DBA

Cursors

Cursors and BULK COLLECT INTO in Oracle

The codes can be found in github.and in gist.

Oracle creates a memory area for processing an SQL Statement, which contains all the information needed for processing the statement called the **context area**. For example, the number of rows processed.

A **cursor** is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows(one or more) returned by a SQL Statement, unlike **SELECT … INTO** Statement which holds only one value and transfers into a variable. The number of rows the cursor holds is referred to as the **active set**.

The two type of cursors are :

1. Implicit Cursor
2. Explicit Cursor

1. **Implicit Cursor:**

Whenever we execute a SQL Statement without explicit cursor assigned, Oracle automatically creates an implicit cursor.

Every DML Statement(Insert, update and Delete) is associated with an Implicit Cursor. For insert operation, Cursor holds the data that needs to be inserted. For Delete and Update operation, the cursor identifies the rows that would be affected.

The implicit cursor is measurably faster than explicit cursor. Also,The implicit cursor is not only faster, but it is actually doing more work, since it includes a `NO_DATA_FOUND` and a `TOO_MANY_ROWS` exception check.

2. **Explicit Cursor:**

Explicit Cursor is a programmer-defined cursors for gaining more control over the context area. An explicit cursor is declared in declaration section of PL/SQL Block and is used with SELECT Statement to point one or more rows.

The steps for creating an explicit cursor is shown below:

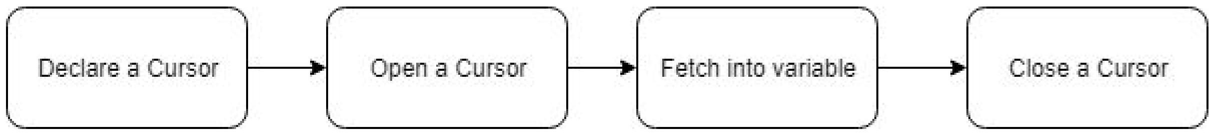


Fig: Steps for Explicit Cursor

The syntax

For Declaring:

```
CURSOR cursor_name IS select_statement;
```

For Opening:

```
OPEN cursor_name;
```

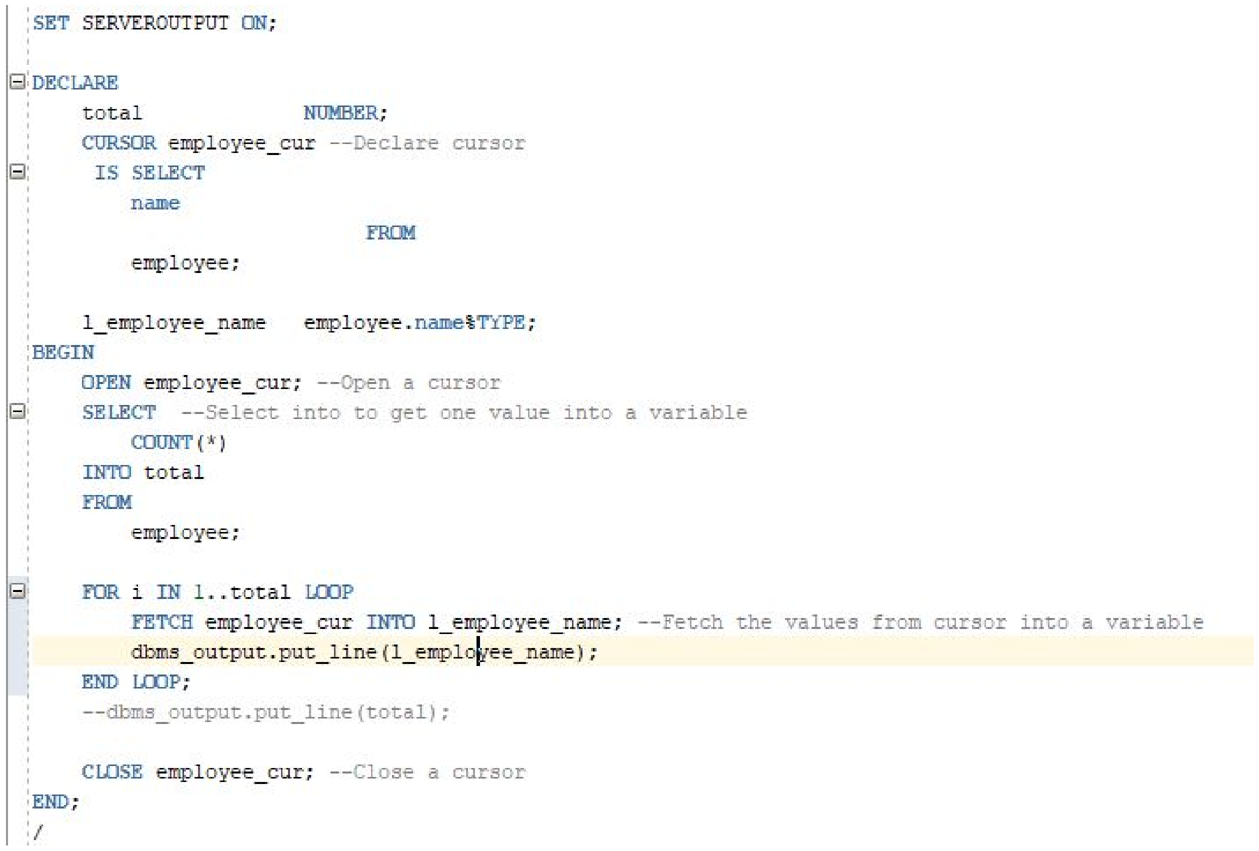
For Fetching a cursor into a variable:

```
FETCH cursor_name INTO variable_name;
```

