## **Cursors**

(Notes #4)

Topics

1. Implicit cursors
2. Explicit cursors (declare, open, fetch, close)
3. Definition (page 3)
4. Fetch data with cursor (page 4)
5. Cursor based PL/SQL records (%ROWTYPE) (page 5)
6. Cursor For loops (page 7)
7. Cursors that accepts parameters (page 9)
8. Cursor for UPDATE (page 12)
9. Cursor Variable (page 13) (skip for this week. No time to discuss this in week #3)

Oracle creates a memory area, known as the context area, which contains all the information needed for processing the statement; for example, the number of rows processed, etc~~.~~

A cursor is a pointer to a private SQL area that stores information about processing a

specific SELECT or DML statement.

A (static) cursor is an object, which residents in SGA (System Global Area). PL/SQL controls the context area through a cursor.

A select statement of SELECT \* FROM emp,

returns a set of records (also called active set) to the context area in memory.

EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO

--------- ---------- --------- --------- --------- ------- -------- --------

7369 SMITH CLERK 7902 17-DEC-80 800 300 20

7499 ALLEN SALESMAN 7698 20-FEB-81 1600 300 30

7521 WARD SALESMAN 7698 22-FEB-81 1250 500 30

7566 JONES MANAGER 7839 02-APR-81 2975 20

7654 MARTIN SALESMAN 7698 28-SEP-81 1250 1400 30

7698 BLAKE MANAGER 7839 01-MAY-81 2850 30

7782 CLARK MANAGER 7839 09-JUN-81 2450 10

. . .

In PL/SQL there are two kinds of cursors: explicit and implicit.

A cursor that is constructed and managed by PL/SQL is an implicit cursor.

A cursor that users construct and manage is an explicit cursor.

1. **Implicit cursors (SQL%)**

Whenever any DML (or SELECT) operations occur in the database, an implicit cursor is created that holds the rows affected, in that particular operation. These cursors cannot be named and, hence they cannot be controlled or referred from another place of the code. We can refer only to the most recent cursor through the cursor attributes.

A cursor is implicitly declared for each data manipulation SQL (DML) statement, including queries (select) that return only one row.

There is no declaration of implicit cursors in the declaration area.

PL/SQL implicitly opens a cursor to process each SQL statement not associated with an explicitly declared cursor.

Implicit Cursor attributes:

SQL%FOUND Returns TRUE if a DML statement affected one or more records;

SQL%NOTFOUND opposite of SQL%FOUND.

SQL%ROWCOUNT Returns the numbers of records affected,

SQL%ISOPEN Always returns FALSE, because the system closes the cursor

automatically after executing its associated SQL statement.

**Example 1** - using the attribute (**SQL%FOUND)**of an implicit cursor for a DML command

DROP TABLE temp\_emp; -- in case you have created before

CREATE TABLE temp\_emp AS SELECT \* FROM emp;

-- Select \* from temp\_emp ;

SET SERVEROUTPUT ON -- you only need to run this one time after login

DECLARE

Dept\_number emp.deptno%TYPE := 30;

BEGIN

DELETE FROM temp\_emp

WHERE deptno = Dept\_number;

IF **SQL%FOUND** THEN

DBMS\_OUTPUT.PUT\_LINE (

'Delete succeeded for department number '||Dept\_number||'.');

ELSE

DBMS\_OUTPUT.PUT\_LINE ('No such a department number ' || Dept\_number);

END IF;

END;

/

Result:

Delete succeeded for department number 30.

(rollback;)

Run the code again while changing the department number from 30 to 33,

Result:

No such a department number 33

**Example 2** - using the attribute (SQL%ROWCOUNT**)**of an implicit cursor

DECLARE

mgr\_no emp.mgr%TYPE := 7566;

BEGIN

DELETE FROM temp\_emp WHERE mgr = mgr\_no;

DBMS\_OUTPUT.PUT\_LINE

('Number of employees deleted: ' || TO\_CHAR(SQL%ROWCOUNT));

END;

/

Result:

Number of employees deleted: 2

1. **Explicit cursors**

When SELECT statement returns two or more than two rows, a simple "SELECT-INTO" clause does not work any more.

You can explicitly define a cursor to

process beyond the first row returned by the query,

keep track of which row is currently being processed.

Cursor allows the PL/SQL to process rows one at a time.

