Table of Contents

[CHAPTER 12: STORED PROCEDURE 2](#_Toc416718849)

[Theory 2](#_Toc416718850)

[AIM 5](#_Toc416718851)

[Lab Exercise 12: STORED PROCEDURE 6](#_Toc416718852)

[1. Create, Call, and Remove Stored Procedure 7](#_Toc416718853)

[2. Parameter Mode 24](#_Toc416718854)

[3. Overloading Procedures 37](#_Toc416718855)

[SUMMARY 44](#_Toc416718856)

[REFERENCES 45](#_Toc416718857)

[INDEX 46](#_Toc416718858)

# CHAPTER 12: STORED PROCEDURE

## Theory

So far, the PL/SQL blocks you have run have no names and cannot be re-executed by other users. On other words, these blocks are anonymous and have not been stored in a database to be re-executed again. This is of course not good for modularity, maintainability, reusability, and performance. For these, reasons among many others, PL/SQL supports stored subprograms.

Stored subprograms include:

1. Stored Procedure.
2. Stored Function.
3. Package
4. Object Type

Somehow, Stored Procedure and Stored Function are the base for other stored subprograms. That is, a Package subprogram actually contains either Function or Procedure. The same is true to Object Type.

In this chapter, you will learn about a Procedure basic structure and usage which includes Nested Procedure and Standalone Procedure. Not all Nested Procedures are Stored Procedure but all Standalone Procedures are Stored Procedure. The basic structure of a procedure is show below:

**Procedure Declaration**

**Declaration part**

**Executable part**

**Exception-Handling part**

**PROCEDURE** proc\_name

(par1 PLS\_INTEGER,

par2 OUT VARCHAR2,

par3 IN OUT VARCHAR2)

**IS**

v\_var1 VARCHAR2(50);

v\_var2 INTEGER;

**BEGIN**

....

....

....

**EXCEPTION**

WHEN .... THEN

......

**END**;

You should be familiar with Declaration, Executable, and Exception-Handling parts. There are the same as anonymous PL/SQL block shown in the previous chapters. However, the Declaration part does not start with DECLARE keyword. It starts with "IS" or "AS" keywords. The Procedure Declaration is the only new section. The structure shown in the figure above represents the basic structure for Nested Procedure. In comparison, Standalone Procedure includes the following additional keywords:

**CREATE OR REPLACE** PROCEDUREproc\_name

(par1 PLS\_INTEGER,

par2 OUT VARCHAR2,

par3 IN OUT VARCHAR2)

IS

.......

Procedure parameters' list is a list of zero or more parameters separated by comma "," that should defined as shown below:

Data type that **can't** include constraints

**COPY :** Default **NOCOPY**

Data Type

Passing Type

- **IN** : Default

- **OUT**

- **IN OUT**

Mode

Name

The Mode and Passing Type is optional but very important to determine the performance of the parameter. The mode determine for what purpose you want to use this parameter:

1. **IN**: you want to pass value from the invoker to the procedure.
2. **OUT**: you want to pass value from the procedure back to the invoker.
3. **IN OUT**: you want to pass a value from the invoker to the procedure and then back to the invoker.

There are two ways that you can pass parameter value from/to procedure:

1. **COPY** (pass **by value**)**:** The value is copied from the invoker variable (Actual Parameter) to the procedure parameter (Formal Parameter).
2. **NOCOPY** (pass **by reference**)**:** A pointer of the invoker variable (Actual Parameter) is given to the procedure parameter (Formal Parameter); so, both variables are referring to the same memory location.

The (**IN**) parameters are passed by reference and treated as CONSTANT inside the procedure; you are allowed to read not to change its value. IN mode is the default and you don't need to mention it explicitly. OUT parameter is initialized first to the default value of its type. By default, actual OUT parameter is passed by value; if you specify NOCOPY, it might be passed by reference. In the other hand, IN OUT parameter, the formal parameter acts like an initialized variable: When the subprogram begins, its value is that of its actual parameter. By default, actual IN OUT parameter is passed by value (in both directions); if you specify NOCOPY, it might be passed by reference.

Procedure can be called as PL/SQL statement executed. For example:

1. *proc\_name(par1, par2, .....);*
2. *CALL proc\_name(par1, par2, .....);*
3. *EXECUTE|EXEC proc\_name(par1, par2, .....);*

Please note: the first and second methods are be used to execute procedure from inside other PL/SQL block. The third method is used to execute procedure directly from SQL\*Plus console.

PL/SQL lets you overload nested subprograms, package subprograms, and type methods. You can use the same name for several different subprograms if their formal parameters differ in name, number, order, or data type family.

## AIM

The AIM of the following exercise is to demonstrate how to use Stored Procedure in PL/SQL.

The steps involved will include:

* Create, Call, and Remove Stored Procedure.
* Parameter Mode
* Overloading Procedures

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.

Estimated Completion Time:

25 minutes

# Lab Exercise 12: STORED PROCEDURE

|  |
| --- |
|  |

## Create, Call, and Remove Stored Procedure

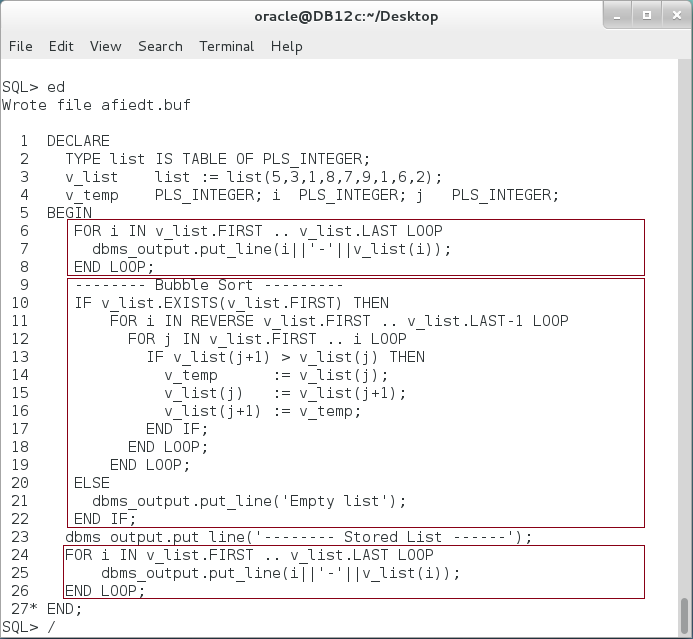
**Step 1:** Open the Terminal, open SQL\*Plus console and connect to hr schema.

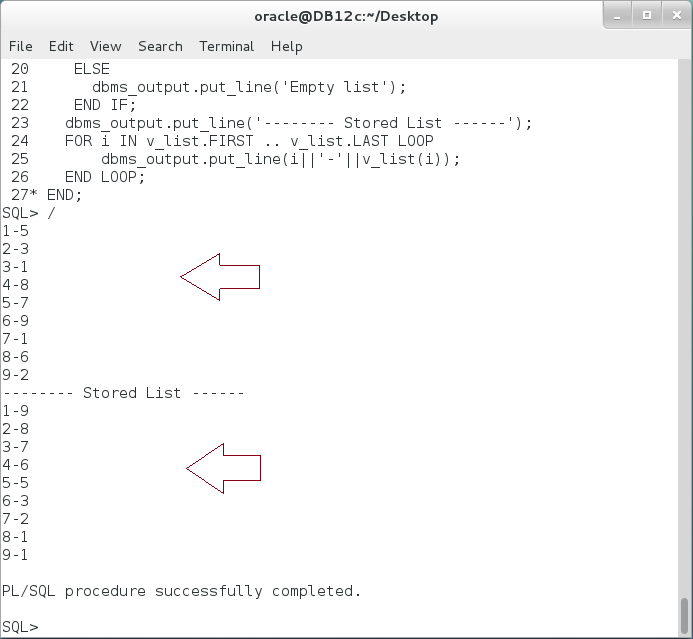
|  |  |
| --- | --- |
| Command | Description |
| sqlplus | Open SQL\*Plus console. |
| hr/oracle | connect to **hr** schema. |

