Table of Contents

[CHAPTER 13: STORED FUNCTION 2](#_Toc416721475)

[Theory 2](#_Toc416721476)

[AIM 4](#_Toc416721477)

[Lab Exercise 13: STORED FUNCTION 5](#_Toc416721478)

[1. Create, Call, and Remove Stored Procedure 6](#_Toc416721479)

[2. Use Functions in SQL 22](#_Toc416721480)

[3. Recursive Function 33](#_Toc416721481)

[SUMMARY 36](#_Toc416721482)

[REFERENCES 37](#_Toc416721483)

[INDEX 38](#_Toc416721484)

# CHAPTER 13: STORED FUNCTION

## Theory

Stored Function is much like Stored Procedure. However, a Stored Function is used not perform action but to compute and return value.

The basic structure of a function is show below:

**Executable part**

**Declaration part**

**Function Declaration**

**Exception-Handling part**

**FUNCTION** function\_name

(par1 PLS\_INTEGER,

par2 VARCHAR2) **RETURN NUMBER**

**IS**

v\_var1 VARCHAR2(50);

v\_var2 NUMBER;

**BEGIN**

....

**RETURN v\_var2;**

....

**EXCEPTION**

WHEN .... THEN

......

**END**;

The basic structure of the function is very similar to the procedure structure but it has one additional element: RETURN part. In Function Declaration part, you must mention the data type (without constraint) that the function should return. Inside Executable Part, you must guarantee to return value in every path the function can follow.

In contrast to Procedure, the Function can be executed as expression not statement. For example:

1. var := *function\_name();*
2. 5\*3+*function\_name()*
3. SELECT emp\_name, *function\_name()*, ... INTO ..

The structure shown in the figure above represents the basic structure for Nested Function. In comparison, Standalone Function includes the following additional keywords:

**CREATE OR REPLACE** FUNCTIONfunc\_name

(par1 PLS\_INTEGER,

par2 VARCHAR2)

RETURN NUMBER

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

Oracle highly recommend **not to use** OUT or IN OUT parameter mode in Function. A function should return a single value and it may be used in expression!

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. Remember, standalone function can't be overloaded.

PL/SQL supports recursive function, in which the function calls itself. In recursive subprograms, it must be at least two tracks: (1) calls itself, (2) ends the execution.

## AIM

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

The steps involved will include:

* Create, Call, and Remove Stored Procedure.
* Use Functions in SQL
* Recursive Function

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

Estimated Completion Time:

25 minutes

# Lab Exercise 13: STORED FUNCTION

|  |
| --- |
|  |

## Create, Call, and Remove Stored Procedure

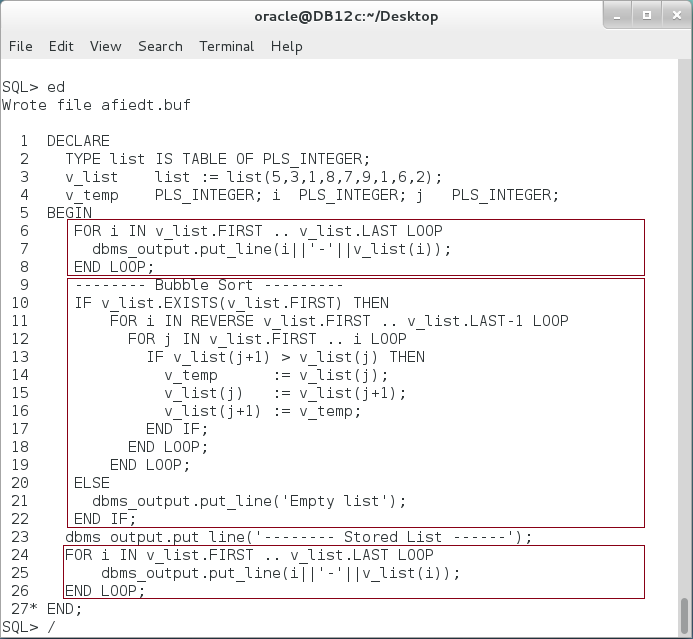
**Step 1:** Open the Terminal, open SQL\*Plus console and connect to hr schema.

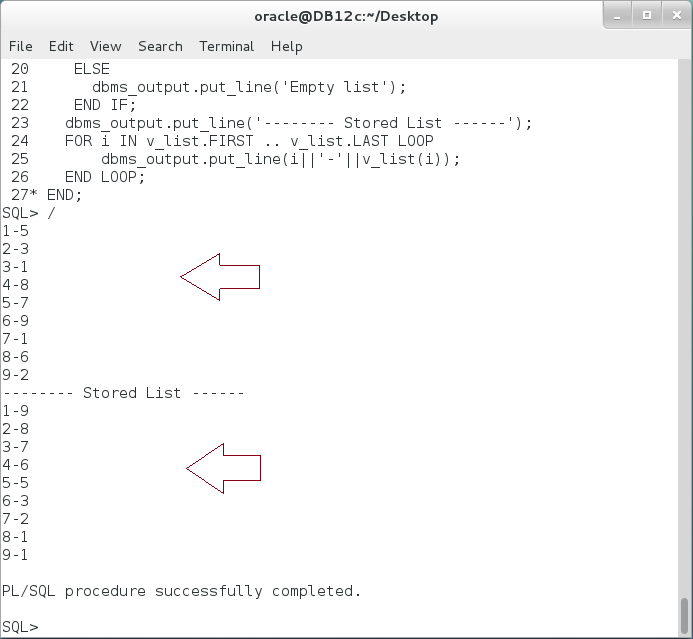
|  |  |
| --- | --- |
| Command | Description |
| sqlplus | Open SQL\*Plus console. |
| hr/oracle | connect to **hr** schema. |