**Declare/Define** :in declaration section.

**Open**  : initialize the cursor before any rows are returned by the query.

**Fetch**  : to retrieve the cursor' current row, advances the cursor to the next row. Fetch

can be executed repeatedly until all rows have been retrieved.

**Close**  : close the cursor.

Explicit Cursor attributes:

cursor\_name %notfound If true, there are no more rows in the cursor. If false,

there are remaining rows in the cursor.

cursor\_name %found If true, this attribute tells that last row fetched from a cursor

returned data. If false, there are no more rows in the cursor.

cursor\_name %rowcount the number of rows already processed in the cursor.

cursor\_name %isopen If true, the cursor is open. If false, the cursor is closed.

These attributes can help you to process data and maintain your program.

Basic steps of define a user-defined cursor, open, Fetch-cursor-into, close cursor, using

%NOTFOUND as exit loop condition.

DECLARE

num1 table\_name.col\_name1%TYPE;

num2 table\_name.col\_name2%TYPE;

CURSOR X **IS** -- pay attention to syntax

SELECT col\_name1, col\_name2

FROM table\_name

WHERE predicate\_expression;

BEGIN

OPEN X;

LOOP

FETCH X INTO num1, num2;

-- refer “select select\_list into”; matching the data item and type.

EXIT WHEN X%NOTFOUND;

executable statements;

END LOOP;

CLOSE X;

END;

1. **Declaring/Defining:**

An explicit cursor definition

DECLARE

CURSOR *cursor\_name* [*parameter\_list*] [RETURN *return\_type*]

IS *select\_statement*;

Example:

DECLARE

CURSOR c\_emp return emp%ROWTYPE IS

Select \* from emp;

or

CURSOR c\_emp return IS

Select \* from emp;

Note (for future):

at the most times in PL/SQL, we declare and define the explicit cursor at the same time as the example above. But you may declare a cursor first, then define it later in the same block (we will discuss that in subprogram, package).

An explicit cursor declaration (only, no definition)

DECLARE

CURSOR *cursor\_name* [*parameter\_list*] RETURN *return\_type*;

Return spec: row type structure. It could be a defined RECORD TYPE.

1. **Fetching data with cursor**

As it has mentioned before, The FETCH statement retrieves the current row of the result set, stores the column values of that row into the variables or record, and advances the cursor to the next row.

The INTO clause is either a list of variables or a single record variable. For each column that the query returns, the variable list or record must have a corresponding type-compatible variable or field.

Fetch can be executed repeatedly until all rows have been retrieved. Typically, it is processed inside the loop statement. Cursor\_name%NOTFOUND attribute can be used inside the loop to exit.

The %TYPE and %ROWTYPE attributes are useful for declaring variables and records for use in FETCH statements.

**Example 3 -** typical basic steps of “open, loop, fetch-cursor-into, close”, using cursor\_name%NOTFOUND for exit loop condition, while process every row retrieved by cursor.

DECLARE

CURSOR cemp IS

SELECT ename, sal

FROM emp

WHERE deptno = 10

ORDER BY sal DESC;

Emp\_name emp.ename%TYPE; -- variable to store the value of column

salary emp.sal%TYPE;

BEGIN

OPEN cemp;

LOOP

FETCH cemp INTO Emp\_name, salary; -- INTO list of variables

EXIT WHEN cemp%NOTFOUND; -- Control EXIT

DBMS\_OUTPUT.put\_line

('current row number is '|| cemp%ROWCOUNT || ': ' ||

RPAD (emp\_name, 10) || ': ' ||TO\_CHAR (salary, '$999,999'));

END LOOP;

CLOSE cemp;

END;

/

Output:

current row number is 1: KING : $5,000

current row number is 2: CLARK : $2,450

current row number is 3: MILLER : $1,300

**Example 4**. This sample declares a record variable (%ROWTYPE) which represents a full row of the database table employees. Use dot notation to access record variable field (component).

