

# TEMĂ

În această temă voi pune în evidență de ce este mai recomandat, dar și optim să folosim un cursor explicit (FETCH BULK COLLECT INTO) și nu un cursor implicit (SELECT BULK COLLECT INTO). De asemenea voi prezenta efectul utilizării mai multor comenzi FETCH limit 10 (exemplul 5.7 din curs).

## **Cursor explicit (FETCH BULK COLLECT INTO) vs Cursor implicit (SELECT BULK COLLECT INTO).**

Spre deosebire de SELECT BULK COLLECT INTO, care implică un cursor *implicit*, FETCH BULK COLLECT INTO, indică faptul că, apelăm un cursor *explicit* (definit de noi în zona DECLARE), care poate fi deschis o singură odată (când avem nevoie de el). Atunci când noi ne declarăm un cursor *explicit*, acesta își alocă practic memorie dinamic, iar această zonă de memorie este accesibilă doar atunci când folosim comanda *open nume\_cursor*. Ea este ștearsă atunci când cursorul este închis (nu mai pointează către acea zonă), folosind comanda *close nume\_cursor*.

Folosind un cursor *explicit* avem un avantaj semnificativ asupra controlului cererilor noastre SQL, în sensul că, putem decide când să se efectueze FETCH pe înregistrările noastre, sau câte înregistrări vrem să recuperăm odată. Aceste aspecte fiind imposibile de realizat prin intermediul unui cursor *implicit* (SELECT BULK COLLECT INTO...). Eu consider că, este mai bine să folosim cursorul *explicit* din exemplul 5.6 din curs decât un cursor implicit (SELECT BULK COLLECT INTO) deoarece, un cursor *implicit* care este creat de baza de date Oracle, și nu oferă flexibilitatea pe care o oferă un cursor *explicit* (dacă dorim să executăm o instrucțiune DML, de exemplu, cursorul *implicit* nu oferă programatorului un control sporit asupra ei). Deși, dacă vom testa în SQL Developer pentru a vedea, care este mai rapid din punct de vedere al timpului, vom observa că cursorul implicit se execută mai repede decât cursorul explicit.

Vom crea un tabel categorii, în care vom insera multe înregistrări random (la fel ca exemplul 5.6 din curs). Pentru a vedea diferențele între cele două vom folosi funcția ***dbms\_utility.get\_time()***; cu care vom calcula timpul de execuție pentru ambele (timp\_execuție = stop\_time – start\_time). Mai jos am atașat codul și câteva PRINT-SCREEN-uri.

