Table of Contents

[CHAPTER 24: DYNAMIC SQL 2](#_Toc417247010)

[Theory 2](#_Toc417247011)

[AIM 3](#_Toc417247012)

[Lab Exercise 24: DYNAMIC SQL 4](#_Toc417247013)

[1. OPEN FOR statement 5](#_Toc417247014)

[2. EXECUTE IMMEDIATE 14](#_Toc417247015)

[3. DBMS\_SQL 29](#_Toc417247016)

[4. SQL Injection 43](#_Toc417247017)

[SUMMARY 51](#_Toc417247018)

[REFERENCES 52](#_Toc417247019)

[INDEX 53](#_Toc417247020)

# CHAPTER 24: DYNAMIC SQL

## Theory

Oracle supports developing an elastic application where the user has the ability to play with code. Sometimes, it is not for fun; it is the only way to satisfy the business requirements. Dynamic SQL is mainly standing in contrast to Static SQL. All the codes you have been looked at was belonging to Static SQL. In Static SQL, code is known and written before a compile-time. In contrast, in Dynamic SQL, the developer does not know the full SQL command in compile-time. Therefore, you may use a Dynamic SQL when:

1. You don't know the SQL the user want to execute. Thus, the user is free to execute his/her own SQL statement.
2. You want the user to execute DDL commands.

Oracle supports two types of Dynamic SQL:

1. Native Dynamic SQL: which includes
   1. EXECUTE IMMEDIATE statement
   2. OPEN FOR statement
2. DBMS\_SQL package

Native Dynamic SQL gives you the freedom to construct the SQL command at run-time, but you still need to know two things to execute you SQL successfully:

* SELECT list: you know the columns the query should return.
* Binding variables: you know all variables which should bind to the SQL statement.

DBMS\_SQL package gives you more general and flexibility. You may not know SELECT list or the binding variable in compile-time. However, you can't use DBMS\_SQL to retrieves rows into records or use the SQL cursor attributes such as %FOUND, %ISOPEN and %ROWCOUNT.

Be careful when using Dynamic SQL. If not designed well, Dynamic SQL may make your application prone to SQL Injection. Use parameterized SQL and check the user code to avoid malicious actions.

## AIM

The AIM of the following exercise is to use Dynamic SQL statements.

The steps involved will include:

* OPEN FOR statement
* EXECUTE IMMEDIATE
* DBMS\_SQL
* SQL Injection

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

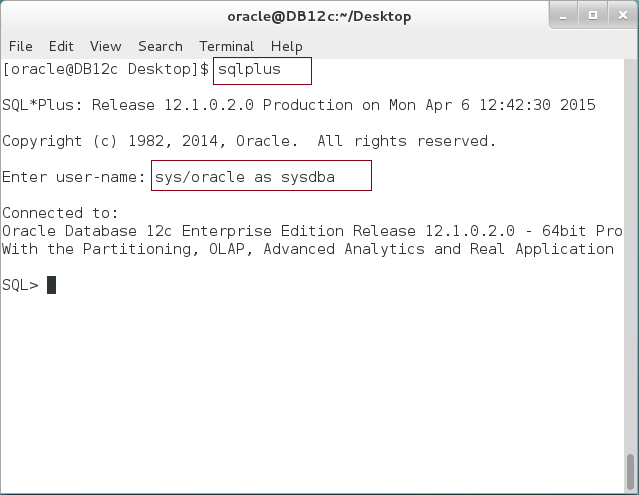
# Lab Exercise 24: DYNAMIC SQL

|  |
| --- |
|  |

## OPEN FOR statement

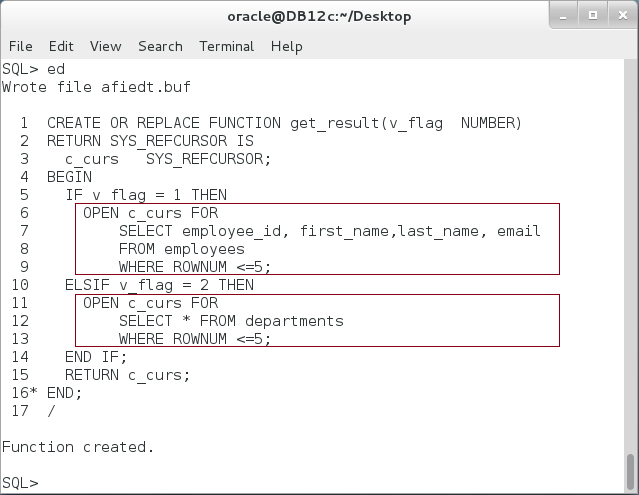
**Step 1:** Open the Terminal, open SQL\*Plus console and connect to hr schema.

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

****

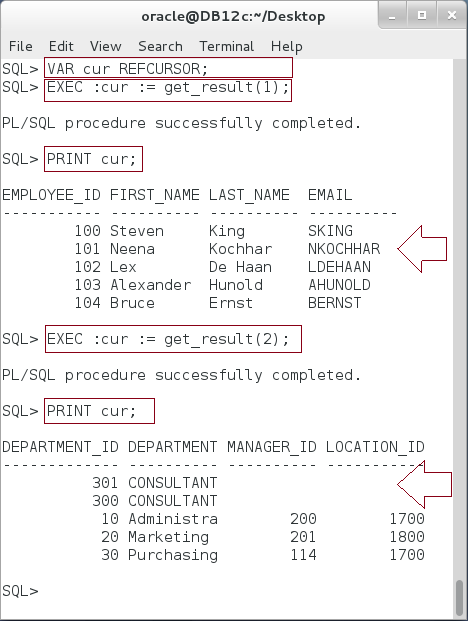
**Step 2:** Create the following function:

|  |  |
| --- | --- |
| Command | Description |
| CREATE OR REPLACE FUNCTION get\_result(v\_flag NUMBER) | A function returns a ref cursor and takes a flag variable to determine which query to execute. |
| RETURN SYS\_REFCURSOR IS |
| c\_curs SYS\_REFCURSOR; |
| BEGIN |
| **IF v\_flag = 1 THEN** |
| **OPEN c\_curs FOR** |
| SELECT employee\_id, first\_name,last\_name, email |
| FROM employees |  |
| WHERE ROWNUM <=5; |  |
| **ELSIF v\_flag = 2 THEN** |  |
| **OPEN c\_curs FOR** |  |
| SELECT \* FROM departments |  |
| WHERE ROWNUM <=5; |  |
| END IF; |  |
| RETURN c\_curs; |  |
| END; |  |
| / |  |



**Step 3:** Test the previous function as shown below:

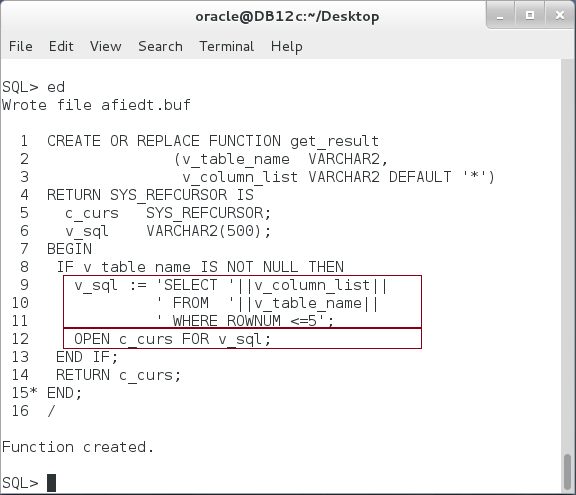
|  |  |
| --- | --- |
| Command | Description |
| VAR cur REFCURSOR; | Return an employees' information |
| **EXEC :cur := get\_result(1);** |
| PRINT cur; |
| EXEC :cur := get\_result(2); | Returns departments' information |
| PRINT cur; |

****

**Please note:** so far, nothing is new. That is a normal ref cursor expression. Now, what would you do if your employer asks you to **return queries on all tables**? would you go with "FLAG" choice? **Can you allow the user determine the returned columns?**