****

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

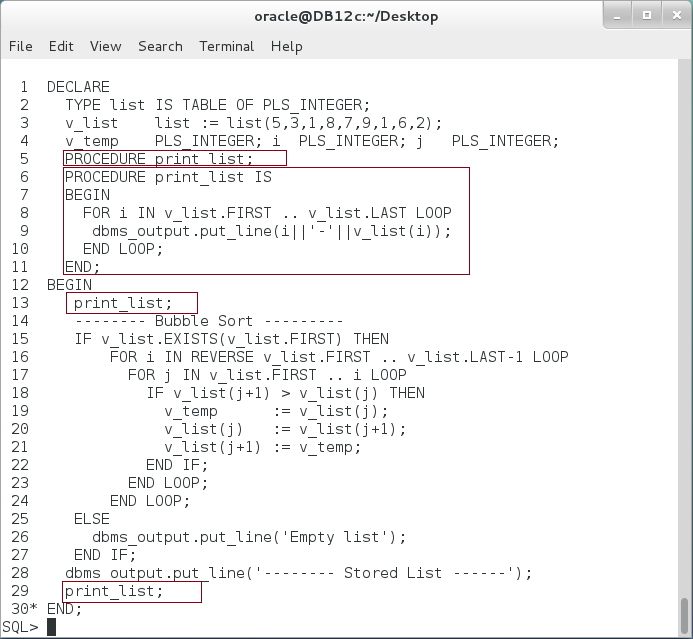
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| TYPE list IS TABLE OF PLS\_INTEGER; |  |
| v\_list list := list(5,3,1,8,7,9,1,6,2); |  |
| v\_temp PLS\_INTEGER; i PLS\_INTEGER; j PLS\_INTEGER; |  |
| BEGIN |  |
| FOR i IN v\_list.FIRST .. v\_list.LAST LOOP | **Print the list** |
| dbms\_output.put\_line(i||'-'||v\_list(i)); |
| END LOOP; |
| -------- Bubble Sort --------- |  |
| IF v\_list.EXISTS(v\_list.FIRST) THEN | Bubble Sort Algorithm |
| FOR i IN REVERSE v\_list.FIRST .. v\_list.LAST-1 LOOP |
| FOR j IN v\_list.FIRST .. i LOOP |
| IF v\_list(j+1) > v\_list(j) THEN |
| v\_temp := v\_list(j); |
| v\_list(j) := v\_list(j+1); |
| v\_list(j+1) := v\_temp; |
| END IF; |
| END LOOP; |
| END LOOP; |
| ELSE |
| dbms\_output.put\_line('Empty list'); |
| END IF; |
| dbms\_output.put\_line('-------- Stored List ------'); | Print the list gain. |
| FOR i IN v\_list.FIRST .. v\_list.LAST LOOP |
| dbms\_output.put\_line(i||'-'||v\_list(i)); |
| END LOOP; |
| END; |
| / |  |

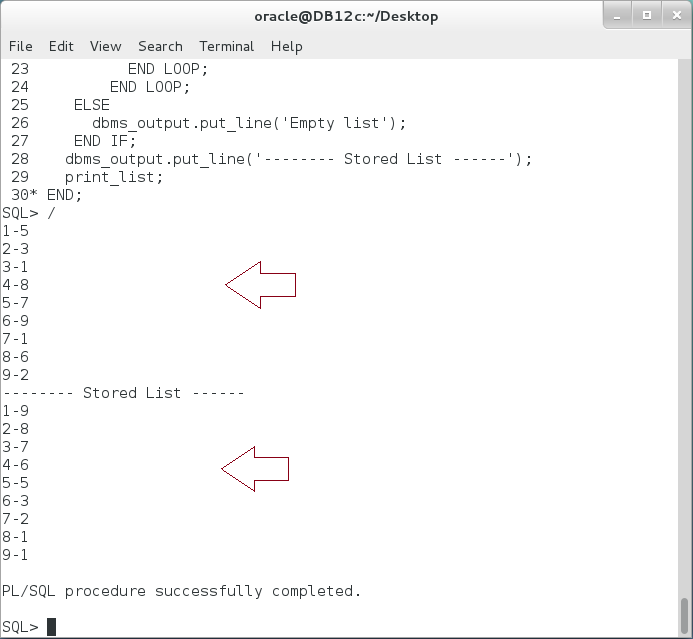
****

****

**Step 3:** In the previous, you may notice that you print the list twice. You may also need to reprint it again after any modification on the list. In such cases, it would be better to define it as Nested Procedure instead of rewrite it again. Modify the previous PL/SQL block as show below:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| TYPE list IS TABLE OF PLS\_INTEGER; |  |
| v\_list list := list(5,3,1,8,7,9,1,6,2); |  |
| v\_temp PLS\_INTEGER; i PLS\_INTEGER; j PLS\_INTEGER; |  |
| **PROCEDURE** print\_list; | **Declare procedure** |
| **PROCEDURE** print\_list **IS** | **Define procedure** |
| **BEGIN** |
| FOR i IN v\_list.FIRST .. v\_list.LAST LOOP |
| dbms\_output.put\_line(i||'-'||v\_list(i)); |
| END LOOP; |
| **END**; |
| BEGIN |  |
| **print\_list;** | **Call procedure** as statement |
| -------- Bubble Sort --------- |  |
| IF v\_list.EXISTS(v\_list.FIRST) THEN |  |
| FOR i IN REVERSE v\_list.FIRST .. v\_list.LAST-1 LOOP |  |
| FOR j IN v\_list.FIRST .. i LOOP |  |
| IF v\_list(j+1) > v\_list(j) THEN |  |
| v\_temp := v\_list(j); |  |
| v\_list(j) := v\_list(j+1); |  |
| v\_list(j+1) := v\_temp; |  |
| END IF; |  |
| END LOOP; |  |
| END LOOP; |  |
| ELSE |  |
| dbms\_output.put\_line('Empty list'); |  |
| END IF; |  |
| dbms\_output.put\_line('-------- Stored List ------'); |  |
| **print\_list;** | **Call procedure** as statement |
| END; |  |
| / |  |

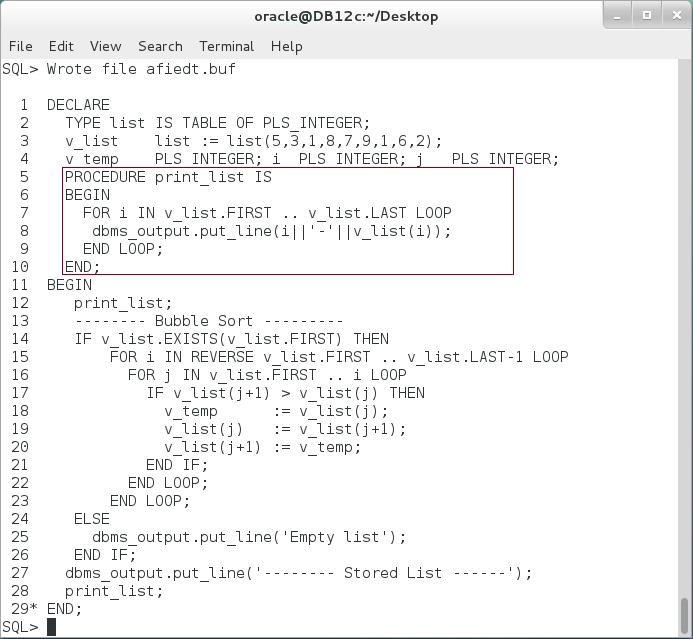


****

**Please note**: the Nested Procedure can access the block variable. In our case, v\_list nested table is accessed from the Nested Procedure.

