Table of Contents

[CHAPTER 10: BULK SQL AND BULK BINDING 2](#_Toc416706107)

[Theory 2](#_Toc416706108)

[AIM 4](#_Toc416706109)

[Lab Exercise 10: BULK SQL AND BULK BINDING 5](#_Toc416706110)

[1. FORALL 6](#_Toc416706111)

[2. FORALL with Sparse Nested Table 14](#_Toc416706112)

[3. BULK\_ROWCOUNT Attribute 17](#_Toc416706113)

[4. BULK COLLECT 21](#_Toc416706114)

[5. Limiting Bulk Selection 30](#_Toc416706115)

[SUMMARY 36](#_Toc416706116)

[REFERENCES 37](#_Toc416706117)

[INDEX 38](#_Toc416706118)

# CHAPTER 10: BULK SQL AND BULK BINDING

## Theory

It is very easy to loop through collection elements as we see in the previous chapter. However, it is also very time and memory consumer. It is essential to review your loop statements and use an optimized technique for better performance and less memory consumption. Oracle recommends using FORALL and BULK COLLECT clauses as such big optimizing technique for normal loop with SELECT or DML statements.

For each PL/SQL block containing SQL statements (SELECT, DML statements), the PL/SQL engine sends the SQL statements one at time to SQL engine. For each SQL statement, SQL engine returns the result(s) one at time back to PL/SQL engine.

SELECT INTO (1)

Bind SQL variables to PL/SQL variables

SELECT INTO (2)

Bind SQL variables to PL/SQL variables

FORALL..IN  UPDATE

RETURNING BULK COLLECT INTO

SELECT INTO BULK COLLECT

Bind SQL Collection variables to PL/SQL Collection variables

Execute   
UPDATE with  
RETURNING BULK COLLECT INTO

BEGIN  
....  
 **SELECT INTO BULK COLLECT;**....  
... **...  
FORALL ..  
UPDATE ...  
RETURNING ...;**....  
END;

Execute   
SELECT INTO BULK COLLECT

SQL Engine

BEGIN  
....  
SELECT INTO;  
....  
...  
...  
SELECT INTO;  
...  
END;

Execute   
SELECT INTO (2)

Execute   
SELECT INTO (1)

PL/SQL Engine

In contrast, you can use FORALL to group all similar SELECT or DML statements and send them all together and for once. If there are many returning values, you may use BULK COLLECT clause to fetch them all together and for once. Using this technique, you make use of less memory and faster query or DML statement.

## AIM

The AIM of the following exercise is to demonstrate how to use bulk SQL and bulk binding in PL/SQL block.

The steps involved will include:

* FORALL
* FORALL with Sparse Nested Table
* BULK\_ROWCOUNT Attribute
* BULK COLLECT
* Limiting Bulk Selection

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 10: BULK SQL AND BULK BINDING

|  |
| --- |
|  |

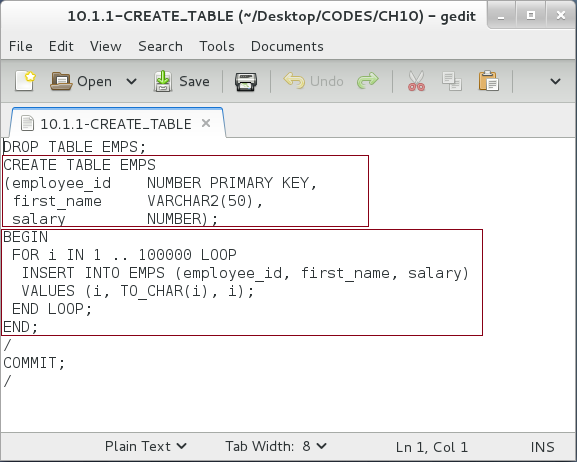
## FORALL

**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:** Start by preparing for the lab. Ask your trainer to provide you with a text file " **10.1.1-CREATE\_TABLE** ".

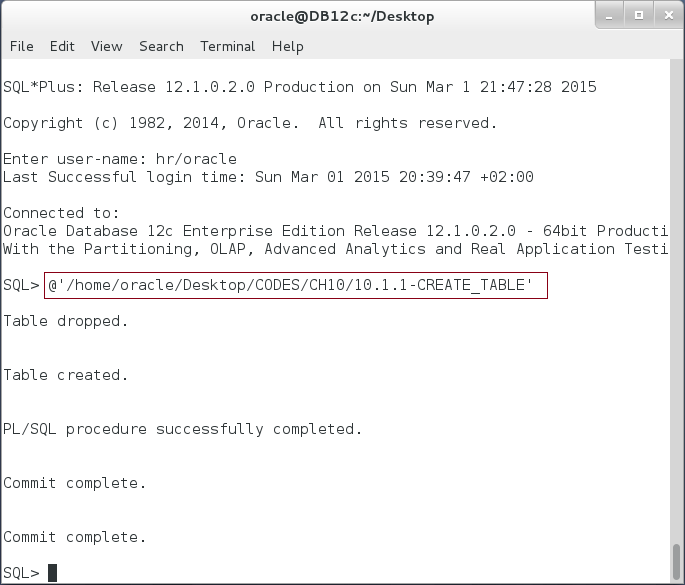


**Please note**: The file is mainly create a temporary table "**EMPS**" and populate it with 100,000 records.

**Step 3:** Run the script file as show below:

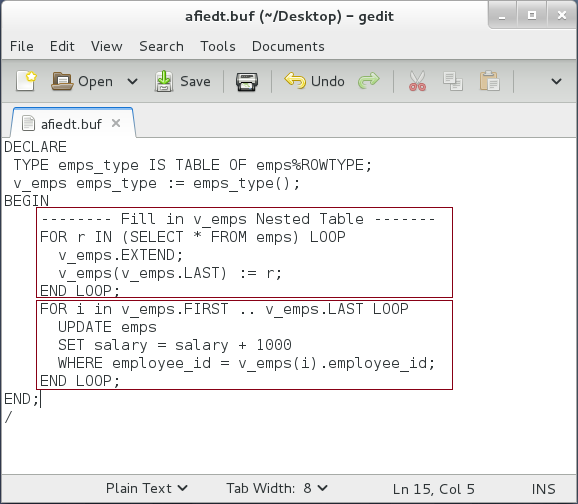
|  |  |
| --- | --- |
| Command | Description |
| @'/home/oracle/Desktop/CODES/CH10/10.1.1-CREATE\_TABLE' | Run script |