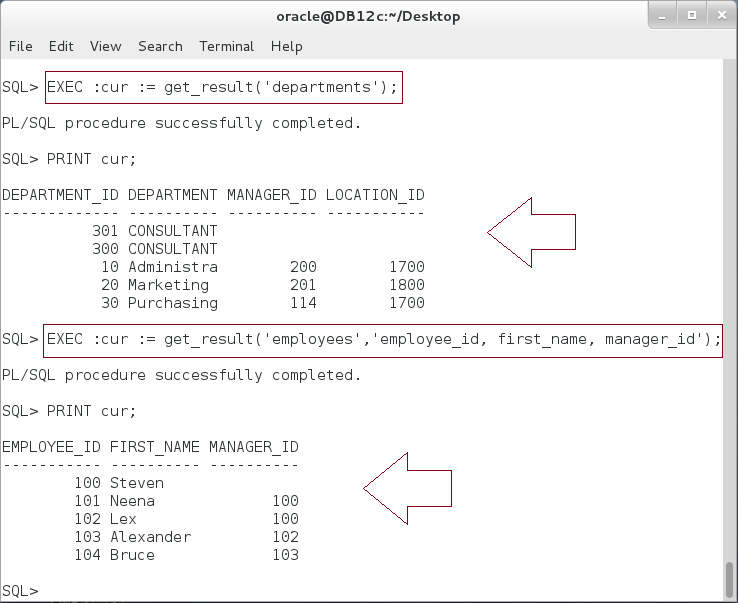
**Step 4:** It is clear that you can't satisfy all dynamic queries using a normal OPEN FOR expression with Ref Cursor. For more flexibility, you should allow a user edit the query by him/herself. Re-create the previous function as shown below:

|  |  |
| --- | --- |
| Command | Description |
| CREATE OR REPLACE FUNCTION get\_result | Allow the user selecting the table name and the columns list. |
| (**v\_table\_name VARCHAR2**, |
| **v\_column\_list VARCHAR2 DEFAULT '\*'**) |
| RETURN SYS\_REFCURSOR IS |
| c\_curs SYS\_REFCURSOR; |  |
| v\_sql VARCHAR2(500); |  |
| BEGIN |  |
| IF v\_table\_name IS NOT NULL THEN | The select statement is built using a simple text variable. |
| **v\_sql** := 'SELECT '||v\_column\_list|| |
| ' FROM '||v\_table\_name|| |
| ' WHERE ROWNUM <=5'; |
| **OPEN c\_curs FOR v\_sql;** | Open a cursor with generated text. |
| END IF; |
| RETURN c\_curs; |
| END; |  |
| / |  |



**Step 5:** Test the previous function as shown below:

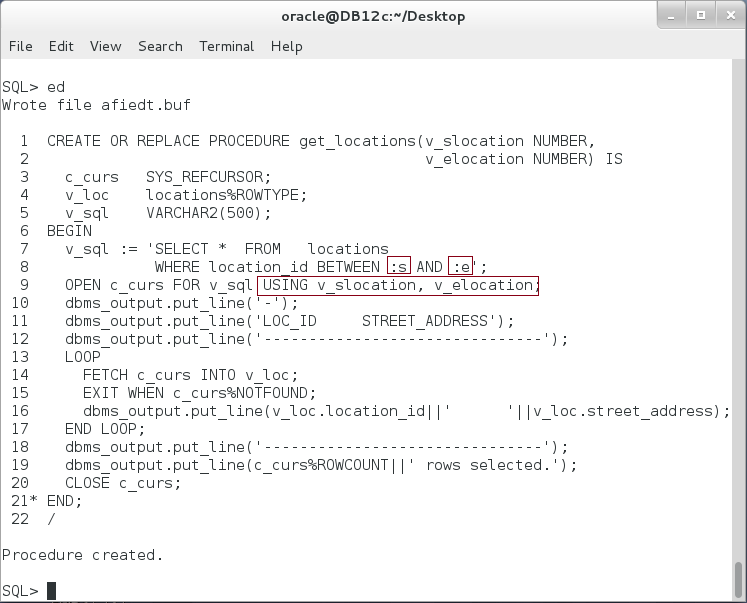
|  |  |
| --- | --- |
| Command | Description |
| EXEC :cur := **get\_result('departments');** |  |
| PRINT cur; |  |
| EXEC :cur := **get\_result('employees','employee\_id, first\_name, manager\_id');** |  |
| PRINT cur; |

****

**Please note:** test the previous function with any other table and columns list you want.

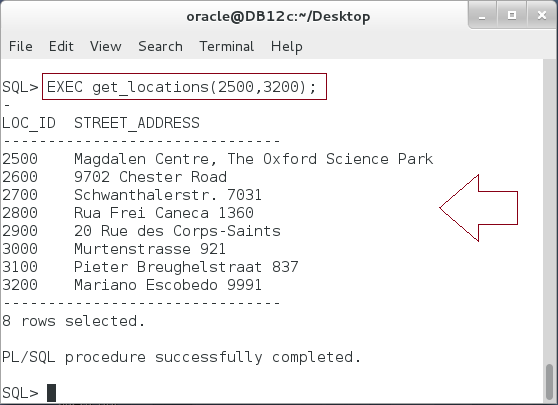
**Step 6:** Sometimes, you need to pass some values from the invoker to the dynamic cursor. In such cases, user placeholders ( started with ":") and USING clause as shown below:

|  |  |
| --- | --- |
| Command | Description |
| CREATE OR REPLACE PROCEDURE get\_locations  (v\_slocation NUMBER, |  |
| v\_elocation NUMBER) IS |  |
| c\_curs SYS\_REFCURSOR; |  |
| v\_loc locations%ROWTYPE; |  |
| v\_sql VARCHAR2(500); |  |
| BEGIN |  |
| v\_sql := 'SELECT \* FROM locations | Open cursor with "USING" clause. |
| WHERE location\_id **BETWEEN :s AND :e'**; |
| OPEN c\_curs FOR v\_sql **USING v\_slocation, v\_elocation;** |
|  |
| dbms\_output.put\_line('-'); |  |
| dbms\_output.put\_line('LOC\_ID STREET\_ADDRESS'); |  |
| dbms\_output.put\_line('-------------------------------'); |  |
| LOOP |  |
| FETCH c\_curs INTO v\_loc; |  |
| EXIT WHEN c\_curs%NOTFOUND; |  |
| dbms\_output.put\_line(v\_loc.location\_id||' '||v\_loc.street\_address); |  |
| END LOOP; |  |
| dbms\_output.put\_line('-------------------------------'); |  |
| dbms\_output.put\_line(c\_curs%ROWCOUNT||' rows selected.'); |  |
| CLOSE c\_curs; |  |
| END; |  |
| / |  |

****

**Step 7:** Test the previous function as shown below:

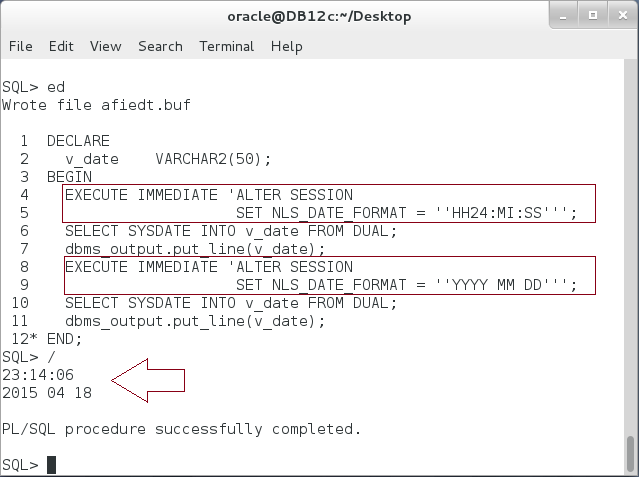
|  |
| --- |
| Command |
| EXEC get\_locations(2500,3200); |

****

## EXECUTE IMMEDIATE

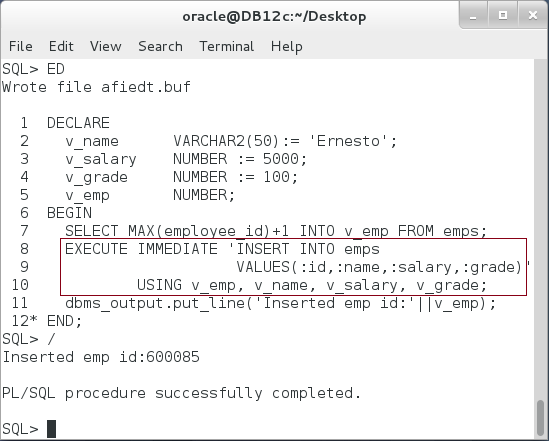
**Step 1:** EXECUTE IMMEDIATE command is another way to execute a dynamic SQL. However, you may also use EXECUTE IMMEDIATE to execute DDL commands as shown below:

|  |
| --- |
| Command |
| DECLARE |
| v\_date VARCHAR2(50); |
| BEGIN |
| **EXECUTE IMMEDIATE 'ALTER SESSION** |
| **SET NLS\_DATE\_FORMAT = ''HH24:MI:SS''';** |
| SELECT SYSDATE INTO v\_date FROM DUAL; |
| dbms\_output.put\_line(v\_date); |
| **EXECUTE IMMEDIATE 'ALTER SESSION** |
| **SET NLS\_DATE\_FORMAT = ''YYYY MM DD''';** |
| SELECT SYSDATE INTO v\_date FROM DUAL; |
| dbms\_output.put\_line(v\_date); |
| END; |
| / |

