

TEMĂ

În această temă voi prezenta *WHERE CURRENT OF* (cursor explicit) vs *FORALL* (cursor implicit definit de Oracle) pe exemplul 5.12 din Curs, exemplul 5.12 (*WHERE CURRENT OF*) vs 5.13 (*WHERE ROWID = i.ROWID*), cursoare *imbricate* 5.16 vs cursoare *parametrizate* 5.10. De asemenea voi prezenta și costul pentru fiecare dintre cele 3 instrucțiuni SQL din exemplul 5.14

Exemplul 5.12 (WHERE CURRENT OF vs FORALL cu BULK COLLECT INTO) din Curs.

Pentru a vedea diferențele între cele două vom folosi funcția **sysimestamp** cu care vom vedea timpul la care a început/sfârșit fiecare instrucțiune, *WHERE CURRENT OF NUME_CURSOR*, respective *FORALL*.

Rezolvare:

```
DECLARE
```

```
    CURSOR c IS
```

```
    SELECT employee_id
```

```
    FROM employees
```

```
    WHERE department_id IN
```

```
        (SELECT department_id
```

```
        FROM departments
```

```
        WHERE department_name = 'IT')
```

```
    FOR UPDATE OF salary NOWAIT;
```

```
    TYPE tab_ind IS TABLE OF employees%ROWTYPE INDEX BY PLS_INTEGER;
```

```
    t tab_ind;
```

```
BEGIN
```

```
    dbms_output.put_line('Inainte de WHERE CURRENT OF: ' || sysimestamp);
```

```
    FOR i IN c LOOP
```

```
        UPDATE employees
```

```
        SET salary = (salary*0.5)*2 + 199*0.95
```

```
        WHERE CURRENT OF c;
```

```
    END LOOP;
```

```
dbms_output.put_line('Dupa de WHERE CURRENT OF: ' || systimestamp);  
ROLLBACK;
```

```
SELECT * BULK COLLECT INTO t  
FROM employees  
WHERE department_id IN  
    (SELECT department_id  
    FROM departments  
    WHERE department_name = 'IT');
```

```
dbms_output.put_line('Inainte de FORALL: ' || systimestamp);  
FORALL i in 1..t.LAST  
    UPDATE employees  
    SET salary = (salary*0.5)*2 + 199*0.95  
    WHERE t(i).employee_id = employee_id;  
dbms_output.put_line('Dupa de FORALL: ' || systimestamp);  
COMMIT;
```

```
END;
```

```
/
```

Print-Screen:

```
1 DECLARE
2   CURSOR c IS
3     SELECT employee_id
4     FROM employees
5     WHERE department_id IN
6       (SELECT department_id
7        FROM departments
8        WHERE department_name = 'IT')
9     FOR UPDATE OF salary NOWAIT;
10
11 TYPE tab_ind IS TABLE OF employees%ROWTYPE INDEX BY PLS_INTEGER;
12 t tab_ind;
13 BEGIN
14   dbms_output.put_line('Inainte de WHERE CURRENT OF: ' || systimestamp);
15   FOR i IN c LOOP
16     UPDATE employees
17     SET salary = (salary*0.5)*2 + 199*0.95
18     WHERE CURRENT OF c;
19   END LOOP;
20   dbms_output.put_line('Dupa de WHERE CURRENT OF: ' || systimestamp);
21   ROLLBACK;
22
23   SELECT * BULK COLLECT INTO t
24   FROM employees
25   WHERE department_id IN
26     (SELECT department_id
27     FROM departments
28     WHERE department_name = 'IT');
29   dbms_output.put_line('Inainte de FORALL: ' || systimestamp);
30   FORALL i IN 1..t.LAST
31     UPDATE employees
```

Output Window:

```
Inainte de WHERE CURRENT OF: 26-NOV-21 06.46.05.944230000 AM +02:00
Dupa de WHERE CURRENT OF: 26-NOV-21 06.46.05.945821000 AM +02:00
Inainte de FORALL: 26-NOV-21 06.46.05.946540000 AM +02:00
Dupa de FORALL: 26-NOV-21 06.46.05.946897000 AM +02:00
```

Script Output:

```
PL/SQL procedure successfully completed.
```

Putem observa că blocul nostru PL/SQL s-a executat cu succes. În dreapta vedem pe prima linie, respectiv a doua, ora la care a început/sfârșit monitorizarea pentru loop-ul cu *WHERE CURRENT OF NUME_CURSOR*, respectiv pentru *FORALL*.

Diferențele sunt infime! Ele se pot observa la ordinul milisecundelor. Pentru a vedea cu exactitate vom face un diferență între timpul de început și timpul de sfârșit folosind calculatorul.

