

Gestiunea cursorurilor în *PL/SQL*

Sistemul *Oracle* folosește, pentru a procesa o comandă *SQL*, o zonă de memorie cunoscută sub numele de zonă context (*context area*). Când este procesată o instrucțiune *SQL*, *server-ul Oracle* deschide această zonă de memorie în care comanda este analizată sintactic și este executată.

Zona conține informații necesare procesării comenzii, cum ar fi:

- numărul de rânduri procesate de instrucțiune;
- un *pointer* către reprezentarea internă a comenzii;
- în cazul unei cereri, mulțimea rândurilor rezultate în urma execuției acestei comenzi (*active set*).

Un cursor este un *pointer* la această zonă context. Prin intermediul cursorurilor, un program *PL/SQL* poate controla zona context și transformările petrecute în urma procesării comenzii.

Există două tipuri de cursoruri:

- implicite, generate de *server-ul Oracle* când în partea executabilă a unui bloc *PL/SQL* apare o instrucțiune *SQL*;
- explicite, declarate și definite de către utilizator atunci când o cerere (*SELECT*), care apare într-un bloc *PL/SQL*, întoarce mai multe linii ca rezultat.

Atât cursorurile implicite cât și cele explicite au o serie de atribute ale căror valori pot fi folosite în expresii. Lista atributelor este următoarea:

- *%ROWCOUNT*, care este de tip întreg și reprezintă numărul liniilor încărcate de cursor;
- *%FOUND*, care este de tip boolean și ia valoarea *TRUE* dacă ultima operație de încărcare (*FETCH*) dintr-un cursor a avut succes (în cazul cursorurilor explicite) sau dacă instrucțiunea *SQL* a întors cel puțin o linie (în cazul cursorurilor implicite);
- *%NOTFOUND*, care este de tip boolean și are semnificație opusă față de cea a atributului *%FOUND*;
- *%ISOPEN*, care este de tip boolean și indică dacă un cursor este deschis (în cazul cursorurilor implicite, acest atribut are întotdeauna valoarea *FALSE*, deoarece un cursor implicit este închis de sistem imediat după executarea instrucțiunii *SQL* asociate).

Atributele pot fi referite prin expresia *SQL%nume_atribut*, în cazul cursorurilor implicite, sau prin *nume_cursor%nume_atribut*, în cazul unui cursor explicit. Ele pot să apară în comenzi *PL/SQL*, în funcții, în secțiunea de tratare a erorilor, dar nu pot fi utilizate în comenzi *SQL*.

Cursoare implicite

Când se procesează o comandă *LMD*, motorul *SQL* deschide un cursor implicit. Atributele scalare ale cursorului implicit (*SQL%ROWCOUNT*, *SQL%FOUND*, *SQL%NOTFOUND*, *SQL%ISOPEN*) furnizează informații referitoare la ultima comandă *INSERT*, *UPDATE*, *DELETE* sau *SELECT INTO* executată. Înainte ca *Oracle* să deschidă cursorul *SQL* implicit, atributele acestuia au valoarea *null*.

În *Oracle9i*, pentru cursoare implicite a fost introdus atributul compus *%BULK_ROWCOUNT*, care este asociat comenzii *FORALL*. Atributul are semantica unui tablou indexat. Componenta *%BULK_ROWCOUNT(j)* conține numărul de linii procesate de a *j*-a execuție a unei comenzi *INSERT*, *DELETE* sau *UPDATE*. Dacă a *j*-a execuție nu afectează nici o linie, atunci atributul returnează valoarea 0. Comanda *FORALL* și atributul *%BULK_ROWCOUNT* au aceeași indici, deci folosesc același domeniu. Dacă *%BULK_ROWCOUNT(j)* este zero, atributul *%FOUND* este *FALSE*.

Exemplu:

În exemplul care urmează, comanda *FORALL* inserează un număr arbitrar de linii la fiecare iterație, iar după fiecare iterație atributul *%BULK_ROWCOUNT* returnează numărul acestor linii inserate.

```
SET SERVEROUTPUT ON
DECLARE
  TYPE alfa IS TABLE OF NUMBER;
  beta alfa;
BEGIN
  SELECT cod_artist BULK COLLECT INTO beta FROM artist;
  FORALL j IN 1..beta.COUNT
    INSERT INTO tab_art
      SELECT cod_artist,cod_opera
      FROM   opera
      WHERE  cod_artist = beta(j);
  FOR j IN 1..beta.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE ('Pentru artistul avand codul ' ||
      beta(j) || ' au fost inserate ' ||
      SQL%BULK_ROWCOUNT(j)
      || inregistrari (opere de arta)');
  END LOOP;
  DBMS_OUTPUT.PUT_LINE ('Numarul total de inregistrari
    inserate este ' || SQL%ROWCOUNT);
END;
/
SET SERVEROUTPUT OFF
```

