the package from an anonymous block as shown below:

|  |  |
| --- | --- |
| Command | Description |
| BEGIN |  |
| **private\_pkg.print\_seperator;** |  |
| END; |  |
| / |  |



**Step 4:** Try to access "private\_pkg" from "clients" package. **Does it work?**

# SUMMARY

A package is a logical group of subprograms that has two main parts: Specification and Body. Package Specification is a layer of abstraction that hides the programs details and presents only what the users are really care about. Therefore, all the variables, types, exceptions, cursors, functions and procedures defined in the package specification are public; accessible to the users. Package Body is where you implement the subprograms defined in the Package Specification. In addition, you may define private variables, types, cursors, exceptions, functions and procedures in the Package Body; not accessible to users. The package retains its state in a session level; variable's value and cursor's states remain the same for the next reference in the session. A SERIALLY\_REUSABLE package, however, retains its state in work unit level. ACCESSIBLE BY clause is used to create a white list that can access the package. Other packages and anonymous blocks can't access it.

After completing this lab exercise, you should be able to create and manipulate package.

# REFERENCES

* http://docs.oracle.com/database/121/LNPLS/packages.htm#LNPLS00902

# INDEX

Package Body 2, 33

Package Specification 2, 5, 7, 33