For Closing:

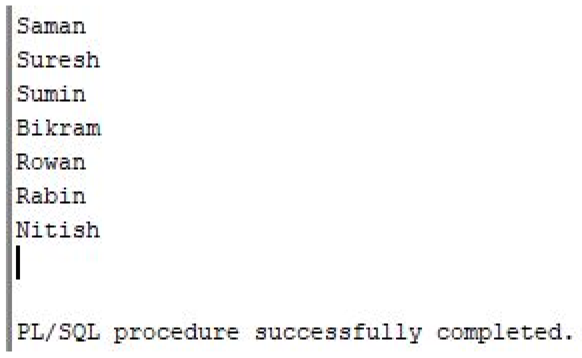
```
CLOSE cursor_name;
```

An example of explicit cursor is:



```
SET SERVEROUTPUT ON;

DECLARE
    total              NUMBER;
    CURSOR employee_cur --Declare cursor
     IS SELECT
        name
                          FROM
        employee;

    l_employee_name   employee.name%TYPE;
BEGIN
    OPEN employee_cur; --Open a cursor
    SELECT  --Select into to get one value into a variable
        COUNT(*)
    INTO total
    FROM
        employee;

    FOR i IN 1..total LOOP
        FETCH employee_cur INTO l_employee_name; --Fetch the values from cursor into a variable
        dbms_output.put_line(l_employee_name);
    END LOOP;
    --dbms_output.put_line(total);

    CLOSE employee_cur; --Close a cursor
END;
/
```

Fig: Explicit Cursor example

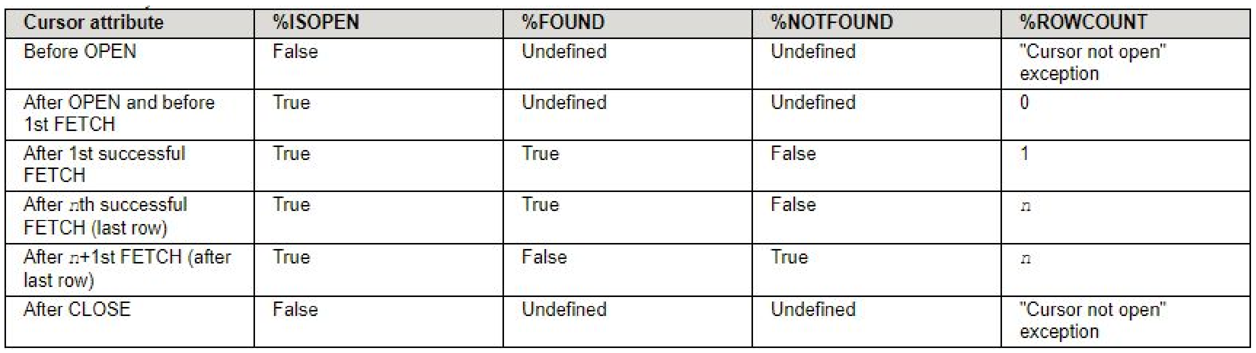


```
Saman
Suresh
Sumin
Bikram
Rowan
Rabin
Nitish


PL/SQL procedure successfully completed.
```

