

# **SISTEME DE GESTIUNE A BAZELOR DE DATE**

**AN UNIVERSITAR 2018-2019  
INFORMATICĂ AN III SEM I**

**Lect. Univ. Dr. Gabriela Mihai**

## 5. PL/SQL – GESTIUNEA CURSOARELOR

- Cursorare implicite
- Cursorare explicite
  - Gestiunea cursorarelor explicite
  - Cursorare parametrizate
  - Cursorare SELECT FOR UPDATE
  - Cursorare dinamice

## 5. PL/SQL – Gestiunea cursorurilor



### ■ Un cursor

- este un pointer către o zonă de memorie

  - *Private SQL Area*

- în care sunt stocate informații despre procesarea unei comenzi *SELECT* sau *LMD*.



În acest capitol se discută cursorurile la nivel sesiune.



## 5. PL/SQL – Gestiunea cursorurilor

- Cursorurile la nivel de sesiune:
  - sunt diferite față de cursorurile ce folosesc zona privată *SQL* din *PGA* (*Program Global Area*);
  - există în memoria alocată sesiunii până la momentul încheierii acesteia.
- Vizualizarea *V\$OPEN\_CURSOR*
  - informații despre cursorurile deschise la nivel de sesiune ale fiecărei sesiuni utilizator.

## 5. PL/SQL – Gestiunea cursorurilor



În continuare, din motive de simplificare a exprimării, pentru un cursor la nivel de sesiune se va utiliza termenul de cursor.





## 5. PL/SQL – Gestiunea cursorurilor

■ Atributele care furnizează informații despre cursoruri:

■ pot fi referite doar de comenzi procedurale

■ pot fi referite utilizând sintaxa

■ pentru cursorurile implicite

`SQL%nume_atribut`

■ pentru cursorurile explicite

`nume_cursor%nume_atribut`

### ■ **%ROWCOUNT**

- este de tip întreg (*PLS\_INTEGER*);
- are valoarea *NULL* dacă nu a fost rulată nicio comandă *SELECT* sau *LMD*;
- reprezintă numărul liniilor întoarse de ultima comandă *SELECT* sau numărul de linii afectate de ultima comandă *LMD*;
- dacă numărul de linii este mai mare decât valoarea maximă permisă de tipul *PLS\_INTEGER* (2.147.483.647), atunci întoarce o valoare negativă.



## 5. PL/SQL – Gestiunea cursorurilor

### ■ **%FOUND**

- este de tip boolean;
- are valoarea *NULL* dacă nu a fost rulată nicio comandă *SELECT* sau *LMD*;
- în cazul cursorurilor implicite
  - are valoarea *TRUE* dacă ultima comandă *SELECT* a întors cel puțin o linie sau ultima comandă *LMD* a afectat cel puțin o linie;
- în cazul cursorurilor explicite
  - are valoarea *TRUE* dacă ultima operație de încărcare (*FETCH*) dintr-un cursor a avut succes.





## 5. PL/SQL – Gestiunea cursorilor

### ■ **%NOTFOUND**

- are semnificație opusă față de cea a atributului *%FOUND*

### ■ **%BULK\_ROWCOUNT**

- vezi comanda *FORALL*

### ■ **%BULK\_EXCEPTIONS**

- vezi comanda *FORALL*



## 5. PL/SQL – Gestiunea cursorurilor

### ■ **%ISOPEN**

- este de tip boolean;
- indică dacă un cursor este deschis;
- în cazul cursorurilor implicite
  - are întotdeauna valoarea *FALSE*, deoarece un cursor implicit este închis de sistem imediat după execuția instrucțiunii *SQL* asociate.



## 5.1. Cursoare implicite

- *PL/SQL* deschide automat un cursor implicit la nivel de sesiune de fiecare dată când este rulată o comandă *SELECT* sau *LMD*.
- Mai sunt denumite și cursoare *SQL*.
- Cursorul implicit este închis automat, atunci când comanda se încheie.
- Valorile atributelor asociate cursorului rămân disponibile până când este rulată o altă comandă *SELECT* sau *LMD*.



## 5.1. Cursoare implicite

### Exemplul 5.1

```
BEGIN
  DELETE FROM tip_plata
  WHERE   id_tip_plata NOT IN
          (SELECT id_tip_plata FROM facturi);

  -- cursor deschis?
  IF SQL%ISOPEN
  THEN
    DBMS_OUTPUT.PUT_LINE('Cursor deschis');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Cursor inchis');
  END IF;
  -- SQL% se refera la comanda DELETE sau la comanda SELECT?
```



## 5.1. Cursoare implicite

```
-- a gasit linii?  
IF SQL%FOUND  
THEN  
    DBMS_OUTPUT.PUT_LINE('A fost gasita cel putin o linie');  
ELSE  
    DBMS_OUTPUT.PUT_LINE('Nu a fost gasita nicio linie');  
END IF;  
-- cate linii a gasit  
DBMS_OUTPUT.PUT_LINE('Au fost sterse ' || SQL%ROWCOUNT  
    || ' linii');  
END;  
/
```