### **Rezolvare:**

```
CREATE TABLE categorii (id_categorie INT PRIMARY KEY,  
                        denumire VARCHAR2(20),  
                        id_parinte INT DEFAULT NULL);
```

```
INSERT INTO categorii (id_categorie)
```

```
SELECT LEVEL  
FROM DUAL  
CONNECT BY LEVEL <= 3000000;
```

```
SELECT * FROM categorii;
```

```
COMMIT;
```

```
DECLARE  
    TYPE tab_ind IS TABLE OF categorii%ROWTYPE INDEX BY PLS_INTEGER;  
    t tab_ind;  
    t2 tab_ind;
```

```
CURSOR c IS  
    SELECT * FROM categorii  
    WHERE id_parinte IS NULL;
```

```
    timp_executie NUMBER(10);  
    start_time NUMBER(10);  
    stop_time NUMBER(10);
```

```
BEGIN  
    start_time := dbms_utility.get_time();  
    SELECT * BULK COLLECT INTO t2  
    FROM categorii  
    WHERE id_parinte IS NULL;  
    stop_time := dbms_utility.get_time();
```

```

    timp_executie := stop_time - start_time;

    dbms_output.put_line('SELECT BULK COLLECT INTO a durat: ' || timp_executie);

start_time := dbms_utility.get_time();

OPEN c;

FETCH c BULK COLLECT INTO t;

CLOSE c;

stop_time := dbms_utility.get_time();

timp_executie := stop_time - start_time;

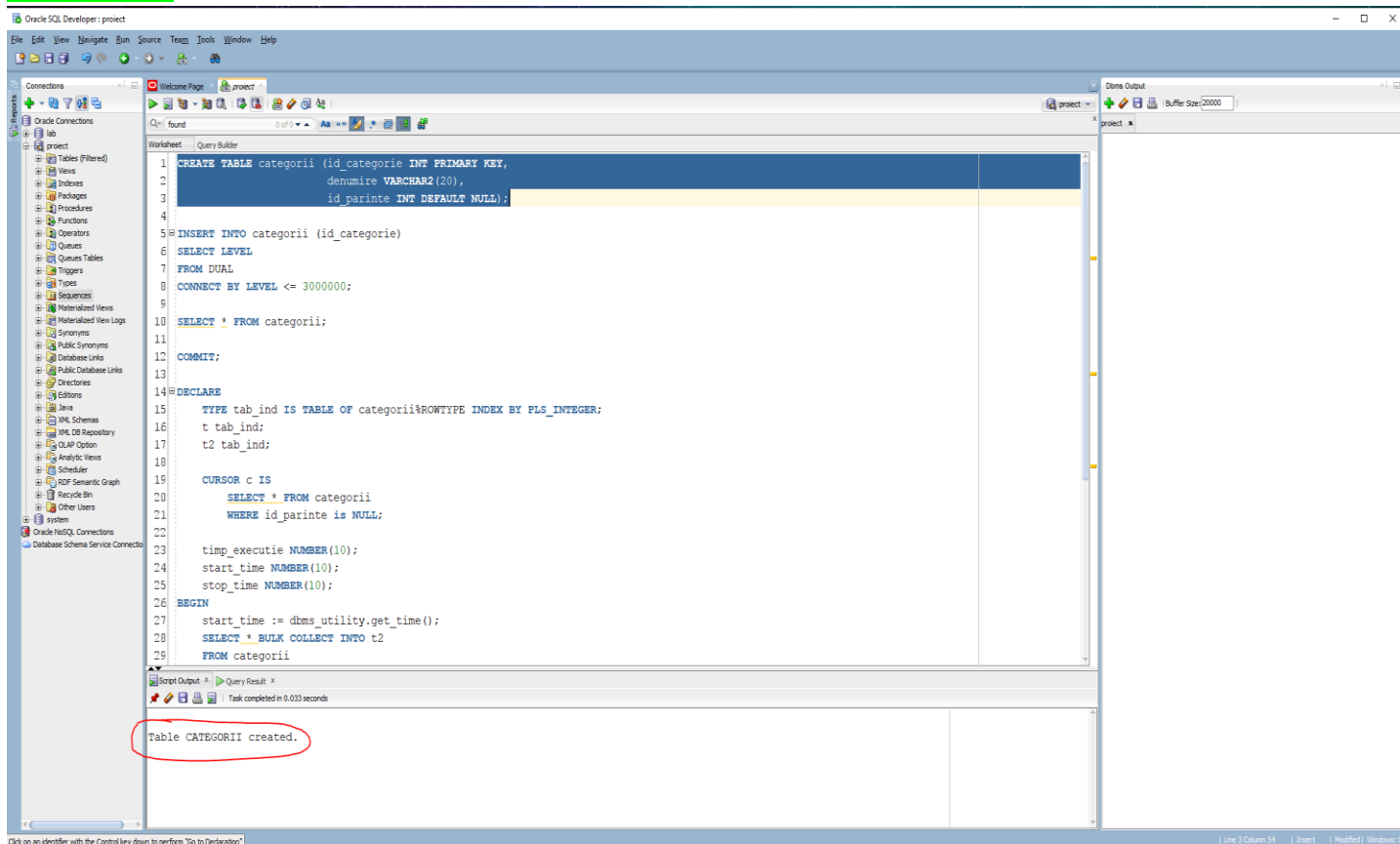
dbms_output.put_line('FETCH BULK COLLECT INTO a durat: ' || timp_executie);

END;

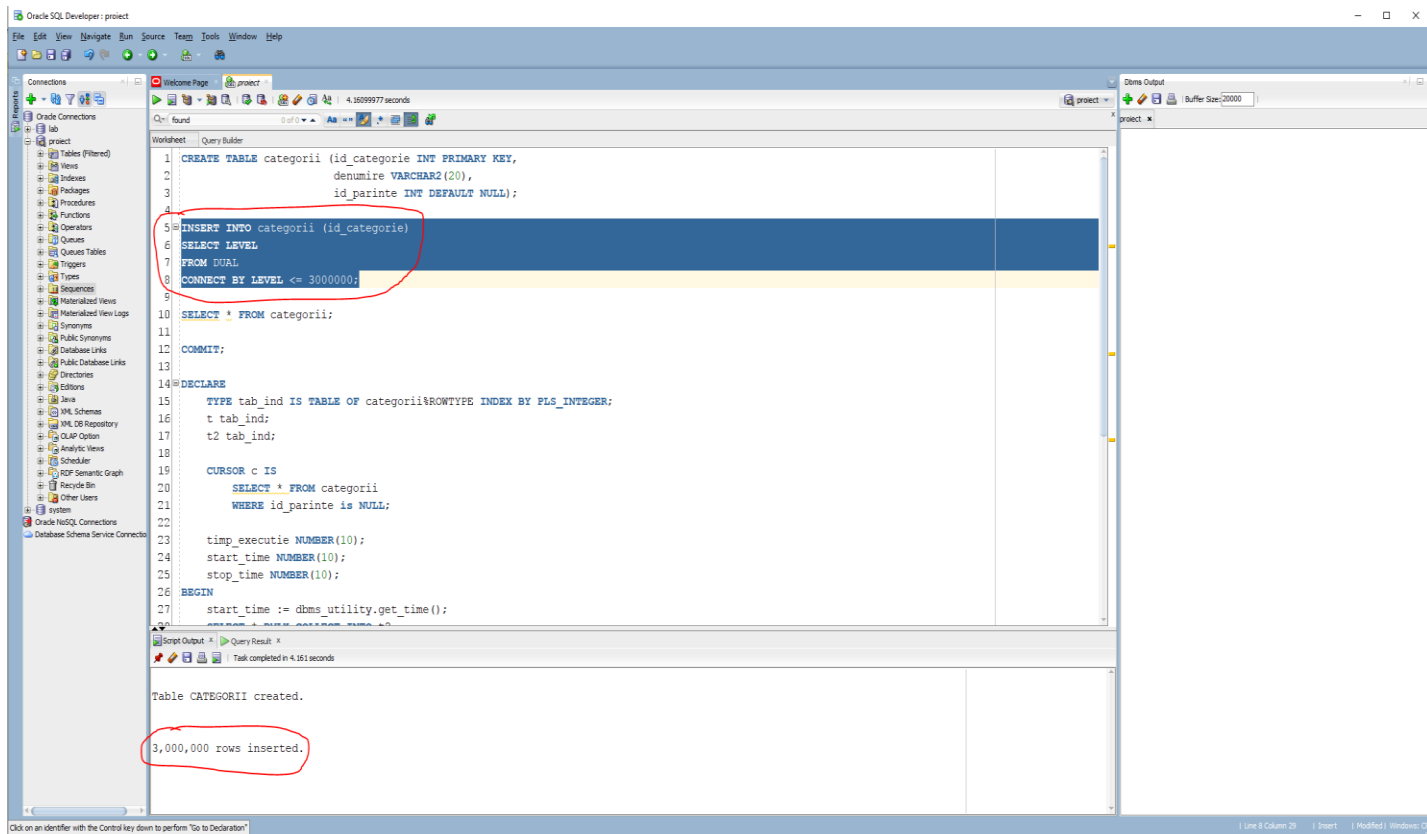
/