****

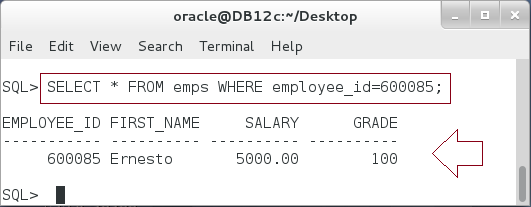
**Step 2:** You can also use EXECUTE IMMEDIATE to run DML commands as shown below:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| v\_name VARCHAR2(50):= 'Ernesto'; |  |
| v\_salary NUMBER := 5000; |  |
| v\_grade NUMBER := 100; |  |
| v\_emp NUMBER; |  |
| BEGIN |  |
| SELECT MAX(employee\_id)+1 INTO v\_emp FROM emps; |  |
| **EXECUTE IMMEDIATE** 'INSERT INTO emps | insert into EMPS using placeholders. |
| VALUES(:id,:name,:salary,:grade)' |
| **USING** v\_emp, v\_name, v\_salary, v\_grade; |
| dbms\_output.put\_line('Inserted emp id:'||v\_emp); |  |
| END; |  |
| / |  |



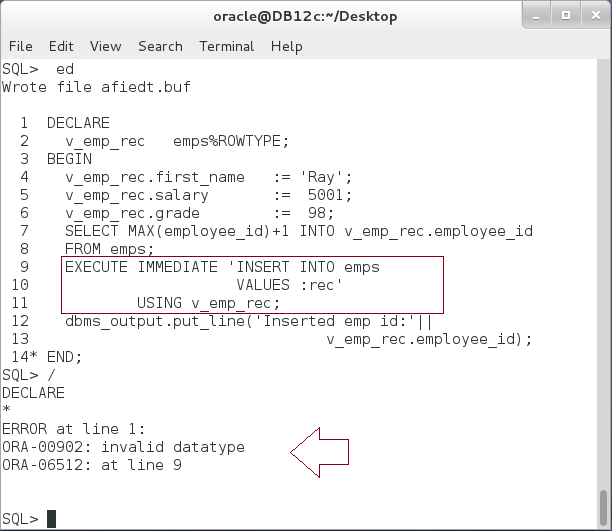
**Step 3:** Execute the following query to make sure that the previous record has been inserted into EMPS table as shown below:

|  |  |
| --- | --- |
| Command | Description |
| SELECT \* FROM emps WHERE employee\_id=**600085** | replace the emp no. with your previous result. |
| / |

****

**Step 4:** Some consideration should be taken when you work with record. Execute the following block:

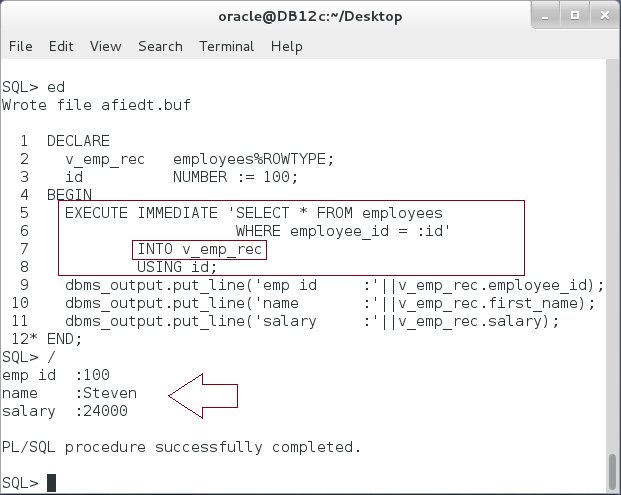
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| v\_emp\_rec emps%ROWTYPE; |
| BEGIN |  |
| v\_emp\_rec.first\_name := 'Ray'; |  |
| v\_emp\_rec.salary := 5001; | Trying to insert record. |
| v\_emp\_rec.grade := 98; |
| SELECT MAX(employee\_id)+1 INTO v\_emp\_rec.employee\_id |
| FROM emps; |
| **EXECUTE IMMEDIATE 'INSERT INTO emps** |
| **VALUES :rec'** |
| **USING v\_emp\_rec;** |  |
| dbms\_output.put\_line('Inserted emp id:'|| |  |
| v\_emp\_rec.employee\_id); |  |
| END; |
| / |  |



**Please note:** The previous block failed. Do you have any idea to make it right?

**Step 5:** EXECUTE IMMEDIATE can be used to retrieve value from SELECT statement as shown below:

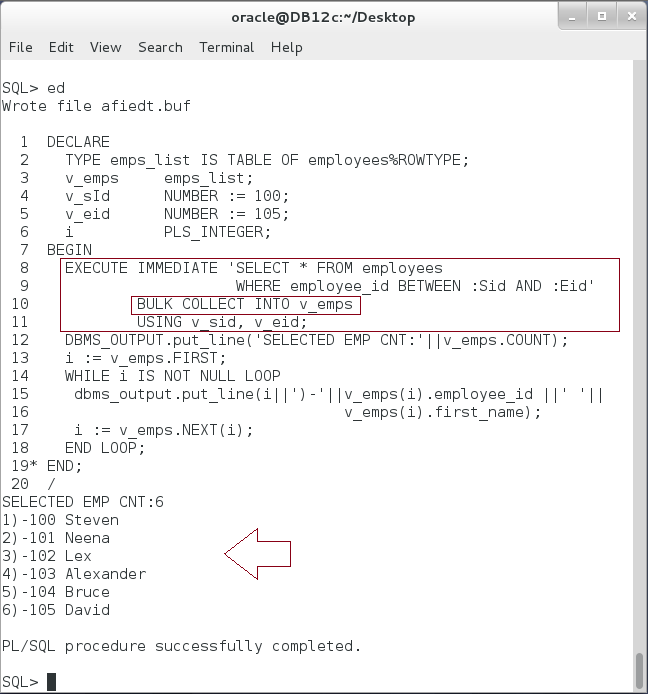
|  |
| --- |
| Line |
| DECLARE |
| v\_emp\_rec employees%ROWTYPE; |
| id NUMBER := 100; |
| BEGIN |
| **EXECUTE IMMEDIATE** 'SELECT \* FROM employees |
| WHERE employee\_id = :id' |
| **INTO** v\_emp\_rec |
| **USING** id; |
| dbms\_output.put\_line('emp id :'||v\_emp\_rec.employee\_id); |
| dbms\_output.put\_line('name :'||v\_emp\_rec.first\_name); |
| dbms\_output.put\_line('salary :'||v\_emp\_rec.salary); |
| END; |
| / |



**Please note:** Use "INTO" to get a single value from the select statement.

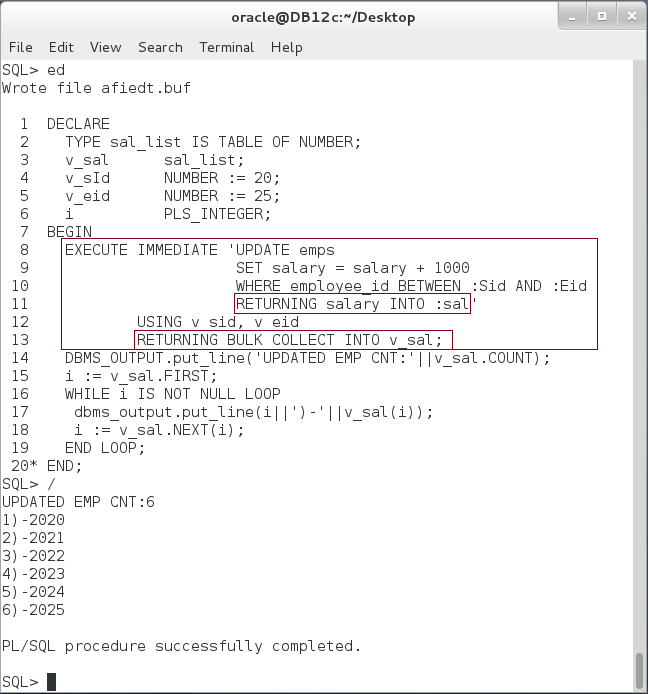
**Step 6:** BULK COLLECT INTO clause is supported for EXECUTE IMMEDIATE. Execute the following block:

|  |  |
| --- | --- |
| Line |  |
| DECLARE |  |
| TYPE emps\_list IS TABLE OF employees%ROWTYPE; |  |
| v\_emps emps\_list; |  |
| v\_sId NUMBER := 100; |  |
| v\_eid NUMBER := 105; |  |
| i PLS\_INTEGER; |  |
| BEGIN |  |
| **EXECUTE IMMEDIATE** 'SELECT \* FROM employees | Use BULK COLLECT INTO expression to get a collection of values |
| WHERE employee\_id BETWEEN :Sid AND :Eid' |
| **BULK COLLECT INTO** v\_emps |
| USING v\_sid, v\_eid; |
| DBMS\_OUTPUT.put\_line('SELECTED EMP CNT:'||v\_emps.COUNT); |
| i := v\_emps.FIRST; |  |
| WHILE i IS NOT NULL LOOP |  |
| dbms\_output.put\_line(i||')-'||v\_emps(i).employee\_id ||' '|| |  |
| v\_emps(i).first\_name); |  |
| i := v\_emps.NEXT(i); |  |
| END LOOP; |  |
| END; |  |
| / |  |