DECLARE

CURSOR c IS

SELECT \* FROM employees

WHERE department\_ID = 20;

v\_employees employees%ROWTYPE; -- row variable for table

BEGIN

OPEN c;

LOOP -- Fetches entire row into a row type variable

FETCH c INTO v\_employees;

EXIT WHEN c%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(v\_employees.last\_name || ' : ' ||

v\_employees.salary ); -- using dot notation for column

-- using RPAD(v\_employees.last\_name, 12, ' ') to get better output

END LOOP;

CLOSE c;

END;

OUTPUT

Hartstein: 13000

Fay: 6000

- - - -- using RPAD(v\_employees.last\_name, 12)

Hartstein :13000

Fay :6000

1. **Cursor Based PL/SQL records,** cursor\_name%ROWTYPE

PL/SQL record is composite data structure, it can be table-based, cursor-based and user defined (we will discuss that later in class note #6).

The %ROWTYPE attribute lets you declare a record that represents either a full or partial

row of a database table or view (record anchoring).

For every visible column of the full or partial row, the record has a field with the same

name and data type. If the structure of the row changes, then the structure of the

record changes accordingly.

The record fields do not inherit the constraints or initial values of the corresponding

columns.

variable\_name1 table\_or\_view\_name%ROWTYPE

variable\_name2 cursor\_name%ROWTYPE

(cursor's definition is similar to a view, just a select statement)

**A cursor-based record type variable** can make the fetch-cursor-into clause simple.

record\_var\_name cursor\_name%ROWTYPE

A PL/SQL record whose structure is based on the list of a cursor is called a cursor-based record.

**Example 5.** - using cursor\_name%ROWTYPE variable**,** (anchored declarations)

DECLARE

CURSOR c\_1 IS

SELECT d.department\_name, sum(e.salary) AS tot\_salary

FROM departments d, employees e

WHERE d.department\_id = e.department\_id

GROUP BY department\_name;

v\_sal\_rec c\_1%ROWTYPE;

BEGIN

OPEN c\_1;

LOOP

FETCH c\_1 INTO v\_sal\_rec;

EXIT WHEN c\_1%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE (rpad(v\_sal\_rec.department\_name, 17) ||

TO\_CHAR (v\_sal\_rec.tot\_salary, '$9,999,999'));

END LOOP;

CLOSE c\_1;

END;

OUTPUT:

Administration $4,400

Accounting $20,300

Purchasing $24,900

...

**Example 6.** Similar to example #3**,** using %ROWTYPE here

DECLARE

CURSOR cemp IS

SELECT ename, sal

FROM emp

WHERE deptno = 10

ORDER BY sal DESC;

v\_cemp cemp%ROWTYPE;

BEGIN

OPEN cemp;

LOOP

FETCH cemp INTO v\_cemp; -- INOT a single composite record variable

EXIT WHEN cemp%NOTFOUND;

DBMS\_OUTPUT.put\_line (

'current row number is '|| cemp%ROWCOUNT ||

': ' || RPAD (v\_cemp.ename, 10) || ': ' ||

TO\_CHAR (v\_cemp.sal, '$999,999'));

END LOOP;

-- using dot notation to access the field of record variable

CLOSE cemp; -- It gets the same result as example #3.

END;

/

1. **Cursor for LOOP**

The cursor FOR LOOP statement lets you run a SELECT statement and then immediately loop through the rows of the result set. This coding provides the same role as the loop – fetch coding when the program processes the cursor active set through first to the last record.

After the declare/defining cursor, the loop-fetch program usually consists of these steps:

OPEN cursor\_name;

LOOP (repeat for each record in the cursor active set)

FETCH cursor INTO list of variables, or record type variable.

EXIT if cursor\_name$NOTFOUND -- end of the active set of the select statement

Your codes for processing the data (stored in variables) from each record,

END LOOP;

CLOSE cursor\_name;.

Syntax of “cursor for loop”:

FOR record\_index IN cursor\_name LOOP

-- record\_index is dummy index, no need to declare

-- access the column: record\_index.column\_name

-- loop body

END LOOP;

The syntax For-cursor will take care of the commands related to the cursor:

open, close, Fetch, Exit when cursor\_name%NOTFOUND.

System automatically OPENs and CLOSEs Cursor for the program.

If you write any command (such as “Open cursor\_name”, “Fetch”) mentioned above again while you use cursor-for-loop, you will get error message.

(Review: for *index*  in [reverse] *lower\_bound* ***..*** *upper\_bound* loop

There, user cannot manually manage the dummy index)

Here, record\_index is a dummy index, you do not need to declare it. The scope of the record\_index is this LOOP only. It is useful for the situation where all rows of the cursor are to be processed. (If the program does not need to process all the rows, an extra counter is needed.)