Cursoare explicite

Pentru gestiunea cursoroarelor explicite sunt necesare următoarele etape:

- declararea cursorului (atribuirea unui nume și asocierea cu o comandă *SELECT*);
- deschiderea cursorului pentru cerere (executarea interogării asociate și determinarea mulțimii rezultat);
- recuperarea liniilor rezultatului în variabile *PL/SQL*;
- închiderea cursorului (eliberarea resurselor relative la cursor).

Prin urmare, pentru a utiliza un cursor, el trebuie declarat în secțiunea declarativă a programului, trebuie deschis în partea executabilă, urmând să fie utilizat apoi pentru extragerea datelor. Dacă nu mai este necesar în restul programului, cursorul trebuie să fie închis.

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

Pentru a controla activitatea unui cursor sunt utilizate comenzile *DECLARE*, *OPEN*, *FETCH* și *CLOSE*.

Declararea unui cursor explicit

Prin declarația *CURSOR* în cadrul comenzii *DECLARE* este definit un cursor explicit și este precizată structura cererii care va fi asociată acestuia.

Declarația *CURSOR* are următoarea formă sintactică:

CURSOR *nume_cursor* **IS** *comanda_select*

Identificatorul *nume_cursor* este numele cursorului, iar *comanda_select* este cererea *SELECT* care va fi procesată.

Observații:

- Comanda *SELECT* care apare în declararea cursorului, nu trebuie să includă clauza *INTO*.
- Dacă se cere procesarea liniilor într-o anumită ordine, atunci în cerere este utilizată clauza *ORDER BY*.

- Variabilele care sunt referite în comanda de selectare trebuie declarate înaintea comenzii *CURSOR*. Ele sunt considerate variabile de legătură.
- Dacă în lista comenzii *SELECT* apare o expresie, atunci pentru expresia respectivă trebuie utilizat un *alias*, iar câmpul expresie se va referi prin acest *alias*.
- Numele cursorului este un identificator unic în cadrul blocului, care nu poate să apară într-o expresie și căruia nu i se poate atribui o valoare.

Deschiderea unui cursor explicit

Comanda *OPEN* execută cererea asociată cursorului, identifică mulțimea liniilor rezultat și poziționează cursorul înaintea primei linii.

Deschiderea unui cursor se face prin comanda:

OPEN *nume_cursor*;

Identificatorul *nume_cursor* reprezintă numele cursorului ce va fi deschis.

La deschiderea unui cursor se realizează următoarele operații:

- se evaluează cererea asociată (sunt examinate valorile variabilelor de legătură ce apar în declarația cursorului);
- este determinată mulțimea rezultat (*active set*) prin executarea cererii *SELECT*, având în vedere valorile de la pasul anterior;
- *pointer*-ul este poziționat la prima linie din mulțimea activă.

Încărcarea datelor dintr-un cursor explicit

Comanda *FETCH* regăsește liniile rezultatului din mulțimea activă. *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 spre ultima înregistrare);
- citește datele liniei curente în variabile *PL/SQL*;
- dacă *pointer*-ul este poziționat la sfârșitul mulțimii active atunci se iese din bucla cursorului.

Comanda *FETCH* are următoarea sintaxă:

FETCH *nume_cursor*

INTO {*nume_variabilă* [, *nume_variabilă*] ... / *nume_înregistrare*};

Identificatorul *nume_cursor* reprezintă numele unui cursor declarat și deschis anterior. Variabila sau lista de variabile din clauza *INTO* trebuie să fie compatibilă (ca ordine și tip) cu lista selectată din cererea asociată cursorului.

La un moment dat, comanda *FETCH* regăsește o singură linie. Totuși, în ultimele versiuni *Oracle* pot fi încărcate mai multe linii (la un moment dat) într-o colecție, utilizând clauza *BULK COLLECT*.