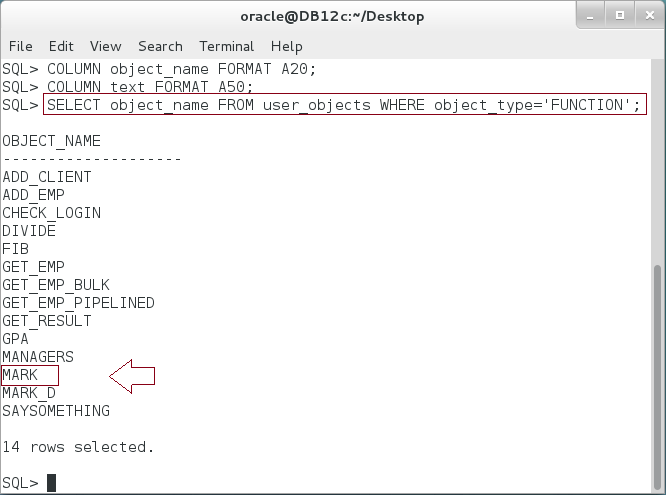
**Step 7:** You can also use BULK COLLECT to return values from update, insert, and delete statements as shown below:

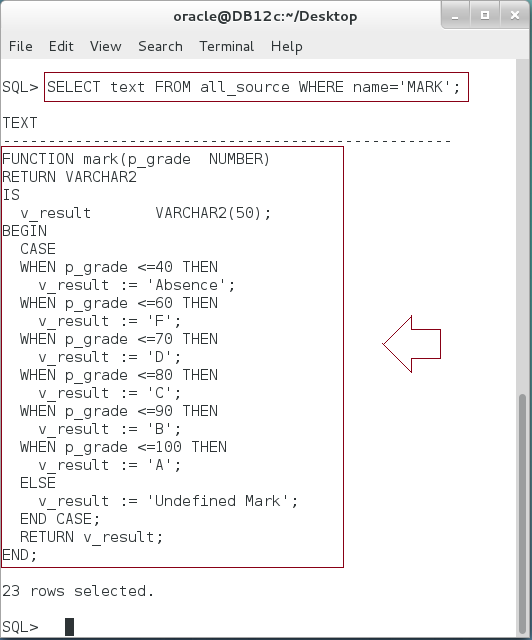
|  |
| --- |
| Line |
| DECLARE |
| TYPE sal\_list IS TABLE OF NUMBER; |
| v\_sal sal\_list; |
| v\_sId NUMBER := 20; |
| v\_eid NUMBER := 25; |
| i PLS\_INTEGER; |
| BEGIN |
| **EXECUTE IMMEDIATE** 'UPDATE emps |
| SET salary = salary + 1000 |
| WHERE employee\_id BETWEEN :Sid AND :Eid |
| **RETURNING salary INTO :sal'** |
| USING v\_sid, v\_eid |
| **RETURNING BULK COLLECT INTO v\_sal;** |
| DBMS\_OUTPUT.put\_line('UPDATED EMP CNT:'||v\_sal.COUNT); |
| i := v\_sal.FIRST; |
| WHILE i IS NOT NULL LOOP |
| dbms\_output.put\_line(i||')-'||v\_sal(i)); |
| i := v\_sal.NEXT(i); |
| END LOOP; |
| END; |
| / |



**Step 8:** You can also use EXECUTE IMMEDIATE to run a PL/SQL block. We will select one of the previous functions, MARK. Execute the following queries:

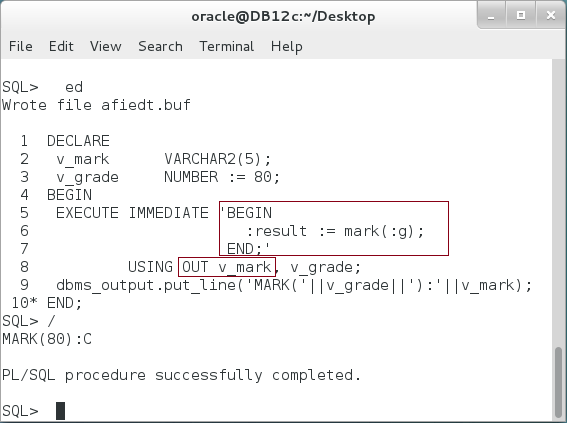
|  |
| --- |
| Line |
| COLUMN object\_name FORMAT A20; |
| COLUMN text FORMAT A50; |
| SELECT object\_name FROM user\_objects WHERE object\_type='FUNCTION' |
| / |
| SELECT text FROM all\_source WHERE name='MARK' |
| / |





**Step 9:** Use EXECUTE IMMEDIATE to run a PL/SQL block as shown:

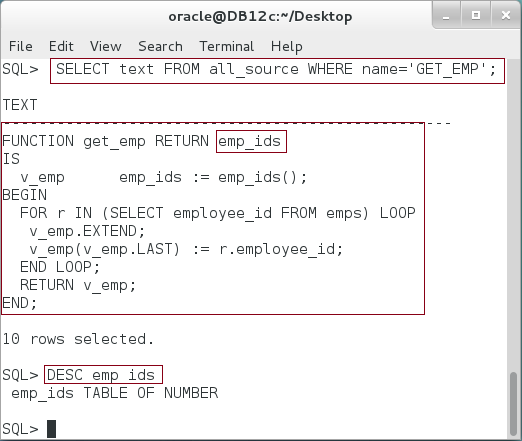
|  |
| --- |
| Line |
| DECLARE |
| v\_mark VARCHAR2(5); |
| v\_grade NUMBER := 80; |
| BEGIN |
| EXECUTE IMMEDIATE **'BEGIN** |
| :result := mark(:g); |
| **END**;' |
| USING **OUT v\_mark**, v\_grade; |
| dbms\_output.put\_line('MARK('||v\_grade||'):'||v\_mark); |
| END; |
| / |



**Please note:** when you want to get a value of the variable out of the PL/SQL use OUT variable.

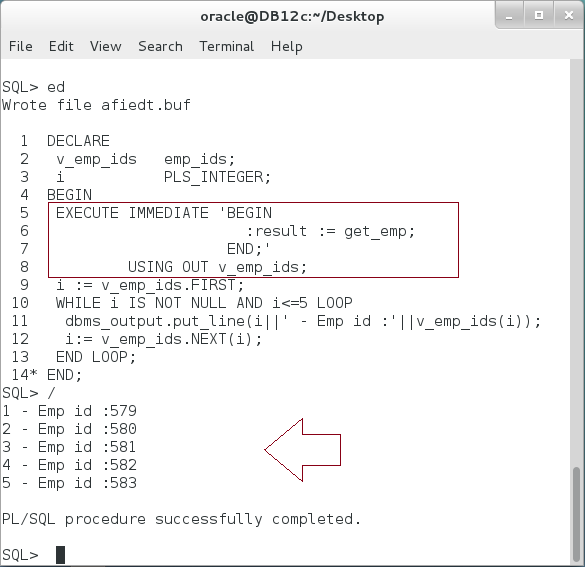
**Step 10:** We will select another stored function, one that use NESTED TABLE as an output. Execute the following query:

|  |
| --- |
| Line |
| SELECT text FROM all\_source WHERE name=**'GET\_EMP'** |
| / |
| DESC emp\_ids |



**Step 11:** EXECUTE IMMEDIATE supports nested table variable. Execute the following block :

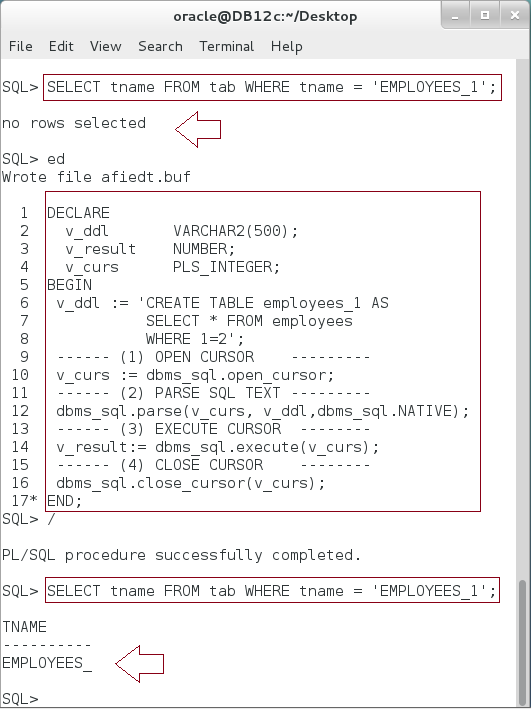
|  |
| --- |
| Line |
| DECLARE |
| **v\_emp\_ids emp\_ids;** |
| i PLS\_INTEGER; |
| BEGIN |
| **EXECUTE IMMEDIATE** 'BEGIN |
| :result := get\_emp; |
| END;' |
| USING **OUT v\_emp\_ids;** |
| i := v\_emp\_ids.FIRST; |
| WHILE i IS NOT NULL AND i<=5 LOOP |
| dbms\_output.put\_line(i||' - Emp id :'||v\_emp\_ids(i)); |
| i:= v\_emp\_ids.NEXT(i); |
| END LOOP; |
| END; |
| / |