## 5.1. Cursoare implicite

### ▣ Output

Cursor inchis

A fost gasita cel putin o linie

Au fost sterse 2 linii



## 5.1. Cursoare implicite



În cazul comenzii *SELECT INTO* valoarea atributului ***SQL%NOTFOUND*** nu este utilă:

- ❖ dacă nu întoarce linii, atunci apare imediat excepția *NO\_DATA\_FOUND*
  - înainte să se poată verifica valoarea atributului
- ❖ dacă în lista *SELECT* a comenzii se utilizează funcții agregat, atunci este întoarsă întotdeauna o valoare.
  - valoarea atributului este *FALSE*

## 5.1. Cursoare implicite



- ❖ Dacă o comandă *SELECT INTO* (nu este folosită clauza *BULK COLLECT*) întoarce mai multe linii, atunci apare imediat excepția *TOO\_MANY\_ROWS*.
- ❖ În acest caz, atributul *SQL%ROWCOUNT* are valoarea 1 (nu numărul de linii care satisfac cererea).





## 5.2. Cursoare explicite

- Cursoarele explicite
  - sunt cursoare la nivel de sesiune
  - sunt definite și gestionate de către utilizator
- Un cursor explicit
  - are specificat un nume
  - este asociat cu o comandă SELECT ce întoarce de obicei mai multe linii



## 5.2. Cursoare explicite

- Mulțimea rezultat a cererii asociate poate fi procesată folosind una dintre variantele următoare:


- **Varianta 1**

- se deschide cursorul (*OPEN*)
- se încarcă liniile cursorului în variabile (*FETCH*)
- se închide cursorul (*CLOSE*)

- **Varianta 2**

- se utilizează cursorul într-o comandă *FOR LOOP*

## 5.2.1. Gestiunea cursoarelor explicite



```
DECLARE
    declarare cursor
BEGIN
    deschidere cursor (OPEN)
    WHILE rămân linii de recuperat LOOP
        recuperare linie rezultat (FETCH)
    END LOOP
    închidere cursor (CLOSE)
END;
```

## 5.2.1. Gestiunea cursorurilor explicite



### ■ Declararea unui cursor explicit

- Sintaxa de declarare, fără a asocia comanda *SELECT*

```
CURSOR nume_cursor [RETURN tip];
```

- Sintaxa de declarare, cu asociere a comanda *SELECT*

```
CURSOR nume_cursor [RETURN tip]  
IS comanda_SELECT;
```

## 5.2.1. Gestiunea cursorurilor explicite



### Exemplul 5.2

```
DECLARE
    CURSOR c1 RETURN produse%ROWTYPE;

    CURSOR c2 IS
        SELECT id_produ, denumire FROM produse;

    CURSOR c1 RETURN produse%ROWTYPE IS
        SELECT * FROM PRODUSE;

BEGIN
    NULL;
END;
```

## 5.2.1. Gestiunea cursoarelor explicite



- ❖ Numele cursorului:
  - ❖ este un identificator unic în cadrul blocului
  - ❖ nu poate să apară într-o expresie
  - ❖ nu i se poate atribui o valoare
  
- ❖ Comanda *SELECT* care apare în declararea cursorului, **nu trebuie să includă clauza *INTO*.**

## 5.2.1. Gestiunea cursoarelor explicite



- ❖ Dacă se cere procesarea liniilor într-o anumită ordine, atunci
  - ❖ în cerere este utilizată clauza *ORDER BY*
  
- ❖ Dacă în lista *SELECT* apare o expresie, atunci
  - ❖ pentru expresia respectivă trebuie utilizat un alias
  - ❖ câmpul expresie se va referi prin acest alias

## 5.2.1. Gestiunea cursoarelor explicite



### ■ Deschiderea unui cursor explicit

```
OPEN nume_cursor;
```

- Dacă se încearcă deschiderea unui cursor deja deschis, atunci pare excepția *CURSOR\_ALREADY\_OPEN*.



## 5.2.1. Gestiunea cursoarelor explicite



- Operații realizate la deschiderea unui cursor
  - se alocă resursele necesare pentru a procesa cererea
  - se procesează cererea
    - se identifică mulțimea activă prin execuția cererii *SELECT*
    - dacă cererea include clauza *FOR UPDATE*, atunci liniile din mulțimea activă sunt blocate
- se poziționează *pointer*-ul înaintea primei linii din mulțimea activă

## 5.2.1. Gestiunea cursoarelor explicite

### Exemplul 5.3

■ Ce se afișează în urma execuției blocului PL/SQL?

```
DECLARE
  CURSOR c1 IS
    SELECT * FROM categorii WHERE id_parinte IS NULL;
  CURSOR c2 IS
    SELECT * FROM categorii WHERE 1=2;
BEGIN
  OPEN c1;
  IF c1%FOUND THEN
    DBMS_OUTPUT.PUT_LINE('c1 - cel puțin o linie');
  ELSE
    DBMS_OUTPUT.PUT_LINE('c1 - nicio linie');
  END IF;
```