Exemplu:

În exemplul care urmează se încarcă date dintr-un cursor în două colecții.

```
DECLARE
  TYPE    ccopera IS TABLE OF opera.cod_opera%TYPE;
  TYPE    ctopera IS TABLE OF opera.titlu%TYPE;
  cod1    ccopera;
  titlu1  ctopera;
  CURSOR  alfa IS SELECT cod_opera, titlu
                  FROM    opera
                  WHERE   stil = 'impresionism';

BEGIN
  OPEN alfa;
  FETCH alfa BULK COLLECT INTO cod1, titlu1;
  ...
  CLOSE alfa;
END;
```

Închiderea unui cursor explicit

După ce a fost procesată mulțimea activă, cursorul trebuie închis. Prin această operație, *PL/SQL* este informat că programul a terminat folosirea cursorului și resursele asociate acestuia pot fi eliberate. Aceste resurse includ spațiul utilizat pentru memorarea mulțimii active și spațiul temporar folosit pentru determinarea mulțimii active.

Cursorul va fi închis prin comanda *CLOSE*, care are următoarea sintaxă:

CLOSE *nume_cursor*;

Identificatorul *nume_cursor* este numele unui cursor deschis anterior.

Pentru a reutiliza cursorul este suficient ca acesta să fie redeschis. Dacă se încearcă încărcarea datelor dintr-un cursor închis, atunci apare excepția *INVALID_CURSOR*. Un bloc *PL/SQL* poate să se termine fără a închide cursoarele, dar acest lucru nu este indicat, deoarece este bine ca resursele să fie eliberate.

Exemplu:

Pentru toți artiștii care au opere de artă expuse în muzeu să se insereze în tabelul *temp* informații referitoare la numele acestora și anul nașterii.

```
DECLARE
  v_nume      artist.nume%TYPE;
  v_an_nas    artist.an_nastere%TYPE;
  CURSOR info IS
    SELECT DISTINCT nume, an_nastere
    FROM    artist;

BEGIN
```

```

OPEN info;
LOOP
    FETCH info INTO v_nume, v_an_nas;
    EXIT WHEN info%NOTFOUND;
    INSERT INTO temp
    VALUES (v_nume || TO_CHAR(v_an_nas));
END LOOP;
CLOSE info;
COMMIT;
END;

```

Valorile atributelor unui cursor explicit sunt prezentate în următorul tabel:

		<i>%FOUND</i>	<i>%ISOPEN</i>	<i>%NOTFOU ND</i>	<i>%ROWCOUNT</i>
OPEN	Înainte	Excepție	<i>False</i>	Excepție	Excepție
	După	<i>Null</i>	<i>True</i>	<i>Null</i>	0
Prima Încărcare	Înainte	<i>Null</i>	<i>True</i>	<i>Null</i>	0
	După	<i>True</i>	<i>True</i>	<i>False</i>	1
Următoarea încărcare	Înainte	<i>True</i>	<i>True</i>	<i>False</i>	1
	După	<i>True</i>	<i>True</i>	<i>False</i>	Depinde de date
Ultima încărcare	Înainte	<i>True</i>	<i>True</i>	<i>False</i>	Depinde de date
	După	<i>False</i>	<i>True</i>	<i>True</i>	Depinde de date
CLOSE	Înainte	<i>False</i>	<i>True</i>	<i>True</i>	Depinde de date
	După	Excepție	<i>False</i>	Excepție	Excepție

După prima încărcare, dacă mulțimea rezultat este vidă, *%FOUND* va fi *FALSE*, *%NOTFOUND* va fi *TRUE*, iar *%ROWCOUNT* este 0.

Într-un pachet poate fi separată specificarea unui cursor de corpul acestuia. Cursorul va fi declarat în specificația pachetului prin comanda:

```

CURSOR nume_cursor [ (parametru [, parametru]...) ]
RETURN tip_returnat;

```

În felul acesta va crește flexibilitatea programului, putând fi modificat doar corpul cursorului, fără a schimba specificația.

Exemplu:

```

CREATE PACKAGE exemplu AS
    CURSOR alfa (p_valoare_min NUMBER) RETURN opera%ROWTYPE;
    -- declaratie specificatie cursor
    ...
END exemplu;

```

```

CREATE PACKAGE BODY exemplu AS
  CURSOR alfa (p_valoare_min  NUMBER) RETURN opera%ROWTYPE
IS
  SELECT * FROM opera WHERE valoare > p_valoare_min;
  -- definire corp cursor
  ...
END exemplu;

```

Procesarea liniilor unui cursor explicit

Pentru procesarea diferitelor linii ale unui cursor explicit se folosește operația de ciclare (*LOOP*, *WHILE*, *FOR*), prin care la fiecare iterație se va încărca o nouă linie. Comanda *EXIT* poate fi utilizată pentru ieșirea din ciclu, iar valoarea atributului *%ROWCOUNT* pentru terminarea ciclului.