Fig: Output of an explicit cursor(above code)

Cursor Attributes:

Some of the cursor attributes are



Cursor attribute	%ISOPEN	%FOUND	%NOTFOUND	%ROWCOUNT
Before OPEN	False	Undefined	Undefined	"Cursor not open" exception
After OPEN and before 1st FETCH	True	Undefined	Undefined	0
After 1st successful FETCH	True	True	False	1
After *n*th successful FETCH (last row)	True	True	False	*n*
After *n*+1st FETCH (after last row)	True	False	True	*n*
After CLOSE	False	Undefined	Undefined	"Cursor not open" exception

Fig: Cursor Attributes

Performance:

Besides the use of cursor, as a DBA, we should also be concerned with the performance including the speed, memory and cost.

Thanks to Oracle-base, It is clear that implicit cursors are faster than the explicit cursor and the speed can further be enhanced with **Cursor FOR Loop** .

Oracle Experts Recommends:

- Never use a cursor FOR loop when you're writing new code for normal production deployment in a multi-user application.
- Use **SELECT ... INTO** Statement if you expects to retrieve only a single row.
- If you expect to retrieve multiple rows of data and you **know the upper limit** (as in, "I will never get more than 100 rows in this query"), use **BULK COLLECT** into a collection of type **varray** whose upper limit matches what you know about the query.
- If you expect to retrieve multiple rows of data and you **do not know the upper limit**, use **BULK COLLECT with a FETCH statement** that relies on a **LIMIT clause** to ensure that you do not consume too much per-session memory.
- If your existing code contains a **cursor FOR loop**, you should perform a cost-benefit analysis on converting that code, based on these recommendations.

BULK COLLECT:

BULK COLLECT is very handy when we are trying to query large volume of records. BULK COLLECT offers a faster way of querying data; however, it consumes more memory.

"BULK COLLECT runs faster but consume more memory."