## 5.2.1. Gestiunea cursorurilor explicite



```
OPEN c2;  
IF c2%NOTFOUND THEN  
    DBMS_OUTPUT.PUT_LINE('c2 - nicio linie');  
ELSE  
    DBMS_OUTPUT.PUT_LINE('c2 - cel putin o linie');  
END IF;  
  
CLOSE c1;  
CLOSE c2;  
END;
```

## 5.2.1. Gestiunea cursoarelor explicite



### ■ Output

c1 - nicio linie

c2 - cel puțin o linie



## 5.2.1. Gestiunea cursoarelor explicite

### ■ Încărcarea datelor dintr-un cursor explicit

```
FETCH nume_cursor INTO {nume_variabilă  
    [, nume_variabilă] ... | nume_înregistrare};
```

```
FETCH nume_cursor BULK COLLECT INTO  
    {nume_variabilă_colecție  
    [,nume_variabilă_colecție]
```

## 5.2.1. Gestiunea cursorurilor explicite



- *FETCH* realizează următoarele operații:
  - avansează *pointer*-ul la următoarea linie în mulțimea activă
    - *pointer*-ul poate avea doar un sens de deplasare de la prima înregistrare spre ultima
  - citește datele liniei curente în variabile *PL/SQL*
  - dacă *pointer*-ul este poziționat la sfârșitul mulțimii active, atunci se iese din bucla cursorului

## 5.2.1. Gestiunea cursoarelor explicite



Comanda *FETCH INTO* regăsește la un moment dat o singură linie.



Comanda *FETCH BULK COLLECT INTO* încarcă la un moment mai multe linii în colecții.

## 5.2.1. Gestiunea cursorurilor explicite



### Exemplul 5.4

```
DECLARE
  CURSOR c IS
    SELECT id_categorie, denumire FROM categorii
    WHERE id_parinte IS NULL;
  v_id_categorie categorii.id_categorie%TYPE;
  v_denumire      categorii.denumire%TYPE;
BEGIN
  OPEN c;
  LOOP
    FETCH c INTO v_id_categorie, v_denumire;
    EXIT WHEN c%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_id_categorie || ' ' || v_denumire);
  END LOOP;
  CLOSE c;
END;
```



## 5.2.1. Gestiunea cursoarelor explicite



### ▣ Output

289 Industriale

290 IT

291 Papetarie

292 Mobilier

## 5.2.1. Gestiunea cursorurilor explicite

### Exemplul 5.5

```
DECLARE
  CURSOR c IS
    SELECT * FROM categorii
    WHERE id_parinte IS NULL;
  v_categorii categorii%ROWTYPE;
BEGIN
  OPEN c;
  FETCH c INTO v_categorii;
  WHILE c%FOUND LOOP
    DBMS_OUTPUT.PUT_LINE(v_categorii.id_categorie || ' ' ||
      v_categorii.denumire);
    FETCH c INTO v_categorii;
  END LOOP;
  CLOSE c;
END;
```

## 5.2.1. Gestiunea cursorurilor explicite



- ❖ Atunci când un cursor încarcă o linie, acesta realizează o „schimbare de context”
  - ❖ controlul este preluat de motorul *SQL* care va obține datele
- ❖ Motorul *SQL* plasează datele în memorie și are loc o altă „schimbare de context”
  - ❖ controlul este preluat de motorul *PL/SQL*
- ❖ Schimbările de context sunt foarte rapide, dar numărul prea mare de astfel de operații poate implica performanță scăzută

## 5.2.1. Gestiunea cursorurilor explicite



### ❖ Metoda *BULK COLLECT*

- ❖ sunt obținute mai multe linii, implicând doar 2 schimbări de context.

### ❖ Începând cu *Oracle 10g*, un cursor poate determina ca *PL/SQL* să realizeze implicit operații *BULK COLLECT*, încărcând câte 100 linii la un moment dat, fără a mai fi necesară utilizarea colecțiilor.

- ❖ Utilizarea colecțiilor se poate dovedi utilă, doar dacă sunt încărcate mai multe sute de linii.

## 5.2.1. Gestiunea cursoroarelor explicite



### Exemplul 5.6

```
DECLARE
TYPE tab_imb IS TABLE OF categorii%ROWTYPE;
v_categorii tab_imb;
CURSOR c IS
    SELECT * FROM    categorii
    WHERE   id_parinte IS NULL;
BEGIN
    OPEN c;
    FETCH c BULK COLLECT INTO v_categorii;
    CLOSE c;
    FOR i IN 1..v_categorii.LAST LOOP
        DBMS_OUTPUT.PUT_LINE(v_categorii(i).id_categorie || ' '
                               || v_categorii(i).denumire);
    END LOOP;
END;
```

## 5.2.1. Gestiunea cursoarelor explicite



### Exemplul 5.7

```
--limitarea numarului de linii incarcate

DECLARE
  TYPE tab_imb IS TABLE OF produse.denumire%TYPE;
  v_produce tab_imb;
  v_denumire produse.denumire%TYPE;
  CURSOR c1 IS
    SELECT denumire
    FROM   produse
    WHERE  ROWNUM <=10;

  CURSOR c2 IS
    SELECT denumire
    FROM   produse;
```