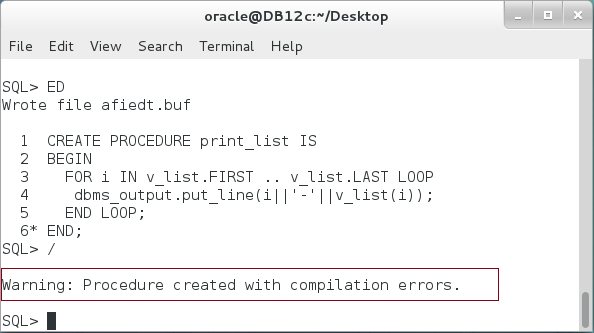
**Step 4:** In previous block, the procedure is declared and then defined. You may declare and define at once as shown in the next block:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| TYPE list IS TABLE OF PLS\_INTEGER; |  |
| v\_list list := list(5,3,1,8,7,9,1,6,2); |  |
| v\_temp PLS\_INTEGER; i PLS\_INTEGER; j PLS\_INTEGER; |  |
| **PROCEDURE** print\_list **IS** | **Declare and Define procedure** |
| **BEGIN** |
| FOR i IN v\_list.FIRST .. v\_list.LAST LOOP |
| dbms\_output.put\_line(i||'-'||v\_list(i)); |
| END LOOP; |
| **END**; |
| BEGIN |  |
| **print\_list;** | **Call procedure** as statement |
| -------- Bubble Sort --------- |  |
| IF v\_list.EXISTS(v\_list.FIRST) THEN |  |
| FOR i IN REVERSE v\_list.FIRST .. v\_list.LAST-1 LOOP |  |
| FOR j IN v\_list.FIRST .. i LOOP |  |
| IF v\_list(j+1) > v\_list(j) THEN |  |
| v\_temp := v\_list(j); |  |
| v\_list(j) := v\_list(j+1); |  |
| v\_list(j+1) := v\_temp; |  |
| END IF; |  |
| END LOOP; |  |
| END LOOP; |  |
| ELSE |  |
| dbms\_output.put\_line('Empty list'); |  |
| END IF; |  |
| dbms\_output.put\_line('-------- Stored List ------'); |  |
| **print\_list;** | **Call procedure** as statement |
| END; |  |
| / |  |

** Please note:** Nested Procedure is not stored in the database; no other user can see or use this procedure. It is just away to organize your coding.

**Step 5:** To create a stored procedure to do the same job as the previous Nested Procedure. Execute the following DDL statement:

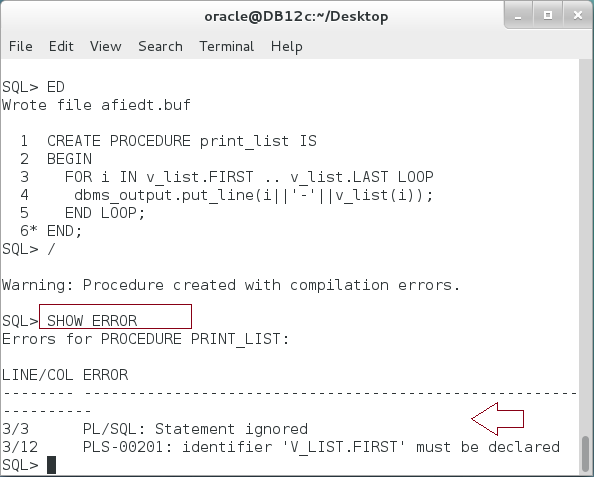
|  |  |
| --- | --- |
| Command | Description |
| **CREATE PROCEDURE** print\_list **IS** | **Create Stored Procedure**. |
| **BEGIN** |
| FOR i IN v\_list.FIRST .. v\_list.LAST LOOP |
| dbms\_output.put\_line(i||'-'||v\_list(i)); |
| END LOOP; |
| **END**; |



**Please note**: The Stored Procedure create but with compilation errors! **How to view those errors?**

**Step 6:** To view the error, execute the following command:

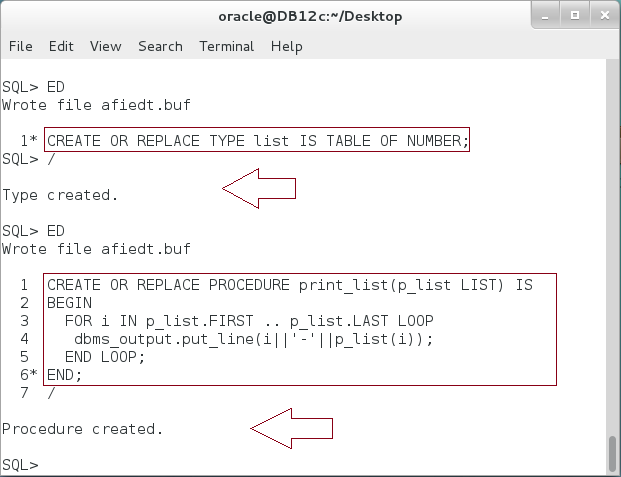
|  |  |
| --- | --- |
| Command | Description |
| SHOW ERROR | Display the error message. |



**Please Note:** The message indicates that V\_LIST is not declared. You should either declare an inner variable or add parameter to the procedure to hold this variable. In our case, we should add parameter. **How we can modify a previously created procedure?**

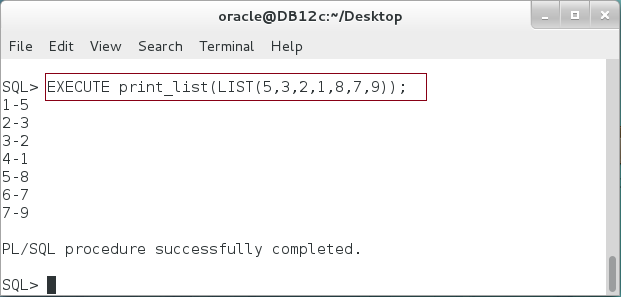
**Step 7:** To modify a procedure, execute the following DDL statements:

|  |  |
| --- | --- |
| Command | Description |
| **CREATE OR REPLACE TYPE list IS TABLE OF NUMBER;** | Create stored collection. |
| CREATE **OR REPLACE** PROCEDURE print\_list(**p\_list LIST**) IS | Alter the procedure. |
| BEGIN |
| FOR i IN p\_list.FIRST .. p\_list.LAST LOOP |
| dbms\_output.put\_line(i||'-'||p\_list(i)); |
| END LOOP; |
| END; |

****

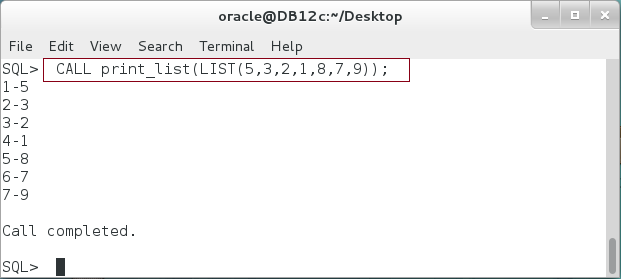
**Step 8:** You can execute the stored procedure from the SQL\*Plus as shown below:

|  |  |
| --- | --- |
| Command | Description |
| EXECUTE print\_list(LIST(5,3,2,1,8,7,9)); | Use EXECUTE command. |

****

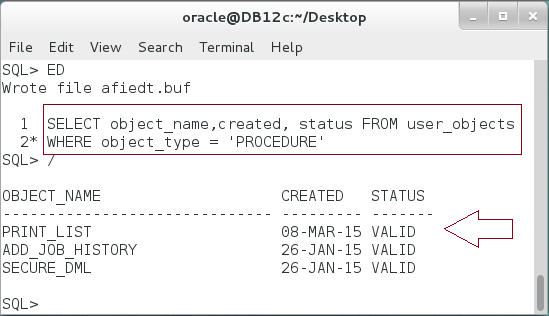
**Step 9:** You can also execute the stored procedure from the SQL\*Plus as shown below:

|  |  |
| --- | --- |
| Command | Description |
| **CALL** print\_list(LIST(5,3,2,1,8,7,9)); | Use **CALL** command. |