Oracle SQL Developer interface showing a PL/SQL procedure execution. The procedure uses a cursor loop with `WHERE CURRENT OF` to update salaries. The execution output shows the duration of the loop as 1,591,000 milliseconds.

```
1 DECLARE
2 CURSOR c IS
3 SELECT employee_id
4 FROM employees
5 WHERE department_id IN
6 (SELECT department_id
7 FROM departments
8 WHERE department_name = 'IT')
9 FOR UPDATE OF salary NOWAIT;
10
11 TYPE tab_ind IS TABLE OF employees%ROWTYPE INDEX BY PLS_INTEGER;
12 t tab_ind;
13 BEGIN
14 dms_output.put_line('Inainte de WHERE CURRENT OF: ' || systimestamp);
15 FOR i IN c LOOP
16 UPDATE employees
17 SET salary = (salary*0.5)*2 + 199*0.95
18 WHERE CURRENT OF c;
19 END LOOP;
20 dms_output.put_line('Dupa de WHERE CURRENT OF: ' || systimestamp);
21 ROLLBACK;
22
23 SELECT * BULK COLLECT INTO t
24 FROM employees
25 WHERE department_id IN
26 (SELECT department_id
27 FROM departments
28 WHERE department_name = 'IT');
29 dms_output.put_line('Inainte de FORALL: ' || systimestamp);
30 FORALL i in 1..t.LAST
31 UPDATE employees
```

Execution Output:

```
Inainte de WHERE CURRENT OF: 26-NOV-21 06.46.05.944230000 AM +02:00
Dupa de WHERE CURRENT OF: 26-NOV-21 06.46.05.945821000 AM +02:00
Inainte de FORALL: 26-NOV-21 06.46.05.946540000 AM +02:00
Dupa de FORALL: 26-NOV-21 06.46.05.946897000 AM +02:00
```

Calculator showing the calculation: $945821000 - 944230000 = 1,591,000$

Observăm că a durat **1.591.000** unități (ordinul milisecundelor), loop-ul pentru cursorul explicit cu `WHERE CURRENT OF`. Acum vom face calculul pentru loop-ul `FORALL`.

Oracle SQL Developer interface showing the same PL/SQL procedure execution, but with the `FORALL` loop. The execution output shows the duration of the loop as 357,000 milliseconds.

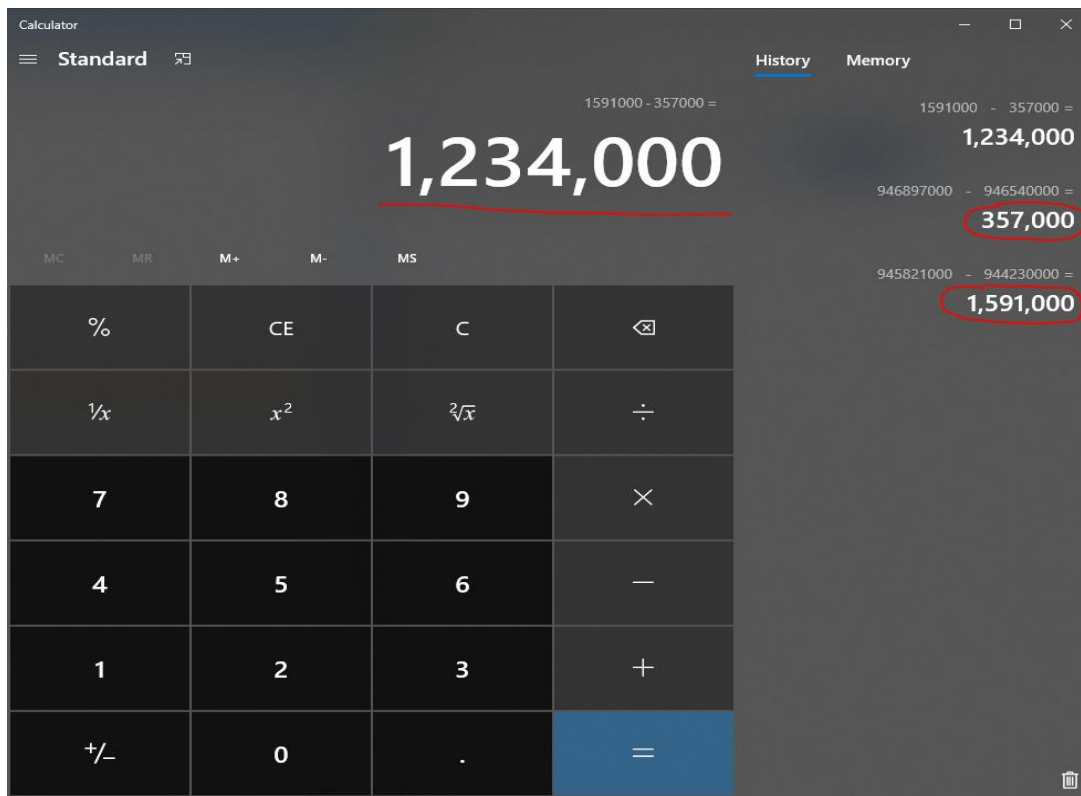
```
1 DECLARE
2 CURSOR c IS
3 SELECT employee_id
4 FROM employees
5 WHERE department_id IN
6 (SELECT department_id
7 FROM departments
8 WHERE department_name = 'IT')
9 FOR UPDATE OF salary NOWAIT;
10
11 TYPE tab_ind IS TABLE OF employees%ROWTYPE INDEX BY PLS_INTEGER;
12 t tab_ind;
13 BEGIN
14 dms_output.put_line('Inainte de WHERE CURRENT OF: ' || systimestamp);
15 FOR i IN c LOOP
16 UPDATE employees
17 SET salary = (salary*0.5)*2 + 199*0.95
18 WHERE CURRENT OF c;
19 END LOOP;
20 dms_output.put_line('Dupa de WHERE CURRENT OF: ' || systimestamp);
21 ROLLBACK;
22
23 SELECT * BULK COLLECT INTO t
24 FROM employees
25 WHERE department_id IN
26 (SELECT department_id
27 FROM departments
28 WHERE department_name = 'IT');
29 dms_output.put_line('Inainte de FORALL: ' || systimestamp);
30 FORALL i in 1..t.LAST
31 UPDATE employees
```