## DBMS\_SQL

**Step 1:** DBMS\_SQL package is a more general method to execute dynamic query. DBMS\_SQL can be used to execute DDL statements as shown below:

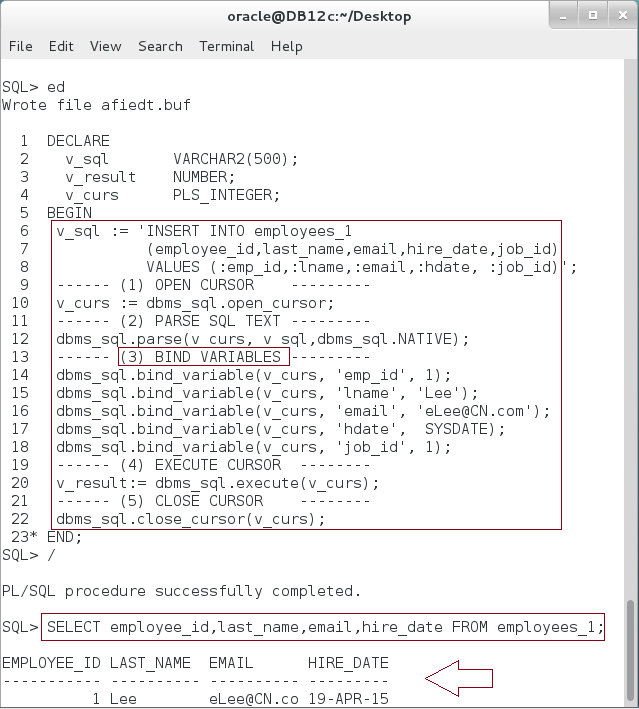
|  |  |
| --- | --- |
| Line | Description |
| SELECT tname FROM tab WHERE tname = 'EMPLOYEES\_1' | Check if the table exists. |
| / |
| DECLARE |  |
| v\_ddl VARCHAR2(500); |  |
| v\_result NUMBER; |  |
| v\_curs PLS\_INTEGER; |  |
| BEGIN |  |
| v\_ddl := 'CREATE TABLE employees\_1 AS | Prepare DDL statement |
| SELECT \* FROM employees |
| WHERE 1=2'; |
| ------ (**1) OPEN CURSOR** --------- | **Step 1:** open cursor |
| v\_curs := dbms\_sql.open\_cursor; |
| ------ **(2) PARSE SQL TEXT** --------- | **Step 2**: parse DDL statement |
| dbms\_sql.parse(v\_curs, v\_ddl,dbms\_sql.NATIVE); |
| ------ **(3) EXECUTE CURSOR** -------- | **Step 3**: Execute the cursor. |
| v\_result:= dbms\_sql.execute(v\_curs); |
| ------ **(4) CLOSE CURSOR** -------- | **Step 4:** Close the cursor. |
| dbms\_sql.close\_cursor(v\_curs); |
| END; |  |
| / |  |
| SELECT tname FROM tab WHERE tname = 'EMPLOYEES\_1' |  |
| / |  |



**Please note:** executing DDL statement using DBMS\_SQL involves 4 general steps: (1) open cursor, (2) parse DDL statement, (3) execute cursor, and (4) close the cursor.

**Step 2:** To view your own triggers and their statuses, use the following command:

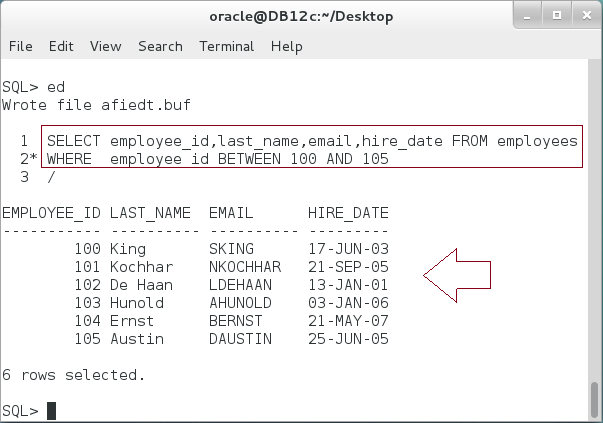
|  |  |
| --- | --- |
| Line | Description |
| **SELECT** employee\_id,last\_name,email,hire\_date FROM employees\_1 |  |
| / |  |
| DECLARE |  |
| v\_sql VARCHAR2(500); |  |
| v\_result NUMBER; |  |
| v\_curs PLS\_INTEGER; |  |
| BEGIN | Prepare DML statement |
| v\_sql := 'INSERT INTO employees\_1 |
| (employee\_id,last\_name,email,hire\_date,job\_id) |
| VALUES (:emp\_id,:lname,:email,:hdate, :job\_id)'; |
| **------ (1) OPEN CURSOR ---------** |
| v\_curs := dbms\_sql.open\_cursor; | **Step 1:** Open cursor |
| **------ (2) PARSE SQL TEXT ---------** |
| dbms\_sql.parse(v\_curs, v\_sql,dbms\_sql.NATIVE); | **Step 2:** Parse DML |
| **------ (3) BIND VARIABLES ---------** |
| dbms\_sql.bind\_variable(v\_curs, 'emp\_id', 1); | Step 3: Bind variable to DML statement. |
| dbms\_sql.bind\_variable(v\_curs, 'lname', 'Lee'); |
| dbms\_sql.bind\_variable(v\_curs, 'email', 'eLee@CN.com'); |
| dbms\_sql.bind\_variable(v\_curs, 'hdate', SYSDATE); |
| dbms\_sql.bind\_variable(v\_curs, 'job\_id', 1); |
| **------ (4) EXECUTE CURSOR --------** | **Step 4**: Execute cur |
| v\_result:= dbms\_sql.execute(v\_curs); |
| **------ (5) CLOSE CURSOR --------** | **Step 5**: close cursor. |
| dbms\_sql.close\_cursor(v\_curs); |
| END; |  |
| / |  |
| SELECT employee\_id,last\_name,email,hire\_date FROM employees\_1 | Check if the record is inserted. |
| / |

****

**Please note:** use BIND\_VARIABLE method to attach a simple data type variable from invoker to DBMS\_SQL code and via versa.

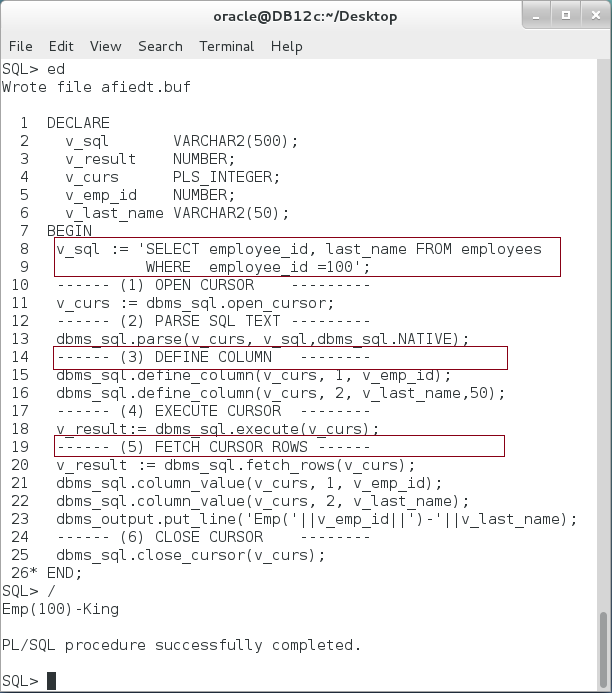
**Step 3:** Suppose you want to retrieve more than one record using DBMS\_SQL package. As the following query does:

|  |
| --- |
| Line |
| **SELECT** employee\_id,last\_name,email,hire\_date **FROM** employees |
| **WHERE** employee\_id **BETWEEN** **100 AND 105** |
| / |