**Please:** replace the path to reflect yours.

****

**Step 4:** Execute the following block:

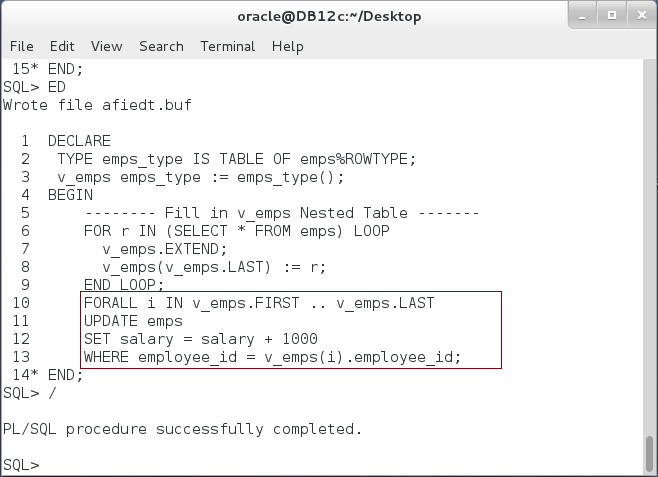
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| TYPE emps\_type IS TABLE OF emps%ROWTYPE; |  |
| v\_emps emps\_type := emps\_type(); |  |
| BEGIN |  |
| -------- Fill in v\_emps Nested Table ------- | Populate v\_emps Nested Table with values. |
| **FOR r IN (SELECT \* FROM emps) LOOP** |
| v\_emps.EXTEND; |
| v\_emps(v\_emps.LAST) := r; |
| **END LOOP;** |
| **FOR i in v\_emps.FIRST .. v\_emps.LAST LOOP** | Loop through v\_emps Nested Table and update each record. |
| **UPDATE** emps |
| SET salary = salary + 1000 |
| WHERE employee\_id = v\_emps(i).employee\_id; |
| **END LOOP;** |
| END; |  |
| / |  |



**Please note:** In the previous block, you use normal FOR LOOP to go through v\_emps Nested Table. For each iteration, UPDATE statement is sent to SQL Engine.

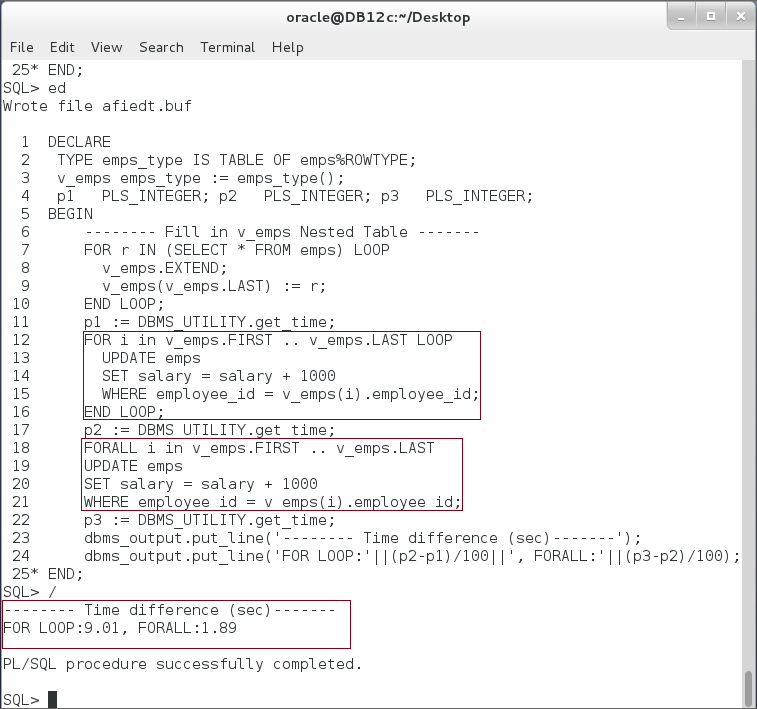
**Step 5:** Modify the previous block as shown below:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| TYPE emps\_type IS TABLE OF emps%ROWTYPE; |  |
| v\_emps emps\_type := emps\_type(); |  |
| BEGIN |  |
| -------- Fill in v\_emps Nested Table ------- | Populate v\_emps Nested Table with values. |
| FOR r IN (SELECT \* FROM emps) LOOP |
| v\_emps.EXTEND; |
| v\_emps(v\_emps.LAST) := r; |
| END LOOP; |
| **FORALL i IN v\_emps.FIRST .. v\_emps.LAST** | Loop through v\_emps Nested Table using **FORALL** and update each record. |
| **UPDATE** emps |
| SET salary = salary + 1000 |
| WHERE employee\_id = v\_emps(i).employee\_id; |
| END; |  |
| / |  |

**Please note:** This block use FORALL statement instead of FOR LOOP. This statement sends a single statement from PL/SQL Engine to SQL Engine.