****

**Step 2:** We will start with the same block as in the Stored Procedure. Execute the following block:

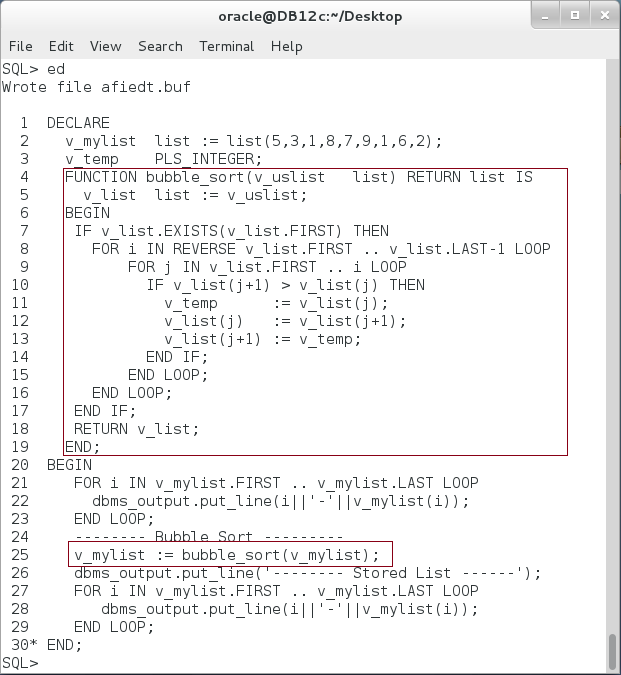
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| TYPE list IS TABLE OF PLS\_INTEGER; |  |
| v\_list list := list(5,3,1,8,7,9,1,6,2); |  |
| v\_temp PLS\_INTEGER; i PLS\_INTEGER; j PLS\_INTEGER; |  |
| BEGIN |  |
| FOR i IN v\_list.FIRST .. v\_list.LAST LOOP | Print the list |
| dbms\_output.put\_line(i||'-'||v\_list(i)); |
| 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; |
| / |  |

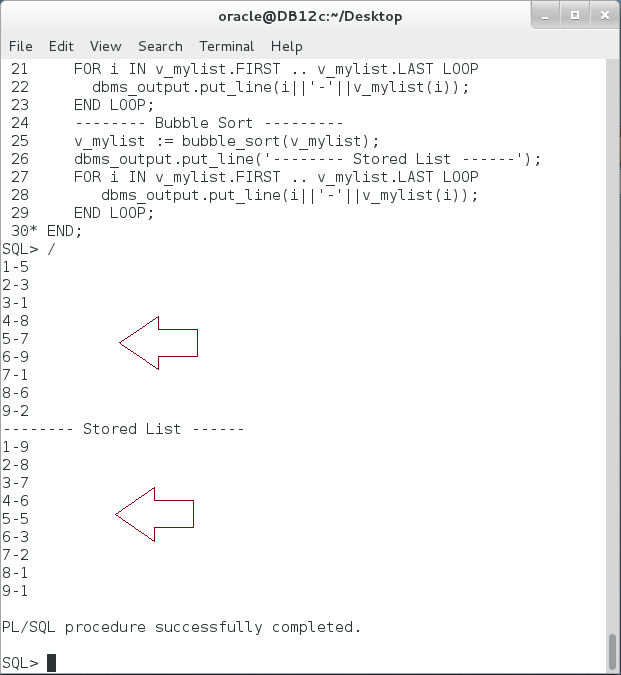
****

****

**Step 3:** In the previous, you may notice that the core bubble sort algorithm can be encapsulated in a separated block of code, in which you get an unsorted collection and return a sorted one. This is where you should function. Modify the previous PL/SQL block as show below:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| v\_mylist list := list(5,3,1,8,7,9,1,6,2); |  |
| v\_temp PLS\_INTEGER; |  |
| **FUNCTION** bubble\_sort(v\_uslist list) **RETURN** list **IS** | **Nested Function** which returns "list" type |
| v\_list list := v\_uslist; |
| **BEGIN** |
| 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; |  |
| END IF; | **Return** the collection |
| **RETURN** v\_list; |
| **END;** |  |
| BEGIN |  |
| FOR i IN v\_mylist.FIRST .. v\_mylist.LAST LOOP |  |
| dbms\_output.put\_line(i||'-'||v\_mylist(i)); |  |
| END LOOP; |  |
| -------- Bubble Sort --------- |  |
| **v\_mylist := bubble\_sort(v\_mylist);** | **Call a function** as expression |
| dbms\_output.put\_line('-------- Stored List ------'); |
| FOR i IN v\_mylist.FIRST .. v\_mylist.LAST LOOP |  |
| dbms\_output.put\_line(i||'-'||v\_mylist(i)); |  |
| END LOOP; |  |
| END; |  |
| / |  |