## 5.2.1. Gestiunea cursoroarelor explicite



```
BEGIN
  OPEN c1;
  LOOP
    FETCH c1 INTO v_denumire;
    EXIT WHEN c1%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_denumire);
  END LOOP;
  CLOSE c1;

  OPEN c2;
  FETCH c2 BULK COLLECT INTO v_produce LIMIT 10;
  CLOSE c2;
  FOR i IN 1..v_produce.LAST LOOP
    DBMS_OUTPUT.PUT_LINE(v_produce(i));
  END LOOP;
END;
```



## 5.2.1. Gestiunea cursoarelor explicite

### ■ Output

Banda adeziva pentru vopsit Tesa, 30 mm x 50 m

Dosar din plastic special Durable, rosu

Biblioraft Standard 50 mm galben

Mapa pentru semnaturi, negru

Capsator C1, albastru fresh

Office Ultimate 2007 pachet RETAIL

Masa calculator, cires

Detergent pentru motor

Brother MFC - 7820N

Tabla fetru PREMIUM, 60x90 cm, albastru



## 5.2.1. Gestiunea cursorurilor explicite



### ■ Închiderea unui cursor explicit

```
CLOSE nume_cursor;
```

- După ce a fost procesată mulțimea activă, cursorul trebuie închis.
  - resursele asociate cursorului sunt eliberate:
    - spațiul utilizat pentru memorarea mulțimii active;
    - spațiul temporar folosit pentru determinarea mulțimii active.

## 5.2.1. Gestiunea cursoarelor explicite



- ❖ Dacă un bloc *PL/SQL* se termină fără a închide un cursor utilizat, sistemul nu va returna o eroare sau un mesaj de avertizare.
- ❖ Se recomandă închiderea cursoarelor pentru a permite sistemului să elibereze resursele alocate.
- ❖ Dacă se încearcă încărcarea datelor dintr-un cursor închis, atunci apare excepția *INVALID\_CURSOR*.

## 5.2.1. Gestiunea cursoarelor explicite

### ■ Valorile atributelor unui cursor explicit

	OPEN		Primul FETCH		Următorul FETCH		Ultimul FETCH		CLOSE	
	Înainte	După	Înainte	După	Înainte	După	Înainte	După	Înainte	După
<b>%ISOPEN</b>	False	True	True	True	True	True	True	True	True	False
<b>%FOUND</b>	Eroare	Null	Null	True	True	True	True	False	False	Eroare
<b>%NOTFOUND</b>	Eroare	Null	Null	False	False	False	False	True	True	Eroare
<b>%ROWCOUNT</b>	Eroare	0	0	1	1	Depinde de date				Eroare

## 5.2.1. Gestiunea cursoarelor explicite



După prima încărcare, dacă mulțimea rezultat este vidă, atunci



- *%FOUND = FALSE*
- *%NOTFOUND = TRUE*
- *%ROWCOUNT = 0*

## 5.2.1. Gestiunea cursorurilor explicite



### ■ Procesarea liniilor unui cursor explicit

- Se utilizează o comandă de ciclare (*LOOP*, *WHILE* sau *FOR*), prin care la fiecare iterație se va încărca o linie nouă.
- Pentru ieșirea din ciclu poate fi utilizată comanda *EXIT*.

### ■ Utilizarea comenzii de ciclare *LOOP*

- vezi exemplul 5.4

### ■ Utilizarea comenzii de ciclare *WHILE*

- vezi exemplul 5.5

## 5.2.1. Gestiunea cursoarelor explicite



Dacă se utilizează una dintre comenzile de ciclare *LOOP* sau *WHILE*, atunci cursorul trebuie:

- declarat
- deschis
- parcurs, încărcând câte o linie la fiecare iterație (trebuie să se asigure ieșirea din buclă atunci când nu mai sunt linii de procesat)
- închis

## 5.2.1. Gestiunea cursoarelor explicite



### ■ Utilizarea comenzii de ciclare *FOR*

- Procesarea liniilor unui cursor explicit se poate realiza și cu ajutorul unui ciclu *FOR* special, numit **ciclu cursor**.



În acest caz cursorul trebuie doar declarat, operațiile de deschidere, încărcare și închidere ale acestuia fiind implicite.



## 5.2.1. Gestiunea cursorurilor explicite

```
FOR nume_înregistrare IN nume_cursor LOOP  
    secvență_de_instrucțiuni;  
END LOOP;
```



Variabila *nume\_înregistrare* nu trebuie declarată.



## 5.2.1. Gestiunea cursorurilor explicite



### Exemplul 5.8

```
DECLARE
  CURSOR c IS
    SELECT *
    FROM   categorii
    WHERE  id_parinte IS NULL;
BEGIN
  FOR i IN c LOOP
    DBMS_OUTPUT.PUT_LINE(i.id_categorie||' '|| i.denumire);
  END LOOP;
END;
```

## 5.2.1. Gestiunea cursoarelor explicite



- Există ciclu cursoare speciale care în comanda *FOR* în loc să refere un cursor declarat, utilizează direct o subcerere (**ciclu cursor cu subcereri**).
  - Nu este necesară nici măcar declararea cursorului.
  - Operațiile de deschidere, încărcare și închidere ale cursorului sunt implicite.