**Example 7**. Change example #3 and #6, using cursor for loop instead

With the “For index in cursor loop”, system will automatically take care these for you:

it uses “indx” as dummy index, and

the loop starts from the first row to the last row in active set of the cursor,

loop automatically ends as “EXIT when cemp$NOTFOUND”,

it automatically executes “CLOSE cemp” for you.

DECLARE

CURSOR cemp IS

SELECT ename, sal

FROM emp

WHERE deptno = 10

ORDER BY sal DESC;

BEGIN

For indx in cemp loop

DBMS\_OUTPUT.put\_line ( 'current row number is '||

cemp%ROWCOUNT || ': ' || RPAD (indx.ename, 10) || ': ' ||

TO\_CHAR (indx.sal, '$999,999'));

End loop;

END;

OUTPUT is same as example #3.

**Example 8a. Explicit cursor (the cursor is declared)**

Declare

CURSOR c IS

SELECT ename, sal

FROM emp

WHERE job = 'CLERK';

BEGIN

FOR indx IN C LOOP

DBMS\_OUTPUT.put\_line ( c%ROWCOUNT || '. Salary of ' ||

indx.ename ||': ' || TO\_CHAR (indx.sal, '$999,999'));

END LOOP;

END;

/

OUTPUT

1. Salary of SMITH: $800

2. Salary of ADAMS: $1,100

3. Salary of JAMES: $950

4. Salary of MILLER: $1,300

**Example 8b** -- using **Implicit Cursor** FOR LOOP

-- (select-statement replaces the Declaration of cursor c)

BEGIN

FOR indx IN (

SELECT ename, sal

FROM emp

WHERE job = 'CLERK')

LOOP -- of course we cannot use cursor attribute here)

DBMS\_OUTPUT.put\_line ( 'Salary of ' || indx.ename ||

': ' || TO\_CHAR (indx.sal, '$999,999'));

END LOOP;

END;

/

1. **Cursor that accept parameters (parameterized cursor)**

You can create an explicit cursor that has formal parameters, and then pass different

actual parameters to the cursor each time you open it.

In the cursor query, you can use a formal cursor parameter anywhere that you can use a constant.

Outside the cursor query, you cannot reference formal cursor parameters.

Pass parameter values to a cursor when the cursor is opened and the query is executed.

Open an explicit cursor several times with a different active set each time.

You cannot pass values out of the cursor through parameters.

Only the data type of the parameter is defined, not its length.

# Many concepts here are related or same as procedures/functions, which we will discuss later, such as parameter mode (IN, OUT), formal and actual parameter, no length constrain for formal parameter, etc. (note7b\_functions)

# Definition:

# CURSOR cursor\_name (parameter\_list) -- default mode of parameter is IN.

# [ return return\_spec ] IS select-statement;

**Example 9a -** Open an explicit cursor two times with passing a different value of the

parameter for each time.

DECLARE

CURSOR c\_emp ( p\_deptno number) IS

SELECT empno, ename, sal

FROM emp

WHERE deptno = p\_deptno ;

v\_empno emp.empno%TYPE;

v\_name emp.ename%TYPE;

v\_sal emp.sal%TYPE;

BEGIN

OPEN c\_emp(20); -- pass value to parameter

DBMS\_OUTPUT.PUT\_LINE('OUTPUT for deptno as 20: --- ');

LOOP

FETCH c\_emp INTO v\_empno, v\_name, v\_sal;

EXIT WHEN c\_emp%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('employee id ' || v\_empno ||', name is '

|| RPAD(v\_name, 10)||'; salary is ' || TO\_CHAR (v\_sal, '$99,999'));

END LOOP;

CLOSE c\_emp;

DBMS\_OUTPUT.PUT\_LINE('--------------');

DBMS\_OUTPUT.PUT\_LINE(' ');

-- repeat the code as above with new value of the parameter from 20 to 10

OPEN c\_emp(10); -- pass value to parameter

DBMS\_OUTPUT.PUT\_LINE('OUTPUT for deptno as 10: --- ');

LOOP

FETCH c\_emp INTO v\_empno, v\_name, v\_sal;

EXIT WHEN c\_emp%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('employee id ' || v\_empno ||', name is '

|| RPAD(v\_name, 10)||'; salary is ' || TO\_CHAR (v\_sal, '$99,999'));