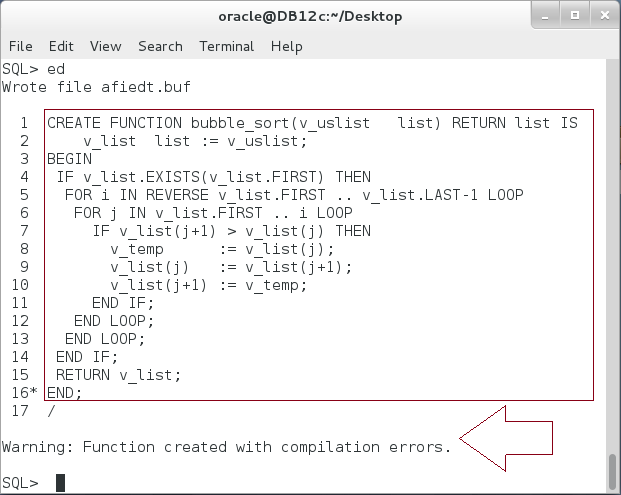




**Please note:** Nested Procedure is not stored in the database; no other user can see or use this procedure. It is just away to organize your coding.

**Step 4:** To create a stored function that do the same job as the previous Nested Function, execute the following DDL statement:

|  |  |
| --- | --- |
| Command | Description |
| **CREATE FUNCTION** bubble\_sort(v\_uslist list) RETURN list IS | Create bubble\_sort Function. |
| v\_list list := v\_uslist; |
| BEGIN |
| 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; |  |
| END IF; |  |
| **RETURN v\_list;** |  |
| **END;** |  |

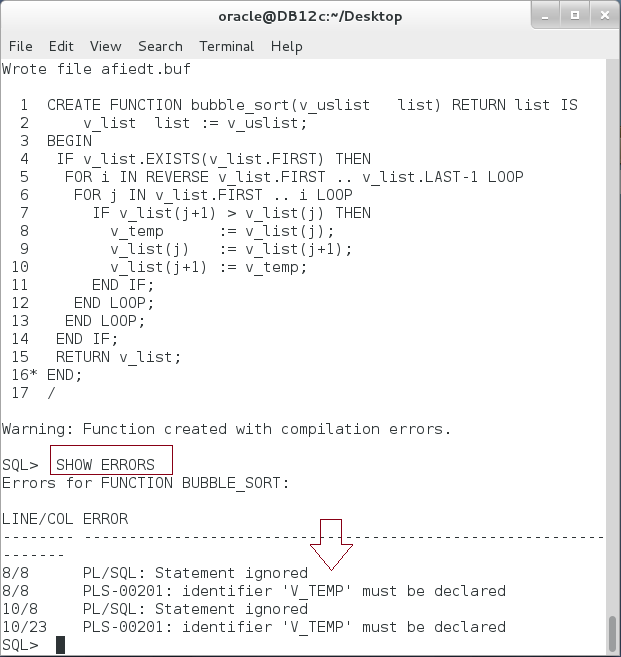


**Please note**: The Stored Function was created but with compilation errors! **How to view those errors?**

**Step 5:** To view the error, execute the following command:

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

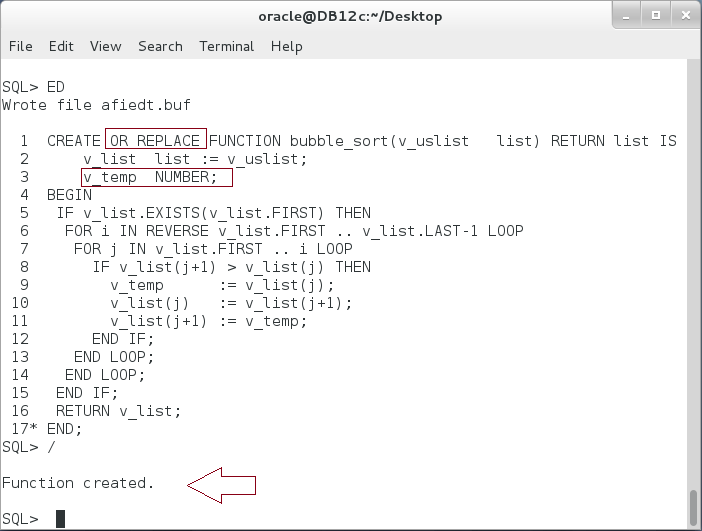
**Please Note:** The message indicates that V\_LIST is not declared. You should either declare an inner variable or add parameter to the procedure to hold this variable. In our case, we should add parameter. **How we can modify a previously created procedure?**

****

**Please note:** V\_TEMP is a variable that was declared previously in the outer block. The nested function was able to reach but not anymore.