**Step 6:** To compare between both method, run them in the same block and the time consumed for each as shown in the following block:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| TYPE emps\_type IS TABLE OF emps%ROWTYPE; |  |
| v\_emps emps\_type := emps\_type(); |  |
| p1 PLS\_INTEGER; p2 PLS\_INTEGER; p3 PLS\_INTEGER; |  |
| BEGIN |  |
| -------- Fill in v\_emps Nested Table ------- |  |
| FOR r IN (SELECT \* FROM emps) LOOP |  |
| v\_emps.EXTEND; |  |
| v\_emps(v\_emps.LAST) := r; |  |
| END LOOP; |  |
| p1 := DBMS\_UTILITY.get\_time; |  |
| **FOR i in v\_emps.FIRST .. v\_emps.LAST LOOP** | **Run FOR LOOP** |
| UPDATE emps |
| SET salary = salary + 1000 |
| WHERE employee\_id = v\_emps(i).employee\_id; |
| END LOOP; |
| p2 := DBMS\_UTILITY.get\_time; |  |
| **FORALL i in v\_emps.FIRST .. v\_emps.LAST** | **Run FORALL** |
| UPDATE emps |
| SET salary = salary + 1000 |
| WHERE employee\_id = v\_emps(i).employee\_id; |
| p3 := DBMS\_UTILITY.get\_time; |  |
| dbms\_output.put\_line('--- Time difference (sec) -----'); |  |
| dbms\_output.put\_line('FOR LOOP:'||(p2-p1)/100||', FORALL:'||(p3-p2)/100); | Compare between both of them. |
| END; |  |
| / |  |

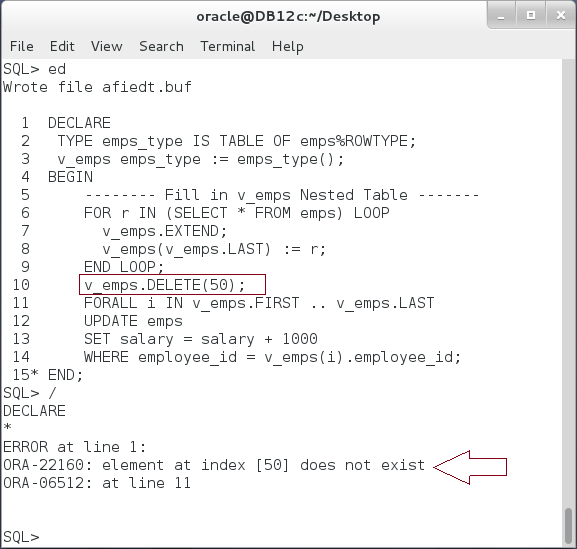


**Please note:** The time required for "FOR LOOP" is much more than the time required for "FORALL".

## FORALL with Sparse Nested Table

**Step 1:** The Nested Table may become sparse after deleting some elements. Execute the following block:

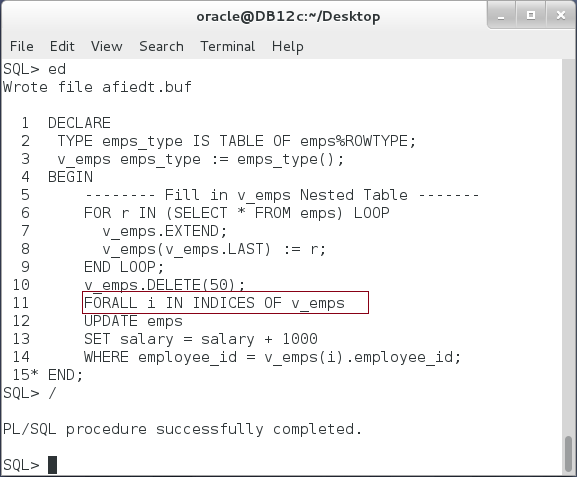
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| TYPE emps\_type IS TABLE OF emps%ROWTYPE; |  |
| v\_emps emps\_type := emps\_type(); |  |
| BEGIN |  |
| -------- Fill in v\_emps Nested Table ------- |  |
| FOR r IN (SELECT \* FROM emps) LOOP |  |
| v\_emps.EXTEND; |  |
| v\_emps(v\_emps.LAST) := r; |  |
| END LOOP; |  |
| **v\_emps.DELETE(50);** | Delete the element number 50. |
| **FORALL i IN v\_emps.FIRST .. v\_emps.LAST** | Use FORALL to send UPDATE statement. |
| UPDATE emps |
| SET salary = salary + 1000 |
| WHERE employee\_id = v\_emps(i).employee\_id; |
| END; |  |
| / |  |



**Please note:** The block fails because the element at index 50 does not exist.

**Step 2:** Modify the previous block as shown below:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| TYPE emps\_type IS TABLE OF emps%ROWTYPE; |  |
| v\_emps emps\_type := emps\_type(); |  |
| BEGIN |  |
| -------- Fill in v\_emps Nested Table ------- |  |
| FOR r IN (SELECT \* FROM emps) LOOP |  |
| v\_emps.EXTEND; |  |
| v\_emps(v\_emps.LAST) := r; |  |
| END LOOP; |  |
| v\_emps.DELETE(50); | Delete element |
| **FORALL i IN INDICES OF v\_emps** | Use FORALL with **INDICES OF** clause. |
| UPDATE emps |
| SET salary = salary + 1000 |
| WHERE employee\_id = v\_emps(i).employee\_id; |
| END; |  |
| / |  |

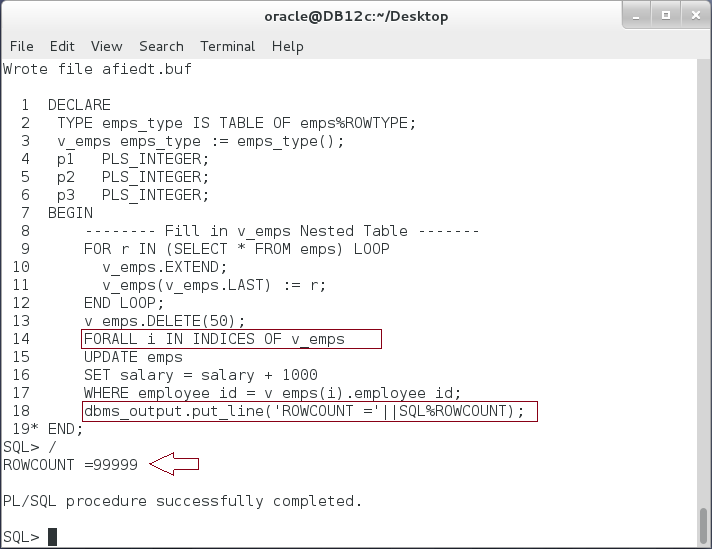


**Please note:** The block succeeded using **INDICES OF** clause when the nested table is sparse.

## BULK\_ROWCOUNT Attribute

**Step 1:** To get the total number of rows affected by the FORALL statement, use the implicit cursor attribute SQL%ROWCOUNT. Execute the following block:

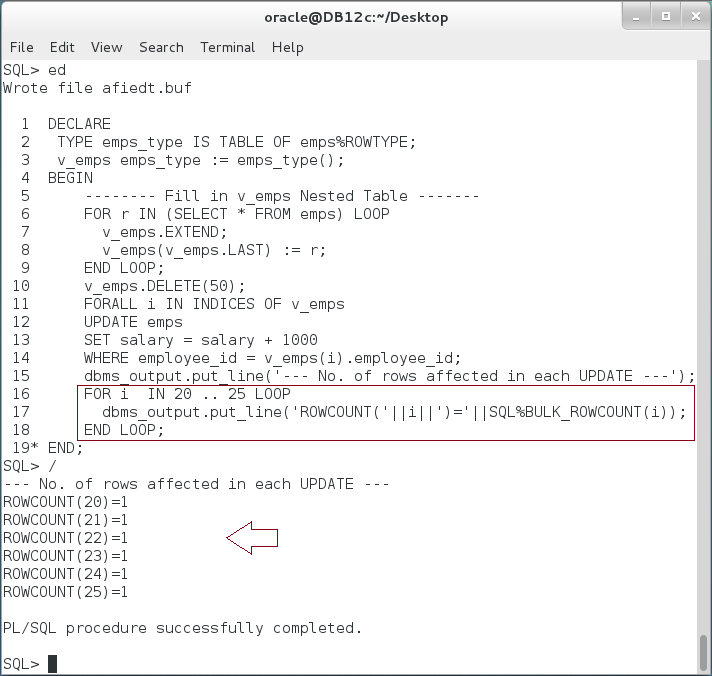
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| TYPE emps\_type IS TABLE OF emps%ROWTYPE; |  |
| v\_emps emps\_type := emps\_type(); |  |
| BEGIN |  |
| -------- Fill in v\_emps Nested Table ------- |  |
| FOR r IN (SELECT \* FROM emps) LOOP |  |
| v\_emps.EXTEND; |  |
| v\_emps(v\_emps.LAST) := r; |  |
| END LOOP; |  |
| v\_emps.DELETE(50); |  |
| **FORALL i IN INDICES OF v\_emps** |  |
| UPDATE emps |  |
| SET salary = salary + 1000 |  |
| WHERE employee\_id = v\_emps(i).employee\_id; |  |
| **dbms\_output.put\_line**  **('ROWCOUNT ='||SQL%ROWCOUNT);** | Get the total number of rows affected by the last DML (FORALL). |
| END; |  |
| / |  |

****

**Please note:** SQL%ROWCOUNT shows the total number of rows affected by the last DML statement, that is FORALL. The number is 99,999 which is 100,000 (total records) - 1 (deleted record).

**Step 2:** Extend the previous block as the following:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| TYPE emps\_type IS TABLE OF emps%ROWTYPE; |  |
| v\_emps emps\_type := emps\_type(); |  |
| BEGIN |  |
| -------- Fill in v\_emps Nested Table ------- |  |
| **FOR r IN (SELECT \* FROM emps) LOOP** |  |
| v\_emps.EXTEND; |  |
| v\_emps(v\_emps.LAST) := r; |  |
| END LOOP; |  |
| v\_emps.DELETE(50); |  |
| **FORALL i IN INDICES OF v\_emps** |  |
| UPDATE emps |  |
| SET salary = salary + 1000 |  |
| WHERE employee\_id = v\_emps(i).employee\_id; |  |
| dbms\_output.put\_line  ('--- No. of rows affected in each UPDATE ---'); |  |
| **FOR i IN 20 .. 25 LOOP** | Display the number of rows affected in each UPDATE of FORALL DML. |
| dbms\_output.put\_line  ('ROWCOUNT('||i||')='||**SQL%BULK\_ROWCOUNT(i)**); |
| **END LOOP;** |
| END; |  |
| / |  |

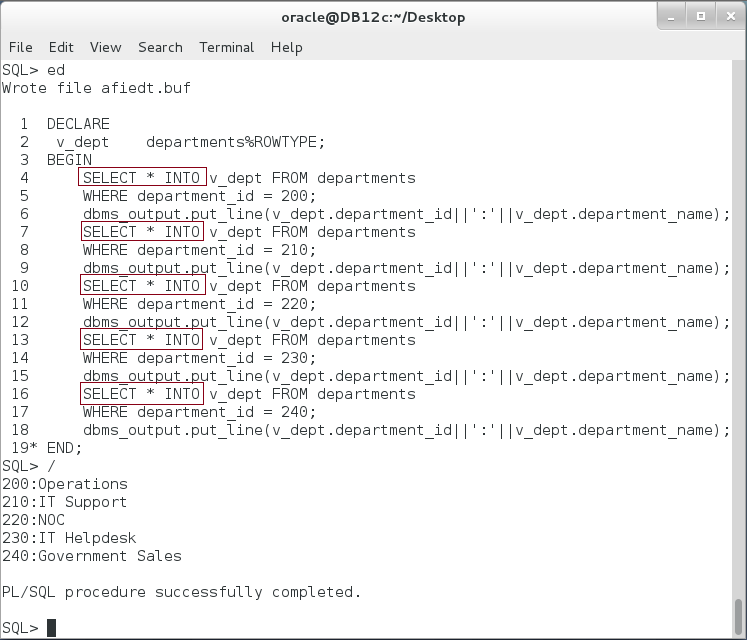


**Please note:** The number of rows affected by **each** UPDATE statement is always 1 because we use the primary key in the statement. However, it might be different if you use other criteria in FORALL UPDATE statement.