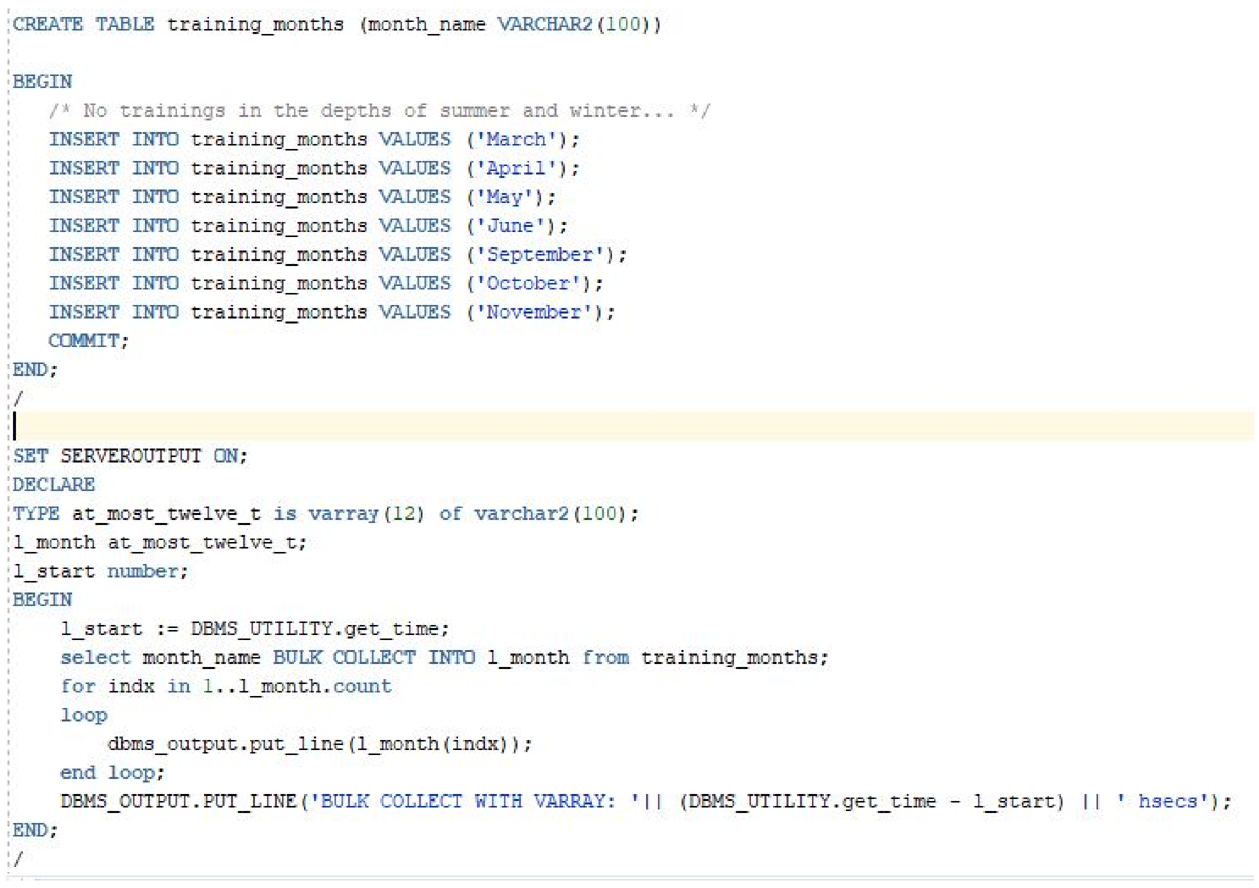
Convert existing cursor FOR loops only when necessary. Oracle Database does automatically optimize the performance of cursor FOR loops. They do not generally execute as efficiently as explicitly coded BULK COLLECT statements, but they are much more performant than single-row fetches. Consequently, Expert suggest that you convert cursor FOR loops to BULK COLLECT retrievals only if you identify a performance bottleneck in that part of our code. Otherwise, leave the cursor FOR loop in place.

BULK COLLECT WHEN YOU KNOW THE UPPER LIMIT:

As mentioned earlier, when we know the upper limit in bulk collect, it is good to use BULK COLLECT into a varry.

A **varray** is a collection that has an upper limit on the number of elements that can be defined in the collection. This upper limit is specified when the varray type is declared; it can also be modified afterward.

Varrays offer a very nice mechanism when we need to retrieve multiple rows of data efficiently and the number of rows should never exceed a certain limit.



```
CREATE TABLE training_months (month_name VARCHAR2(100))

BEGIN
   /* No trainings in the depths of summer and winter... */
   INSERT INTO training_months VALUES ('March');
   INSERT INTO training_months VALUES ('April');
   INSERT INTO training_months VALUES ('May');
   INSERT INTO training_months VALUES ('June');
   INSERT INTO training_months VALUES ('September');
   INSERT INTO training_months VALUES ('October');
   INSERT INTO training_months VALUES ('November');
   COMMIT;
END;
/

SET SERVEROUTPUT ON;
DECLARE
TYPE at_most_twelve_t is varray(12) of varchar2(100);
l_month at_most_twelve_t;
l_start number;
BEGIN
    l_start := DBMS_UTILITY.get_time;
    select month_name BULK COLLECT INTO l_month from training_months;
    for indx in 1..l_month.count
    loop
        dbms_output.put_line(l_month(indx));
    end loop;
    DBMS_OUTPUT.PUT_LINE('BULK COLLECT WITH VARRAY: '|| (DBMS_UTILITY.get_time - l_start) || ' hsecs');
END;
/
```