```

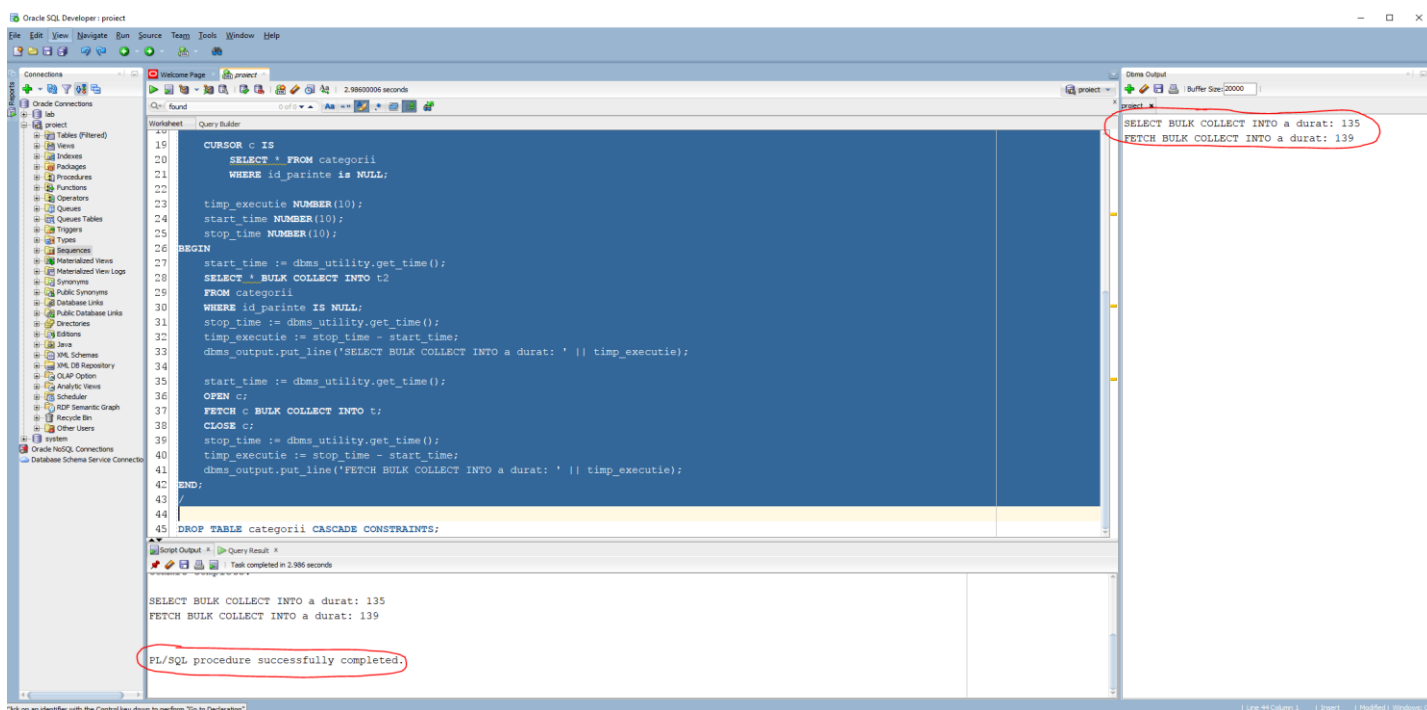
### Print-Screen:



Tabelul nostru s-a creat cu succes, acum vom insera multe înregistrări random în el.