****

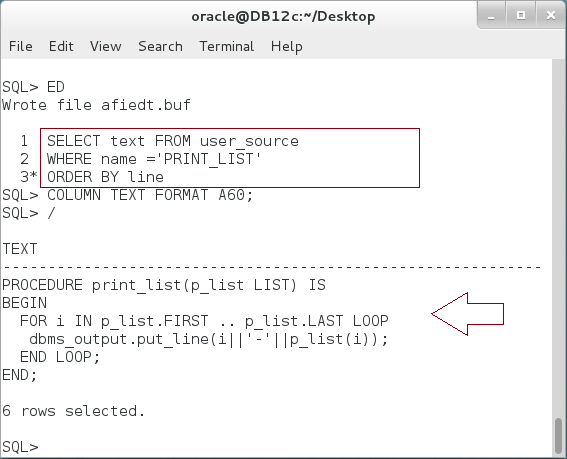
**Step 10:** You may worry how to know about what procedures you have in your schema. To list all procedures, execute the following query:

|  |  |
| --- | --- |
| Command | Description |
| SELECT object\_name,created, status FROM user\_objects  WHERE object\_type = **'PROCEDURE'** | View all procedures in your schema. |

****

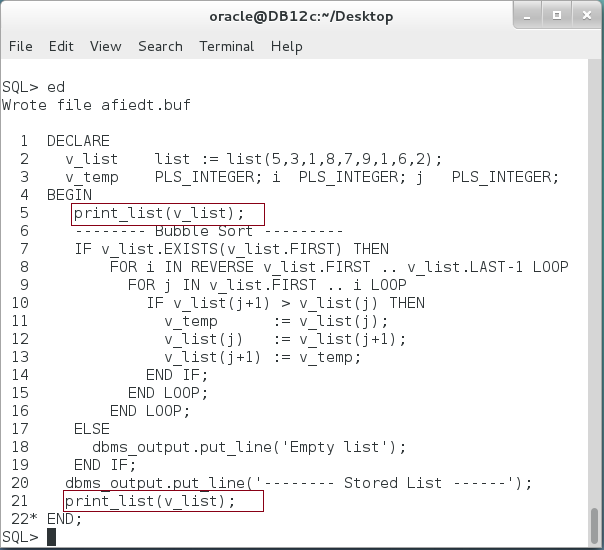
**Step 11:** If you interested in viewing the procedure source code, execute the following query:

|  |  |
| --- | --- |
| Command | Description |
| SELECT text FROM user\_source | Query the source code of the procedure. |
| WHERE name =**'PRINT\_LIST**' |
| ORDER BY line |

****

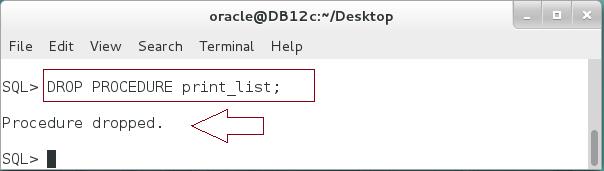
**Step 12:** You can now use the stored procedure in the first PL/SQL block as shown below:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| v\_list list := list(5,3,1,8,7,9,1,6,2); |  |
| v\_temp PLS\_INTEGER; i PLS\_INTEGER; j PLS\_INTEGER; |  |
| BEGIN |  |
| **print\_list(v\_list);** | **Call procedure** as statement |
| -------- Bubble Sort --------- |  |
| IF v\_list.EXISTS(v\_list.FIRST) THEN |  |
| FOR i IN REVERSE v\_list.FIRST .. v\_list.LAST-1 LOOP |  |
| FOR j IN v\_list.FIRST .. i LOOP |  |
| IF v\_list(j+1) > v\_list(j) THEN |  |
| v\_temp := v\_list(j); |  |
| v\_list(j) := v\_list(j+1); |  |
| v\_list(j+1) := v\_temp; |  |
| END IF; |  |
| END LOOP; |  |
| END LOOP; |  |
| ELSE |  |
| dbms\_output.put\_line('Empty list'); |  |
| END IF; |  |
| dbms\_output.put\_line('-------- Stored List ------'); |  |
| **print\_list(v\_list);** | **Call procedure** as statement |
| END; |  |
| / |  |

****

**Step 13:** To drop a Stored Procedure, execute the following DDL statement:

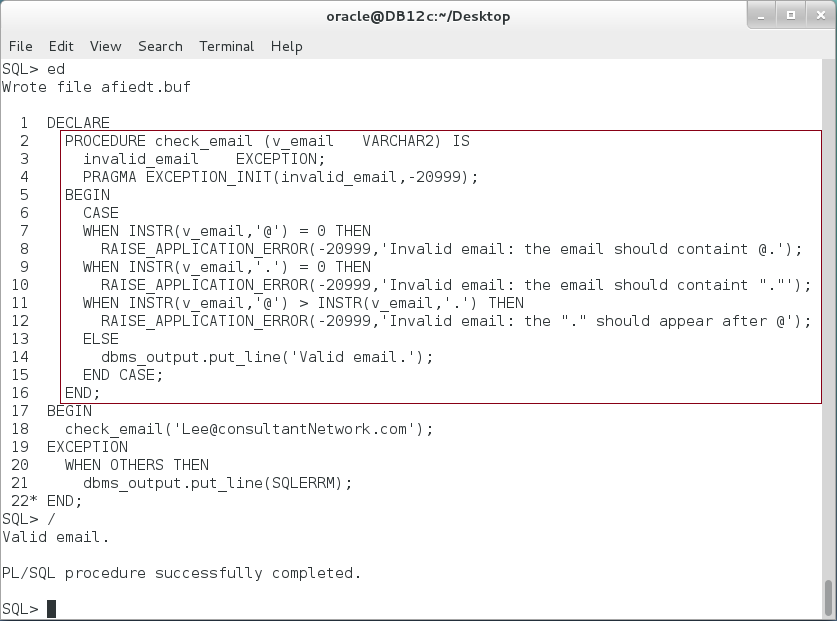
|  |  |
| --- | --- |
| Command | Description |
| **DROP PROCEDURE** print\_list; | Drop Stored Procedure. |

****

## Parameter Mode

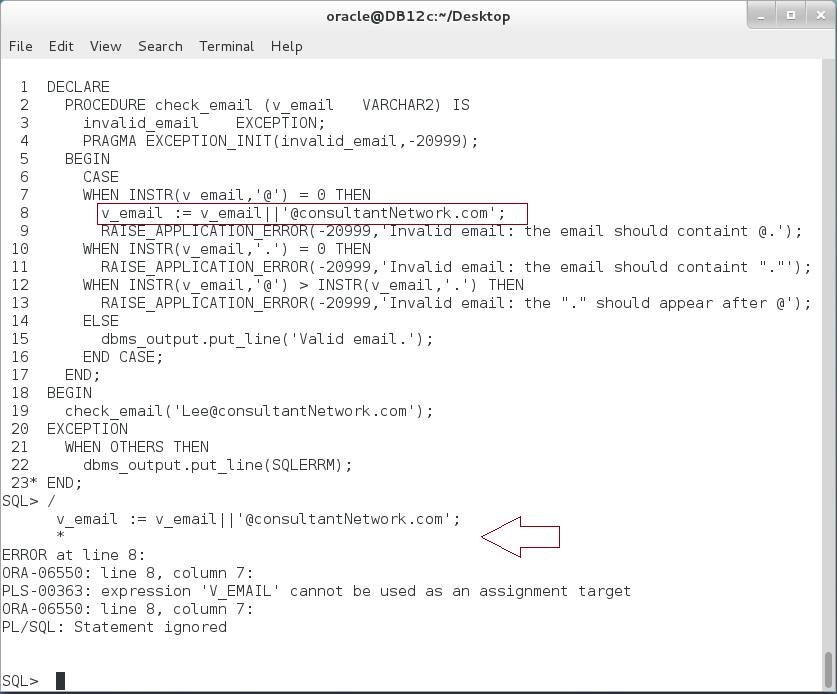
**Step 1:** Execute the following PL/SQL block:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| **PROCEDURE check\_email (v\_email VARCHAR2) IS** | **Declare and Define** nested procedure chekc\_email. |
| invalid\_email EXCEPTION; |
| PRAGMA EXCEPTION\_INIT(invalid\_email,-20999); |
| **BEGIN** |
| CASE |
| WHEN INSTR(v\_email,'@') = 0 THEN |
| RAISE\_APPLICATION\_ERROR  (-20999,'Invalid email: the email should containt @.'); |
| WHEN INSTR(v\_email,'.') = 0 THEN |
| RAISE\_APPLICATION\_ERROR  (-20999,'Invalid email: the email should containt "."'); |
| WHEN INSTR(v\_email,'@') > INSTR(v\_email,'.') THEN |
| RAISE\_APPLICATION\_ERROR  (-20999,'Invalid email: the "." should appear after @'); |
| ELSE |
| dbms\_output.put\_line('Valid email.'); |
| END CASE; |
| **END**; |
| BEGIN |  |
| check\_email('Lee@consultantNetwork.com'); | Use the procedure |
| EXCEPTION |  |
| WHEN OTHERS THEN |  |
| dbms\_output.put\_line(SQLERRM); |  |
| END; |  |
| / |  |



