## TEMATICA – CURSULUI

1. PL/SQL – CONCEPTE GENERALE
2. BLOCURI PL/SQL; INSTRUCTIUNI
3. TIPURI DE DATE IN PL/SQL
4. GESTIUNEA CURSOARELOR IN PL/SQL
5. SUBPROGRAME IN PL/SQL
6. PACHETE IN PL/SQL
7. DECLANSATORI IN PL/SQL
8. TRATAREA ERORILOR

## 1. PL/SQL – CONCEPTE GENERALE

***P**rocedural **L**anguage/**S**tructured **Q**uery **L**anguage* (*PL/SQL*) este extensia **procedurală** a limbajului *SQL*.

*PL/SQL* este un limbaj de programare sofisticat care asigură accesarea datelor unei **baze de date relaționale orientate obiect** şi permite gruparea unei mulțimi de comenzi într-un **bloc** unic de tratare a datelor. Programul este format din unul sau mai multe blocuri care pot conține blocuri încuibărite.

*PL/SQL* include atât instrucțiuni *SQL* pentru manipularea datelor şi pentru gestiunea tranzacțiilor, cât şi instrucțiuni proprii. Limbajul combină construcțiile procedurale ale unui limbaj *LG3* cu puterea şi flexibilitatea lui *SQL* (*LG4*). Combinația a generat un limbaj puternic pentru modelarea aplicațiilor complexe.

*PL/SQL* extinde *SQL* prin construcții specifice limbajelor procedurale (definirea variabilelor, declararea tipurilor, utilizarea structurilor de control, implementarea procedurilor şi funcțiilor, introducerea tipurilor obiect şi metodelor etc.). *PL/SQL* oferă posibilități moderne de tratare a informației: încapsularea

datelor, analiza specială a erorilor, mascarea informației, orientarea obiect. Posibilitățile lui *SQL* sunt folosite pentru un acces rafinat la date, iar facilitățile oferite de *PL/SQL* sunt folosite pentru fluxul controlului procesării datelor.

Dintre funcționalitățile limbajului *PL/SQL* care determină ca acesta să fie frecvent utilizat se remarcă următoarele facilități:

- integrarea comenzilor *SQL* de bază;
- integrarea cu *server*-ul *Oracle* şi cu utilitare *Oracle*;
- oferirea unui suport pentru programarea orientată obiect;
- asigurarea securității informației;
- definirea şi gestiunea blocurilor de instrucțiuni;
- gestiunea variabilelor, constantelor şi a cursoarelor;
- modularizarea programelor (subprograme, pachete);
- implementarea şi utilizarea declanşatorilor;
- utilizarea structurilor de control fundamentale;
- detectarea şi gestiunea erorilor de execuție şi a situațiilor excepționale;
- dezvoltarea de aplicații *Web*.

*PL/SQL* este o tehnologie utilizată de *server*-ul *Oracle* şi de anumite utilitare *Oracle*. Blocurile *PL/SQL* sunt transmise unui **motor *PL/SQL*** şi procesate (compilate şi executate) de acesta. Motorul *PL/SQL* poate să se afle pe *server*-ul *Oracle* sau într-un utilitar, iar utilizarea sa depinde de unde se invocă *PL/SQL*. Multe utilitare *Oracle* (inclusiv *Developer/2000*) au propriul lor motor *PL/SQL* care este independent de motorul prezent pe *server*-ul *Oracle*.

Blocurile *PL/SQL* pot fi executate pe stația *client* fără interacțiune cu *server*-ul sau în întregime pe *server*. Când blocurile *PL/SQL* sunt referite dintr-un program *PRO**, din *iSQL*Plus*, sau de către *Server Manager*, motorul *PL/SQL* de pe *server*-ul *Oracle* va procesa aceste blocuri. Acesta descompune blocul în instrucțiuni *SQL* şi le trimite executorului de instrucțiuni *SQL* (*SQL Statement Executor*) de pe *server*-ul *Oracle*. Fără *PL/SQL*, instrucțiunile *SQL* ar fi procesate separat, fiecare la un moment dat, fiecare implicând un apel la *server*-ul *Oracle*.