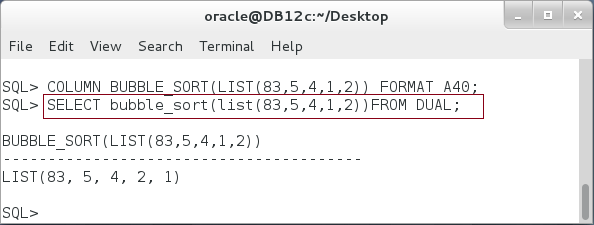
**Step 6:** To modify the function, execute the following DDL statement:

|  |  |
| --- | --- |
| Command | Description |
| CREATE **OR REPLACE** FUNCTION bubble\_sort(v\_uslist list) RETURN list IS | Alter function using OR REPLACE keywords. |
| v\_list list := v\_uslist; |
| **v\_temp NUMBER;** |
| BEGIN |
| 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; |  |
| END IF; |  |
| RETURN v\_list; |  |
| END; |  |
| / |  |

****

**Step 7:** You can execute the stored function from the SQL\*Plus as shown below:

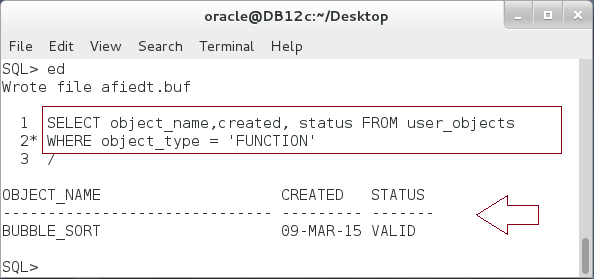
|  |  |
| --- | --- |
| Command | Description |
| **SELECT** bubble\_sort(list(83,5,4,1,2))FROM DUAL | Use **SELECT** statement. |

****

**Please note:** You can't use EXECUTE or CALL commands to execute a Stored Function from the SQL\*Plus.

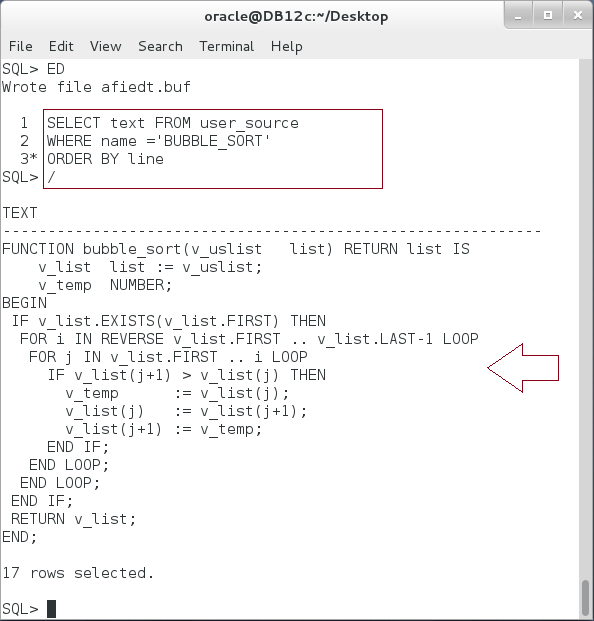
**Step 8:** To list all Functions in your schema, execute the following query:

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

****

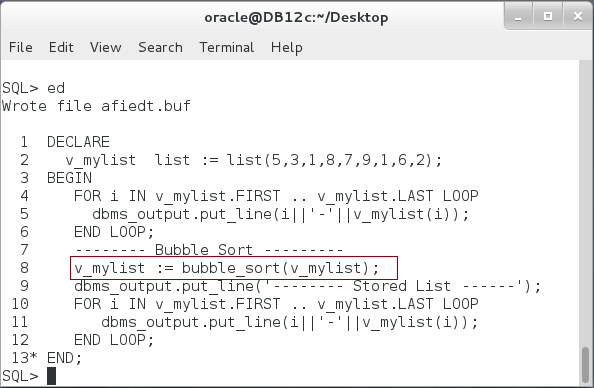
**Step 9:** To view the function source code, execute the following query:

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

****

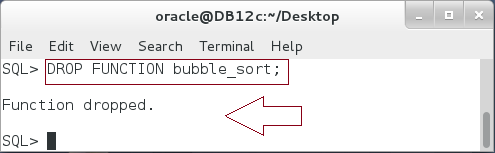
**Step 10:** You can now use the stored function in the first PL/SQL block as shown below:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| v\_mylist list := list(5,3,1,8,7,9,1,6,2); |  |
| BEGIN |  |
| FOR i IN v\_mylist.FIRST .. v\_mylist.LAST LOOP |  |
| dbms\_output.put\_line(i||'-'||v\_mylist(i)); |  |
| END LOOP; |  |
| -------- Bubble Sort --------- |  |
| **v\_mylist := bubble\_sort(v\_mylist);** | **Call a stored function** as expression |
| dbms\_output.put\_line('-------- Stored List ------'); |
| FOR i IN v\_mylist.FIRST .. v\_mylist.LAST LOOP |  |
| dbms\_output.put\_line(i||'-'||v\_mylist(i)); |  |
| END LOOP; |  |
| END; |  |
| / |  |

****

**Step 11:** To drop a Stored Function, execute the following DDL statement:

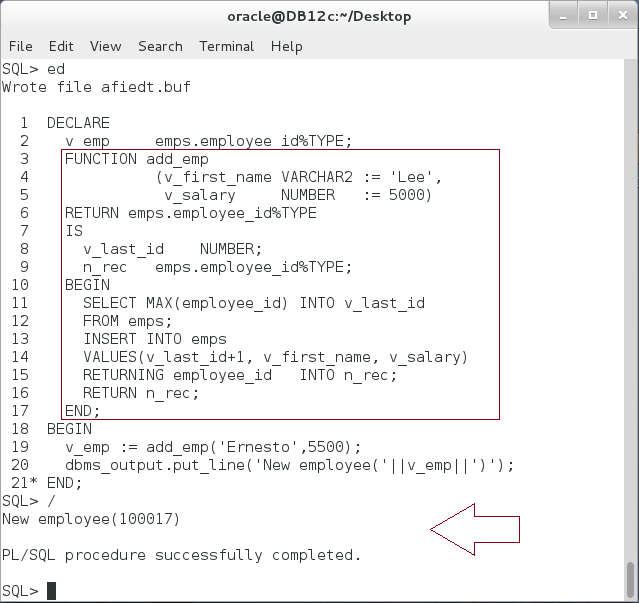
|  |  |
| --- | --- |
| Command | Description |
| **DROP FUNCTION** bubble\_sort; | Drop Stored Procedure. |