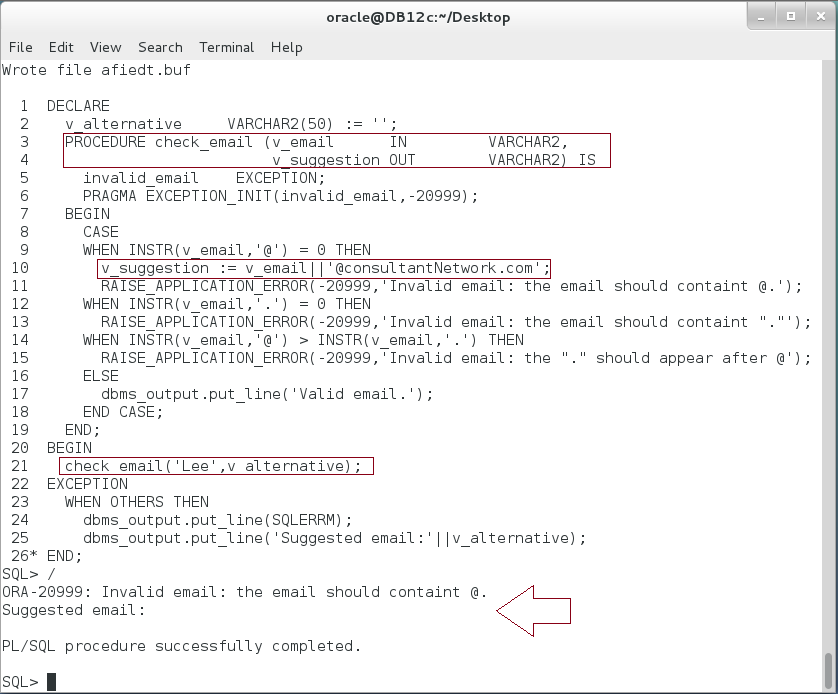
**Step 2:** In the previous block, check\_email nested procedure has one parameter. By default, it is acts as IN parameter. Modify the previous block so the procedure return reasonable suggestions for valid email.

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| **PROCEDURE** check\_email (**v\_email VARCHAR2**) **IS** | You can't assign value to **IN** parameter. It acts as CONSTANT. |
| invalid\_email EXCEPTION; |
| PRAGMA EXCEPTION\_INIT(invalid\_email,-20999); |
| **BEGIN** |
| CASE |
| WHEN INSTR(v\_email,'@') = 0 THEN |
| **v\_email := v\_email||'@consultantNetwork.com';** |
| RAISE\_APPLICATION\_ERROR  (-20999,'Invalid email: the email should containt @.'); |
| WHEN INSTR(v\_email,'.') = 0 THEN |
| RAISE\_APPLICATION\_ERROR  (-20999,'Invalid email: the email should containt "."'); |
| WHEN INSTR(v\_email,'@') > INSTR(v\_email,'.') THEN |
| RAISE\_APPLICATION\_ERROR  (-20999,'Invalid email: the "." should appear after @'); |
| ELSE |
| dbms\_output.put\_line('Valid email.'); |
| END CASE; |
| **END**; |  |
| BEGIN | Use the procedure |
| check\_email('Lee@consultantNetwork.com'); |  |
| EXCEPTION |  |
| WHEN OTHERS THEN |  |
| dbms\_output.put\_line(SQLERRM); |  |
| END; |  |
| / |  |

****

**Step 3:** You may now think of adding new **OUT** variable. Thus, this variable represents a logical alternative to what you enter as email. Modify the previous block as show below:

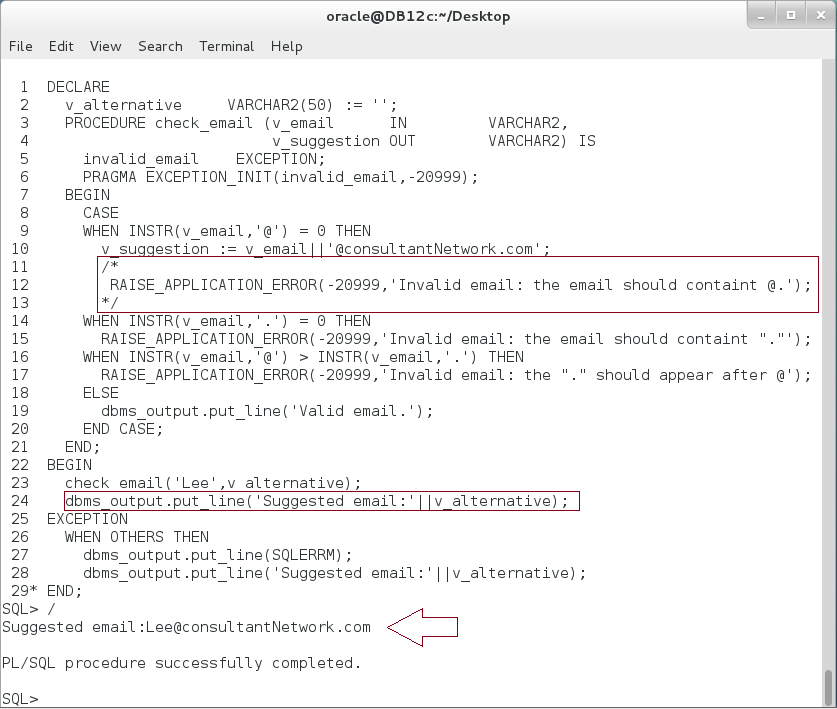
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| v\_alternative VARCHAR2(50) := ''; |  |
| **PROCEDURE check\_email (v\_email IN VARCHAR2,** |  |
| **v\_suggestion OUT VARCHAR2**) IS | Add an **OUT** parameter |
| invalid\_email EXCEPTION; |  |
| PRAGMA EXCEPTION\_INIT(invalid\_email,-20999); |  |
| **BEGIN** |  |
| CASE |  |
| WHEN INSTR(v\_email,'@') = 0 THEN |  |
| **v\_suggestion:= v\_email||'@consultantNetwork.com';** |  |
| RAISE\_APPLICATION\_ERROR  (-20999,'Invalid email: the email should containt @.'); |  |
| WHEN INSTR(v\_email,'.') = 0 THEN |  |
| RAISE\_APPLICATION\_ERROR  (-20999,'Invalid email: the email should containt "."'); |  |
| WHEN INSTR(v\_email,'@') > INSTR(v\_email,'.') THEN |  |
| RAISE\_APPLICATION\_ERROR  (-20999,'Invalid email: the "." should appear after @'); |  |
| ELSE |  |
| dbms\_output.put\_line('Valid email.'); |  |
| END CASE; |  |
| END; |  |
| BEGIN |  |
| **check\_email('Lee',v\_alternative);** |  |
| EXCEPTION |  |
| WHEN OTHERS THEN |  |
| dbms\_output.put\_line(SQLERRM); |  |
| dbms\_output.put\_line  ('Suggested email:'||v\_alternative); |  |
| END; |  |
| / |  |



**Opps:** The suggested email does not appear in the output. This means the invoker variable (Actual Parameter) : v\_alternative does not get the value of procedure parameter (Formal Parameter): v\_suggestion. Please note: By default, Oracle passes OUT parameter by value and at the end of the procedure. If the procedure failed by exception, OUT parameter does not get the value.