Restul comenzilor (procedurale) sunt procesate de către executorul instrucțiunilor procedurale (*PSE – Procedural Statement Executor*) care este în motorul *PL/SQL*. *PSE* poate procesa datele care sunt locale aplicației, reducându-se astfel activitatea de transfer spre *server*-ul *Oracle* şi numărul de cursoare solicitate. În felul acesta, este necesar un singur transfer pentru a trimite blocul din

aplicație către *server*.

O aplicație bază de date poate fi structurată în trei părți:

- interfața utilizator (utilizatorul introduce anumite informații şi obține nişte rezultate în urma executării aplicației);
- aplicația logică efectivă;
- baza de date.

Există două modele pentru proiectarea unei aplicații bază de date:

modelul *client-server* (*two-tier*);

modelul *three-tier*.

Multe dintre aplicațiile baze de date sunt construite folosind modelul clasic *client-server*, descris succint anterior pentru *PL/SQL*. Modelul este caracterizat de cele două componente: *client* şi *server*. *Client*-ul mânuieşte interfața, iar *server*-ul conține baza de date. Aplicația logică este scindată între *client* şi *server*. De remarcat această caracteristică fundamentală a modelului că aplicația comunică direct cu *server*-ul. Există un motor *PL/SQL* pe *server*, iar în anumite cazuri şi pe *client*.

Dacă motorul *PL/SQL* este pe *server*, atunci aplicația (care poate fi scrisă în *Pro*C*, *JDBC*, *OCI* sau alte limbaje) care rezidă pe *client* trimite cereri la un *server* de date. Cererile sunt rezolvate utilizând *SQL*. Diferite cereri *SQL* pot fi grupate într-un bloc *PL/SQL* şi trimise ca o singură entitate *server*-ului.

Vom considera un scenariu în care există două motoare *PL/SQL*, unul pe stația *client* (local) şi un motor *PL/SQL* pe *server*. De exemplu, un declanşator ce se execută pe stația *client* şi care apelează un subprogram stocat în baza de date. În acest caz, blocurile anonime sunt trimise motorului *PL/SQL* de pe stația *client*, care procesează local comenzile procedurale. Comenzile neprocedurale din interiorul blocului sunt trimise executorului de instrucțiuni *SQL* de pe *server*. De asemenea, apelurile procedurilor care sunt stocate pe *server* sunt trimise tot motorului de pe *server* pentru procesare.

# 2. BLOCURI PL/SQL

## Controlul execuției unui bloc *PL/SQL*

*PL/SQL* este un limbaj cu structură de bloc, adică programele sunt compuse din blocuri care pot fi complet separate sau imbricate. Structura unui bloc poate fi obținută combinând subprograme, pachete, blocuri imbricate. Blocurile pot fi folosite în utilitarele *Oracle*.

Pentru modularizarea unui program este necesară:

- gruparea logică a instrucțiunilor în blocuri;
- imbricarea de subblocuri în blocuri mai mari;
- descompunerea unei probleme complexe într-o mulțime de module logice şi implementarea acestora cu ajutorul blocurilor;
- plasarea în biblioteci a codului *PL/SQL* reutilizabil, de unde poate fi folosit de aplicații;
- depunerea codului într-un *server Oracle*, de unde este accesibil oricărei aplicații care interacționează cu baza de date *Oracle*.

Un program *PL/SQL* poate cuprinde unul sau mai multe blocuri. Un bloc poate fi anonim sau neanonim.

Blocurile **anonime** sunt blocuri *PL/SQL* fără nume, care sunt construite dinamic şi sunt executate o singură dată. Acest tip de bloc nu are argumente şi nu returnează un rezultat. Ele sunt declarate într-un punct al aplicației, unde vor fi executate (trimise motorului *PL/SQL*). În blocurile anonime pot fi declarate proceduri şi funcții *PL/SQL*.