## 5.2.1. Gestiunea cursoroarelor explicite



### Exemplul 5.9

```
BEGIN
  FOR i IN (SELECT *
            FROM   categorii
            WHERE  id_parinte IS NULL) LOOP
    DBMS_OUTPUT.PUT_LINE(i.id_categorie || ' ' || i.denumire);
  END LOOP;
END;
```

## 5.2.2. Cursoare parametrizate



- ❖ Unei variabile de tip cursor îi corespunde o comandă *SELECT*.
  - ❖ Comanda nu poate fi modificată pe parcursul programului.
- ❖ Cursoarele parametrizate sunt cursoare ale căror comenzi *SELECT* depind de parametri ce pot fi modificați la momentul execuției.
  - ❖ Transmiterea de parametri se realizează în mod similar procedurilor stocate.





## 5.2.2. Cursoare parametrizate

### ■ Declararea unui cursor parametrizat

#### ■ Sintaxa de declarare, fără a asocia comanda *SELECT*

```
CURSOR nume_cursor (declarare_parametru  
                    [,declarare_parametru ...])  
    [RETURN tip];
```

#### ■ Sintaxa de declarare, cu asocierea comenzii *SELECT*

```
CURSOR nume_cursor (declarare_parametru  
                    [,declarare_parametru ...])  
    [RETURN tip]  
IS comanda_SELECT;
```



## 5.2.2. Cursoare parametrizate

- sintaxa pentru *declarare\_parametru*

```
nume_parametru [IN] tip_date_scalar  
    [ {:= | DEFAULT} expresie]
```



Parametrul unui cursor nu poate fi declarat *NOT NULL*.



## 5.2.2. Cursoare parametrizate

### ■ Deschiderea unui cursor parametrizat

- Se realizează asemănător apelului unei funcții, specificând lista parametrilor actuali ai cursorului.
  - Asocierea dintre parametrii formali și cei actuali se realizează prin:
    - poziție (parametrii actuali sunt separați prin virgulă, respectând ordinea);
    - nume (parametrii actuali sunt aranjați într-o ordine arbitrară, dar cu o corespondență de forma *parametru formal => parametru actual*).





## 5.2.2. Cursoare parametrizate

- Dacă în definiția cursorului, toți parametrii au valori implicite (*DEFAULT*), cursorul poate fi deschis fără a specifica vreun parametru.
- În determinarea mulțimii active se vor folosi valorile actuale ale parametrilor.

```
OPEN nume_cursor  
[ (valoare_parametru [, valoare_parametru] ...) ];
```





## 5.2.2. Cursoare parametrizate

### ■ Procesarea liniilor unui cursor parametrizat

- Dacă sunt utilizate comenzile de ciclare *LOOP* sau *WHILE*, atunci nu apar modificări de sintaxă.
- Dacă este utilizat un ciclu cursor, atunci se va utiliza:

```
FOR nume_înregistrare IN nume_cursor  
  [(valoare_parametru [, valoare_parametru] ...)] LOOP  
    secvență_de_instrucțiuni;  
END LOOP;
```

### ■ Închiderea unui cursor parametrizat

- Nu apar modificări de sintaxă



## 5.2.2. Cursoare parametrizate

### Exemplul 5.10

```
DECLARE
  CURSOR categ IS SELECT id_categorie, denumire
                    FROM   categorii
                    WHERE  nivel = 5;

  CURSOR prod(v_categ categorii.id_categorie%TYPE)
            IS SELECT MAX(p.denumire), SUM(cantitate)
               FROM   produse p, facturi_produce fp
               WHERE  v_categ = p.id_categorie
               AND    p.id_produc = fp.id_produc
               GROUP BY p.id_produc
               ORDER BY 1,2 desc;

  c_denumire categorii.denumire%TYPE;
  c_id categorii.id_categorie%TYPE;
```



## 5.2.2. Cursoare parametrizate

```
p_denumire produse.denumire%TYPE;  
p_cantitate NUMBER;  
BEGIN  
  OPEN categ;  
  LOOP  
    FETCH categ INTO c_id, c_denumire;  
    EXIT WHEN categ%NOTFOUND;  
    DBMS_OUTPUT.PUT_LINE(c_denumire);  
    DBMS_OUTPUT.PUT_LINE('-----');  
    OPEN prod(c_id);  
    LOOP  
      FETCH prod INTO p_denumire, p_cantitate;  
      EXIT WHEN prod%NOTFOUND OR prod%ROWCOUNT>3;  
      DBMS_OUTPUT.PUT_LINE(prod%ROWCOUNT|| '. ' ||  
                           p_denumire || ' ' || p_cantitate);  
    END LOOP;  
  END LOOP;
```



## 5.2.2. Cursoare parametrizate

```
IF prod%ROWCOUNT = 0 THEN
    DBMS_OUTPUT.PUT_LINE('Nu are produse vandute!');
END IF;
CLOSE prod;
DBMS_OUTPUT.NEW_LINE;
END LOOP;
CLOSE categ;
END;
```

## 5.2.1. Gestiunea cursoarelor explicite



### ■ Output

Mape din carton pentru documente

-----

1. Mape din carton plastifiat cu elastic, albastru 22
2. Mape din carton plastifiat cu elastic, galben 5
3. Mape din carton plastifiat cu elastic, rosu 9

Caiete cu separatoare coperti carton

-----

1. Caiet Exclusive 9

Unitati optice

-----

Nu are produse vandute!