****

## Use Functions in SQL

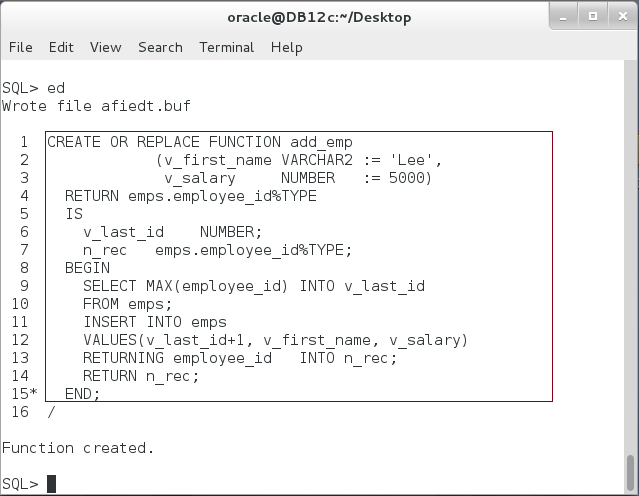
**Step 1:** Execute the following PL/SQL block:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| v\_emp emps.employee\_id%TYPE; | **Declare and Define** nested function add\_emp. |
| **FUNCTION add\_emp** |
| (v\_first\_name VARCHAR2 := 'Lee', |
| v\_salary NUMBER := 5000) |
| **RETURN emps.employee\_id%TYPE** |
| **IS** |
| v\_last\_id NUMBER; |
| n\_rec emps.employee\_id%TYPE; |
| **BEGIN** |
| SELECT MAX(employee\_id) INTO v\_last\_id |
| FROM emps; |
| INSERT INTO emps |
| VALUES(v\_last\_id+1, v\_first\_name, v\_salary) |
| RETURNING employee\_id INTO n\_rec; |
| RETURN n\_rec; |
| **END**; |
| BEGIN |  |
| **v\_emp := add\_emp('Ernesto',5500);** | Use assignment to call the function |
| dbms\_output.put\_line('New employee('||v\_emp||')'); |
| END; |  |
| / |  |



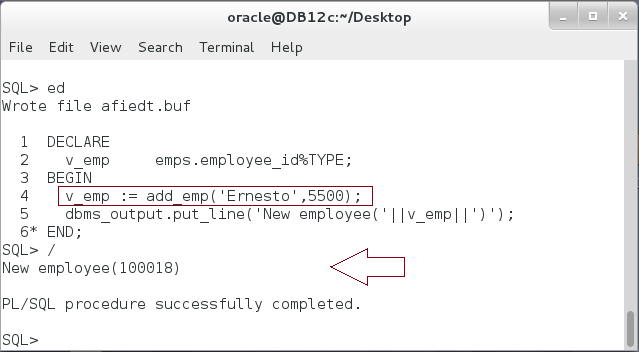
**Step 2:** Create a stored procedure "**add\_emp**" as show below:

|  |  |
| --- | --- |
| Line | Description |
| **CREATE OR REPLACE FUNCTION add\_emp** |  |
| (v\_first\_name VARCHAR2 := 'Lee', |
| v\_salary NUMBER := 5000) |
| RETURN emps.employee\_id%TYPE |
| IS |
| v\_last\_id NUMBER; |
| n\_rec emps.employee\_id%TYPE; |
| BEGIN |
| SELECT MAX(employee\_id) INTO v\_last\_id |
| FROM emps; |
| INSERT INTO emps |
| VALUES(v\_last\_id+1, v\_first\_name, v\_salary) |
| RETURNING employee\_id INTO n\_rec; |
| RETURN n\_rec; |
| END; |
| / |  |

****

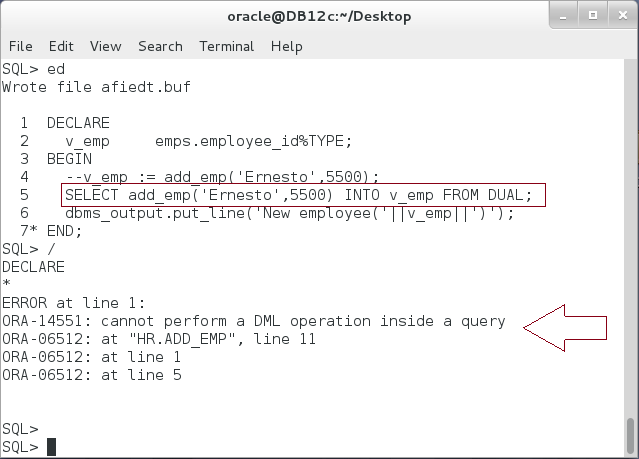
**Step 3:** Call the stored function as shown below:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| v\_emp emps.employee\_id%TYPE; |  |
| BEGIN |  |
| **v\_emp := add\_emp('Ernesto',5500);** | Call the function using assignment |
| dbms\_output.put\_line('New employee('||v\_emp||')'); |
| END; |  |
| / |  |