Blocurile anonime pot să apară într-un program ce lucrează cu precompilator sau în *SQL*Plus*. De obicei, blocurile anonime sunt plasate într-un fişier, iar apoi fişierul este executat din *SQL*Plus*. De asemenea, declanşatorii din componentele *Developer Suite* constau din astfel de blocuri.

Blocurile **neanonime** sunt fie blocuri cu nume (etichetate) construite static sau dinamic şi executate o singură dată, fie subprograme, pachete sau declanşatori.

Subprogramele sunt proceduri sau funcții depuse în baza de date. Aceste blocuri sunt executate de mai multe ori şi, în general, nu mai sunt modificate după ce au fost construite. Procedurile şi funcțiile stocate sunt depuse pe *server*-ul *Oracle*, acceptă parametri şi pot fi apelate prin nume. Procedurile şi funcțiile aplicație sunt depuse într-o aplicație *Developer Suite* sau într-o bibliotecă.

Pachetele (stocate sau aplicație) sunt blocuri neanonime care grupează

proceduri, funcţii, cursoare, tipuri, constante, variabile într-o unitate logică, în baza de date.

Declanşatorii sunt blocuri *PL/SQL* neanonime depuse în baza de date, care pot fi asociaţi bazei, iar în acest caz sunt executaţi implicit ori de câte ori apare un anumit eveniment declanşator (de exemplu, instrucţiuni *INSERT*, *UPDATE* sau *DELETE* ce se execută asupra unui tabel al bazei de date) sau pot fi asociaţi unei aplicaţii (de exemplu, declanşator *SQL*Forms*), ceea ce presupune că se execută automat, în funcţie de anumite condiţii sistem.

**Structura unui bloc *PL/SQL***

Un bloc *PL/SQL* este compus din trei secţiuni distincte.

- Secţiunea declarativă (opţională) conţine declaraţii pentru toate variabilele, constantele, cursoarele şi erorile definite de utilizator la care se face referinţă în secţiunea executabilă sau chiar în cea declarativă. De asemenea, pot fi declarate subprograme locale care sunt vizibile doar în blocul respectiv.
- Secţiunea executabilă conţine instrucţiuni neprocedurale *SQL* pentru prelucrarea datelor din baza de date şi instrucţiuni *PL/SQL* pentru prelucrarea datelor în cadrul blocului.
- Secţiunea pentru tratarea erorilor (opţională) specifică acţiunile ce vor fi efectuate atunci când în execuţia blocului apar erori sau condiţii anormale.

Blocul *PL/SQL* are următoarea structură generală:

[<<*nume_bloc*>>]
[***DECLARE***
*instrucţiuni de declarare*]
***BEGIN***
*instrucţiuni executabile* (*SQL sau PL/SQL*)
[***EXCEPTION***
*tratarea erorilor*]
***END*** [*nume_bloc*];

Dacă blocul *PL/SQL* este executat fără erori, invariant va apărea mesajul:

*PL/SQL procedure successfully completed*

***Exemplu* (SELECT cu clauza INTO)**

Să se creeze un bloc anonim în care se declară o variabilă *v_job* de tip *job_title* (%TYPE) a cărei valoare va fi titlul jobului salariatului având codul 200.

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2    v_job jobs.job_title%TYPE;
  3  BEGIN
  4    SELECT job_title
  5    INTO   v_job
  6    FROM   employees e, jobs j
  7    WHERE  e.job_id=j.job_id
  8    AND    employee_id=200;
  9    DBMS_OUTPUT.PUT_LINE('jobul este '|| v_job);
 10  END;
 11  /
jobul este Administration Assistant

PL/SQL procedure successfully completed.
```

**Varianta 2**

Să se rezolve problema anterioară utilizând **variabile de legătură**. Să se afişeze rezultatul atât din bloc, cât şi din exteriorul acestuia.