Execution Output:

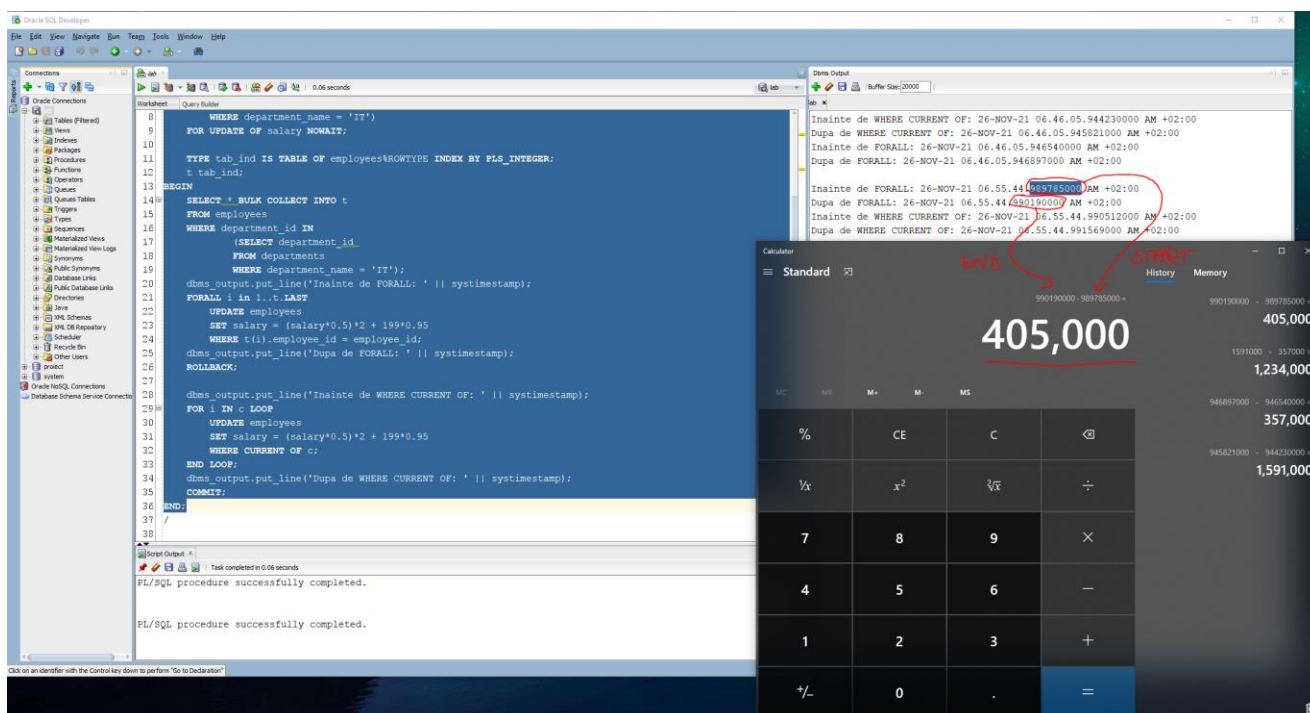
```
Inainte de WHERE CURRENT OF: 26-NOV-21 06.46.05.944230000 AM +02:00
Dupa de WHERE CURRENT OF: 26-NOV-21 06.46.05.945821000 AM +02:00
Inainte de FORALL: 26-NOV-21 06.46.05.946540000 AM +02:00
Dupa de FORALL: 26-NOV-21 06.46.05.946897000 AM +02:00
```

Calculator showing the calculation: $946897000 - 946540000 = 357,000$

Observăm că loop-ul cu FORALL a durat **357.000** unități (ordinul milisecundelor). Deci s-a executat cu **1.234.000** de unități mai repede decât loop-ul cursorului cu *WHERE CURRENT OF*!



Dacă testăm mai întâi pentru *FORALL* și după aceea pentru *WHERE CURRENT OF*, vom observa că indiferent de ordinea execuției *FORALL* se va executa mult mai repede decât *WHERE CURRENT OF*.