**Step 4:** In the previous block, everything goes smoothly and as expected. Modify the previous block so you call a function using SELECT statement:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| v\_emp emps.employee\_id%TYPE; |  |
| BEGIN |  |
| --v\_emp := add\_emp('Ernesto',5500); | Call the function using **SQL** |
| **SELECT add\_emp('Ernesto',5500) INTO v\_emp**  **FROM DUAL;** |
| dbms\_output.put\_line('New employee('||v\_emp||')'); |
| END; |
| / |  |



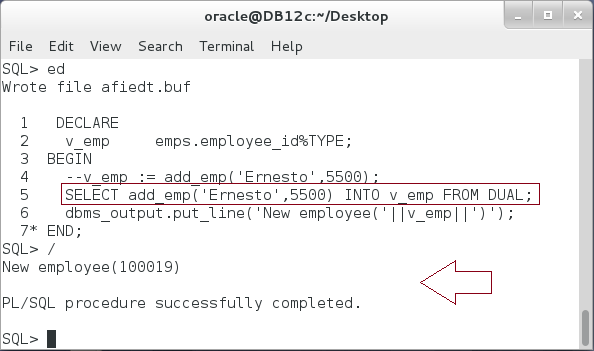
**Please note:** When invoked from a SELECT statement or a parallelized INSERT, UPDATE, DELETE, or MERGE statement, the subprogram cannot modify any database tables. Therefore, you are not allowed to call a function that performs DML statements in a SELECT statement.

**Step 5:** Modify the function so it contains no DML statments. Execute the following block:

|  |  |
| --- | --- |
| Line | Description |
| **CREATE OR REPLACE FUNCTION** add\_emp |  |
| (v\_first\_name VARCHAR2 := 'Lee', |  |
| v\_salary NUMBER := 5000) |  |
| **RETURN** emps.employee\_id%TYPE |  |
| **IS** |  |
| v\_last\_id NUMBER; |  |
| n\_rec emps.employee\_id%TYPE; |  |
| **BEGIN** |  |
| SELECT MAX(employee\_id) INTO v\_last\_id |  |
| FROM emps; |  |
| /\* INSERT INTO emps | Comment INSERT statement |
| VALUES(v\_last\_id+1, v\_first\_name, v\_salary) |
| RETURNING employee\_id INTO n\_rec; \*/ |
| RETURN v\_last\_id+1; |  |
| **END**; |  |
| / |  |

**Step 6:** Re-execute the block as show below:

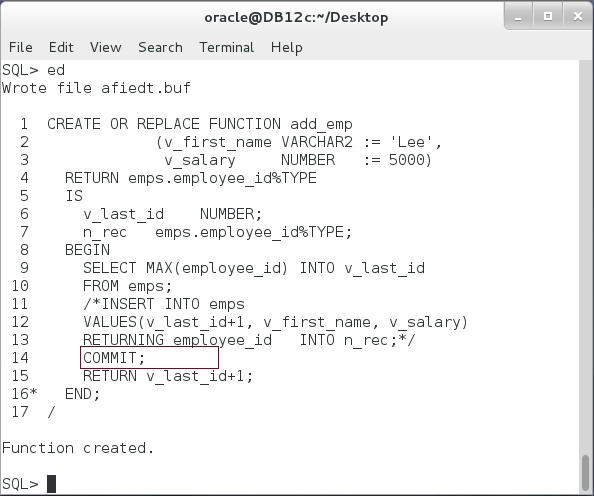
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| v\_emp emps.employee\_id%TYPE; |  |
| BEGIN |  |
| --v\_emp := add\_emp('Ernesto',5500); |  |
| SELECT add\_emp('Ernesto',5500) INTO v\_emp FROM DUAL; |  |
| dbms\_output.put\_line('New employee('||v\_emp||')'); |  |
| END; |  |
| / |  |

****

**Please note:** The block successfully completed.

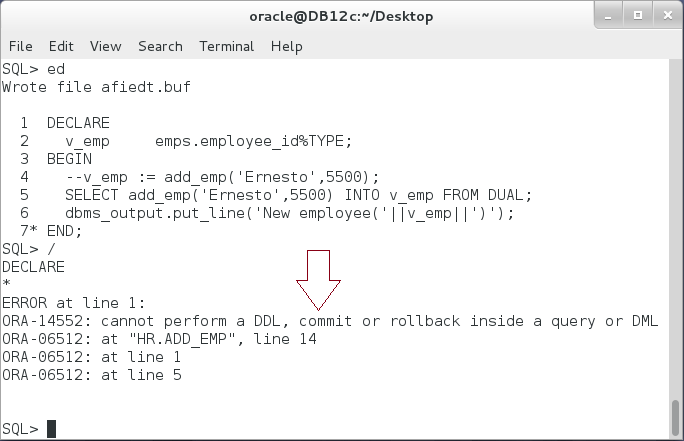
**Step 7:** Re-Create the function so you add "**COMMIT**" statement as show below:

|  |  |
| --- | --- |
| Line | Description |
| **CREATE OR REPLACE FUNCTION add\_emp** |  |
| (v\_first\_name VARCHAR2 := 'Lee', |  |
| v\_salary NUMBER := 5000) |  |
| RETURN emps.employee\_id%TYPE |  |
| IS |  |
| v\_last\_id NUMBER; |  |
| n\_rec emps.employee\_id%TYPE; |  |
| BEGIN |  |
| SELECT MAX(employee\_id) INTO v\_last\_id |  |
| FROM emps; |  |
| /\*INSERT INTO emps |  |
| VALUES(v\_last\_id+1, v\_first\_name, v\_salary) |  |
| RETURNING employee\_id INTO n\_rec;\*/ |  |
| **COMMIT;** |  |
| RETURN v\_last\_id+1; |  |
| END; |  |
| / |  |