## BULK COLLECT

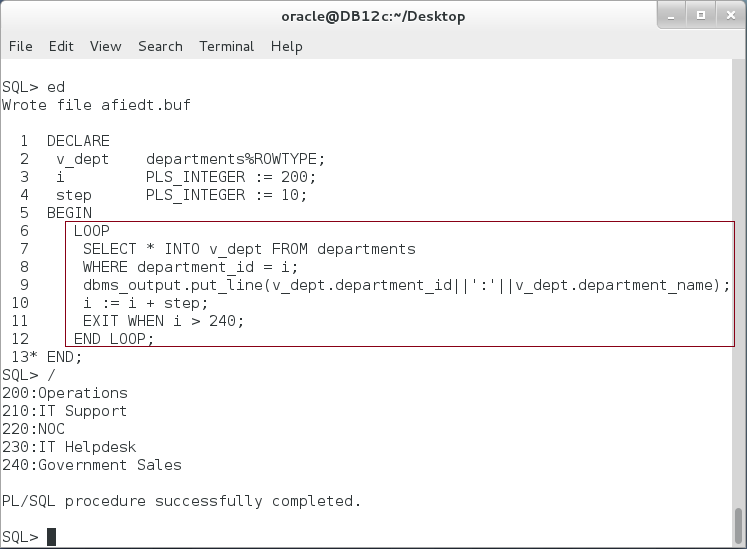
**Step 1:** Execute the following block:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| v\_dept departments%ROWTYPE; |  |
| BEGIN |  |
| **SELECT \* INTO** v\_dept FROM departments | **Send first** query and **fetch** the return value into variable |
| WHERE department\_id = **200**; |
| dbms\_output.put\_line  (v\_dept.department\_id||':'||v\_dept.department\_name); |
| **SELECT \* INTO** v\_dept FROM departments | **Send second** query and **fetch** the return value into variable |
| WHERE department\_id = **210**; |
| dbms\_output.put\_line  (v\_dept.department\_id||':'||v\_dept.department\_name); |
| **SELECT \* INTO** v\_dept FROM departments | **Send third** query and **fetch** the return value into variable |
| WHERE department\_id = **220**;   dbms\_output.put\_line  (v\_dept.department\_id||':'||v\_dept.department\_name); |
| **SELECT \* INTO** v\_dept FROM departments | **Send fourth** query and **fetch** the return value into variable |
| WHERE department\_id = **230**; |
| dbms\_output.put\_line  (v\_dept.department\_id||':'||v\_dept.department\_name); |
| **SELECT \* INTO** v\_dept FROM departments | **Send fifth** query and **fetch** the return value into variable |
| WHERE department\_id = **240**; |
| dbms\_output.put\_line  (v\_dept.department\_id||':'||v\_dept.department\_name); |
| END; |  |
| / |  |

****

**Step 2:** The previous block seems irrational. It repeats relatively similar code multiple times. The first thought is to use one of loop statements. Modify the previous block as the following:

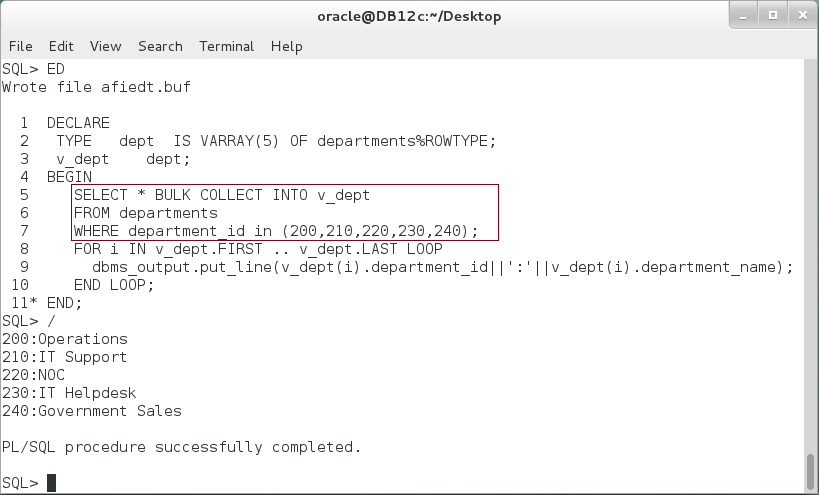
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| v\_dept departments%ROWTYPE; |  |
| i PLS\_INTEGER := 200; |  |
| step PLS\_INTEGER := 10; |  |
| BEGIN |  |
| **LOOP** | Basic LOOP is used to do the repeated job. |
| **SELECT \* INTO** v\_dept FROM departments |
| WHERE department\_id = i; |
| dbms\_output.put\_line  (v\_dept.department\_id||':'||v\_dept.department\_name); |
| i := i + step; |
| **EXIT WHEN i > 240;** |  |
| **END LOOP;** |  |
| END; |  |
| / |  |



**Please note:** This block seems more rational. Although, it does the same steps shown in the first block; it sends the query to SQL engine repeatedly and fetch the data for each.