****

**Step 4:** We will move step by step; first, retrieve one record. Execute the following block:

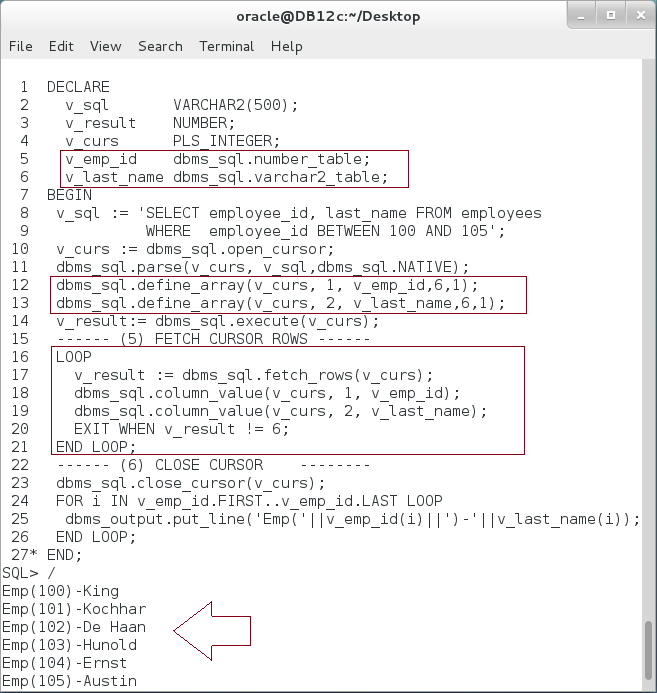
|  |
| --- |
| Line |
| DECLARE |
| v\_sql VARCHAR2(500); |
| v\_result NUMBER; |
| v\_curs PLS\_INTEGER; |
| v\_emp\_id NUMBER; |
| v\_last\_name VARCHAR2(50); |
| BEGIN |
| v\_sql := **'SELECT employee\_id, last\_name FROM employees** |
| **WHERE employee\_id =100';** |
| ------ **(1) OPEN CURSOR** --------- |
| v\_curs := dbms\_sql.open\_cursor; |
| ------ **(2) PARSE SQL TEXT --**------- |
| dbms\_sql.parse(v\_curs, v\_sql,dbms\_sql.NATIVE); |
| ------ **(3) DEFINE COLUMN** -------- |
| dbms\_sql.define\_column(v\_curs, 1, v\_emp\_id); |
| dbms\_sql.define\_column(v\_curs, 2, v\_last\_name,50); |
| ------ **(4) EXECUTE CURSOR** -------- |
| v\_result:= dbms\_sql.execute(v\_curs); |
| ------ **(5) FETCH CURSOR ROWS** ------ |
| v\_result := dbms\_sql.fetch\_rows(v\_curs); |
| dbms\_sql.column\_value(v\_curs, 1, v\_emp\_id); |
| dbms\_sql.column\_value(v\_curs, 2, v\_last\_name); |
| dbms\_output.put\_line('Emp('||v\_emp\_id||')-'||v\_last\_name); |
| ------ **(6) CLOSE CURSOR** -------- |
| dbms\_sql.close\_cursor(v\_curs); |
| END; |
| / |



**Please note:** To execute query, you should add two steps: (1) Define Column and (2) Fetch Cursor Rows.

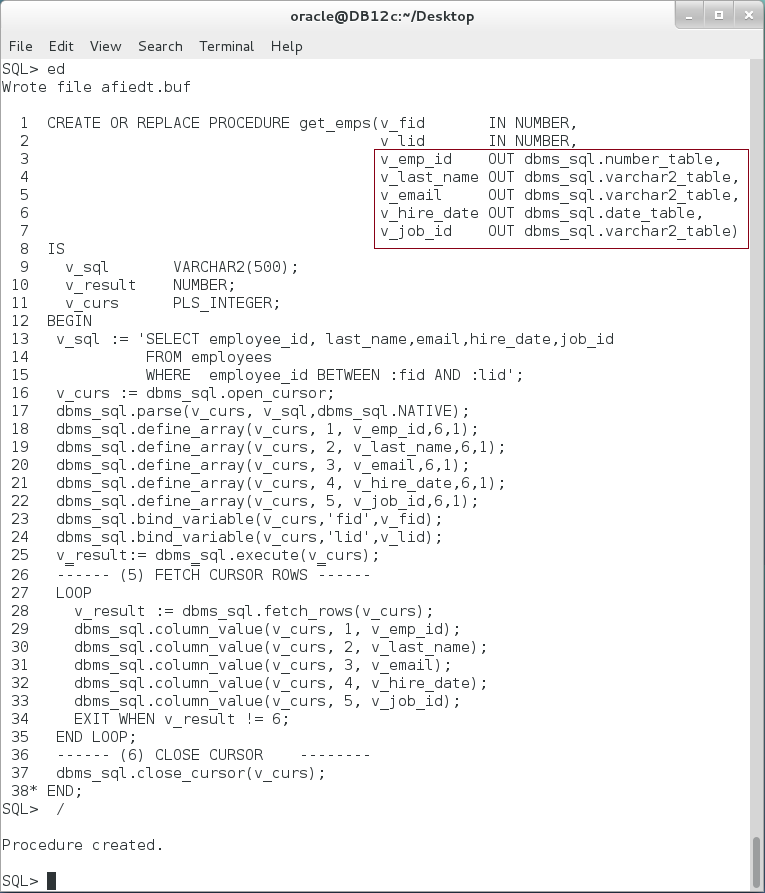
**Step 5:** Now, we are ready to move forward and fetch multiple rows. Modify the previous block as shown below:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| v\_sql VARCHAR2(500); |
| v\_result NUMBER; |
| v\_curs PLS\_INTEGER; |
| v\_emp\_id  **dbms\_sql.number\_table;** |  |
| v\_last\_name **dbms\_sql.varchar2\_table;** |  |
| BEGIN |  |
| v\_sql := 'SELECT employee\_id, last\_name FROM employees |
| **WHERE employee\_id BETWEEN 100 AND 105';** |
| v\_curs := dbms\_sql.open\_cursor; |  |
| dbms\_sql.parse(v\_curs, v\_sql,dbms\_sql.NATIVE); | Define array instead of column. |
| **dbms\_sql.define\_array**(v\_curs, 1, v\_emp\_id,6,1); |
| **dbms\_sql.define\_array**(v\_curs, 2, v\_last\_name,6,1); |
| v\_result:= dbms\_sql.execute(v\_curs); |  |
| ------ (5) FETCH CURSOR ROWS ------ | Loop through the rows to fill in column arrays. |
| **LOOP** |
| v\_result := dbms\_sql.fetch\_rows(v\_curs); |
| dbms\_sql.column\_value(v\_curs, 1, v\_emp\_id); |
| dbms\_sql.column\_value(v\_curs, 2, v\_last\_name); |
| EXIT WHEN v\_result != 6; |
| **END LOOP;** |  |
| ------ (6) CLOSE CURSOR -------- |  |
| dbms\_sql.close\_cursor(v\_curs); |  |
| FOR i IN v\_emp\_id.FIRST..v\_emp\_id.LAST LOOP |  |
| dbms\_output.put\_line('Emp('||v\_emp\_id(i)||')-'||v\_last\_name(i)); |  |
| END LOOP; |  |
| END; |  |
| / |  |



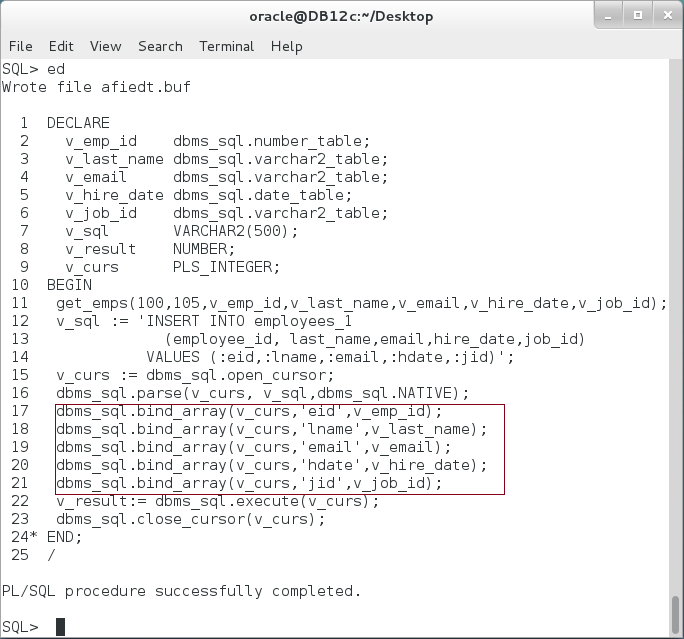
**Step 6:** Create a stored procedure to fetch all mandatory columns of EMPLOYEES table and return them into DBMS\_SQL.XXXX\_TABLE type, where XXXX refers to the column type: DATE, VARCHAR2, or NUMBER. Execute the following DDL statement:

|  |
| --- |
| Line |
| CREATE OR REPLACE PROCEDURE get\_emps(v\_fid IN NUMBER, |
| v\_lid IN NUMBER, |
| **v\_emp\_id OUT dbms\_sql.number\_table,** |
| **v\_last\_name OUT dbms\_sql.varchar2\_table,** |
| **v\_email OUT dbms\_sql.varchar2\_table,** |
| **v\_hire\_date OUT dbms\_sql.date\_table,** |
| **v\_job\_id OUT dbms\_sql.varchar2\_table)** |
| IS |
| v\_sql VARCHAR2(500); |
| v\_result NUMBER; |
| v\_curs PLS\_INTEGER; |
| BEGIN |
| v\_sql := 'SELECT employee\_id, last\_name,email,hire\_date,job\_id |
| FROM employees |
| WHERE employee\_id BETWEEN :fid AND :lid'; |
| v\_curs := dbms\_sql.open\_cursor; |
| dbms\_sql.parse(v\_curs, v\_sql,dbms\_sql.NATIVE); |
| dbms\_sql.define\_array(v\_curs, 1, v\_emp\_id,6,1); |
| dbms\_sql.define\_array(v\_curs, 2, v\_last\_name,6,1); |
| dbms\_sql.define\_array(v\_curs, 3, v\_email,6,1); |
| dbms\_sql.define\_array(v\_curs, 4, v\_hire\_date,6,1); |
| dbms\_sql.define\_array(v\_curs, 5, v\_job\_id,6,1); |
| dbms\_sql.bind\_variable(v\_curs,'fid',v\_fid); |
| dbms\_sql.bind\_variable(v\_curs,'lid',v\_lid); |
| v\_result:= dbms\_sql.execute(v\_curs); |
| ------ (5) FETCH CURSOR ROWS ------ |
| LOOP |
| v\_result := dbms\_sql.fetch\_rows(v\_curs); |
| dbms\_sql.column\_value(v\_curs, 1, v\_emp\_id); |
| dbms\_sql.column\_value(v\_curs, 2, v\_last\_name); |
| dbms\_sql.column\_value(v\_curs, 3, v\_email); |
| dbms\_sql.column\_value(v\_curs, 4, v\_hire\_date); |
| dbms\_sql.column\_value(v\_curs, 5, v\_job\_id); |
| EXIT WHEN v\_result != 6; |
| END LOOP; |
| ------ (6) CLOSE CURSOR -------- |
| dbms\_sql.close\_cursor(v\_curs); |
| END; |
| / |



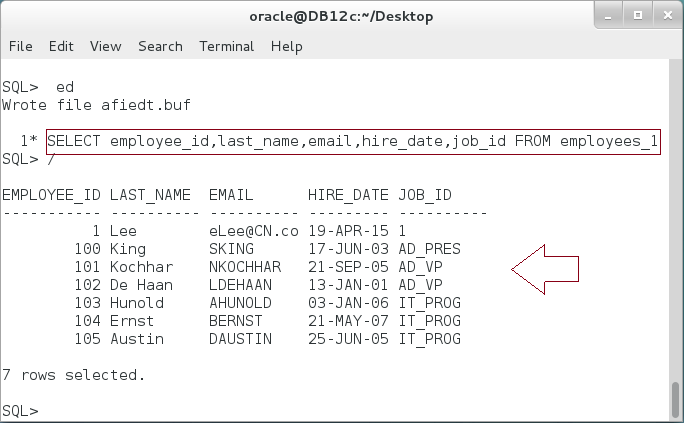
**Step 7:** You can DBMS\_SQL table type to bulk insert record into table. Use the previous procedure to fill data in DBMS\_SQL tables and then insert them into EMPLOYEES\_1 table as shown below:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| v\_emp\_id dbms\_sql.number\_table; |  |
| v\_last\_name dbms\_sql.varchar2\_table; |  |
| v\_email dbms\_sql.varchar2\_table; |  |
| v\_hire\_date dbms\_sql.date\_table; |  |
| v\_job\_id dbms\_sql.varchar2\_table; |  |
| v\_sql VARCHAR2(500); |  |
| v\_result NUMBER; |  |
| v\_curs PLS\_INTEGER; |  |
| BEGIN |  |
| **get\_emps**(100,105,v\_emp\_id,v\_last\_name,v\_email,v\_hire\_date ,v\_job\_id); | Use **get\_emps** Function. |
| v\_sql := 'INSERT INTO employees\_1 | Use regular INSERT statement |
| (employee\_id, last\_name,email,hire\_date,job\_id) |
| VALUES (:eid,:lname,:email,:hdate,:jid)'; |
| v\_curs := dbms\_sql.open\_cursor; |  |
| dbms\_sql.parse(v\_curs, v\_sql,dbms\_sql.NATIVE); | Now, you should use BIND\_ARRARY instead of BIND\_VARIABLE |
| **dbms\_sql.bind\_array**(v\_curs,'eid',v\_emp\_id); |
| **dbms\_sql.bind\_array**(v\_curs,'lname',v\_last\_name); |
| **dbms\_sql.bind\_array**(v\_curs,'email',v\_email); |
| **dbms\_sql.bind\_array**(v\_curs,'hdate',v\_hire\_date); |
| **dbms\_sql.bind\_array**(v\_curs,'jid',v\_job\_id); |
| v\_result:= dbms\_sql.execute(v\_curs); |  |
| dbms\_sql.close\_cursor(v\_curs); |  |
| END; |  |
| / |  |



**Step 8:** Check if the previous block has inserted rows into "EMPLOYEE\_1" table as shown below:

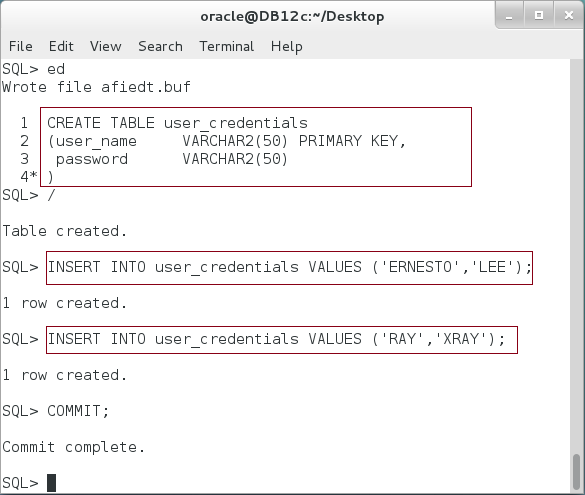
|  |
| --- |
| Line |
| SELECT employee\_id,last\_name,email,hire\_date,job\_id FROM employees\_1 |
| / |



## SQL Injection

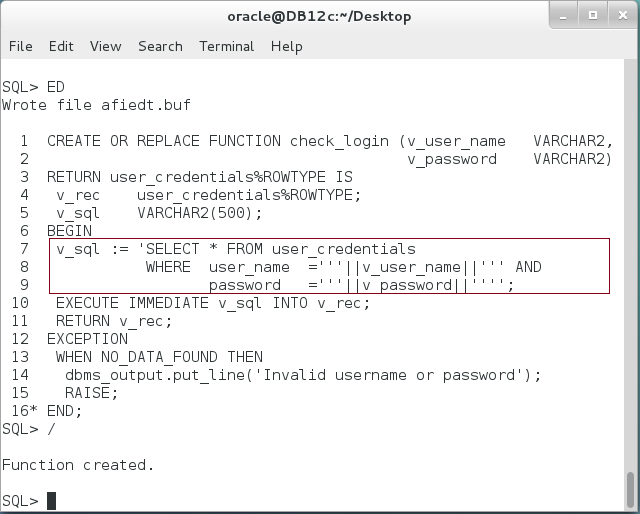
**Step 1:** Create a credintional :

|  |  |
| --- | --- |
| Line | Description |
| CREATE TABLE user\_credentials | Create USER\_CREDENTIALS table |
| (user\_name VARCHAR2(50) PRIMARY KEY, |
| password VARCHAR2(50) |
| ) |
| / |
| INSERT INTO user\_credentials VALUES ('ERNESTO','LEE') | Insert some records. |
| / |
| INSERT INTO user\_credentials VALUES ('RAY','XRAY') |
| / |



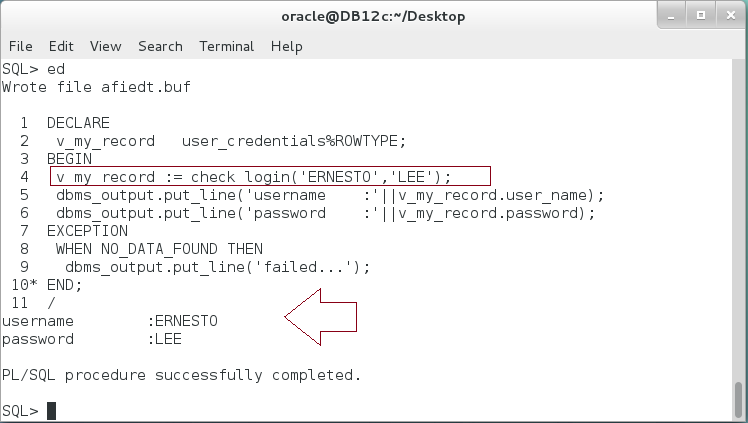
**Step 2:** Create a function to check if the user has a valid username/password combination as shown below:

|  |  |
| --- | --- |
| Line | Description |
| CREATE OR REPLACE FUNCTION check\_login (v\_user\_name VARCHAR2, | Create function |
| v\_password VARCHAR2) |
| RETURN user\_credentials%ROWTYPE IS |
| v\_rec user\_credentials%ROWTYPE; |
| v\_sql VARCHAR2(500); |  |
| BEGIN |  |
| v\_sql := 'SELECT \* FROM user\_credentials | Simple concatenation |
| WHERE **user\_name ='''||v\_user\_name||'''** AND |
| **password ='''||v\_password||''''**; |
| EXECUTE IMMEDIATE v\_sql INTO v\_rec; |
| RETURN v\_rec; |  |
| EXCEPTION |  |
| WHEN NO\_DATA\_FOUND THEN |  |
| dbms\_output.put\_line('Invalid username or password'); |  |
| RAISE; |  |
| END; |  |
| / |  |