Procesarea liniilor unui cursor explicit se poate realiza și cu ajutorul unui ciclu *FOR* special, numit ciclu cursor. Pentru acest ciclu este necesară doar declararea cursorului, operațiile de deschidere, încărcare și închidere ale acestuia fiind implicite.

Comanda are următoarea sintaxă:

```

FOR nume_înregistrare IN nume_cursor LOOP
    secvență_de_instrucțiuni;
END LOOP;

```

Variabila *nume_înregistrare* (care controlează ciclul) nu trebuie declarată. Domeniul ei este doar ciclul respectiv.

Pot fi utilizate cicluri cursor speciale care folosesc subcereri, iar în acest caz nu mai este necesară nici declararea cursorului. Exemplul care urmează este concludent în acest sens.

Exemplu:

Să se calculeze, utilizând un ciclu cursor cu subcereri, valoarea operelor de artă expuse într-o galerie al cărei cod este introdus de la tastatură. De asemenea, să se obțină media valorilor operelor de artă expuse în galeria respectivă.

```

SET SERVEROUTPUT ON
ACCEPT p_galerie PROMPT 'Dati codul galeriei:'
DECLARE
  v_cod_galerie  galerie.cod_galerie%TYPE:=&p_galerie;
  val            NUMBER;
  media         NUMBER;
  i             INTEGER;
BEGIN
  val:=0;
  i:=0;
  FOR numar_opera IN

```

```

        (SELECT  cod_opera, valoare
          FROM    opera
          WHERE    cod_galerie = v_cod_galerie) LOOP
    val := val + numar_opera.valoare;
    i := i+1;
END LOOP;--închidere implicită
DBMS_OUTPUT.PUT_LINE('Valoarea operelor de arta din
galeria cu numarul ' || TO_CHAR(v_cod_galerie) || '
este ' || TO_CHAR(val));
IF i=0 THEN
    DBMS_OUTPUT.PUT_LINE('Galeria nu are opere de arta');
ELSE
    media := val/i;
    DBMS_OUTPUT.PUT_LINE('Media valorilor operelor de arta
    din galeria cu numarul ' || TO_CHAR(v_cod_galerie)
    || ' este ' || TO_CHAR(media));
END IF;
END;
/
SET SERVEROUTPUT OFF

```

Cursoare parametrizate

Unei variabile de tip cursor îi corespunde o comandă *SELECT*, care nu poate fi schimbată pe parcursul programului. Pentru a putea lucra cu niște cursoare ale căror comenzi *SELECT* atașate depind de parametri ce pot fi modificați la momentul execuției, în *PL/SQL* s-a introdus noțiunea de cursor parametrizat. Prin urmare, un cursor parametrizat este un cursor în care comanda *SELECT* atașată depinde de unul sau mai mulți parametri.

Transmiterea de parametri unui cursor parametrizat se face în mod similar procedurilor stocate. Un astfel de cursor este mult mai ușor de interpretat și de întreținut, oferind și posibilitatea reutilizării sale în blocul *PL/SQL*.

Declararea unui astfel de cursor se face respectând următoarea sintaxă:

```

CURSOR nume_cursor [ (nume_parametru[, nume_parametru ...] ) ]
    [RETURN tip_returnat]
    IS comanda_select;

```

Identificatorul *comanda_select* este o instrucțiune *SELECT* fără clauza *INTO*, *tip_returnat* reprezintă un tip înregistrare sau linie de tabel, iar *nume_parametru* are sintaxa:

```

nume_parametru [IN] tip_parametru [ {:= | DEFAULT} expresie]

```

În această declarație, atributul *tip_parametru* reprezintă tipul parametrului,

care este un tip scalar. Parametrii formali sunt de tip *IN* și, prin urmare, nu pot returna valori parametrilor actuali. Ei nu suportă constrângerea *NOT NULL*.

Deschiderea unui astfel de cursor se face asemănător apelului unei funcții, specificând lista parametrilor actuali ai cursorului. În determinarea mulțimii active se vor folosi valorile actuale ale acestor parametri.

Sintaxa pentru deschiderea unui cursor parametrizat este:

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

Parametrii sunt specificați similar celor de la subprograme. Asocierea dintre parametrii formali și cei actuali se face prin:

- poziție – parametrii formali și actuali sunt separați prin virgulă;
- nume – parametrii actuali sunt aranjați într-o ordine arbitrară, dar cu o corespondență de forma *parametru formal => parametru actual*.

Dacă în definiția cursorului, toți parametrii au valori implicite (*DEFAULT*), cursorul poate fi deschis fără a specifica vreun parametru.