****

**Step 8:** Re-execute the block as shown below:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| v\_emp emps.employee\_id%TYPE; |  |
| BEGIN |  |
| --v\_emp := add\_emp('Ernesto',5500); |  |
| **SELECT add\_emp('Ernesto',5500) INTO v\_emp FROM DUAL;** |  |
| dbms\_output.put\_line('New employee('||v\_emp||')'); |  |
| END; |  |
| / |  |

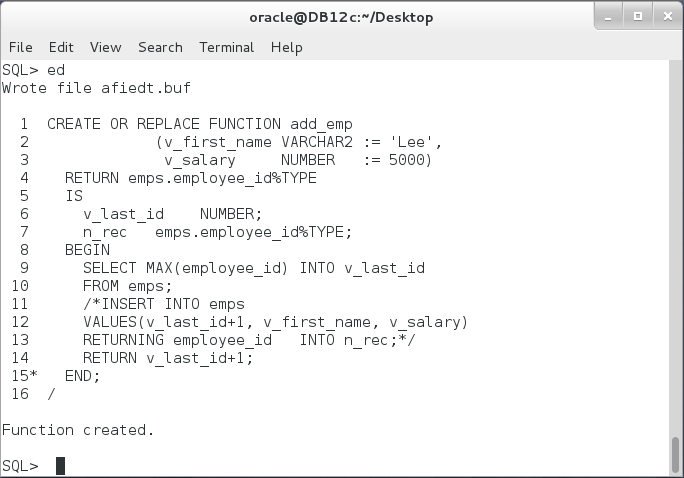
****

**Please note:** When invoked from a SELECT, INSERT, UPDATE, DELETE, or MERGE statement, the subprogram cannot execute any of the following SQL statements:

* COMMIT/ROLLBACK
* SET ROLE
* ALTER SYSTEM/ALTER SESSION.
* DDL statement

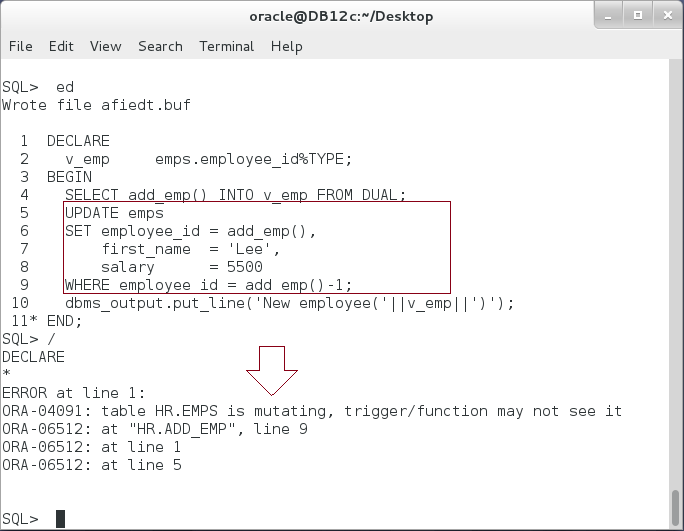
**Step 9:** Re-Create the function so you delete "**COMMIT**" statement as show below:

|  |  |
| --- | --- |
| Line | Description |
| **CREATE OR REPLACE FUNCTION add\_emp** |  |
| (v\_first\_name VARCHAR2 := 'Lee', |  |
| v\_salary NUMBER := 5000) |  |
| RETURN emps.employee\_id%TYPE |  |
| IS |  |
| v\_last\_id NUMBER; |  |
| n\_rec emps.employee\_id%TYPE; |  |
| BEGIN |  |
| SELECT MAX(employee\_id) INTO v\_last\_id |  |
| FROM emps; |  |
| /\*INSERT INTO emps |  |
| VALUES(v\_last\_id+1, v\_first\_name, v\_salary) |  |
| RETURNING employee\_id INTO n\_rec;\*/ |  |
| RETURN v\_last\_id+1; |  |
| END; |  |
| / |  |

****

**Step 10:** Re-execute the block so you add "**UPDATE**" statement on "**emps**" in which you call the function "**add\_emp**" as show below:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| v\_emp emps.employee\_id%TYPE; |  |
| BEGIN |  |
| SELECT add\_emp() INTO v\_emp FROM DUAL; |  |
| **UPDATE** emps | Update emps using the function add\_emp that refer to the same table. |
| **SET** employee\_id = **add\_emp()**, |
| first\_name = 'Lee', |
| salary = 5500 |
| **WHERE** employee\_id = **add\_emp()-1;** |
| dbms\_output.put\_line('New employee('||v\_emp||')'); |
| END; |  |
| / |  |