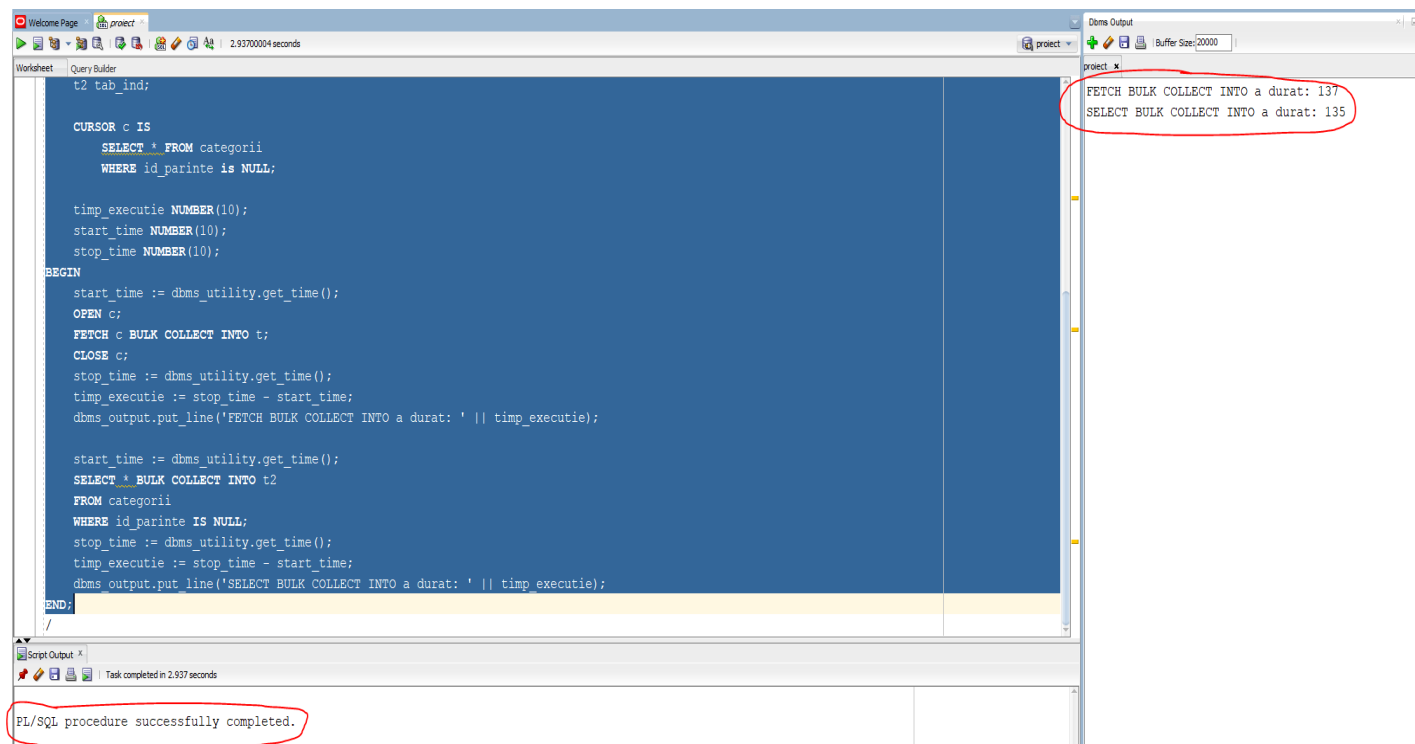
Observăm că am reușit să inserăm cu succes 3.000.000 de înregistrări random. În continuare vom vedea care s-a executat mai rapid.



Observăm că **SELECT BULK COLLECT INTO** a durat **135** unități, iar **FETCH BULK COLLECT INTO** a durat **139** unități. Deci, cursorul *implicit*, definit de ORACLE, **SELECT BULK COLLECT INTO** s-a executat cu **4** unități mai repede decât cursorul nostru *explicit* cu **FETCH BULK COLLECT INTO**!

Din acest punct de vedere, cursorul *implicit* **SELECT BULK COLLECT INTO** este mai eficient decât un cursor *explicit*, **FETCH BULK COLLECT INTO**.