**Step 3:** Test the previous function with valid username/password as shown below:

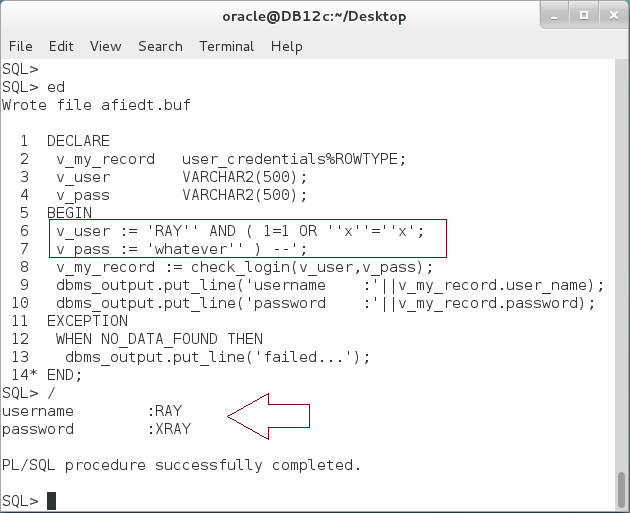
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| v\_my\_record user\_credentials%ROWTYPE; |  |
| BEGIN |  |
| **v\_my\_record := check\_login('ERNESTO','LEE');** |  |
| dbms\_output.put\_line('username :'||v\_my\_record.user\_name); |  |
| dbms\_output.put\_line('password :'||v\_my\_record.password); |  |
| EXCEPTION |  |
| WHEN NO\_DATA\_FOUND THEN |  |
| dbms\_output.put\_line('failed...'); |  |
| END; |  |
| / |  |

****

**Please note:** retest the previous block with invalid username/password and record your notes.

**Step 4:** So far so good. This is time to introduce the SQL injection. Execute the following block to access RAY account without knowing his password:

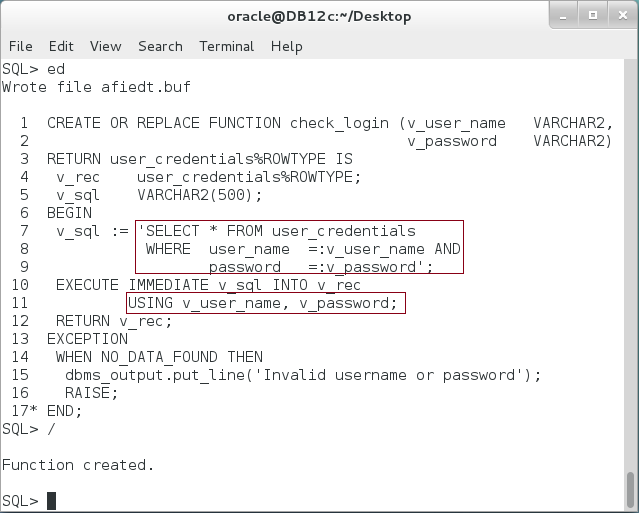
|  |
| --- |
| Line |
| DECLARE |
| v\_my\_record user\_credentials%ROWTYPE; |
| v\_user VARCHAR2(500); |
| v\_pass VARCHAR2(500); |
| BEGIN |
| **v\_user := 'RAY'' AND ( 1=1 OR ''x''=''x';** |
| **v\_pass := 'whatever'' ) --';** |
| **v\_my\_record := check\_login(v\_user,v\_pass);** |
| dbms\_output.put\_line('username :'||v\_my\_record.user\_name); |
| dbms\_output.put\_line('password :'||v\_my\_record.password); |
| EXCEPTION |
| WHEN NO\_DATA\_FOUND THEN |
| dbms\_output.put\_line('failed...'); |
| END; |
| / |

****

**Please note:** the previous user name and password text is called an SQL Injection, where the user attempt to modify the query of the developer to access more information.

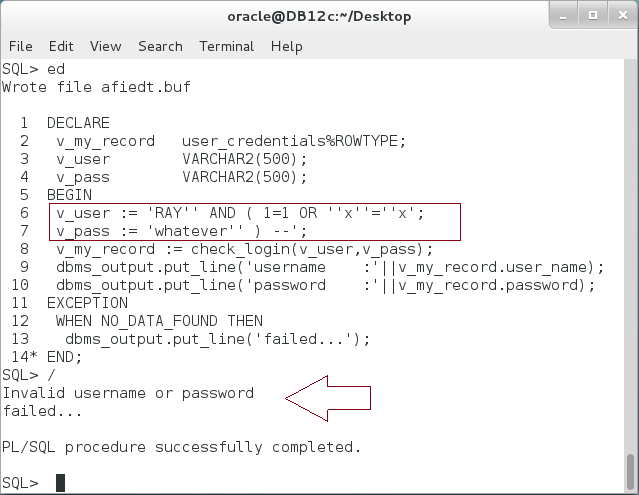
**Step 5:** To create a robust function, use parameterize query with placeholder and USING clause as shown below:

|  |
| --- |
| Line |
| CREATE OR REPLACE FUNCTION check\_login (v\_user\_name VARCHAR2, |
| v\_password VARCHAR2) |
| RETURN user\_credentials%ROWTYPE IS |
| v\_rec user\_credentials%ROWTYPE; |
| v\_sql VARCHAR2(500); |
| BEGIN |
| v\_sql := 'SELECT \* FROM user\_credentials |
| WHERE user\_name **=:v\_user\_name** AND |
| password **=:v\_password'**; |
| EXECUTE IMMEDIATE v\_sql INTO v\_rec |
| **USING v\_user\_name, v\_password;** |
| RETURN v\_rec; |
| EXCEPTION |
| WHEN NO\_DATA\_FOUND THEN |
| dbms\_output.put\_line('Invalid username or password'); |
| RAISE; |
| END; |
| / |

****

**Step 6:** Test the previous function by trying sql injection as shown below:

|  |
| --- |
| Line |
| DECLARE |
| v\_my\_record user\_credentials%ROWTYPE; |
| v\_user VARCHAR2(500); |
| v\_pass VARCHAR2(500); |
| BEGIN |
| **v\_user := 'RAY'' AND ( 1=1 OR ''x''=''x';** |
| **v\_pass := 'whatever'' ) --';** |
| **v\_my\_record := check\_login(v\_user,v\_pass);** |
| dbms\_output.put\_line('username :'||v\_my\_record.user\_name); |
| dbms\_output.put\_line('password :'||v\_my\_record.password); |
| EXCEPTION |
| WHEN NO\_DATA\_FOUND THEN |
| dbms\_output.put\_line('failed...'); |
| END; |
| / |

****

# SUMMARY

Dynamic SQL allows you to develop more flexible application, in which the users can play with the SQL statement as they need. Oracle supports three ways to deliver dynamic SQL: OPEN FOR, EXECUTE IMMEDIATE, and DBMS\_SQL package. OPEN FOR allows you to open a cursor based on text variable and then fetch rows as needed. EXECUTE IMMEDIATE is the most used method of dynamic SQL methods. EXECUTE IMMEDIATE can easily execute DDL, SELECT, and DML statements. If the statement needs an input from the invoker, you can use USING clause for bind variables. If you need to get a value out of SELECT or DML statement, use INTO clause. Moreover, Oracle supports bulk collection in both directions. The third method is DBMS\_SQL package. DBMS\_SQL package is the more general method to execute dynamic SQL but it requires much coding than other methods. Be careful when using dynamic SQL. Allowing the user to write his/her SQL statement makes your application fragile and prone to misuse. Make sure not use parameterize SQL instead of string concatenation to minimize inappropriate text and don't use a dynamic SQL unless there is no other means you can do it

After completing this lab exercise, you should be able to use Dynamic SQL with OPEN FOR, EXECUTE IMMEIDATE, or DBMS\_SQL package.

# REFERENCES

* http://docs.oracle.com/database/121/LNPLS/dynamic.htm#LNPLS01115
* http://docs.oracle.com/cd/E21901\_01/timesten.1122/e21645/d\_sql.htm#TTPLP71258
* http://www.dba-oracle.com/plsql/t\_plsql\_dynamic.htm
* http://psoug.org/reference/dbms\_sql.html

# INDEX

DBMS\_SQL package 2, 29, 32, 51

Dynamic SQL 2, 3, 51

EXECUTE IMMEDIATE 1, 2, 3, 14, 15, 17, 18, 19, 20, 22, 23, 25, 27, 44, 48, 51

OPEN FOR statement 1, 2, 3, 5

SQL Injection 1, 2, 3, 43, 47