Exemplu:

Utilizând un cursor parametrizat să se obțină codurile operelor de artă din fiecare sală, identificatorul sălii și al galeriei. Rezultatele să fie inserate în tabelul *mesaje*.

```
DECLARE
  v_cod_sala      sala.cod_sala%TYPE;
  v_cod_galerie   galerie.cod_galerie%TYPE;
  v_car           VARCHAR2(75);
  CURSOR sala_cursor IS
    SELECT  cod_sala, cod_galerie
    FROM    sala;
  CURSOR ope_cursor (v_id_sala NUMBER, v_id_galerie NUMBER) IS
    SELECT  cod_opera || cod_sala || cod_galerie
    FROM    opera
    WHERE   cod_sala = v_id_sala
    AND     cod_galerie = v_id_galerie;
BEGIN
  OPEN sala_cursor;
  LOOP
    FETCH sala_cursor INTO v_cod_sala, v_cod_galerie;
    EXIT WHEN sala_cursor%NOTFOUND;
    IF ope_cursor%ISOPEN THEN
      CLOSE ope_cursor;
    END IF;
    OPEN ope_cursor (v_cod_sala, v_cod_galerie);
    LOOP
      FETCH ope_cursor INTO v_car;
      EXIT WHEN ope_cursor%NOTFOUND;
      INSERT INTO mesaje (rezultat)
```

```

VALUES (v_car);
END LOOP;
CLOSE ope_cursor;
END LOOP;
CLOSE sala_cursor;
COMMIT;
END;

```

Cursoare *SELECT FOR UPDATE*

Uneori este necesară blocarea liniilor înainte ca acestea să fie șterse sau reactualizate. Blocarea se poate realiza (atunci când cursorul este deschis) cu ajutorul comenzii *SELECT* care conține clauza *FOR UPDATE*.

Declararea unui astfel de cursor se face conform sintaxei:

```

CURSOR nume_cursor IS
    comanda_select
    FOR UPDATE [OF lista_câmpuri] [NOWAIT];

```

Identificatorul *lista_câmpuri* este o listă ce include câmpurile tabelului care vor fi modificate. Atributul *NOWAIT* returnează o eroare dacă liniile sunt deja blocate de altă sesiune. Liniile unui tabel sunt blocate doar dacă clauza *FOR UPDATE* se referă la coloane ale tabelului respectiv.

În momentul deschiderii unui astfel de cursor, liniile corespunzătoare mulțimii active, determinate de clauza *SELECT*, sunt blocate pentru operații de scriere (reactualizare sau ștergere). În felul acesta este realizată consistența la citire a sistemului. De exemplu, această situație este utilă când se reactualizează o valoare a unei linii și trebuie avută siguranța că linia nu este schimbată de alt utilizator înaintea reactualizării. Prin urmare, alte sesiuni nu pot schimba liniile din mulțimea activă până când tranzacția nu este permanentizată sau anulată. Dacă altă sesiune a blocat deja liniile din mulțimea activă, atunci comanda *SELECT ... FOR UPDATE* va aștepta (sau nu) ca aceste blocări să fie eliberate. Pentru a trata această situație se utilizează clauza *WAIT*, respectiv *NOWAIT*.

În *Oracle9i* este utilizată sintaxa:

```

SELECT ... FROM ... FOR UPDATE [OF lista_campuri]
    [ { WAIT n / NOWAIT } ];

```

Valoarea lui *n* reprezintă numărul de secunde de așteptare. Dacă liniile nu sunt deblocate în *n* secunde, atunci se declanșează eroarea *ORA-30006*, respectiv eroarea *ORA-00054*, după cum este specificată clauza *WAIT*, respectiv *NOWAIT*. Dacă nu este specificată nici una din clauzele *WAIT* sau *NOWAIT*, sistemul așteaptă până ce linia este deblocată și atunci returnează rezultatul comenzii *SELECT*.

Dacă un cursor este declarat cu clauza *FOR UPDATE*, atunci comenzile

DELETE și *UPDATE* corespunzătoare trebuie să conțină clauza *WHERE CURRENT OF nume_cursor*.

Această clauză referă linia curentă care a fost găsită de cursor, permițând ca reactualizările și ștergerile să se efectueze asupra acestei linii, fără referirea explicită a cheii primare sau pseudocoloanei *ROWID*. De subliniat că instrucțiunile *UPDATE* și *DELETE* vor reactualiza numai coloanele listate în clauza *FOR UPDATE*.

Pseudocoloana *ROWID* poate fi utilizată dacă tabelul referit în interogare nu are o cheie primară specificată. *ROWID*-ul fiecărei linii poate fi încărcat într-o variabilă *PL/SQL* (declarată de tipul *ROWID* sau *UROWID*), iar această variabilă poate fi utilizată în clauza *WHERE* (*WHERE ROWID = v_rowid*).