```
SQL> VARIABLE rezultat VARCHAR2(35)
SQL> BEGIN
  2    SELECT job_title
  3    INTO   :rezultat
  4    FROM   employees e, jobs j
  5    WHERE  e.job_id=j.job_id AND employee_id=200;
  6    DBMS_OUTPUT.PUT_LINE('rezultatul este '|| :rezultat);
  7  END;
  8  /
rezultatul este Administration Assistant

PL/SQL procedure successfully completed.


SQL> PRINT REZULTAT

REZULTAT

------------------------------
```

## Compatibilitate *SQL*

Din punct de vedere al compatibilității dintre *PL/SQL* şi *SQL*, se remarcă următoarele reguli de bază:

- *PL/SQL* furnizează toate comenzile *LMD* ale lui *SQL*, comanda *SELECT* cu clauza *INTO*, comenzile *LCD*, funcțiile, pseudocoloanele şi operatorii *SQL*;
- *PL/SQL* nu furnizează comenzile *LDD*.

Totuşi, în ultimele sale versiuni, *Oracle* permite folosirea dinamică a comenzilor *SQL*, utilizând tehnica oferită de *SQL* dinamic. În felul acesta, orice comandă *SQL* (inclusiv comandă *LDD*) poate să fie utilizată în *PL/SQL*.

Majoritatea funcțiilor *SQL* sunt disponibile în *PL/SQL*. Există însă funcții specifice *PL/SQL*, cum sunt funcțiile *SQLCODE* şi *SQLERRM*. De asemenea, există funcții *SQL* care nu sunt disponibile în instrucțiuni procedurale (*DECODE*, funcțiile grup), dar care sunt disponibile în instrucțiunile *SQL* dintr-un bloc *PL/SQL*. *SQL* nu poate folosi funcții sau atribute specifice *PL/SQL*.

Funcțiile grup trebuie folosite cu atenție, deoarece clauza *GROUP BY* nu are sens să apară în instrucțiunea *SELECT ... INTO*. *Oracle9i* introduce clauza *OVER*, care permite ca funcția grup căreia îi este asociată să fie considerată o funcție analitică (poate returna mai multe linii pentru fiecare grup).

Următoarele funcții *SQL* nu sunt permise în *PL/SQL*: *WIDTH_BUCKET*, *BIN_TO_NUM*, *COMPOSE*, *DECOMPOSE*, *TO_LOB*, *DECODE*, *DUMP*, *EXISTSNODE*, *TREAT*, *NULLIF*, *SYS_CONNECT_BY_PATH*, *SYS_DBURIGEN*, *EXTRACT*.

## Instrucțiuni *PL/SQL*

Orice program poate fi scris utilizând structuri de control de bază care sunt combinate în diferite moduri pentru rezolvarea problemei propuse. *PL/SQL* dispune de comenzi ce permit controlul execuției unui bloc. Instrucțiunile limbajului pot fi: iterative (*LOOP*, *WHILE*, *FOR*), de atribuire (:=), condiționale (*IF*, *CASE*), de salt (*GOTO*, *EXIT*) şi instrucțiunea vidă (*NULL*).