Fig : BULK COLLECT with VARRAY

BULK COLLECT WHEN YOU DON'T KNOW THE UPPER LIMIT OR WHEN THE UPPER LIMIT IS TOO HIGH:

OR

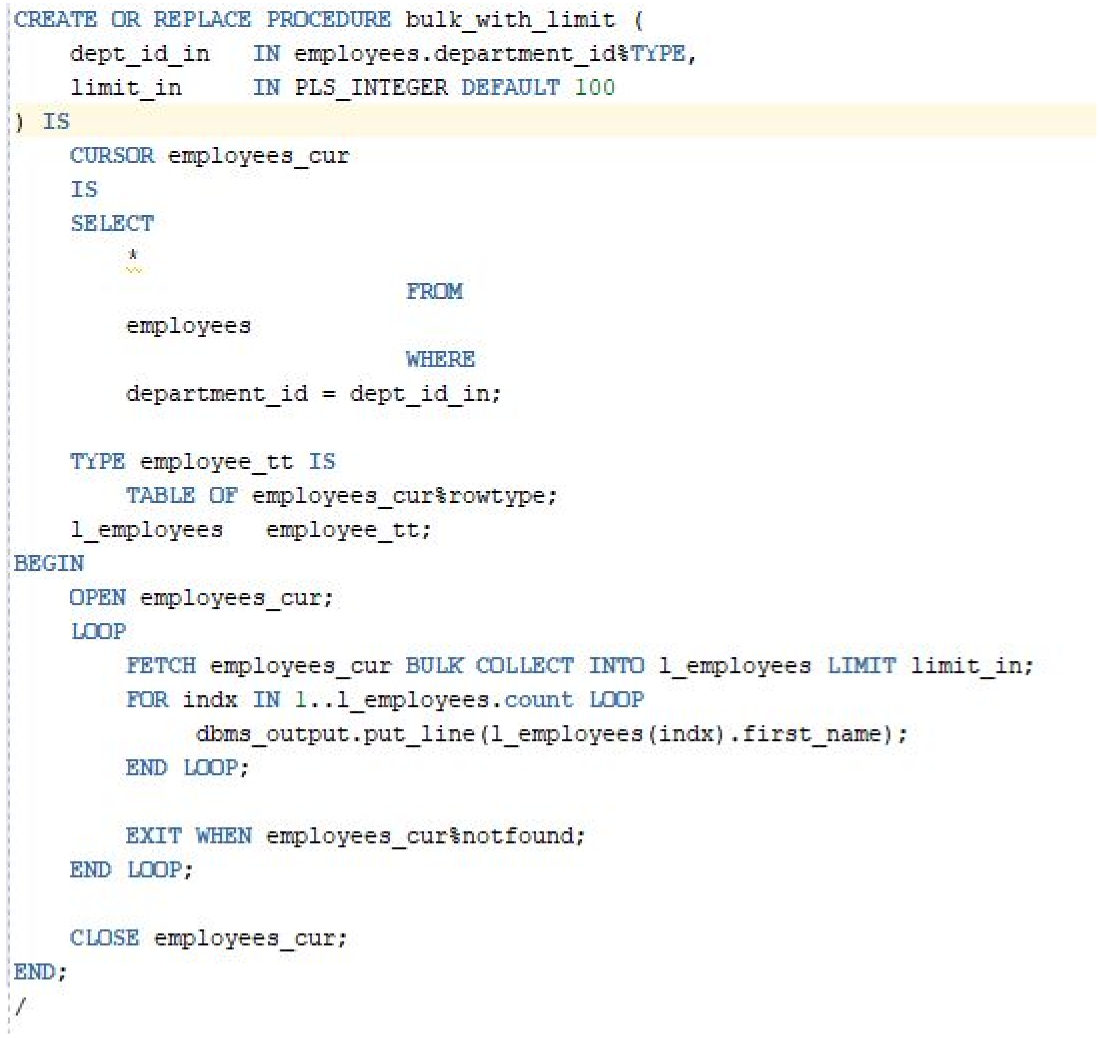
BULK COLLECT WITH LIMIT CLAUSE:

What if you know the maximum number of elements that can appear in the varray and that maximum is 1,000,000? This technique will "work," but the program will consume a *dangerously large amount of per-session memory*. In this case, you should **forsake the varray**. Instead, switch to an associative array or nested table and use the **LIMIT clause with BULK COLLECT**.