### 5.2.3. Cursoare *SELECT FOR UPDATE*



- Este necesară blocarea liniilor înainte ca acestea să fie șterse sau reactualizate
  - blocarea se poate realiza cu ajutorul clauzei *FOR UPDATE* a comenzii *SELECT* din definiția cursorului
- Cursorul trebuie să fie deschis.

```
CURSOR nume_cursor IS  
    comanda_select  
FOR UPDATE [OF listă_coloane]  
            [NOWAIT | WAIT n | SKIP LOCKED];
```



## 5.2.3. Cursoare *SELECT FOR UPDATE*

### Exemplul 5.11 **explicații**

```
--sesiune 1
SELECT * FROM produse
WHERE id_produs=10 FOR UPDATE;
--commit;

--sesiune 2
SELECT * FROM curs_plsql.produse
WHERE id_produs=10
FOR UPDATE NOWAIT;

SELECT * FROM curs_plsql.produse
WHERE id_produs=1000
FOR UPDATE WAIT 10;
```



### 5.2.3. Cursoare *SELECT FOR UPDATE*



În momentul deschiderii unui cursor *FOR UPDATE*, liniile din mulțimea activă, determinată de clauza *SELECT*, sunt blocate pentru operații de scriere (reactualizare sau ștergere).

- În felul acesta este realizată consistența la citire a sistemului.

### 5.2.3. Cursoare *SELECT FOR UPDATE*



#### **Exemplu de utilizare**



- Se reactualizează o valoare a unei linii și trebuie avută siguranța că linia nu este schimbată de un alt utilizator înaintea reactualizării.
  - Astfel, alte sesiuni nu pot schimba liniile din mulțimea activă până când tranzacția nu este permanentizată sau anulată.

### 5.2.3. Cursoare *SELECT FOR UPDATE*



Comenzile *DELETE/UPDATE* corespunzătoare trebuie să conțină clauza *WHERE CURRENT OF nume\_cursor*.

- clauza referă linia curentă care a fost găsită de cursor
- permite ca reactualizările/ștergerile să se efectueze asupra acestei linii, fără referirea explicită a cheii primare sau pseudocoloanei *ROWID*.

### 5.2.3. Cursoare *SELECT FOR UPDATE*



- ❖ Deoarece cursorul lucrează doar cu niște copii ale liniilor existente în tabele, după închiderea cursorului este necesară comanda *COMMIT* pentru a realiza scrierea efectivă a modificărilor.
- ❖ Deoarece blocările implicate de clauza *FOR UPDATE* vor fi eliberate de comanda *COMMIT*, nu este recomandată utilizarea comenzii *COMMIT* în interiorul ciclului în care se fac încărcări de date.
  - ❖ Orice *FETCH* executat după *COMMIT* va eșua.



## 5.2.3. Cursoare *SELECT FOR UPDATE*

### Exemplul 5.12

```
DECLARE
  CURSOR c IS
    SELECT id_produș
    FROM   produse
    WHERE  id_categorie IN
          (SELECT id_categorie
           FROM   categorii
           WHERE  denumire = 'Placi de rețea Wireless')
    FOR UPDATE OF pret_unitar NOWAIT;
```

### 5.2.3. Cursoare *SELECT FOR UPDATE*



```
BEGIN
  FOR i IN c LOOP
    UPDATE  produse
    SET     pret_unitar = pret_unitar*0.95
    WHERE CURRENT OF c;
  END LOOP;
  -- permanentizare si eliberare blocari
  COMMIT;
END;
```

## 5.2.3. Cursoare *SELECT FOR UPDATE*



### Exemplul 5.13

```
-- utilizare ROWID in loc de CURRENT OF  
DECLARE  
  CURSOR c IS  
    SELECT id_produc, rowid  
    FROM   produse  
    WHERE  id_categorie IN  
           (SELECT id_categorie  
            FROM   categorii  
            WHERE  denumire = 'Placi de retea Wireless')  
    FOR UPDATE OF pret_unitar NOWAIT;
```



## 5.2.3. Cursoare *SELECT FOR UPDATE*



```
BEGIN
  FOR i IN c LOOP
    UPDATE  produse
    SET     pret_unitar = pret_unitar*0.95
    WHERE   ROWID = i.ROWID;
  END LOOP;
  COMMIT;
END;
```



## 5.2.4. Cursoare dinamice

### ■ Cursor static

- este un cursor a cărui comandă *SQL* este cunoscută la momentul compilării blocului
  - toate exemplele explicate anterior