***Observații*:**

- Comentariile sunt ignorate de compilatorul *PL/SQL*. Există comentarii pe o singură linie, prefixate de simbolurile „--“, care încep în orice punct al liniei şi se termină la sfârşitul acesteia. De asemenea, există comentarii pe mai multe linii, care sunt delimitate de simbolurile „/*“ şi „*/“. Nu se

admit comentarii imbricate.

- Caracterul „;“ este separator pentru instrucțiuni.
- Atât operatorii din *PL/SQL*, cât şi ordinea de execuție a acestora, sunt identici cu cei din *SQL*. În *PL/SQL* este introdus un nou operator („**“) pentru ridicare la putere.
- Un identificator este vizibil în blocul în care este declarat şi în toate subblocurile, procedurile şi funcțiile imbricate în acesta. Dacă blocul nu găseşte identificatorul declarat local, atunci îl caută în secțiunea declarativă a blocurilor care includ blocul respectiv şi niciodată nu caută în blocurile încuibărite în acesta.
- Comenzile *SQL*Plus* nu pot să apară într-un bloc *PL/SQL*.
- În comanda *SELECT* trebuie specificate variabilele care recuperează rezultatul acțiunii acestei comenzi. În clauza *INTO*, care este obligatorie, pot fi folosite variabile *PL/SQL* sau variabile de legătură.
- Referirea la o variabilă de legătură se face prin prefixarea acesteia cu simbolul „:“.
- Cererea dintr-o comandă *SELECT* trebuie să returneze o singură linie drept rezultat. Atunci când comanda *SELECT* întoarce mai multe linii, apare eroarea *TOO_MANY_ROWS*, iar în cazul în care comanda nu găseşte date se generează eroarea *NO_DATA_FOUND*.
- Un bloc *PL/SQL* nu este o unitate tranzacțională. Într-un bloc pot fi mai multe tranzacții sau blocul poate face parte dintr-o tranzacție. Acțiunile *COMMIT*, *SAVEPOINT* şi *ROLLBACK* sunt independente de blocuri, dar instrucțiunile asociate acestor acțiuni pot fi folosite într-un bloc.
- *PL/SQL* nu suportă comenzile *GRANT* şi *REVOKE*, utilizarea lor fiind posibilă doar prin *SQL* dinamic.

Fluxul secvențial de execuție a comenzilor unui program *PL/SQL* poate fi modificat cu ajutorul structurilor de control: *IF*, *CASE*, *LOOP*, *FOR*, *WHILE*, *GOTO*, *EXIT*.

**Instrucțiunea de atribuire**

Instrucțiunea de atribuire se realizează cu ajutorul operatorului de asignare (:=) şi are forma generală clasică (*variabila* := *expresie*). Comanda respectă proprietățile instrucțiunii de atribuire din clasa *LG3*. De remarcat că nu poate fi asignată valoarea *null* unei variabile care a fost declarată *NOT NULL*.

***Exemplu*:**

Următorul exemplu prezintă modul în care acționează instrucțiunea de atribuire în cazul unor tipuri de date particulare.