După închiderea cursorului este necesară comanda *COMMIT* pentru a realiza scrierea efectivă a modificărilor, deoarece cursorul lucrează doar cu niște copii ale liniilor reale existente în tabele.

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. În cazul în care cursorul nu este definit prin *SELECT...FOR UPDATE*, nu sunt probleme în acest sens și, prin urmare, în interiorul ciclului unde se fac schimbări ale datelor poate fi utilizat un *COMMIT*.

Exemplu:

Să se dubleze valoarea operelor de artă pictate pe pânză care au fost achiziționate înainte de 1 ianuarie 1956.

```
DECLARE
  CURSOR calc IS
    SELECT *
    FROM   opera
    WHERE  material = 'panza'
    AND    data_achizitie <= TO_DATE('01-JAN-56', 'DD-MON-YY')
    FOR UPDATE OF valoare NOWAIT;
BEGIN
  FOR x IN calc LOOP
    UPDATE opera
    SET    valoare = valoare*2
    WHERE CURRENT OF calc;
  END LOOP;
  -- se permanentizeaza actiunea si se elibereaza blocarea
  COMMIT;
END;
```

Cursoare dinamice

Toate exemplele considerate anterior se referă la cursoare statice. Unui cursor static i se asociază o comandă *SQL* care este cunoscută în momentul în care blocul este compilat.

În *PL/SQL* a fost introdusă variabila cursor, care este de tip referință. Variabilele cursor sunt similare tipului *pointer* din limbajele *C* sau *Pascal*. Prin urmare, un cursor este un obiect static, iar un cursor dinamic este un *pointer* la un cursor.

În momentul declarării, variabilele cursor nu solicită o comandă *SQL* asociată. În acest fel, diferite comenzi *SQL* pot fi asociate variabilelor cursor, la diferite momente de timp. Acest tip de variabilă trebuie declarată, deschisă, încărcată și închisă în mod similar unui cursor static.

Variabilele cursor sunt dinamice deoarece li se pot asocia diferite interogări atâta timp cât coloanele returnate de fiecare interogare corespund declarației variabilei cursor.

Aceste variabile sunt utile în transmiterea seturilor de rezultate între subprograme *PL/SQL* stocate și diferiți clienți. De exemplu, un *client OCI*, o aplicație *Oracle Forms* și *server-ul Oracle* pot referi aceeași zonă de lucru (care conține mulțimea rezultat). Pentru a reduce traficul în rețea, o variabilă cursor poate fi declarată pe stația *client*, deschisă și se pot încărca date din ea pe *server*, apoi poate continua încărcarea, dar de pe stația *client* etc.

Pentru a crea o variabilă cursor este necesară definirea unui tip *REF CURSOR*, urmând apoi declararea unei variabile de tipul respectiv. După ce variabila cursor a fost declarată, ea poate fi deschisă pentru orice cerere *SQL* care returnează date de tipul declarat.

Sintaxa pentru declararea variabilei cursor este următoarea:

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

Identificatorul *var_cursor* este numele variabilei cursor, *tip_ref_cursor* este un nou tip de dată ce poate fi utilizat în declarațiile următoare ale variabilelor cursor, iar *tip_returnat* este un tip înregistrare sau tipul unei linii dintr-un tabel al bazei. Acest tip corespunde coloanelor returnate de către orice cursor asociat variabilelor cursor de tipul definit. Dacă lipsește clauza *RETURN*, cursorul poate fi deschis pentru orice cerere *SELECT*.

Dacă variabila cursor apare ca parametru într-un subprogram, atunci trebuie specificat tipul parametrului (tipul *REF CURSOR*) și forma acestuia (*IN* sau *IN OUT*).

Există anumite restricții referitoare la utilizarea variabilelor cursor:

- nu pot fi declarate într-un pachet;

- cererea asociată variabilei cursor nu poate include clauza *FOR UPDATE* (restricția dispăre în *Oracle9i*);
- nu poate fi asignată valoarea *null* unei variabile cursor;
- nu poate fi utilizat tipul *REF CURSOR* pentru a specifica tipul unei coloane în comanda *CREATE TABLE*;
- nu pot fi utilizați operatorii de comparare pentru a testa egalitatea, inegalitatea sau valoarea *null* a variabilelor cursor;
- nu poate fi utilizat tipul *REF CURSOR* pentru a specifica tipul elementelor unei colecții (*varray*, *nested table*);
- nu pot fi folosite cu *SQL* dinamic în *Pro*C/C++*.