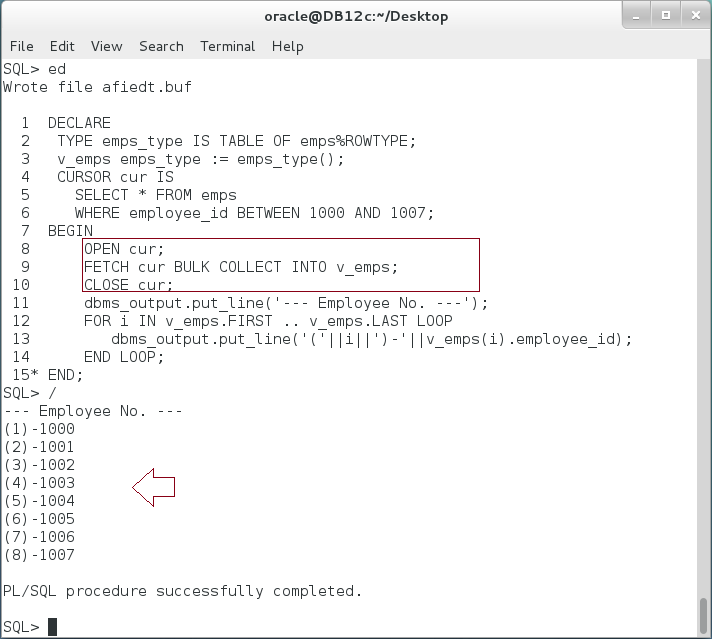
**Step 3:** Change the previous block again as shown below:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| TYPE dept IS VARRAY(5) OF departments%ROWTYPE; |  |
| v\_dept dept; |  |
| BEGIN |  |
| **SELECT \* BULK COLLECT INTO** v\_dept | BULK COLLECT fetches all rows at once. |
| FROM departments |
| WHERE department\_id in (200,210,220,230,240); |
| FOR i IN v\_dept.FIRST .. v\_dept.LAST LOOP |  |
| dbms\_output.put\_line  (v\_dept(i).department\_id||':'||v\_dept(i).department\_name); |  |
| END LOOP; |  |
| END; |  |

****

**Step 4:** BULK COLLECT can also be used in FETCH statements. Execute the following block:

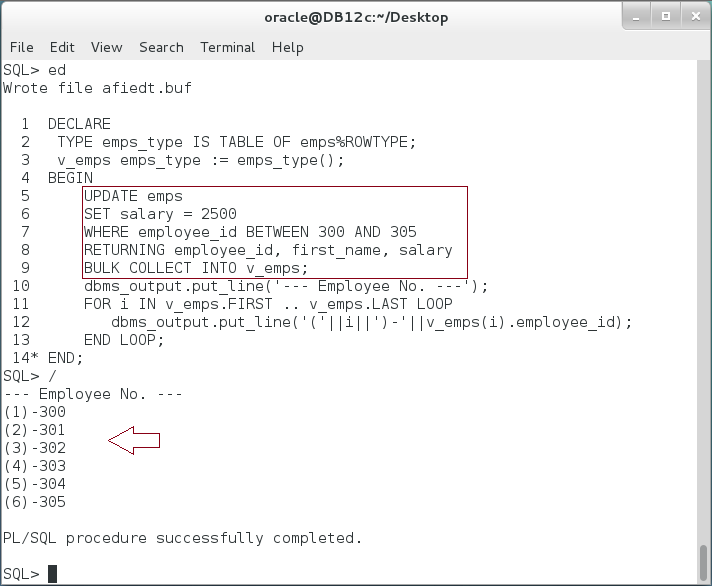
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| TYPE emps\_type IS TABLE OF emps%ROWTYPE; |  |
| v\_emps emps\_type := emps\_type(); |  |
| **CURSOR cur IS** |  |
| SELECT \* FROM emps |  |
| WHERE employee\_id BETWEEN 1000 AND 1007; |  |
| BEGIN |  |
| **OPEN** cur; | BULK COLLECT fetches all rows at once. |
| **FETCH** cur **BULK COLLECT INTO** v\_emps**;** |
| **CLOSE** cur; |
| dbms\_output.put\_line('--- Employee No. ---'); |  |
| FOR i IN v\_emps.FIRST .. v\_emps.LAST LOOP |  |
| dbms\_output.put\_line('('||i||')-'||v\_emps(i).employee\_id); |  |
| END LOOP; |  |
| END; |  |
| / |  |

****

**Please:** Compare between FETCH INTO and FETCH BULK COLLECT INTOin performance.

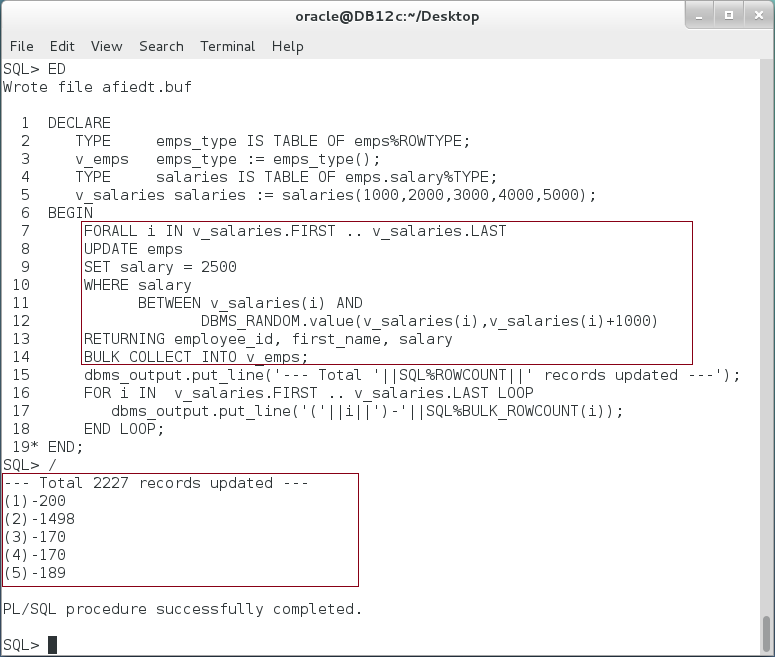
**Step 5:** BULK COLLECTcan be use in RETURNING clause of DML statements. Change the previous block again as shown below:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| TYPE emps\_type IS TABLE OF emps%ROWTYPE; |  |
| v\_emps emps\_type := emps\_type(); |  |
| BEGIN |  |
| **UPDATE** emps | BULK COLLECT fetches all rows at once. |
| **SET** salary = 2500 |
| **WHERE** employee\_id BETWEEN 300 AND 305 |
| **RETURNING** employee\_id, first\_name, salary |
| **BULK COLLECT INTO** v\_emps; |
| dbms\_output.put\_line('--- Employee No. ---'); |  |
| FOR i IN v\_emps.FIRST .. v\_emps.LAST LOOP |  |
| dbms\_output.put\_line('('||i||')-'||v\_emps(i).employee\_id); |  |
| END LOOP; |  |
| END; |  |
| / |  |