Dacă testăm mai întâi **FETCH BULK COLLECT INTO** și după **SELECT BULK COLLECT INTO**, care va fi mai rapid? Tot cursorul implicit **SELECT BULK COLLECT INTO** sau de data aceasta va fi **FETCH BULK COLLECT INTO**? Testăm acest lucru.



```
Worksheet: Query Builder
t2 tab_ind;

CURSOR c IS
  SELECT * FROM categorii
  WHERE id_parinte IS NULL;

timp_executie NUMBER(10);
start_time NUMBER(10);
stop_time NUMBER(10);

BEGIN
  start_time := dbms_utility.get_time();
  OPEN c;
  FETCH c BULK COLLECT INTO t;
  CLOSE c;
  stop_time := dbms_utility.get_time();
  timp_executie := stop_time - start_time;
  dbms_output.put_line('FETCH BULK COLLECT INTO a durat: ' || timp_executie);

  start_time := dbms_utility.get_time();
  SELECT * BULK COLLECT INTO t2
  FROM categorii
  WHERE id_parinte IS NULL;
  stop_time := dbms_utility.get_time();
  timp_executie := stop_time - start_time;
  dbms_output.put_line('SELECT BULK COLLECT INTO a durat: ' || timp_executie);
END;
```

Dbms Output

FETCH BULK COLLECT INTO a durat: 137  
SELECT BULK COLLECT INTO a durat: 135

Script Output

Task completed in 2.937 seconds

PL/SQL procedure successfully completed.

Tot **SELECT BULK COLLECT INTO** este mai rapid, în unele cazuri timpul acestuia de execuție pot să fi egal cu cel pentru **FETCH BULK COLLECT INTO**.

**Efectul utilizării mai multor comenzi FETCH limit 10 în exemplul 5.7 din Curs.**

Pentru început vom adăuga mai multe comenzi FETCH în interiorul cursorului, dar și afișări pentru fiecare, pentru a vedea cu exactitate ce se întâmplă.

**Rezolvare:**

DECLARE

TYPE tab\_imb IS TABLE OF employees.employee\_id%TYPE;

CURSOR c1 is

SELECT employee\_id

FROM employees;

v\_coduri tab\_imb;

BEGIN

OPEN c1;

DBMS\_OUTPUT.PUT\_LINE('Facem prima data FETCH:');

FETCH c1 BULK COLLECT INTO v\_coduri LIMIT 10;

FOR i IN 1..v\_coduri.LAST LOOP

DBMS\_OUTPUT.PUT\_LINE(v\_coduri(i));

END LOOP;

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

DBMS\_OUTPUT.PUT\_LINE('Facem a doua oara FETCH:');

FETCH c1 BULK COLLECT INTO v\_coduri LIMIT 10;

FOR i IN 1..v\_coduri.LAST LOOP

DBMS\_OUTPUT.PUT\_LINE(v\_coduri(i));

END LOOP;

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

DBMS\_OUTPUT.PUT\_LINE('Facem a doua oara FETCH:');

FETCH c1 BULK COLLECT INTO v\_coduri LIMIT 10;

FOR i IN 1..v\_coduri.LAST LOOP

DBMS\_OUTPUT.PUT\_LINE(v\_coduri(i));