**Step 4:** Try to stop raising the exception and handle the OUT parameter. Execute the following block:

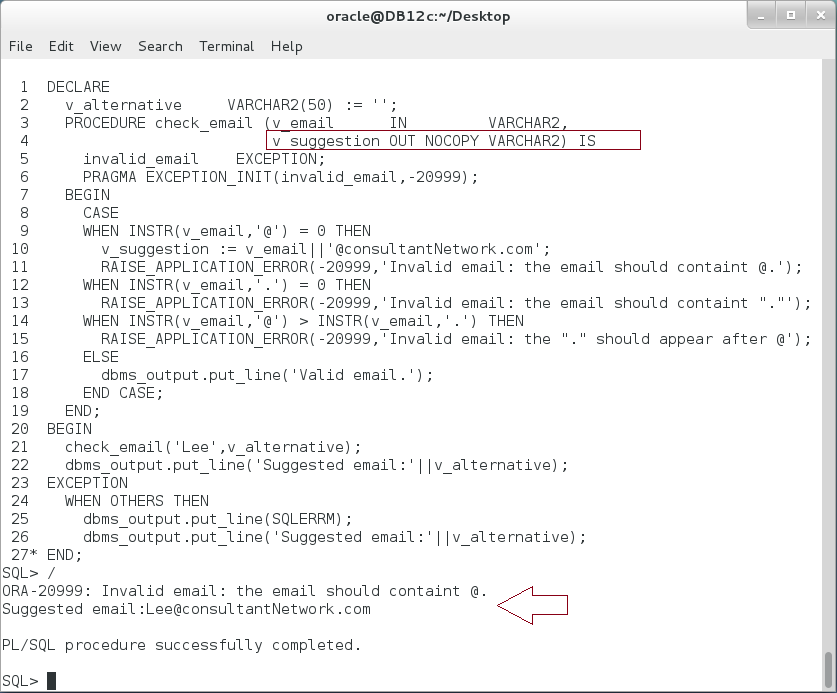
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| v\_alternative VARCHAR2(50) := ''; |  |
| **PROCEDURE check\_email (v\_email IN VARCHAR2,** |  |
| **v\_suggestion OUT VARCHAR2**) IS | Add an **OUT** parameter |
| invalid\_email EXCEPTION; |  |
| PRAGMA EXCEPTION\_INIT(invalid\_email,-20999); |  |
| **BEGIN** |  |
| CASE |  |
| WHEN INSTR(v\_email,'@') = 0 THEN |  |
| **v\_suggestion:= v\_email||'@consultantNetwork.com';** |  |
| **/\* RAISE\_APPLICATION\_ERROR**  **(-20999,'Invalid email: the email should containt @.');  \*/** | Comment the exception |
| WHEN INSTR(v\_email,'.') = 0 THEN |  |
| RAISE\_APPLICATION\_ERROR  (-20999,'Invalid email: the email should containt "."'); |  |
| WHEN INSTR(v\_email,'@') > INSTR(v\_email,'.') THEN |  |
| RAISE\_APPLICATION\_ERROR  (-20999,'Invalid email: the "." should appear after @'); |  |
| ELSE |  |
| dbms\_output.put\_line('Valid email.'); |  |
| END CASE; |  |
| END; |  |
| BEGIN |  |
| **check\_email('Lee',v\_alternative);** |  |
| **dbms\_output.put\_line   ('Suggested email:'||v\_alternative);** |  |
| EXCEPTION |  |
| WHEN OTHERS THEN |  |
| dbms\_output.put\_line(SQLERRM); |  |
| dbms\_output.put\_line  ('Suggested email:'||v\_alternative); |  |
| END; |  |
| / |  |

****

**Please note:** OUT parameter is copied to the Actual Parameter, since the block executed without error. But this certainly is not what you want. The email "Lee" is not a valid email and the code should raise an exception!!

**Step 5:** Undo previous modifications and add NOCOPY hint to OUT parameter. Execute the following block:

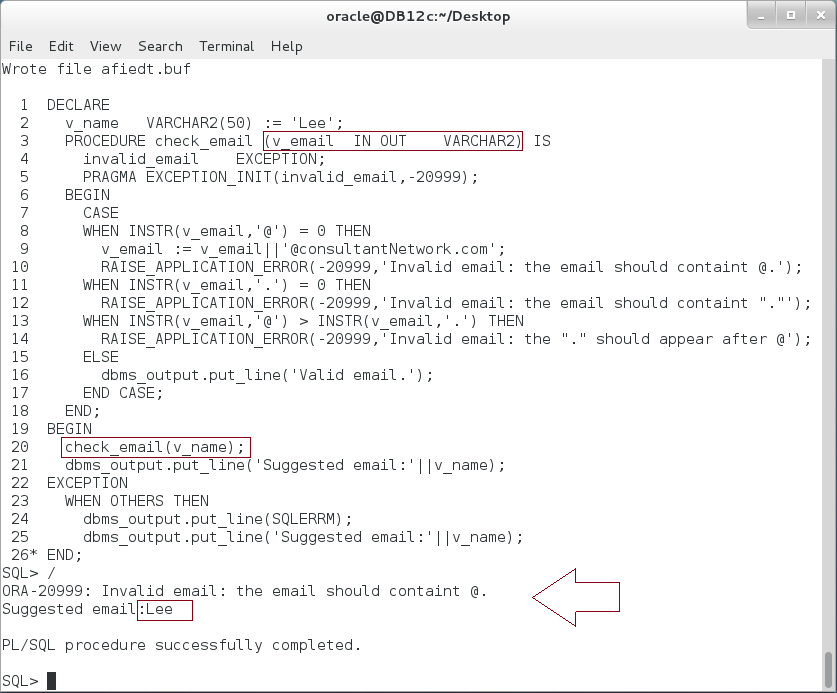
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| v\_alternative VARCHAR2(50) := ''; |  |
| **PROCEDURE** check\_email (v\_email IN VARCHAR2, | Add NOCOPY hint. |
| **v\_suggestion OUT NOCOPY VARCHAR2**) IS |
| invalid\_email EXCEPTION; |
| PRAGMA EXCEPTION\_INIT(invalid\_email,-20999); |  |
| **BEGIN** |  |
| CASE |  |
| WHEN INSTR(v\_email,'@') = 0 THEN |  |
| **v\_suggestion:= v\_email||'@consultantNetwork.com';** |  |
| RAISE\_APPLICATION\_ERROR  (-20999,'Invalid email: the email should containt @.'); |  |
| WHEN INSTR(v\_email,'.') = 0 THEN |  |
| RAISE\_APPLICATION\_ERROR  (-20999,'Invalid email: the email should containt "."'); |  |
| WHEN INSTR(v\_email,'@') > INSTR(v\_email,'.') THEN |  |
| RAISE\_APPLICATION\_ERROR  (-20999,'Invalid email: the "." should appear after @'); |  |
| ELSE |  |
| dbms\_output.put\_line('Valid email.'); |  |
| END CASE; |  |
| END; |  |
| BEGIN |  |
| **check\_email('Lee',v\_alternative);** |  |
| EXCEPTION |  |
| WHEN OTHERS THEN |  |
| dbms\_output.put\_line(SQLERRM); |  |
| dbms\_output.put\_line  ('Suggested email:'||v\_alternative); |  |
| END; |  |
| / |  |



**Please note:** The procedure acts exactly as expected; it raised exception when the email is not valid and return a suggestion at the same time. NOCOPY hint tells the compiler make both the actual and formal parameter referring to the same memory location.