****

**Step 6:** You can gather FORALL and RETURNING BULK COLLECT clauses in the same DML statement. Change the previous block again as shown below:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| TYPE emps\_type IS TABLE OF emps%ROWTYPE; |  |
| v\_emps emps\_type := emps\_type(); |  |
| TYPE salaries IS TABLE OF emps.salary%TYPE; |  |
| v\_salaries salaries := salaries(1000,2000,3000,4000,5000); |  |
| BEGIN |  |
| **FORALL** **i IN v\_salaries.FIRST .. v\_salaries.LAST** | A single DML statement gathers **FORALL** and **RETURNING BULK COLLECT** |
| UPDATE emps |
| SET salary = 2500 |
| WHERE salary |
| BETWEEN v\_salaries(i) AND |
| DBMS\_RANDOM.value(v\_salaries(i),v\_salaries(i)+1000) |
| **RETURNING employee\_id, first\_name, salary** |
| **BULK COLLECT INTO v\_emps;** |
| dbms\_output.put\_line('--- Total '||  SQL%ROWCOUNT||' records updated ---'); |  |
| FOR i IN v\_salaries.FIRST .. v\_salaries.LAST LOOP |  |
| dbms\_output.put\_line('('||i||')-'||  SQL%BULK\_ROWCOUNT(i)); |  |
| END LOOP; |  |
| END; |  |
| / |  |

****

**Please note:** In a FORALL statement, the DML statement can have a RETURNING BULK COLLECT INTO clause. For each iteration of the FORALL statement, the DML statement stores the specified values in the specified collections—without overwriting the previous values

## Limiting Bulk Selection

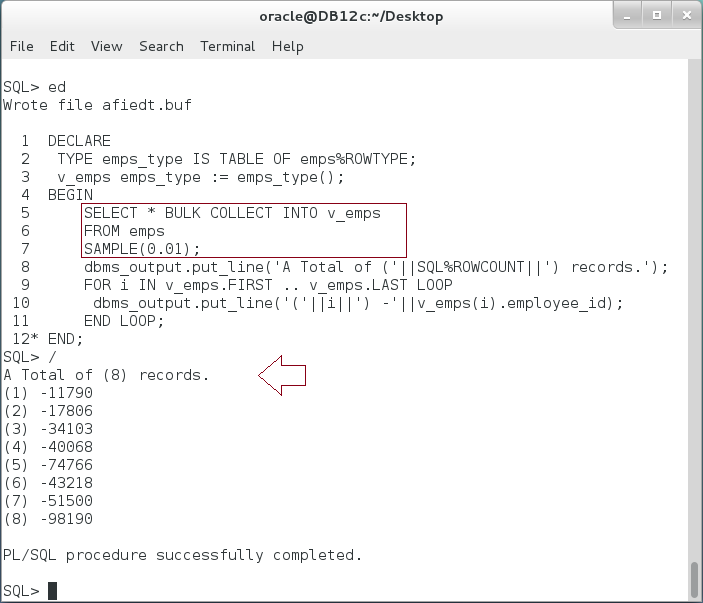
**Step 1:** In some cases, the BULK COLLECT may need consume un-necessary time fetching large collection while you need a subset of data. To limit the number of rows fetched in a bulk statement, you may use WHERE clause with any suitable condition. In addition, you may use **ROWNUM** pseudocolumn, **SAMPLE** clause, or **FETCH FIRST** clause. Execute the following block:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| TYPE emps\_type IS TABLE OF emps%ROWTYPE; |  |
| v\_emps emps\_type := emps\_type(); |  |
| BEGIN |  |
| **SELECT \* BULK COLLECT INTO** v\_emps | Limit the number of rows using ROWNUM |
| FROM emps |
| WHERE **ROWNUM <= 10;** |
| dbms\_output.put\_line('A Total of  ('||SQL%ROWCOUNT||') records.'); |
| FOR i IN v\_emps.FIRST .. v\_emps.LAST LOOP |
| dbms\_output.put\_line('('||i||') -  '||v\_emps(i).employee\_id); |
| END LOOP; |
| END; |
| / |  |

**Please note:** Only 10 out of 100,000 rows was fetched using ROWNUM condition. Rerun the previous block several times and notice if any change appear on the result.