Avem **405.000** unități pentru *FORALL*.

The screenshot shows the Oracle SQL Developer interface. The main window displays a PL/SQL procedure named `FORUPDATE` that updates salaries for employees in department 'IT'. The procedure uses a `FORALL` loop to update salaries in bulk. The execution output window shows the results of the procedure, including the number of rows updated and the total number of rows affected. The output indicates that 405,000 rows were updated.

```
WHERE department_name = 'IT')
FOR UPDATE OF salary NOWAIT:

TYPE tab_ind IS TABLE OF employees%ROWTYPE INDEX BY PLS_INTEGER;
t tab_ind;

BEGIN
  SELECT * BULK COLLECT INTO t
  FROM employees
  WHERE department_id IN
  (SELECT department_id
   FROM departments
   WHERE department_name = 'IT');
  dbms_output.put_line('Inainte de FORALL: ' || systimestamp);
  FORALL i in 1..t.LAST
  UPDATE employees
  SET salary = (salary*0.5)*2 + 199*0.95
  WHERE t(i).employee_id = employee_id;
  dbms_output.put_line('Dupa de FORALL: ' || systimestamp);
  ROLLBACK;
  dbms_output.put_line('Inainte de WHERE CURRENT OF: ' || systimestamp);
  FOR i IN < LOOP
  UPDATE employees
  SET salary = (salary*0.5)*2 + 199*0.95
  WHERE CURRENT OF c;
  END LOOP;
  dbms_output.put_line('Dupa de WHERE CURRENT OF: ' || systimestamp);
  COMMIT;
END;
```

Execution Output:

```
Inainte de WHERE CURRENT OF: 26-NOV-21 06.46.05.944230000 AM +02:00
Dupa de WHERE CURRENT OF: 26-NOV-21 06.46.05.945821000 AM +02:00
Inainte de FORALL: 26-NOV-21 06.46.05.946540000 AM +02:00
Dupa de FORALL: 26-NOV-21 06.46.05.946897000 AM +02:00

Inainte de FORALL: 26-NOV-21 06.55.44.989785000 AM +02:00
Dupa de FORALL: 26-NOV-21 06.55.44.990190000 AM +02:00
Inainte de WHERE CURRENT OF: 26-NOV-21 06.55.44.990512000 AM +02:00
Dupa de WHERE CURRENT OF: 26-NOV-21 06.55.44.991569000 AM +02:00
```

Calculator results:

MC	MR	M+	M-	MS
%	CE	C	<	
1/x	x ²	√x	÷	
7	8	9	×	
4	5	6	-	
1	2	3	+	
+/-	0	.	=	

Avem **1.057.000** unități pentru cursorul explicit, *WHERE CURRENT OF*. Deci din nou avem o diferență observabilă între cele două la nivelul microsecundelor de **652.000** unități.

The screenshot shows a calculator window with the result of the subtraction $1,057,000 - 405,000 = 652,000$. The calculator also displays a history of previous calculations.

Calculator results:

MC	MR	M+	M-	MS
%	CE	C	<	
1/x	x ²	√x	÷	
7	8	9	×	
4	5	6	-	
1	2	3	+	
+/-	0	.	=	

History:

- $1057000 - 405000 = 652,000$
- $1057000 - 990512000 = 1,057,000$
- $991569000 - 990512000 = 405,000$
- $990190000 - 989785000 = 1,234,000$
- $1591000 - 357000 = 357,000$
- $946897000 - 946540000 = 1,591,000$
- $945821000 - 944230000 = 1,591,000$

De aici putem trage concluzia că, *BULK COLLECT INTO* cu *FORALL* (cursor implicit) este mult mai rapid decât *cursoarele explicite cu loop FOR*! Un singur dezavantaj pentru *FORALL* ar fi acela că nu putem folosi decât o instrucțiune DML, nu avem clauza ***FORALL END***;

Exemplul 5.12 (WHERE CURRENT OF) vs Exemplul 5.13 (WHERE ROWID = i.ROWID) din Curs.

Update-ul din exemplul 5.12 este echivalent cu cel din exemplul 5.13 (ambele funcționează). Eu consider că, cel din exemplul 5.12 (WHERE CURRENT OF) este optim în comparație cu cel din exemplul 5.13 (ROWID = i.ROWID) deoarece în cursor facem select doar pe o singură coloană, spre deosebire de exemplul 5.13, unde selectăm și id-ul produsului, dar și row id-ul, acest lucru aducând un consum mai mare de memorie la procesare (ocupă mai multă memorie, ceea ce nu ne dorim, știm acest lucru de la Workshop). De asemenea putem spune că, folosind ROWID (exemplul 5.13) nu avem nicio “garanție” că, până vom încheia noi tranzacția curentă, rowid-ul nu se va modifica. Acest lucru se poate întâmpla spre exemplu atunci când facem operații ALTER TABLE... SHRINK SPACE etc. (operații în masă sau DML-uri foarte mari).