BULK COLLECT with LIMIT when you don't know the upper limit. BULK COLLECT helps retrieve multiple rows of data **quickly**. Rather than retrieve one row of data at a time into a record or a set of individual variables, BULK COLLECT lets us **retrieve hundreds, thousands, even tens of thousands of rows with a single context switch** to the SQL engine and deposit all that data into a collection. The resulting performance improvement can be an order of magnitude or greater.

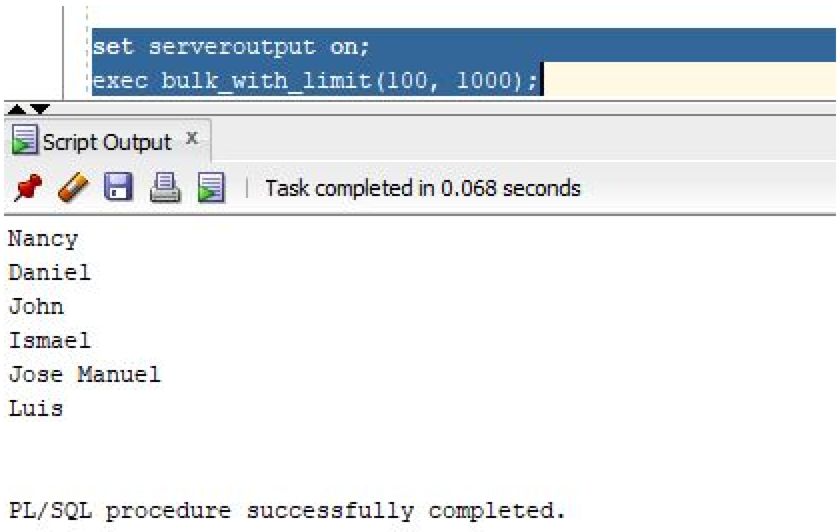
However, in such a case, that boost in performance results in an **increase in the amount of per-session memory** consumed by the collection populated by the query. In addition, each session connected to Oracle Database has its own per-session memory area.

Therefore, we should use **BULK COLLECT with the LIMIT clause.**



```
CREATE OR REPLACE PROCEDURE bulk_with_limit (
    dept_id_in   IN employees.department_id%TYPE,
    limit_in     IN PLS_INTEGER DEFAULT 100
) IS
    CURSOR employees_cur
    IS
    SELECT
        *
                            FROM
        employees
                            WHERE
        department_id = dept_id_in;

    TYPE employee_tt IS
        TABLE OF employees_cur%rowtype;
    l_employees   employee_tt;
BEGIN
    OPEN employees_cur;
    LOOP
        FETCH employees_cur BULK COLLECT INTO l_employees LIMIT limit_in;
        FOR indx IN 1..l_employees.count LOOP
             dbms_output.put_line(l_employees(indx).first_name);
        END LOOP;

        EXIT WHEN employees_cur%notfound;
    END LOOP;

    CLOSE employees_cur;
END;
/
```

Fig: BULK COLLECT WITH LIMIT CLAUSE



```
set serveroutput on;
exec bulk_with_limit(100, 1000);
```

Script Output

Task completed in 0.068 seconds

```
Nancy
Daniel
John
Ismael
Jose Manuel
Luis


PL/SQL procedure successfully completed.
```

Fig : Output for above procedure

The LIMIT value can depend upon data set. I have used a limit value of 1000 and a default value 100 -- incase if it is not provided . This means that, **on each fetch it retrieves 1000 rows** or 100 (if used in default).

References:

1. http://www.oracle.com/technetwork/testcontent/o68plsql-088608.html
2. http://www.oracle.com/technetwork/issue-archive/2013/13-mar/o23plsql-1906474.html
3. http://www.oracle.com/technetwork/issue-archive/2008/08-mar/o28plsql-095155.html