**Step 6:** If it is logical, you may use the same variable to pass value from and to the procedure. :

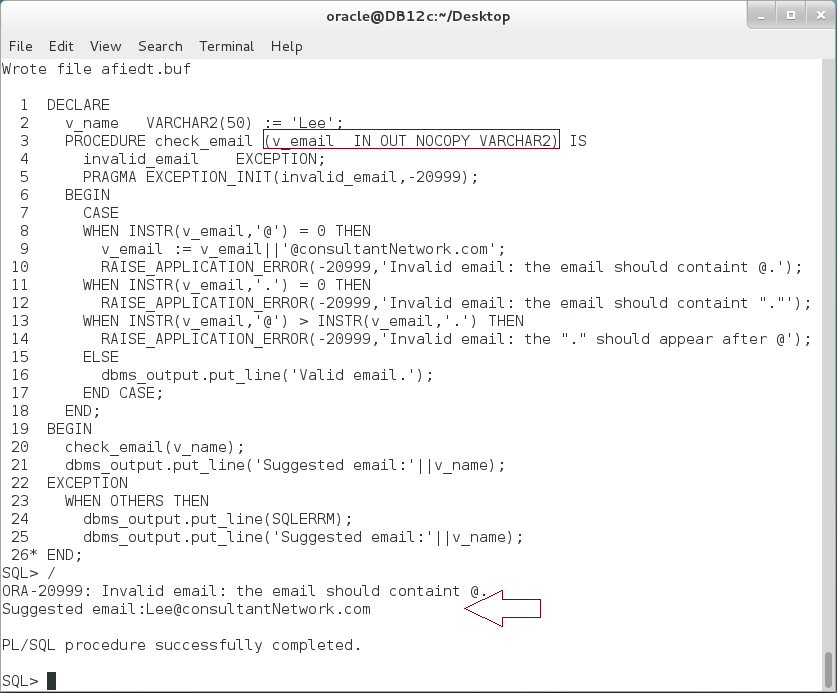
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| **v\_name VARCHAR2(50) := 'Lee';** |  |
| **PROCEDURE** check\_email (**v\_email IN OUT VARCHAR2**) IS | use **IN OUT** variable instead of two variables. |
| invalid\_email EXCEPTION; |
| PRAGMA EXCEPTION\_INIT(invalid\_email,-20999); |  |
| **BEGIN** |  |
| CASE |  |
| WHEN INSTR(v\_email,'@') = 0 THEN |  |
| **v\_email:= v\_email||'@consultantNetwork.com';** |  |
| RAISE\_APPLICATION\_ERROR  (-20999,'Invalid email: the email should containt @.'); |  |
| WHEN INSTR(v\_email,'.') = 0 THEN |  |
| RAISE\_APPLICATION\_ERROR  (-20999,'Invalid email: the email should containt "."'); |  |
| WHEN INSTR(v\_email,'@') > INSTR(v\_email,'.') THEN |  |
| RAISE\_APPLICATION\_ERROR  (-20999,'Invalid email: the "." should appear after @'); |  |
| ELSE |  |
| dbms\_output.put\_line('Valid email.'); |  |
| END CASE; |  |
| END; |  |
| BEGIN |  |
| **check\_email(v\_name);** |  |
| EXCEPTION |  |
| WHEN OTHERS THEN |  |
| dbms\_output.put\_line(SQLERRM); |  |
| dbms\_output.put\_line  ('Suggested email:'||**v\_name**); |  |
| END; |  |
| / |  |

****

**Please note:** In the previous block, the variable "v\_name" should printed a suggested email. However, it prints "Lee". **Explain?**

**Step 7:** Modify the previous block as show below:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| v\_name VARCHAR2(50) := 'Lee'; |  |
| **PROCEDURE** check\_email  (**v\_email IN OUT NOCOPY VARCHAR2**) IS | use **IN OUT** variable with **NOCOPY** hint |
| invalid\_email EXCEPTION; |
| PRAGMA EXCEPTION\_INIT(invalid\_email,-20999); |  |
| BEGIN |  |
| CASE |  |
| WHEN INSTR(v\_email,'@') = 0 THEN |  |
| v\_email:= v\_email||'@consultantNetwork.com'; |  |
| RAISE\_APPLICATION\_ERROR  (-20999,'Invalid email: the email should containt @.'); |  |
| WHEN INSTR(v\_email,'.') = 0 THEN |  |
| RAISE\_APPLICATION\_ERROR  (-20999,'Invalid email: the email should containt "."'); |  |
| WHEN INSTR(v\_email,'@') > INSTR(v\_email,'.') THEN |  |
| RAISE\_APPLICATION\_ERROR  (-20999,'Invalid email: the "." should appear after @'); |  |
| ELSE |  |
| dbms\_output.put\_line('Valid email.'); |  |
| END CASE; |  |
| END; |  |
| BEGIN |  |
| check\_email(v\_name); |  |
| EXCEPTION |  |
| WHEN OTHERS THEN |  |
| dbms\_output.put\_line(SQLERRM); |  |
| dbms\_output.put\_line  ('Suggested email:'||**v\_name**); |  |
| END; |  |
| / |  |

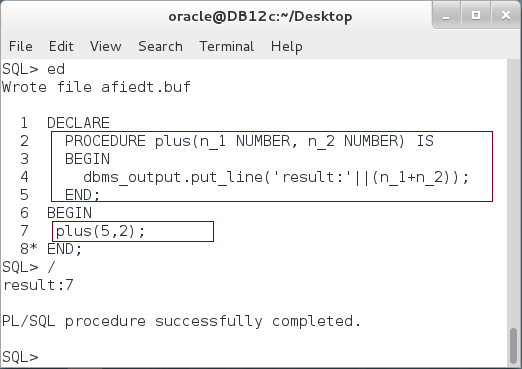
****

**Explain the result.**

## Overloading Procedures

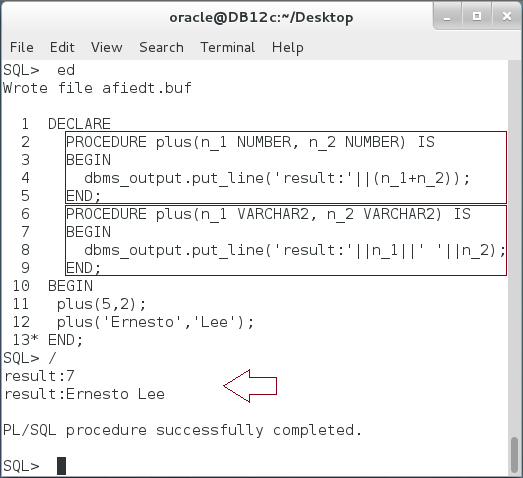
**Step 1:** PL/SQL lets you overload nested subprograms, package subprograms, and type methods. You can use the same name for several different subprograms if their formal parameters differ in name, number, order, or data type family. Execute the following block:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| **PROCEDURE plus(n\_1 NUMBER, n\_2 NUMBER) IS** | Nested Procedure. |
| BEGIN |
| dbms\_output.put\_line('result:'||(n\_1+n\_2)); |
| END; |  |
| BEGIN |  |
| plus(5,2); |  |
| END; |  |
| / |  |