**Step 2:** Modify the previous block as shown in the following block:

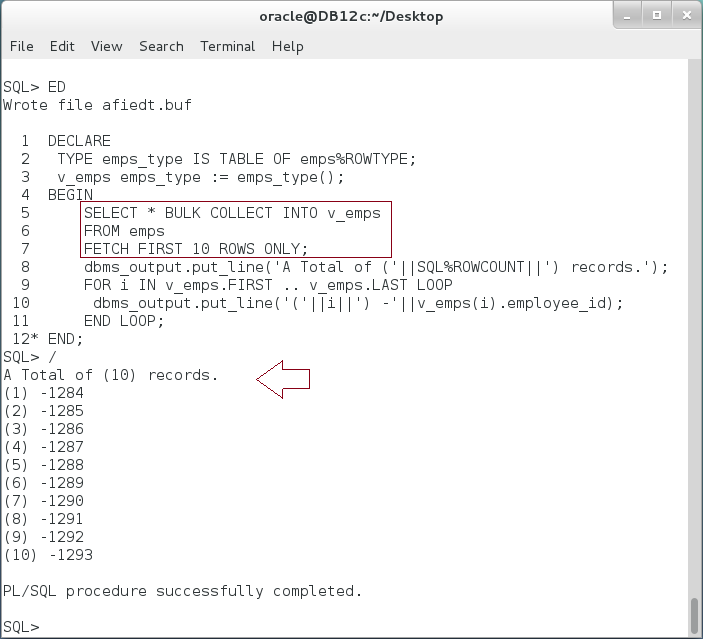
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| TYPE emps\_type IS TABLE OF emps%ROWTYPE; |  |
| v\_emps emps\_type := emps\_type(); |  |
| BEGIN |  |
| **SELECT \* BULK COLLECT INTO** v\_emps | Limit the number of rows using **SAMPLE** condition. |
| FROM emps |
| **SAMPLE(0.01);** |
| dbms\_output.put\_line('A Total of  ('||SQL%ROWCOUNT||') records.'); |
| FOR i IN v\_emps.FIRST .. v\_emps.LAST LOOP |
| dbms\_output.put\_line('('||i||') -  '||v\_emps(i).employee\_id); |
| END LOOP; |
| END; |
| / |  |



**Please note:** Re-execute the block again and notice the difference. The SAMPLE clause lets you instruct the database to select from a random sample of data from the table, rather than from the entire table.

**Step 3:** Modify the previous block as shown below:

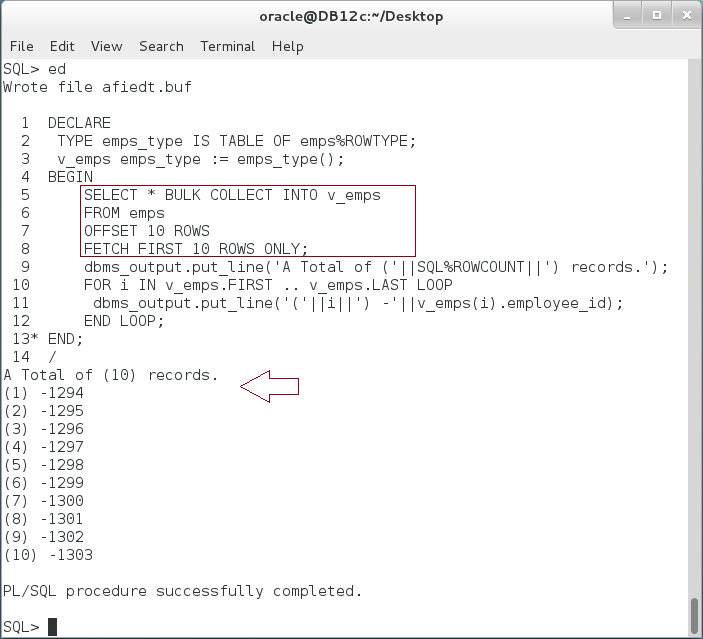
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| TYPE emps\_type IS TABLE OF emps%ROWTYPE; |  |
| v\_emps emps\_type := emps\_type(); |  |
| BEGIN |  |
| **SELECT \* BULK COLLECT INTO** v\_emps | Limit the number of rows using **FETCH FIRST** condition. |
| FROM emps |
| **FETCH FIRST 10 ROWS ONLY;** |
| dbms\_output.put\_line('A Total of  ('||SQL%ROWCOUNT||') records.'); |
| FOR i IN v\_emps.FIRST .. v\_emps.LAST LOOP |
| dbms\_output.put\_line('('||i||') -  '||v\_emps(i).employee\_id); |
| END LOOP; |
| END; |
| / |  |

****

**Please note:** FETCH FIRST clause may help you to subset the result in pages and get top records (instead of RANK OVER).

**Step 4:** To fetch the next page using FETCH FIRST clause, you need to extend the previous block as shown below:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| TYPE emps\_type IS TABLE OF emps%ROWTYPE; |  |
| v\_emps emps\_type := emps\_type(); |  |
| BEGIN |  |
| **SELECT \* BULK COLLECT INTO** v\_emps | Use **FETCH NEXT** to get the next page. |
| FROM emps |
| **OFFSET 10 ROWS** |
| **FETCH NEXT 10 ROWS ONLY;** |
| dbms\_output.put\_line('A Total of  ('||SQL%ROWCOUNT||') records.'); |
| FOR i IN v\_emps.FIRST .. v\_emps.LAST LOOP |
| dbms\_output.put\_line('('||i||') -  '||v\_emps(i).employee\_id); |
| END LOOP; |
| END; |
| / |  |

****

# SUMMARY

In addition to a normal loop statements, Oracle supports FORALL in which you may submit hundreds of DML statements once from PL/SQL Engine to SQL Engine. This draws a considerable performance enhancement especially for large number of statements. Oracle recommends using FORALL for 4 or more statements. To retrieve data back from SQL Engine to PL/SQL Engine, you may consider using BULK COLLECT clause. It gathers multiple rows into one associative array and sends it once to PL/SQL Engine. BULK COLLECT can be easily integrated with SELECT INTO, FETCH INTO, and RETURNING INTO. Moreover, BULK COLLECT clause and FORALL can also be used in the same DML statement.

After completing this lab exercise, you should be able to use FORALL and BULK COLLECT clauses in PL/SQL.

# REFERENCES

* http://docs.oracle.com/database/121/LNPLS/tuning.htm#LNPLS01205

# INDEX

BULK COLLECT 2, 3, 4, 18, 20, 21, 22, 23, 24, 25, 26, 27, 28, 30

FORALL 2, 3, 4, 6, 9, 10, 11, 12, 13, 14, 15, 16, 17, 23, 24, 25, 30