În cazul variabilelor cursor, instrucțiunile de deschidere (*OPEN*), încărcare (*FETCH*), închidere (*CLOSE*) vor avea o sintaxă similară celor comentate anterior.

Comanda *OPEN...FOR* asociază o variabilă cursor cu o cerere multilinie, execută cererea, identifică mulțimea rezultat și poziționează cursorul la prima linie din mulțimea rezultat. Sintaxa comenzii este:

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

Identificatorul *variabila_cursor* specifică o variabilă cursor declarată anterior, dar fără opțiunea *RETURN tip*, *cerere_select* este interogarea pentru care este deschisă variabila cursor, iar *șir_dinamic* este o secvență de caractere care reprezintă cererea multilinie.

Opțiunea *șir_dinamic* este specifică prelucrării dinamice a comenzilor, iar posibilitățile oferite de *SQL* dinamic vor fi analizate într-un capitol separat. Identificatorul *:variabila_cursor_host* reprezintă o variabilă cursor declarată într-un mediu gazdă *PL/SQL* (de exemplu, un program *OCI*).

Comanda *OPEN - FOR* poate deschide același 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.

Exemplu:

```
CREATE OR REPLACE PACKAGE alfa AS
  TYPE ope_tip IS REF CURSOR RETURN opera%ROWTYPE;
  PROCEDURE deschis_ope (ope_var IN OUT ope_tip,
                        alege IN NUMBER);
END alfa;

CREATE OR REPLACE PACKAGE BODY alfa AS
  PROCEDURE deschis_ope (ope_var IN OUT ope_tip,
```

```

                                alege IN NUMBER) IS
BEGIN
  IF alege = 1 THEN
    OPEN ope_var FOR SELECT * FROM opera;
  ELSIF alege = 2 THEN
    OPEN ope_var FOR SELECT * FROM opera WHERE valoare>2000;
  ELSIF alege = 3 THEN
    OPEN ope_var FOR SELECT * FROM opera WHERE valoare=7777;
  END IF;
  END deschis_ope;
END alfa;

```

Exemplu:

În următorul exemplu se declară o variabilă cursor care se asociază unei comenzi *SELECT* (*SQL* dinamic) ce returnează anumite linii din tabelul *opera*.

```

DECLARE
  TYPE operaref IS REF CURSOR;
  opera_var operaref;
  mm_val INTEGER := 100000;
BEGIN
  OPEN opera_var FOR
    'SELECT cod_opera, valoare FROM opera WHERE valoare> :vv'
    USING mm_val;
  ...
END;

```

Comanda *FETCH* returnează o linie din mulțimea rezultat a cererii multi-linie, atribuie valori componentelor din lista cererii prin clauza *INTO*, avansează cursorul la următoarea linie. Sintaxa comenzii este:

```

FETCH {variabila_cursor / :variabila_cursor_host}
INTO {variabila [, variabila]... / înregistrare}
[BULK COLLECT INTO {nume_colecție [, nume_colecție]...} |
                                {nume_array_host [, nume_array_host]...}
] [LIMIT expresie_numerica]];

```

Clauza *BULK COLLECT* permite încărcarea tuturor liniilor simultan în una sau mai multe colecții. Atributul *nume_colecție* indică o colecție declarată anterior, în care sunt depuse valorile respective, iar *nume_array_host* identifică un vector declarat într-un mediu gazdă *PL/SQL* și trimis lui *PL/SQL* ca variabilă de legătură. Prin clauza *LIMIT* se limitează numărul liniilor încărcate din baza de date.

Exemplu:

```

DECLARE
    TYPE alfa IS REF CURSOR RETURN opera%ROWTYPE;
    TYPE beta IS TABLE OF opera.titlu%TYPE;
    TYPE gama IS TABLE OF opera.valoare%TYPE;
    var1 alfa;
    var2 beta;
    var3 gama;
BEGIN
    OPEN alfa FOR SELECT titlu, valoare FROM opera;
    FETCH var1 BULK COLLECT INTO var2, var3;
    ...
    CLOSE var1;
END;
```

Comanda *CLOSE* dezactivează variabila cursor precizată. Ea are sintaxa:

CLOSE {variabila_cursor / :variabila_cursor_host}

Cursoarele și variabilele cursor nu sunt interoperabile. Nu poate fi folosită una din ele, când este așteptată cealaltă. Următoarea secvență este incorectă.

```

DECLARE
    TYPE beta IS REF CURSOR RETURN opera%ROWTYPE;
    gama beta;
BEGIN
    FOR k IN gama LOOP --nu este corect!
    ...
END;
```

Expresie cursor

În *Oracle9i* a fost introdus conceptul de expresie cursor (*cursor expression*), care returnează un cursor imbricat (*nested cursor*).