```
DECLARE
  alfa  INTERVAL YEAR TO MONTH;
BEGIN
  alfa  := INTERVAL '200-7' YEAR TO MONTH;
     DBMS_OUTPUT.PUT_LINE(alfa);
  -- alfa ia valoarea 200 de ani si 7 luni
  alfa := INTERVAL '200' YEAR;
  -- pot fi specificati numai anii
  alfa := INTERVAL '7' MONTH;
  -- pot fi specificate numai lunile
  alfa := '200-7';
  -- conversie implicita din caracter
END;
SQL>  declare
  2   alfa interval year to month;
  3   begin
  4  alfa :=interval '1 - 7' year to month;
  5    DBMS_OUTPUT.PUT_LINE('alfa = '|| alfa);
  6   end;
  7  /
alfa = +01-07
PL/SQL procedure successfully completed.

SQL>  declare
  2   alfa interval year to month;
  3   begin
  4  alfa :=interval '7' month;
  5   DBMS_OUTPUT.PUT_LINE('alfa = '|| alfa);
  6  end;
  7  /
alfa = +00-07
PL/SQL procedure successfully completed.

SQL>  declare
  2   alfa interval year to month;
```

```
  3   begin
  4    alfa := '10 - 8';
  5   DBMS_OUTPUT.PUT_LINE('alfa = '|| alfa);
  6  end;
  7  /
alfa = +10-08
PL/SQL procedure successfully completed.
```

```
DECLARE
  beta  opera%ROWTYPE;
  gama  opera%ROWTYPE;
  cursor epsilon IS SELECT * FROM opera;
  delta  epsilon%ROWTYPE;
BEGIN
  beta := gama;  -- corect
  gama := delta;  -- incorect???-testati!
END;
```

**Instrucțiunea *IF***

Un program *PL/SQL* poate executa diferite porțiuni de cod, în funcție de rezultatul unui test (predicat). Instrucțiunile care realizează acest lucru sunt cele condiționale (*IF*, *CASE*).

Structura instrucțiunii *IF* în *PL/SQL* este similară instrucțiunii *IF* din alte limbaje procedurale, permițând efectuarea unor acțiuni în mod selectiv, în funcție de anumite condiții. Instrucțiunea *IF-THEN-ELSIF* are următoarea formă sintactică:

***IF*** *condiție1* ***THEN***
  *secvența_de_comenzi_1*
**[*ELSIF*** *condiție2* ***THEN***
  *secvența_de_comenzi_2*]
...
[***ELSE***
  *secvența_de_comenzi_n*]
***END IF***;

O secvență de comenzi din *IF* este executată numai în cazul în care condiția asociată este *TRUE*. Atunci când condiția este *FALSE* sau *NULL*, secvența nu este executată. Dacă pe ramura *THEN* se dorește verificarea unei alternative, se folosește ramura *ELSIF* (atenție, nu *ELSEIF*) cu o nouă condiție. Este permis un număr arbitrar de opțiuni *ELSIF*, dar poate apărea cel mult o clauză *ELSE*. Aceasta se referă la ultimul *ELSIF*.

***Exemplu*:**

Să se specifice dacă o galerie este *mare*, *medie* sau *mica* după cum numărul operelor de artă expuse în galeria respectivă este mai mare decât 200, cuprins între 100 şi 200 sau mai mic decât 100.

```
DEFINE p_cod_gal = 753
DECLARE
  v_cod_galerie  opera.cod_galerie%TYPE := &p_cod_gal;
  v_numar        NUMBER(3) := 0;
  v_comentariu   VARCHAR2(10);
BEGIN
  SELECT  COUNT(*)
  INTO    v_numar
  FROM    opera
  WHERE   cod_galerie = v_cod_galerie;
  IF v_numar < 100 THEN
    v_comentariu := 'mica';
  ELSIF v_numar BETWEEN 100 AND 200 THEN
    v_comentariu := 'medie';
  ELSE
    v_comentariu := 'mare';
  END IF;
  DBMS_OUTPUT.PUT_LINE('Galeria avand codul '||
    v_cod_galerie ||' este de tip '|| v_comentariu);
END;
/
```

***Exemplu*:**

Să se specifice dacă un angajat dat are salariu *mare*, *mediu* sau *mic* după cum este mai mare decât 20000, cuprins între 10000 şi 20000 sau mai mic decât 10000.

```
SQL> DEFINE p_cod_em = 201
SQL> DECLARE
  2    v_cod_ang  EMPLOYEES. EMPLOYEE_ID%TYPE := &p_cod_em;
  3    v_sal       EMPLOYEES.salary%type;
  4    v_comentariu   VARCHAR2(10);
  5  BEGIN
  6    SELECT  salary
  7    INTO    v_sal
  8    FROM    EMPLOYEES
  9    WHERE   EMPLOYEE_ID =  v_cod_ang;
 10    IF v_sal  < 10000 THEN
 11      v_comentariu := 'mic';
 12    ELSIF v_sal  BETWEEN 10000 AND 20000 THEN
 13      v_comentariu := 'mediu';
 14    ELSE
 15      v_comentariu:= 'mare';
 16    END IF;
 17     DBMS_OUTPUT.PUT_LINE('salariatul avand codul  '|| v_cod_ang ||'  are
salariu '|| v_sal ||  '
considerat  '|| v_comentariu);
 18  END;
 19  /
old   2:   v_cod_ang  EMPLOYEES. EMPLOYEE_ID%TYPE := &p_cod_em;
new   2:   v_cod_ang  EMPLOYEES. EMPLOYEE_ID%TYPE := 201;
salariatul avand codul  201  are salariu 13000 considerat  mediu