### ■ Variabilă cursor

- este de tip referință (similară tipului *pointer* din limbajele *C* sau *Pascal*)
- un cursor dinamic este un *pointer* la un cursor



## 5.2.4. Cursoare dinamice

### ▣ Variabilele cursor

- ▣ sunt dinamice deoarece li se pot asocia diferite cereri (coloanele obținute de fiecare cerere trebuie să corespundă declarației variabilei cursor)
- ▣ trebuie declarate, deschise, încărcate și închise în mod similar unui cursor static
- ▣ la momentul declarării nu solicită o cerere asociată
- ▣ pot primi valori
- ▣ pot fi utilizate în expresii



## 5.2.4. Cursoare dinamice

### ▣ Variabilele cursor

- ▣ pot fi utilizate ca parametrii în subprograme
  - pot fi utilizate pentru a transmite mulțimea rezultat a unei cereri între subprograme
- ▣ pot fi variabile de legătură
  - pot fi utilizate pentru a transmite mulțimea rezultat a unei cereri între subprograme stocate și diferiți clienți
- ▣ nu acceptă parametrii



## 5.2.4. Cursoare dinamice

```
TYPE  tip_ref_cursor IS REF CURSOR [RETURN tip_returnat];  
var_cursor  tip_ref_cursor;
```

- *tip\_returnat* este un tip înregistrare sau tipul unei linii dintr-un tabel
  - corespunde coloanelor întoarse de către orice cursor asociat variabilelor cursor de tipul definit
  - dacă lipsește clauza *RETURN*, cursorul poate fi deschis pentru orice cerere



## 5.2.4. Cursoare dinamice

### Restricții de utilizare a variabilelor cursor

- Variabilele cursor nu pot fi declarate în specificația unui pachet
  - un pachet nu poate avea definită o variabilă cursor ce poate fi referită din afara pachetului
- valoarea unei variabile cursor nu poate fi stocată într-o colecție sau o coloană a unui tabel
- nu pot fi utilizați operatorii de comparare pentru a testa egalitatea/inegalitatea/valoarea *null* a variabilelor cursor
- nu pot fi folosite cu *SQL dinamic* în *Pro\*C/C++*



## 5.2.4. Cursoare dinamice

### Utilizarea unei variabile cursor

#### ▣ Comanda ***OPEN...FOR***


- ▣ asociază o variabilă cursor cu o cerere
- ▣ execută cererea
- ▣ identifică mulțimea rezultat
- ▣ poziționează cursorul înaintea primei linii din mulțimea rezultat





## 5.2.4. Cursoare dinamice

```
OPEN {variabila_cursor | :variabila_cursor_host}  
FOR {cerere_select |  
    şir_dinamic [USING argument_bind [, argument_bind ...]]};
```

- 
- ❖ Comanda *OPEN...FOR* poate deschide acelaşi cursor pentru diferite cereri.
  - ❖ Nu este necesară închiderea variabilei cursor înainte de a o redeschide.
  - ❖ Dacă se redeschide variabila cursor pentru o nouă cerere, cererea anterioară este pierdută.



## 5.2.4. Cursoare dinamice

### Exemplul 5.14

```
DECLARE
    TYPE tip_cursor IS REF CURSOR RETURN produse%ROWTYPE;
    c tip_cursor;
    v_optiune NUMBER(1) := &p_optiune;
    i produse%ROWTYPE;
BEGIN
    IF v_optiune = 1 THEN
        OPEN c FOR
        SELECT * FROM produse p
        WHERE EXISTS (SELECT 1
                      FROM   facturi_produse pf, facturi f
                      WHERE  p.id_produs = pf.id_produs
                      AND    pf.id_factura = f.id_factura
                      AND    TO_CHAR(data, 'q') = 1);
```



## 5.2.4. Cursoare dinamice

```
ELSIF v_optiune = 2 THEN
    OPEN c FOR
    SELECT *
    FROM produse p
    WHERE id_produc IN
            (SELECT id_produc
             FROM facturi_produce pf, facturi f
             WHERE pf.id_factura = f.id_factura
             AND TO_CHAR(data, 'q') = 2);

ELSIF v_optiune = 3 THEN
    OPEN c FOR
    SELECT DISTINCT p.*
    FROM produse p, facturi_produce pf, facturi f
    WHERE p.id_produc = pf.id_produc
    AND pf.id_factura = f.id_factura
    AND TO_CHAR(data, 'q') = 3;
```



## 5.2.4. Cursoare dinamice

```
ELSE
    OPEN c FOR
    SELECT *
    FROM produse p
    WHERE id_produc IN (SELECT id_produc
                        FROM facturi_produce);
END IF;
LOOP
    FETCH c INTO i;
    EXIT WHEN c%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(i.id_produc||' ' ||i.denumire);
END LOOP;
DBMS_OUTPUT.PUT_LINE('Nr produse vandute: '|| c%ROWCOUNT);
CLOSE c;
END;
```