Exemplul 5.14, costul pentru fiecare dintre cele 3 instrucțiuni SQL.

Rezolvare:

-- VARIANTA 1

SELECT *

FROM employees e

WHERE EXISTS (SELECT 1

FROM departments d, job_history j

WHERE e.employee_id = j.employee_id

AND d.department_id = j.department_id

AND TO_CHAR(end_date,'q') = 1);

-- VARIANTA 2

SELECT *

FROM employees e

WHERE employee_id IN (SELECT employee_id

FROM departments d, job_history j

WHERE e.employee_id = j.employee_id

AND TO_CHAR(end_date,'q') = 1);

-- VARIANTA 3

```
SELECT DISTINCT e.*
FROM employees e, departments d, job_history j
WHERE e.employee_id = j.employee_id
AND d.department_id = j.department_id
AND TO_CHAR(end_date,'q') = 1;
```

Print-Screen:

The screenshot displays the Oracle SQL Developer interface. The top pane shows a query labeled 'VARIANTA 1' which is a SELECT DISTINCT query joining employees, departments, and job_history tables. The bottom pane shows the execution plan for this query. The plan is a NESTED LOOPS join, with a SORT operation preceding it. The inner loop accesses the JOB_HISTORY table via a TABLE ACCESS operation, which is filtered by a predicate. The outer loop accesses the EMPLOYEES table via a TABLE ACCESS operation, which is filtered by a predicate. The cost of the entire query is 5, which is circled in red. The plan also includes an index scan for the EMP_EMP_ID_PK index.

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				5
NESTED LOOPS				5
SORT				3
TABLE ACCESS	JOB_HISTORY	UNIQUE		3
Filter Predicates		FULL		3
AND				
J.DEPARTMENT_ID IS NOT NULL				
TO_NUMBER(TO_CHAR(INTERNAL_FUNCTION(END_DATE),'q'))=1				
INDEX	EMP_EMP_ID_PK	UNIQUE SCAN		0
Access Predicates				
J.EMPLOYEE_ID=E.EMPLOYEE_ID				
TABLE ACCESS	EMPLOYEES	BY INDEX ROWID		1

Pentru prima variantă avem costul 5.

lab 0.044 seconds

```

1 -- VARIANTA 1
2 SELECT *
3 FROM employees e
4 WHERE EXISTS (SELECT 1
5               FROM departments d, job_history j
6               WHERE e.employee_id = j.employee_id
7                     AND d.department_id = j.department_id
8                     AND TO_CHAR(end_date,'q') = 1);
9
10 -- VARIANTA 2
11 SELECT *
12 FROM employees e
13 WHERE employee_id IN (SELECT employee_id
14                       FROM departments d, job_history j
15                       WHERE e.employee_id = j.employee_id
16                             AND TO_CHAR(end_date,'q') = 1);
17

```

Query Result x Explain Plan x

SQL 0.044 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	
SELECT STATEMENT				1	7
NESTED LOOPS				1	7
NESTED LOOPS				1	7
HASH	SYS_WW_SQL			27	5
MERGE JOIN		UNIQUE		1	5
TABLE ACCESS	JOB_HISTORY	CARTESIAN		27	5
Filter Predicates		FULL		1	3
TO_NUMBER(TO_CHAR(INTERNAL_FUNCTION(END_DATE),'q'))=1					
BUFFER				27	2
INDEX	DEPT_ID_PK	FAST FULL SCAN		27	2
Access Predicates	EMP_EMP_ID_PK	UNIQUE SCAN		1	0
Filter Predicates					
EMPLOYEE_ID=ITEM_1					
Filter Predicates					
EMPLOYEE_ID=EMPLOYEE_ID					
TABLE ACCESS	EMPLOYEES	BY INDEX ROWID		1	1

Other XML

(info)

- info type="db_version"
 - 11.1.0.6
- info type="parse_schema"
 - "GRPUPA231"
- info type="dynamic_sampling"
 - yes
- info type="plan_hash"
 - 981048607

Pentru a doua variantă avem costul 7.

lab 0.012 seconds

```

7 AND d.department_id = j.department_id
8 AND TO_CHAR(end_date,'q') = 1);
9
10 -- VARIANTA 2
11 SELECT *
12 FROM employees e
13 WHERE employee_id IN (SELECT employee_id
14                       FROM departments d, job_history j
15                       WHERE e.employee_id = j.employee_id
16                             AND TO_CHAR(end_date,'q') = 1);
17
18 -- VARIANTA 3
19 SELECT DISTINCT e.*
20 FROM employees e, departments d, job_history j
21 WHERE e.employee_id = j.employee_id
22 AND d.department_id = j.department_id
23 AND TO_CHAR(end_date,'q') = 1;