END LOOP;

CLOSE c\_emp;

END;

OUTPUT for deptno as 20: ---

employee id 7369, name is SMITH ; salary is $800

employee id 7566, name is JONES ; salary is $2,975

employee id 7788, name is SCOTT ; salary is $3,000

employee id 7876, name is ADAMS ; salary is $1,100

employee id 7902, name is FORD ; salary is $3,000

--------------

OUTPUT for deptno as 10: ---

employee id 7782, name is CLARK ; salary is $2,450

employee id 7839, name is KING ; salary is $5,000

employee id 7934, name is MILLER ; salary is $1,300

**Example 9b.** same sample as 9a, now using nested loop, just for reference.

DECLARE

CURSOR c\_emp ( p\_deptno number) IS

SELECT empno, ename, sal

FROM emp

WHERE deptno = p\_deptno ;

v\_empno emp.empno%TYPE;

v\_name emp.ename%TYPE;

v\_sal emp.sal%TYPE;

**x integer := 10** ;-- **this is new, counter for outer loop**

-- integer is subtype of number,

BEGIN

LOOP

EXIT WHEN x > 21;

OPEN c\_emp(x); -- pass value to parameter

DBMS\_OUTPUT.PUT\_LINE('OUTPUT for deptno as: '||x || ' --- ');

LOOP

FETCH c\_emp INTO v\_empno, v\_name, v\_sal;

EXIT WHEN c\_emp%NOTFOUND ;

DBMS\_OUTPUT.PUT\_LINE('employee id ' || v\_empno ||

', name is '|| RPAD(v\_name, 10) || '; salary is ' ||

TO\_CHAR (v\_sal, '$99,999'));

END LOOP; -- end of inner loop

DBMS\_OUTPUT.PUT\_LINE('--------------');

DBMS\_OUTPUT.PUT\_LINE(' ');

x := x+10;

CLOSE c\_emp;

END LOOP;

END;

**Example 10a -** normal cursor without parameter

DECLARE

CURSOR c1 IS

SELECT \* FROM emp

WHERE job = 'CLERK' AND sal > 1000; -- Using literal values

BEGIN

FOR person IN c1

LOOP

DBMS\_OUTPUT.PUT\_LINE ( 'Name = ' || person.ename || ', salary = ' || person.sal || ', Job = ' || person.job );

END LOOP;

END;

/

OUTPUT

Name = ADAMS, salary = 1100, Job = CLERK

Name = MILLER, salary = 1300, Job = CLERK

**Example 10b**

-- Same as 10a, now passing parameters to Explicit Cursor FOR LOOP Statement

DECLARE

CURSOR c1 (job\_para VARCHAR2, max\_wage NUMBER) IS

-- Define (IN) parameters

-- or (job\_para IN emp.job%TYPE, max\_wage IN emp.sal%TYPE) IS

SELECT \* FROM emp

WHERE job = job\_para AND sal > max\_wage;

BEGIN

FOR person IN c1 ('CLERK', 1000) -- Passing the values to parameters

LOOP

DBMS\_OUTPUT.PUT\_LINE (

'Name = ' || person.ename || ', salary = ' ||

person.sal || ', Job = ' || person.job);

END LOOP;

END;

**Example 11 -** initial value defined for parameter will act as default value

( you may skip this sample now, and revisit it later after finishing Note7b\_Functions)

DECLARE

CURSOR c\_emp (p\_salary NUMBER := 15000) IS

-- define a parameter with a default value

SELECT Last\_name, salary

FROM employees

WHERE salary >= p\_salary

ORDER BY salary DESC;

BEGIN

FOR idx IN c\_emp Loop

-- same as FOR idx c\_emp (15000) loop

DBMS\_OUTPUT.PUT\_LINE (RPAD(idx.Last\_name, 15, ' ')||

' ' || TO\_CHAR (idx.salary, '$99,999'));

END LOOP;

END;

OUTPUT

King $24,000

De Haan $17,000

Kochhar $17,000

1. **Cursor for Update**

In SQL, SELECT FOR UPDATE statement will ensure that no other user can change those values before you update them. When the cursor is opened, the system automatically obtains row-level locks for all the rows identified by the select statement, holding the records for your changes only. No other users can change these records until you perform a commit or rollback (other user can read these records).