****

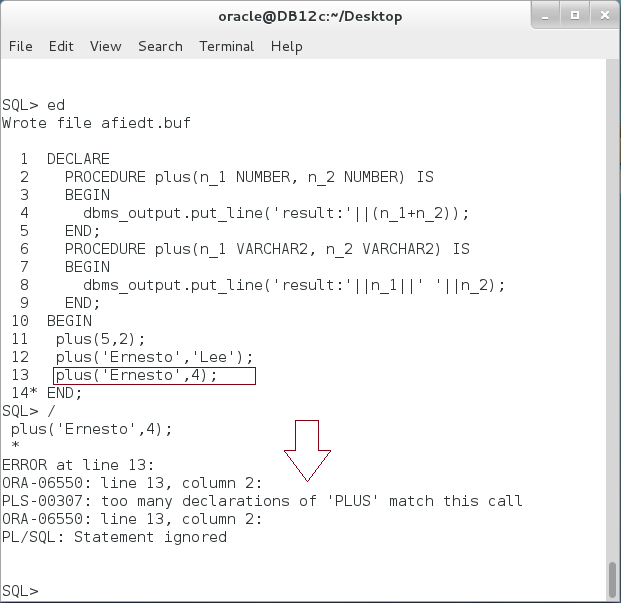
**Step 2:** You want to extend the previous block so PLUS procedure may also used for VARCHAR2 data types. Execute the following block:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| **PROCEDURE plus(n\_1 NUMBER, n\_2 NUMBER) IS** | Nested Procedure. |
| BEGIN |
| dbms\_output.put\_line('result:'||(n\_1+n\_2)); |
| END; |  |
| **PROCEDURE plus(n\_1 VARCHAR2, n\_2 VARCHAR2) IS** | Add an overloading for "**plus**" procedure. |
| BEGIN |
| dbms\_output.put\_line('result:'||n\_1||' '||n\_2); |
| END; |
| BEGIN |  |
| plus(5,2); | The compiler automatically detects which procedure should run the statement |
| **plus('Ernesto','Lee');** |
| END; |
| / |

****

**Step 3:** You want to extend the previous block so PLUS procedure may also used for VARCHAR2 data types. Execute the following block:

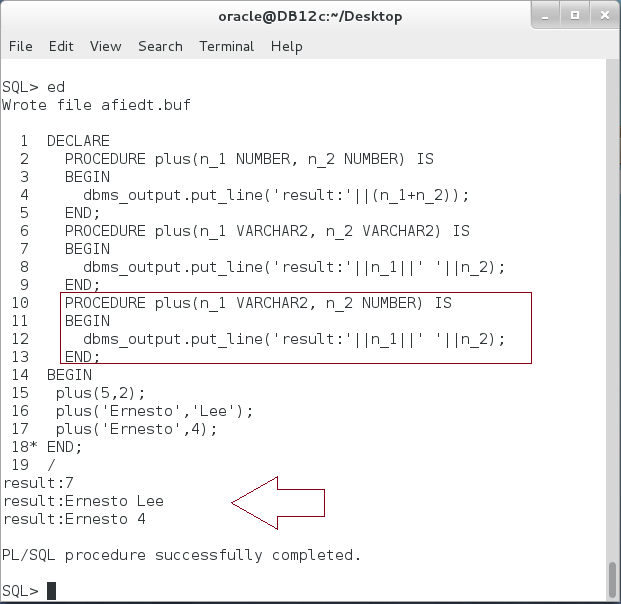
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| **PROCEDURE plus(n\_1 NUMBER, n\_2 NUMBER) IS** | Nested Procedure. |
| BEGIN |
| dbms\_output.put\_line('result:'||(n\_1+n\_2)); |
| END; |  |
| **PROCEDURE plus(n\_1 VARCHAR2, n\_2 VARCHAR2) IS** | Add an overloading for "**plus**" procedure. |
| BEGIN |
| dbms\_output.put\_line('result:'||n\_1||' '||n\_2); |
| END; |
| BEGIN |  |
| plus(5,2); | The compiler failed to find a corresponding plus procedure. |
| plus('Ernesto','Lee'); |
| **plus('Ernesto',4);** |
| END; |
| / |

****

**Please note:** The error message may mislead you. **Is there too many declarations that math this call?**

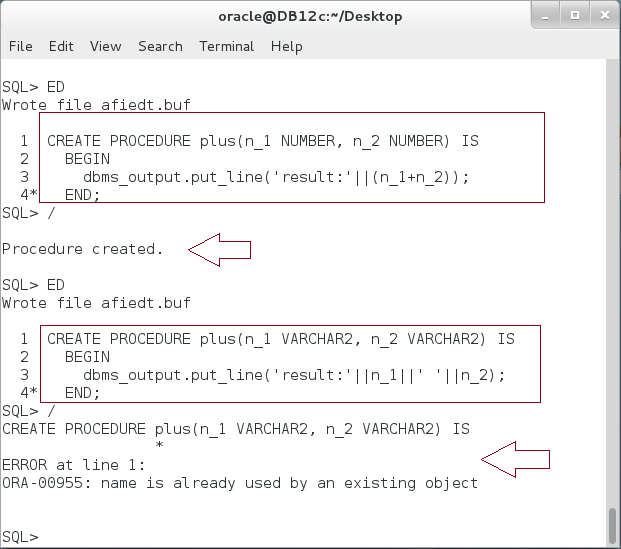
**Step 4:** Extend the previous block as shown below:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| **PROCEDURE plus(n\_1 NUMBER, n\_2 NUMBER) IS** | Nested Procedure. |
| BEGIN |
| dbms\_output.put\_line('result:'||(n\_1+n\_2)); |
| END; |  |
| **PROCEDURE plus(n\_1 VARCHAR2, n\_2 VARCHAR2) IS** | Add an overloading for "**plus**" procedure. |
| BEGIN |
| dbms\_output.put\_line('result:'||n\_1||' '||n\_2); |
| END; |
| **PROCEDURE plus(n\_1 VARCHAR2, n\_2 NUMBER) IS** |  |
| BEGIN |  |
| dbms\_output.put\_line('result:'||n\_1||' '||n\_2); |  |
| END; |  |
| BEGIN |  |
| plus(5,2); |  |
| plus('Ernesto','Lee'); |
| **plus('Ernesto',4);** |
| END; |
| / |

****

**Step 5:** Try to create standalone procedures instead of nested procedures:

|  |  |
| --- | --- |
| Command | Description |
| **CREATE PROCEDURE plus**(n\_1 NUMBER, n\_2 NUMBER) IS | Create first plus Stored procedure. |
| BEGIN |
| dbms\_output.put\_line('result:'||(n\_1+n\_2)); |
| END; |
| **CREATE PROCEDURE plus**(n\_1 VARCHAR2, n\_2 VARCHAR2) IS | The creation of the second plus stored procedure **failed**. |
| BEGIN |
| dbms\_output.put\_line('result:'||n\_1||' '||n\_2); |
| END; |

****

**Please note:** Oracle does not support overloading for standalone procedure. **Drop 'plus' procedure.**

# SUMMARY

Store procedure is the best way to store a repeated action which could be used among different uses. The Stored Procedure is like the anonymous block; it contains declaration, Executable, and Exception-Handling parts. However, it also include a Procedure Definition/Declaration part in which you specify its name and parameters list. There are three main modes you may exchange data with Stored Procedure: using IN, OUT, and IN OUT parameter. You use IN parameter to pass a variable from the invoker to the stored procedure, use OUT parameter to pass variable from the procedure to the invoker, and use IN OUT parameter to pass variable from the invoker to the procedure and then from the procedure back to the invoker. IN parameter is always passed by reference, that is, a pointer of the invoker variable is sent to the procedure variable. Thus, after that, both are referring to the same memory location. In contrast, IN and IN OUT are passed by value by default. You may change the default using NOCOPY keyword to the variable by reference. Oracle supports procedure overloading. Thus, you may use a procedure same name to refer to different stored procedures.

After completing this lab exercise, you should be able to use Stored Procedure.

# REFERENCES

* http://docs.oracle.com/database/121/LNPLS/subprograms.htm#LNPLS008
* http://docs.oracle.com/cd/B19306\_01/server.102/b14200/statements\_4008.htm#BABDEHHG
* http://www.oracle.com/webfolder/technetwork/tutorials/obe/db/sqldev/r30/plsql\_debug\_OBE/plsql\_debug\_otn.htm

# INDEX

Object Type 2

Package 2

Predefined Exception 7

Stored Function 2

Stored Procedure 2, 5