```

Query Result x Explain Plan x

SQL 0.012 seconds

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	
SELECT STATEMENT				1	5
HASH		UNIQUE		1	5
NESTED LOOPS				1	4
NESTED LOOPS				1	4
TABLE ACCESS	JOB_HISTORY	FULL		1	3
Filter Predicates					
AND					
J.DEPARTMENT_ID IS NOT NULL					
TO_NUMBER(TO_CHAR(INTERNAL_FUNCTION(END_DATE),'q'))=1					
INDEX	EMP_EMP_ID_PK	UNIQUE SCAN		1	0
Access Predicates					
E.EMPLOYEE_ID=J.EMPLOYEE_ID					
TABLE ACCESS	EMPLOYEES	BY INDEX ROWID		1	1

Other XML

(info)

- info type="db_version"
 - 11.1.0.6
- info type="parse_schema"
 - "GRPUPA231"
- info type="dynamic_sampling"
 - yes
- info type="plan_hash"
 - 2841582929
- info type="plan_hash_2"
 - 1677045019

(hint)

USE_HASH_AGGREGATION(@"SEL#F7859CDE")

Pentru a treia variantă avem costul 5.

Observăm că prima și a treia variantă au cel mai mic cost, 5. A doua variantă are costul cel mai mare, 7. Putem spune că prima variantă este puțin mai eficientă decât a treia (prima variantă costurile: 5, 3, 3. Total: 5; a treia variantă costurile: 5, 4, 3. Total 5);

Cursoare imbricate vs Cursoare parametrizate (Exemplul 5.16 din Curs).

Pentru a vedea diferențele între cele două cursoare vom folosi funcția **sysimestamp** cu care vom vedea timpul la care a început/sfârșit execuția fiecăruia.

Rezolvare:

DECLARE

-- /CURSORUL IMBRICAT

CURSOR ang IS

SELECT first_name,

CURSOR(SELECT MAX(department_name)

FROM departments d, job_history j

WHERE e.employee_id = j.employee_id

AND d.department_id = j.department_id

GROUP BY d.department_id

ORDER BY 1, SUM(employee_id) desc)

FROM employees e

WHERE salary > 3000;

c_nume_ang2 employees.first_name%TYPE;

v_cursor SYS_REFCURSOR;

TYPE tab_dept IS TABLE OF departments.department_name%TYPE

INDEX BY BINARY_INTEGER;

v_dept tab_dept;

-- /CURSORUL PARAMETRIZAT

CURSOR ang2 IS SELECT employee_id, first_name

```

        FROM employees

        WHERE salary > 3000;

CURSOR dept_param(v_ang2 employees.employee_id%TYPE) IS
    SELECT MAX(d.department_name), SUM(employee_id)
    FROM departments d, job_history j
    WHERE v_ang2 = j.employee_id
    AND d.department_id = j.department_id
    GROUP BY d.department_id
    ORDER BY 1,2 desc;

c2_nume employees.first_name%TYPE;
c_id employees.employee_id%TYPE;
p_dept_nume departments.department_name%TYPE;
p_numar NUMBER;

BEGIN

-- CURSOR IMBRICAT START

dbms_output.put_line('Inainte de a deschide cursorul imbricat: ' || systimestamp);

OPEN ang;

LOOP

    FETCH ang INTO c_nume_ang2, v_cursor;

    EXIT WHEN ang%NOTFOUND;

--    DBMS_OUTPUT.PUT_LINE(c_nume_ang2);
--    DBMS_OUTPUT.PUT_LINE('-----');

```

```

        FETCH v_cursor BULK COLLECT INTO v_dept LIMIT 3;

--      IF v_dept.COUNT = 0 THEN
--          DBMS_OUTPUT.PUT_LINE('Nu avem angajati');
--      ELSE
--          FOR i IN v_dept.FIRST..v_dept.LAST LOOP
--              DBMS_OUTPUT.PUT_LINE(i || ' ' || v_dept(i));
--          END LOOP;
--      END IF;
--
--      DBMS_OUTPUT.NEW_LINE;
END LOOP;

CLOSE ang;

dbms_output.put_line('Dupa ce s-a inchis cursorul imbricat: ' || systimestamp);

-- CURSOR IMBRICAT GATA

-- CURSOR PARAMETRIZAT START

dbms_output.put_line('Inainte de a deschide cursorul parametrizat: ' || systimestamp);

OPEN ang2;

LOOP

    FETCH ang2 INTO c_id, c2_nume;

    EXIT WHEN ang2%NOTFOUND;

--    DBMS_OUTPUT.PUT_LINE(c2_nume);
--    DBMS_OUTPUT.PUT_LINE('-----');

    OPEN dept_param(c_id);

    LOOP

        FETCH dept_param INTO p_dept_nume, p_numar;

        EXIT WHEN dept_param%NOTFOUND OR dept_param%ROWCOUNT > 3;