PL/SQL procedure successfully completed./
```

**Instrucțiunea *CASE***

*Oracle9i* furnizează o nouă comandă (*CASE*) care permite implementarea unor condiții multiple. Instrucțiunea are următoarea formă sintactică:

[<<*eticheta*>>]
***CASE*** *test_var*
  ***WHEN*** *valoare_1* ***THEN*** *secvența_de_comenzi_1*;
  ***WHEN*** *valoare_2* ***THEN*** *secvența_de_comenzi_2*;
  ...
  ***WHEN*** *valoare_k* ***THEN*** *secvența_de_comenzi_k*;
  [***ELSE*** *altă_secvență*;]
***END CASE*** [*eticheta*];

Se va executa *secvența_de_comenzi_p*, dacă valoarea selectorului *test_var* este *valoare_p*. După ce este executată secvența de comenzi, controlul va trece la următoarea instrucțiune după *CASE*. Selectorul *test_var* poate fi o variabilă sau o expresie complexă care poate conține chiar şi apeluri de funcții.

Clauza *ELSE* este opțională. Dacă această clauză este necesară în implementarea unei probleme, dar totuşi lipseşte, iar *test_var* nu ia nici una dintre valorile ce apar în clauzele *WHEN*, atunci se declanşează eroarea predefinită *CASE_NOT_FOUND* (*ORA - 06592*).

Comanda *CASE* poate fi etichetată şi, în acest caz, eticheta poate să apară la sfârşitul clauzei *END CASE*. De remarcat că eticheta după *END CASE* este permisă numai în cazul în care comanda *CASE* este etichetată.

Selectorul *test_var* poate să lipsească din structura comenzii *CASE*, care în acest caz va avea următoarea formă sintactică:

[<<*eticheta*>>]
***CASE***
  ***WHEN*** *condiție_1* ***THEN*** *secvența_de_comenzi_1*;
  ***WHEN*** *condiție_2* ***THEN*** *secvența_de_comenzi_2*;
  ...
  ***WHEN*** *condiție_k* ***THEN*** *secvența_de_comenzi_k*;
  [***ELSE*** *altă_secvență*;]
***END CASE*** [*eticheta*];

Fiecare clauză *WHEN* conține o expresie booleană. Dacă valoarea lui *condiție_p* este *TRUE*, atunci este executată *secvența_de_comenzi_p*.

***Exemplu*:**

În funcție de o valoare introdusă de utilizator, care reprezintă abrevierea zilelor unei săptămâni, să se afişeze (în cele două variante) un mesaj prin care este specificată ziua săptămânii corespunzătoare abrevierii respective.

***Varianta 1*:**

```
SET SERVEROUTPUT ON
DEFINE p_zi = x
DECLARE
  v_zi  CHAR(2) := UPPER('&p_zi');
BEGIN
  CASE v_zi
    WHEN 'L'  THEN DBMS_OUTPUT.PUT_LINE(' Astazi este Luni');
    WHEN 'M'  THEN DBMS_OUTPUT.PUT_LINE(' Astazi este Marti');
    WHEN 'MI' THEN DBMS_OUTPUT.PUT_LINE(' Astazi este Miercuri');
    WHEN 'J'  THEN DBMS_OUTPUT.PUT_LINE(' Astazi este Joi');
    WHEN 'V'  THEN DBMS_OUTPUT.PUT_LINE(' Astazi este Vineri');
```

```
   WHEN 'S'  THEN DBMS_OUTPUT.PUT_LINE(' Astazi este Sambata');
   WHEN 'D'  THEN DBMS_OUTPUT.PUT_LINE(' Astazi este Duminica');
   ELSE DBMS_OUTPUT.PUT_LINE('este o eroare!');
  END CASE;
END;
/
SET SERVEROUTPUT OFF
```

***Varianta* 2*:***