If an explicit cursor is associated with SELECT FOR UPDATE, the cursor is called a FOR UPDATE cursor.

Declare

CURSOR cursor\_name IS

select\_statement -- usually, only one table is involved

FOR UPDATE [OF column\_name]; -- column\_name in the select\_list

When SELECT FOR UPDATE queries multiple tables, it locks only rows whose columns

appear in the FOR UPDATE OF Cloumn\_name clause.

Once the cursor is declared in as for update as above, the following statement can be used to perform a DELETE of a row or modification to the row:

**The WHERE CURRENT OF (cursor\_name) Clause**

(the last, or the most recently fetched row)

UPDATE table\_name

SET clause

**WHERE current of** cursor\_name;

-- “WHERE condition” equals to “the row that the cursor now pointing to”.

or

DELETE table\_name

**WHERE current of** cursor\_name;

**Example 12** (Similar as example 3 on page 4 in this note)

This program will give those employees a raise of salary if their current salary is less than 1000. Run “SELECT \* FROM emp where sal < 1100” to know the table contents.

“Rollback” after the running of the PL/SQL block to recover the original contents of emp.

DECLARE

CURSOR cemp IS

SELECT ename, sal

FROM emp

ORDER BY sal DESC

FOR UPDATE OF sal; -- With FOR UPDATE OF request

salary emp.sal%TYPE;

Emp\_name emp.ename%TYPE;

BEGIN

OPEN cemp;

LOOP

FETCH cemp INTO Emp\_name, salary;

EXIT WHEN cemp%NOTFOUND;

IF SALARY < 1000 THEN

salary := salary \* 1.2;

UPDATE emp

SET sal = salary

**WHERE current of** cemp; -- This is new

DBMS\_OUTPUT.put\_line ('current row number ' || cemp%ROWCOUNT||

': '||emp\_name || ': ' || TO\_CHAR (salary, '$999,999'));

END IF;

END LOOP;

CLOSE cemp;

-- Commit;

END;

OUTPUT

current row number 13: JAMES: $1,140

current row number 14: SMITH: $960

1. **Cursor Variables.** Manual 6.4 (6-30)

An explicit or implicit cursor is the name for the work area for a query result (set). That is static for that special select-statement.

Cursor variable is a variable that points to underlining cursor, and it can be opened for any query. It provides a mechanism for passing results of query between different PL/SQL programs, especially useful for client server programming environment.

7.1**)** Creating Cursor Variable

To create a cursor variable, either declare a variable of the predefined type SYS\_REFCURSOR or define a REF CURSOR type and then declare a variable of that type.

Two steps of declaration:

Declaring REF CURSOR types:

**TYPE** type\_name **IS REF CURSOR** [RETURN return\_type];

Declaring Cursor Variables:

cursor\_var\_name type\_name;

If the cursor variable is declared with the return specification, then it has to match the select list. It is called **STRONG** type.

TYPE empcurtyp IS REF CURSOR RETURN employees%ROWTYPE; -- strong type

If the cursor variable is declared without the return type, the variable can be associated with any kind cursor (or select statement), called **WEAK** type.

TYPE genericcurtyp IS REF CURSOR; -- weak type

Oracle provides a pre-defined weak REF CURSOR type named **SYS\_REFCURSOR**. You no longer need to declare your own weak type. Just use Oracle’s.

Declare my\_cursor\_var SYS\_REFCURSOR;

my\_cursor SYS\_REFCURSOR; -- weak cursor variable

7.2) Opening Cursor variables.

The syntax is as below:

**OPEN** cursor\_var\_name **FOR** select\_statement;

It is not limited to one query.

You can open a cursor variable for a query, process the result set, and then use the

cursor variable for another query.

A cursor variable has this flexibility because it is a pointer; that is, its value is the

address of an item, not the item itself.