```

```

END LOOP;

CLOSE dept_param;

END LOOP;

CLOSE ang2;

dbms_output.put_line('Dupa ce s-a inchis cursorul parametrizat: ' || systimestamp);

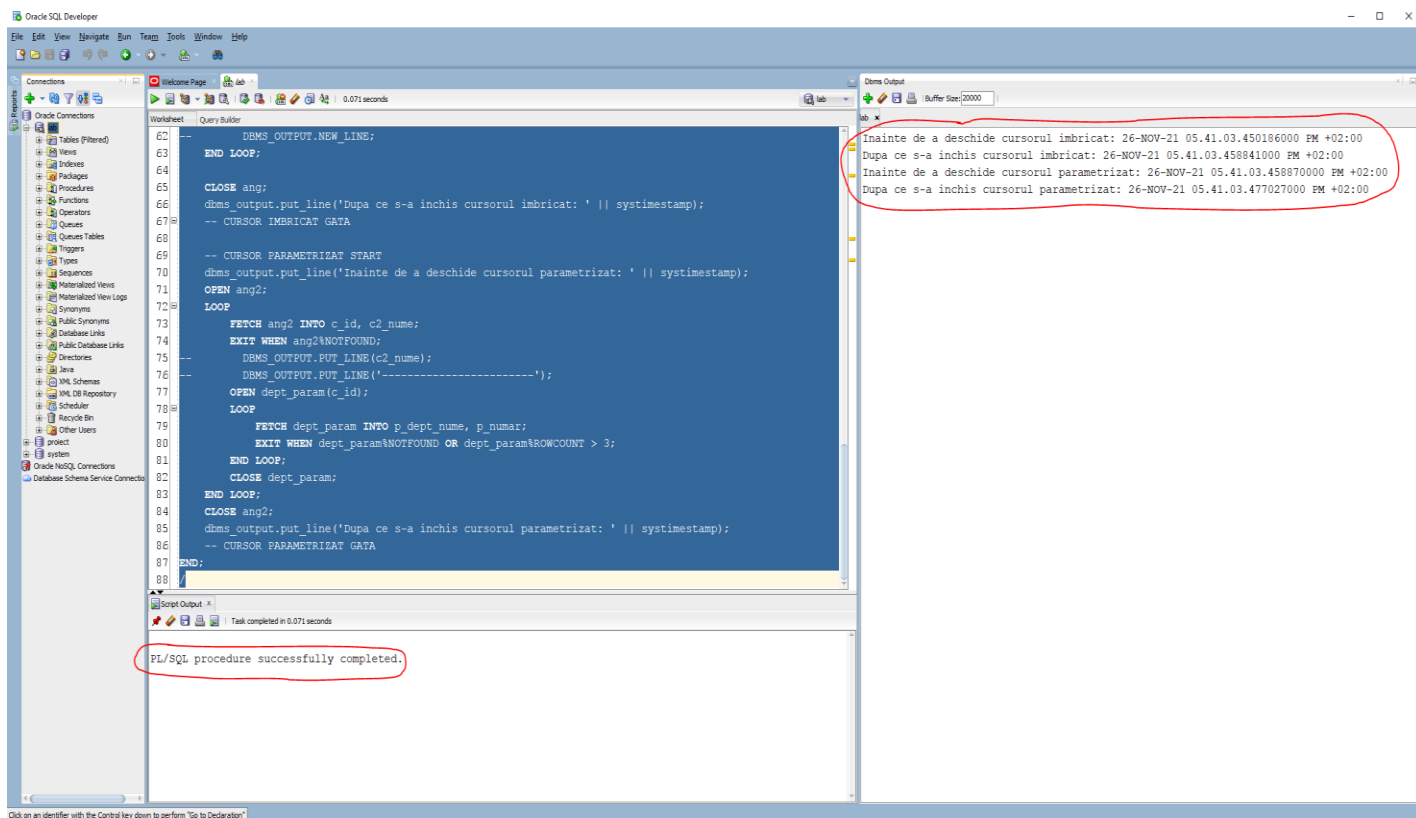
-- CURSOR PARAMETRIZAT GATA

END;

/

```

Print-Screen:



Putem observa că blocul nostru PL/SQL s-a executat cu succes. În dreapta vedem pe prima linie, respectiv a doua, ora la care a început/sfârșit monitorizarea pentru cursorul *imbricat*, respectiv *parametrizat*.

Diferențele sunt infime și în cazul acesta! Ele se pot observa la ordinul milisecundelor. Pentru a vedea cu exactitate vom face un diferență între timpul de început și timpul de sfârșit folosind calculatorul.

The screenshot shows the Oracle SQL Developer interface with a PL/SQL procedure named `DBMS_OUTPUT.PUT_LINE` that uses an `IMBRICAT` cursor. The procedure is executed, and the output window shows the execution time of the cursor. A calculator window is open, showing the calculation $458841000 - 450186000 = 8,655,000$. The result **8,655,000** is highlighted in red.