Expresia cursor are următoarea sintaxă:

CURSOR (subcerere)

Fiecare linie din mulțimea rezultat poate conține valori uzuale și cursoare generate de subcereri. *PL/SQL* acceptă cereri care au expresii cursor în cadrul unei declarații cursor, declarații *REF CURSOR* și a variabilelor cursor.

Prin urmare, expresia cursor poate să apară într-o comandă *SELECT* ce este utilizată pentru deschiderea unui cursor dinamic. De asemenea, expresiile cursor pot fi folosite în cereri *SQL* dinamice sau ca parametri actuali într-un subprogram.

Un cursor imbricat este încărcat automat atunci când liniile care îl conțin sunt încărcate din cursorul „părinte“. El este închis dacă:

- este închis explicit de către utilizator;
- cursorul „părinte“ este reexecutat, închis sau anulat;
- apare o eroare în timpul unei încărcări din cursorul „părinte“.

Există câteva restricții asupra folosirii unei expresii cursor:

- nu poate fi utilizată cu un cursor implicit;
- poate să apară numai într-o comandă *SELECT* care nu este imbricată în altă cerere (exceptând cazul în care este o subcerere chiar a expresiei cursor) sau ca argument pentru funcții tabel, în clauza *FROM* a lui *SELECT*;
- nu poate să apară în interogarea ce definește o vizualizare;
- nu se pot efectua operații *BIND* sau *EXECUTE* cu aceste expresii.

Exemplu:

Să se definească un cursor care furnizează codurile operelor expuse în cadrul unei expoziții având un cod specificat (*val_cod*) și care se desfășoară într-o localitate precizată (*val_oras*). Să se afișeze data când a avut loc vernisajul acestei expoziții.

În acest caz cursorul returnează două coloane, cea de-a doua coloană fiind un cursor imbricat.

```
CURSOR alfa (val_cod NUMBER, val_oras VARCHAR2(20)) IS
  SELECT l.datai,
         CURSOR (SELECT d.cod_expo,
                        CURSOR (SELECT f.cod_opera
                                FROM   figureaza_in f
                                WHERE  f.cod_expo=d.cod_expo) AS xx
                        FROM   expozitie d
                        WHERE  l.cod_expo = d.cod_expo) AS yy
  FROM    locped l
  WHERE   cod_expo = val_cod AND nume_oras= val_oras;
```

Exemplu:

Să se listeze numele galeriilor din muzeu și pentru fiecare galerie să se afișeze numele sălilor din galeria respectivă.

Sunt prezentate două variante de rezolvare. Prima variantă reprezintă o implementare simplă utilizând programarea secvențială clasică, iar a doua utilizează expresii cursor pentru rezolvarea acestei probleme.

Varianta 1:

```
BEGIN
  FOR gal IN (SELECT cod_galerie, nume_galerie
              FROM   galerie)
  LOOP
```



```

DBMS_OUTPUT.PUT_LINE (gal.ume_galerie);
FOR sal IN (SELECT cod_sala, ume_sala
             FROM    sala
             WHERE   cod_galerie = gal.cod.galerie)
LOOP
    DBMS_OUTPUT.PUT_LINE (sal.ume_sala);
END LOOP;
END LOOP;
END;

```

Varianta 2:

```

DECLARE
    CURSOR c_gal IS
        SELECT ume_galerie,
               CURSOR (SELECT ume_sala
                       FROM    sala s
                       WHERE   s.cod_galerie = g.cod_galerie)
        FROM    galerie g;
v_ume_gal    galerie.ume_galerie%TYPE;
v_sala       SYS_REFCURSOR;
TYPE sala_ume IS TABLE OF sala.ume_sala%TYPE
                INDEX BY BINARY_INTEGER;
v_ume_sala   sala_ume;
BEGIN
    OPEN c_gal;
    LOOP
        FETCH c_gal INTO v_ume_gal, v_sala;
        EXIT WHEN c_gal%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE (v_ume_gal);
        FETCH v_sala BULK COLLECT INTO v_ume_sala;
        FOR ind IN v_ume_sala.FIRST..v_ume_sala.LAST
            LOOP
                DBMS_OUTPUT.PUT_LINE (v_ume_sala (ind));
            END LOOP;
        END LOOP;
    CLOSE c_gal;
END;

```