DECLARE

TYPE emp\_curtype IS REF CURSOR RETURN emp%ROWTYPE;

Emp\_curvar emp\_curtype;

counter integer := 1;

BEGIN

If counter = 1 then

OPEN emp\_curvar for

SELECT \* FROM emp where deptno = 10;

ELSE

OPEN emp\_curvar for

SELECT \* FROM emp where deptno = 20;

END if;

END;

/

Sample to create and open cursor variable of Weak type,

DECLARE

TYPE emp\_curtype IS REF CURSOR;

Emp\_curvar emp\_curtype;

BEGIN

OPEN emp\_curvar for SELECT \* FROM dept;

OPEN emp\_curvar for SELECT \* FROM employee;

OPEN emp\_curvar for SELECT empno, ename FROM emp;

END;

While the declaration and open statement for cursor variable are enhanced, some cursor operations are same for cursor variable and static cursor:

Close statement, Cursor attributes, Fetch from the cursor variable,

|  |  |  |
| --- | --- | --- |
|  | Cursor | Cursor Variable (Ref Cursor) |
| definition | is static, to one select-statement | Dynamic, can change the select statement later. |
| Declare | Cursor *curname* IS select statement | Type Ref\_Cur\_Name IS **REF CURSOR** [return datatype], then define:  Cur\_variable\_name Ref\_Cur\_Name |
| Open | Open *cursor\_name* | Open Cur\_variable\_name **FOR** select statement |
| Fetch | Fetch *curname* INTO variable | Same as cursor |
| Close | Close *curnam*e | Same as cursor |
| For loop | For index IN *curname*  loop … end loop; | No  (There is no cursor\_variable for loop) |
| Scope |  | Same as cursor |
|  | You cannot do these (in right side) as for cursor variable | You can assign a value;  You can use it in an expression  Use it as a formal subprogram parameter |
|  | It can accept parameter | You cannot pass parameter to It,  But you can pass whole queries to it. |

7.3) Fetch Cursor\_variable into

It is basically same as for cursor operations.

The return type of cursor variable must be compatible with the into\_clause of Fetch

statement.

If the cursor variable is strong, PL/SQL catches incompatibility at compile time,

If the cursor variable is weak, PL/SQL catches incompatibility at run time,

Raising the predefined exception ROWTYPE\_MISMATCH before the first fetch.

**Example 13** (compare Example 4 on page 6 in this note)

DECLARE

cv SYS\_REFCURSOR; -- cursor variable

v\_lastname employees.last\_name%TYPE; -- variable for last\_name

v\_jobid employees.job\_id%TYPE; -- variable for job\_id

query\_2 VARCHAR2(200) :=

'SELECT \* FROM employees WHERE SALARY > 10000';

v\_employees employees%ROWTYPE; -- record variable, row of table

BEGIN

OPEN cv FOR -- First time Open cursor variable

SELECT last\_name, job\_id FROM employees

WHERE job\_id like 'S%CLERK'

ORDER BY last\_name;

LOOP -- Fetches columns into variables

FETCH cv INTO v\_lastname, v\_jobid;

EXIT WHEN cv%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE( RPAD(v\_lastname, 25, ' ') || v\_jobid );

END LOOP;

DBMS\_OUTPUT.PUT\_LINE( '-------------------------------------' );

OPEN cv FOR query\_2; -- no need to close, second time Open cursor variable cv

LOOP -- This time, fetches entire row into the v\_employees record

FETCH cv INTO v\_employees;

EXIT WHEN cv%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE( RPAD(v\_employees.last\_name, 25, ' ') ||

v\_employees.job\_id );

END LOOP;

CLOSE cv;

END;

OUTPUT

Atkinson ST\_CLERK

Bell SH\_CLERK

...

Vargas ST\_CLERK

Walsh SH\_CLERK

-------------------------------------

King AD\_PRES

Kochhar AD\_VP

...

Higgins AC\_MGR

7.4)Cursor Variable with User-Defined Return Type

In this example, *EmpRecTyp* is a user-defined RECORD type.

DECLARE

**TYPE EmpRecTyp IS RECORD** (

employee\_id NUMBER,

last\_name VARCHAR2(25),

salary NUMBER(8,2));

TYPE EmpCurTyp IS REF CURSOR **RETURN EmpRecTyp**;

emp\_cv EmpCurTyp;

BEGIN

NULL;

END;

/

Using REF CURSOR is one of the most powerful, flexible, and scalable ways to return query results from an Oracle Database to a client application.

A REF CURSOR is PL/SQL data type whose value is the memory address of a query work area on the database. A REF CURSOR refers to a memory address in the database.

You want to use static SQL (with implicit cursors really) first and use a ref cursor only when you absolutely have to.

The diagram depicts the cursor variable referring to a cursor now, and it can be re-assigned to another cursor object later if needed. A cursor (object) is static, pointing a set of returned rows.