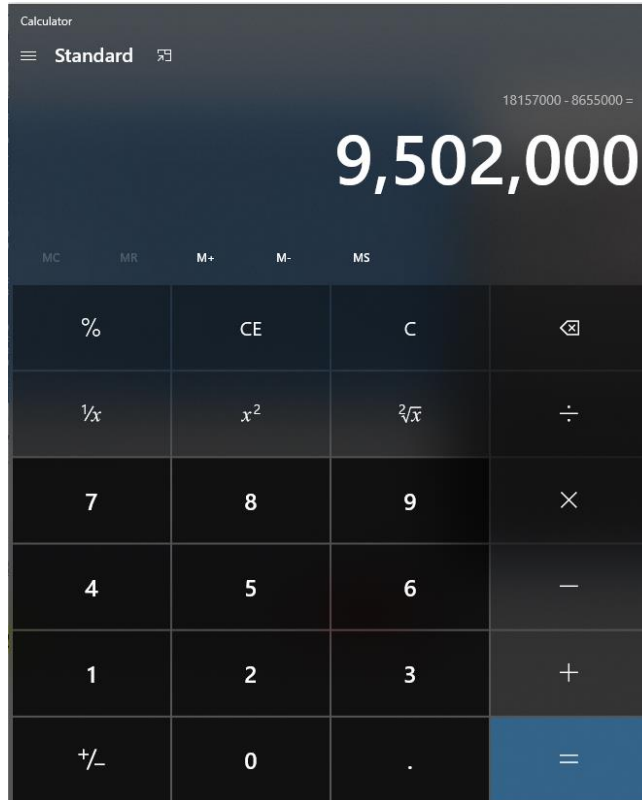
PL/SQL procedure successfully completed.

Observăm că a durat **8.655.000** unități (ordinul milisecundelor), execuția cursorului *imbricat*.
Acum vom face calculul pentru cursorul *parametrizat*.

The screenshot shows the Oracle SQL Developer interface with a PL/SQL procedure named `DBMS_OUTPUT.PUT_LINE` that uses a `PARAMETRIZAT` cursor. The procedure is executed, and the output window shows the execution time of the cursor. A calculator window is open, showing the calculation $477027000 - 458870000 = 18,157,000$. The result **18,157,000** is highlighted in red.

PL/SQL procedure successfully completed.

Observăm că cursorul *parametrizat* a durat **18.157.000** unități (ordinul milisecundelor). Deci s-a executat cu **9.502.000** de unități mai încet decât cursorul *imbricat*!



Dacă testăm mai întâi pentru cursorul *parametrizat* și după aceea pentru cursorul *imbricat*, vom observa că indiferent de ordinea execuției cursorul *imbricat* se va executa mult mai repede decât cursorul *parametrizat*.

The screenshot displays the Oracle SQL Developer environment. The main window shows a PL/SQL script with the following code:

```

62 LOOP
63   FETCH ang INTO c_nume_ang2, v_cursor;
64
65   EXIT WHEN ang%NOTFOUND;
66
67   DBMS_OUTPUT.PUT_LINE(c_nume_ang2);
68   DBMS_OUTPUT.PUT_LINE('-----');
69
70   FETCH v_cursor BULK COLLECT INTO v_dept LIMIT 3;
71
72   IF v_dept.COUNT = 0 THEN
73     DBMS_OUTPUT.PUT_LINE('Nu avem angajati');
74   ELSE
75     FOR i IN v_dept.FIRST..v_dept.LAST LOOP
76       DBMS_OUTPUT.PUT_LINE(i || ' ' || v_dept(i));
77     END LOOP;
78   END IF;
79   DBMS_OUTPUT.PUT_LINE('');
80 END LOOP;
81
82 CLOSE ang;
83 dbms_output.put_line('Dupa ce s-a inchis cursorul imbricat: ' || systimestamp);
84 -- CURSOR IMBRICAT GATA
85 END;
86
87
88

```

The 'DMS Output' window shows the execution results, including the time taken for each cursor. The 'imbricat' cursor is significantly faster than the 'parametrizat' cursor. A calculator is overlaid on the right, showing the calculation 18,384,000.

Avem **18.384.000** unități pentru cursorul parametrizat.

The screenshot shows the Oracle SQL Developer interface. The main window displays a PL/SQL procedure named 'DEMS_OUTPUT.PUT_LINE' with a loop structure. The procedure is designed to fetch data from a cursor and output it. The output window shows the execution results, including timestamps and the number of rows fetched. A calculator window is open in the foreground, displaying the calculation $810430000 - 802192000 = 8,238,000$. The calculator also shows a history of previous calculations.

Avem **8.238.000** unități pentru cursorul *imbricat*. Deci din nou avem o diferență observabilă între cele două, la nivelul microsecundelor de **10.146.000** unități.

The screenshot shows a Windows Calculator application in Standard mode. The display shows the calculation $18384000 - 8238000 = 10,146,000$. The calculator interface includes buttons for basic arithmetic operations and a history of calculations.

De aici putem trage concluzia că, cursoarele *imbricate* sunt mult mai rapide decât cursoarele *parametrizate*! De ce? Pentru că cursoarele *parametrizate* se comportă asemenea unor subprograme (se fac copii pe date, verificări pentru parametri/tipuri etc.).

Popescu Paullo Robertto Karloss

Grupa 231

Temă SGBD #8