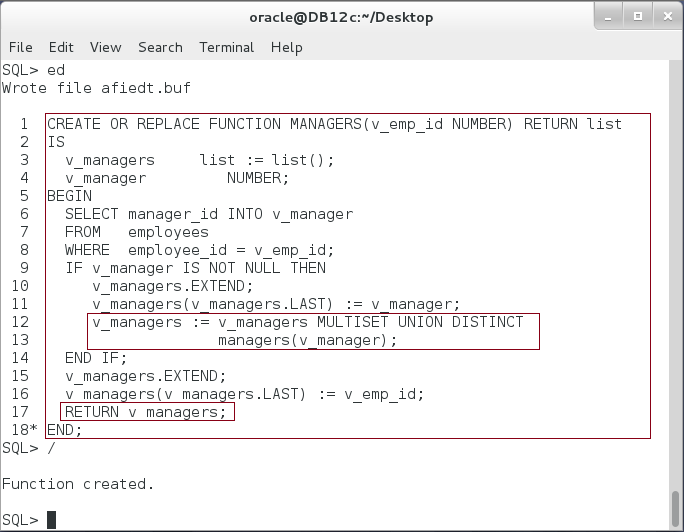
****

**Please note:** If a function either queries or modifies a table, and a DML statement on that table invokes the function, then ORA-04091 (mutating-table error) occurs.

## Recursive Function

**Step 1:** A recursive subprogram invokes itself. Execute the following block:

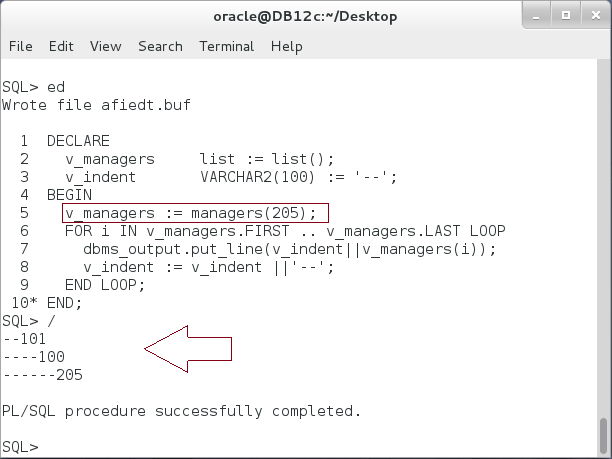
|  |  |
| --- | --- |
| Command | Description |
| **CREATE OR REPLACE FUNCTION MANAGERS**(v\_emp\_id NUMBER) RETURN list |  |
| IS |  |
| v\_managers list := list(); |  |
| v\_manager NUMBER; |  |
| BEGIN |  |
| SELECT manager\_id INTO v\_manager |  |
| FROM employees |  |
| WHERE employee\_id = v\_emp\_id; |  |
| IF v\_manager IS NOT NULL THEN |  |
| v\_managers.EXTEND; |  |
| v\_managers(v\_managers.LAST) := v\_manager; |  |
| **v\_managers := v\_managers MULTISET UNION DISTINCT** | The function calls itself. |
| **managers(v\_manager);** |
| END IF; |  |
| v\_managers.EXTEND; |  |
| v\_managers(v\_managers.LAST) := v\_emp\_id; |  |
| **RETURN v\_managers;** | Terminate |
| END; |  |
| / |  |

****

**Please note:** A recursive subprogram must have at least two execution paths—one leading to the recursive invocation and one leading to a terminating condition. Without the latter, recursion continues until PL/SQL runs out of memory and raises the predefined exception STORAGE\_ERROR.

**Step 2:** To examine the function, execute the following block:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| v\_managers list := list(); |  |
| v\_indent VARCHAR2(100) := '--'; |  |
| BEGIN |  |
| **v\_managers := managers(205);** | Get the manager list |
| FOR i IN v\_managers.FIRST .. v\_managers.LAST LOOP | Print the list |
| dbms\_output.put\_line(v\_indent||v\_managers(i)); |
| v\_indent := v\_indent ||'--'; |
| END LOOP; |
| END; |  |
| / |  |

****

# SUMMARY

Store Function is the best way to calculate something and return a value. The Stored Function is like the anonymous block; it contains declaration, Executable, and Exception-Handling parts. However, it also includes (1) a Function Definition/Declaration part in which you specify its name and parameters list, and (2) a Return keyword. Unlike a Stored Procedure, Stored Function is called as an expression. This means you may call it inside SQL statements: SELECT and DML. However, there are some conditions using it in SQL. For example, you are not allowed to perform DML statements in this function. That is to avoid inconsistency action of performing DML while you are doing SELECT! You also not allowed to query a table inside a function and then call it while performing UPDATE on the same table. This will raise an internally defined exception: table is mutating. Oracle supports recursive function so the function calls itself. Be careful when using recursive function. All recursive function should have at least two paths: one to call itself, and the other to terminate. Without the termination path, the recursive function will executed until it raises out of memory exception.

After completing this lab exercise, you should be able to use Stored Function in a basic setting.

# REFERENCES

* http://docs.oracle.com/database/121/LNPLS/subprograms.htm#LNPLS00810
* http://docs.oracle.com/cd/B19306\_01/server.102/b14200/operators006.htm

# INDEX

recursive function 3, 31

Stored Function 2, 4, 13, 16, 18, 31

Stored Procedure 4