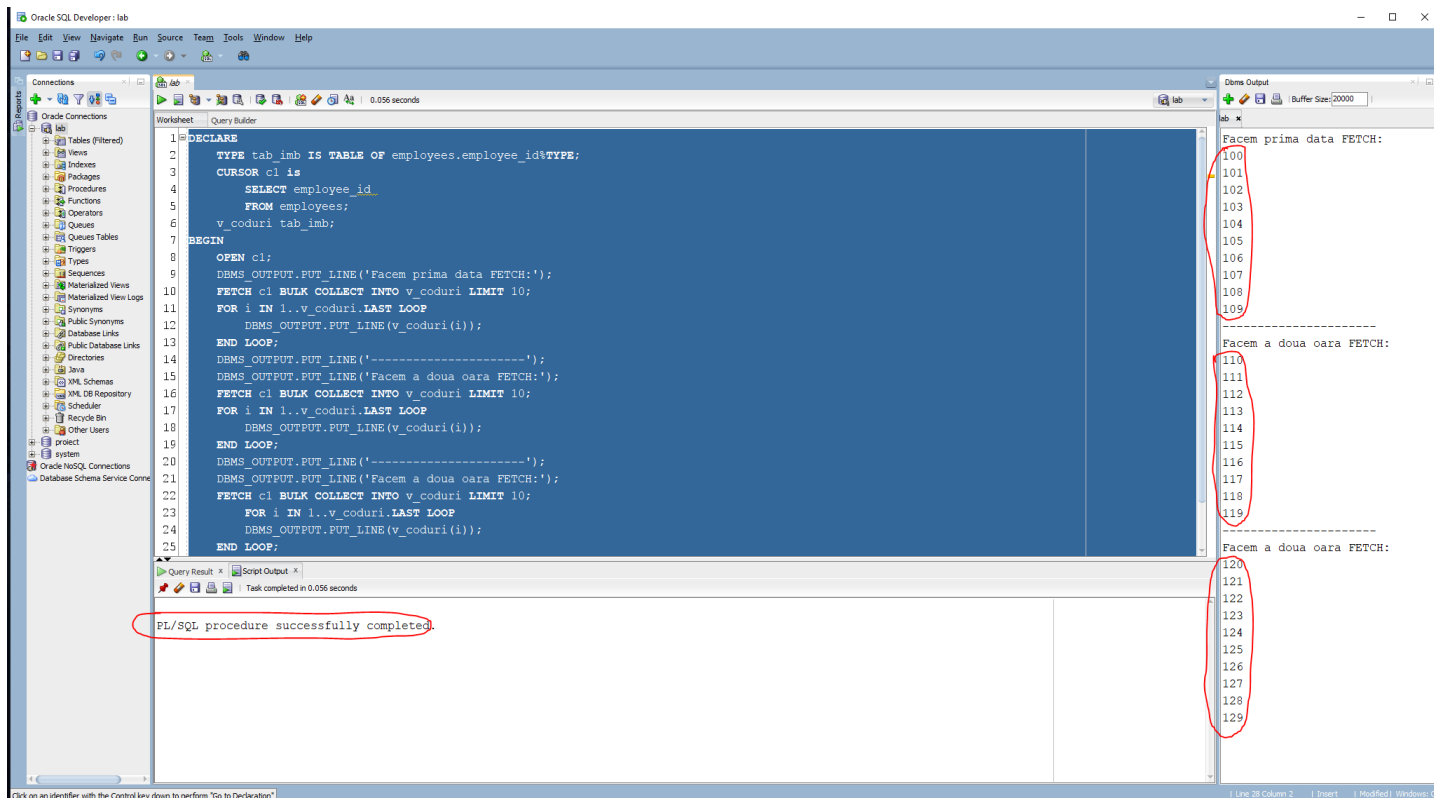
END LOOP;

CLOSE c1;

END;

/

**Print-Screen:**



Observăm că la fiecare FETCH, compilatorul ia câte 10 linii din tabelul employees. La fiecare FETCH avem valori diferite, deci compilatorul știe de unde să continue la fiecare FETCH (continuă de unde a rămas FETCH-ul său anterior).

La primul FETCH compilatorul ia primele 10 linii din tabelul (id-urile 100, 101, ..., 109). El reține în memorie că s-a oprit la a 11-a înregistrare.

La al doilea FETCH continuă de unde acesta s-a oprit, deci de la a 11-a înregistrare. Afișează următoarele 10 linii (110, 111, 112, ..., 119). Reține în memorie că s-a oprit la noua a 11-a înregistrare (la 120).

La ultimul FETCH el continuă de unde a rămas la FETCH-ul anterior, și anume de la id-ul angajatului 120..

De aici putem observa un avantaj al utilizării cursorului *explicit*, care spre deosebire de cursorul *implicit* poate realiza FETCH după felul în care ne dorim noi (oferă flexibilitate programatorului).

Popescu Paullo Robertto Karloss

Grupa 231

Temă SGBD #7