## 5.2.4. Cursoare dinamice

### Exemplul 5.15 – SQL DINAMIC

```
DECLARE
    TYPE tip_cursor IS REF CURSOR;
    -- ? obtin eroare daca includ RETURN produse%ROWTYPE;
    c tip_cursor;
    v_optiune NUMBER(1) := &p_optiune;
    i produse%ROWTYPE;
BEGIN
    OPEN c FOR
        'SELECT DISTINCT p.*
        FROM   produse p, facturi_produse pf, facturi f
        WHERE  p.id_produs = pf.id_produs
        AND    pf.id_factura = f.id_factura
        AND    TO_CHAR(data, 'q') = :v'
        USING v_optiune;
```



## 5.2.4. Cursoare dinamice

```
LOOP
    FETCH c INTO i;
    EXIT WHEN c%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(i.id_produc||' ' ||i.denumire);
END LOOP;
DBMS_OUTPUT.PUT_LINE('Nr produse vandute: '|| c%ROWCOUNT);
CLOSE c;
END;
```

☐ Obținem același rezultat ca în exemplul anterior?



## 5.2.4. Cursoare dinamice

-- modificarea

```
OPEN c FOR
  'SELECT DISTINCT p.*
   FROM produse p, facturi_produse pf, facturi f
   WHERE p.id_produc = pf.id_produc
   AND    pf.id_factura = f.id_factura
   AND    TO_CHAR(data, 'q') =
           CASE WHEN :v1>3 THEN TO_CHAR(data, 'q')
                ELSE TO_CHAR(:v2)
           END'
  USING v_optiune, v_optiune;
```





## 5.2.4. Cursoare dinamice

### Expresie cursor

- Conceptul este introdus în versiunea *Oracle9i*.
- Întoarce un cursor imbricat (*nested cursor*).

```
CURSOR (subcerere)
```

- Semnificație
  - Fiecare linie din mulțimea rezultat poate conține valori uzuale și cursoare generate de subcereri.



## 5.2.4. Cursoare dinamice

- **Încărcarea cursorului imbricat** se realizează
  - automat atunci când liniile care îl conțin sunt încărcate din cursorul „părinte”.
- **Închiderea cursorului imbricat** are loc
  - dacă este realizată explicit de către utilizator;
  - atunci când cursorul „părinte” este reexecutat sau închis;
  - dacă apare o eroare în timpul unei încărcări din cursorul „părinte”.



## 5.2.4. Cursoare dinamice

### Exemplul 5.16

```
DECLARE
  CURSOR categ IS
    SELECT denumire,
      CURSOR(SELECT MAX(denumire)
        FROM   produse p, facturi_produce fp
        WHERE  c.id_categorie = p.id_categorie
        AND    p.id_produc = fp.id_produc
        GROUP BY p.id_produc
        ORDER BY 1, SUM(cantitate) desc)
  FROM   categorii c
  WHERE  nivel = 5;
```



## 5.2.4. Cursoare dinamice

```
c_denumire categorii.denumire%TYPE;  
v_cursor SYS_REFCURSOR;  
TYPE tab_prod IS TABLE OF produse.denumire%TYPE  
                INDEX BY BINARY_INTEGER;  
v_prod tab_prod;  
BEGIN  
  OPEN categ;  
  LOOP  
    FETCH categ INTO c_denumire, v_cursor;  
    EXIT WHEN categ%NOTFOUND;  
    DBMS_OUTPUT.PUT_LINE(c_denumire);  
    DBMS_OUTPUT.PUT_LINE('-----');  
    FETCH v_cursor BULK COLLECT INTO v_prod LIMIT 3;
```



## 5.2.4. Cursoare dinamice

```
IF v_prod.COUNT = 0 THEN
    DBMS_OUTPUT.PUT_LINE('Nu are produse vandute!');
ELSE
    FOR i IN v_prod.FIRST..v_prod.LAST LOOP
        DBMS_OUTPUT.PUT_LINE(i || ' ' || v_prod(i));
    END LOOP;
END IF;
DBMS_OUTPUT.NEW_LINE;
END LOOP;
CLOSE categ;
END;
```



## Bibliografie

- Connolly T.M., Begg C.E., *Database Systems: A Practical Approach to Design, Implementation and Management*, 5th edition, Pearson Education, 2005
- Dollinger R., Andron L. – *Baze de date și gestiunea tranzacțiilor*, Editura Albastră, Cluj-Napoca, 2004
- Oracle and/or its affiliates, *Oracle Database Concepts*, 1993, 2017
- Oracle and/or its affiliates, *Oracle Database Performance Tuning Guide*, 2013, 2017
- Oracle and/or its affiliates, *Oracle Database SQL Language Reference*, 1996, 2017
- Oracle and/or its affiliates, *Oracle Database PL/SQL Language Reference*, 1996, 2017
- Oracle and/or its affiliates, *Oracle Database Administrator's Guide*, 2001, 2010
- Oracle and/or its affiliates, *Pro\*C/C++ Programmer's Guide*, 1996, 2014
- Oracle University, *Oracle Database 11g: PL/SQL Fundamentals, Student Guide*, 2009
- Popescu I., Alecu A., Velcescu L., Florea (Mihai) G., *Programare avansată în Oracle9i*, Ed. Tehnică, 2004