```
SET SERVEROUTPUT ON
DEFINE p_zi = x
DECLARE
  v_zi  CHAR(2) := UPPER('&p_zi');
BEGIN
  CASE
    WHEN v_zi = 'L'  THEN
      DBMS_OUTPUT.PUT_LINE('Astazi este Luni');
    WHEN v_zi = 'M'  THEN
      DBMS_OUTPUT.PUT_LINE(' Astazi este Marti');
    WHEN v_zi = 'MI' THEN
       DBMS_OUTPUT.PUT_LINE(' Astazi este Miercuri');
    WHEN v_zi = 'J'  THEN
       DBMS_OUTPUT.PUT_LINE(' Astazi este Joi');
    WHEN v_zi = 'V'  THEN
       DBMS_OUTPUT.PUT_LINE(' Astazi este Vineri');
    WHEN v_zi = 'S'  THEN
        DBMS_OUTPUT.PUT_LINE(' Astazi este Sambata');
    WHEN v_zi = 'D'  THEN
       DBMS_OUTPUT.PUT_LINE(' Astazi este Duminica');
    ELSE DBMS_OUTPUT.PUT_LINE('Este o eroare!');
  END CASE;
END;
/
SET SERVEROUTPUT OFF
```

În *Oracle9i* poate fi utilizată o construcție *CASE* într-o comandă *SQL* a unui bloc *PL/SQL*.

Expresia *CASE* are sintaxa similară comenzii *CASE*, dar clauzele *WHEN* nu se termină prin caracterul „;“, clauza *END* nu include cuvântul cheie *CASE* şi nu se fac atribuiri în clauza *WHEN*.

Expresia *CASE* returneaza *null* daca nu exista clauza *ELSE* si daca nici o

conditie nu este indeplinita.

```
 SELECT last_name, SALARY,
     (CASE WHEN   salary   <5000   THEN  '    LOW'
               WHEN  salary   <10000 THEN  '    MEDIUM'
                WHEN salary   <15000 THEN  '    GOOD'
                                     ELSE  '    EXCELLENT'
     END) AS calificare
   FROM    EMPLOYEES;
```

| LAST_NAME | SALARY | CALIFICARE |
|---|---|---|
| King | 24000 | EXCELLENT |
| Kochhar | 17000 | EXCELLENT |
| De Haan | 17000 | EXCELLENT |
| Hunold | 9000 | MEDIUM |
| Ernst | 6000 | MEDIUM |
| Austin | 4800 | LOW |
| Pataballa | 4800 | LOW |
| Lorentz | 4200 | LOW |
| Greenberg | 12000 | GOOD |
| Faviet | 9000 | MEDIUM |
| Chen | 8200 | MEDIUM |

**Instrucțiuni iterative**

Există trei tipuri de comenzi iterative: ciclarea simplă *LOOP*, ciclarea *WHILE* şi ciclarea *FOR*.

Acestea permit repetarea (condiționată sau necondiționată) execuției uneia sau mai multor instrucțiuni. Ciclurile pot fi imbricate pe mai multe niveluri. Ele pot fi etichetate, iar ieşirea din ciclu se poate realiza cu ajutorul comenzii *EXIT*.

Se utilizează:

- comanda *LOOP*, dacă instrucțiunile din cadrul ciclului trebuie să se execute cel puțin o dată;
- comanda *WHILE*, în cazul în care condiția trebuie evaluată la începutul fiecărei iterații;
- comanda *FOR*, dacă numărul de iterații este cunoscut.

Instrucțiunea *LOOP* are următoarea formă sintactică:

***LOOP***
*secvența_de_comenzi*;
***END LOOP***;

Ciclarea simplă cuprinde o mulțime de comenzi incluse între cuvintele cheie *LOOP* şi *END LOOP*. Aceste comenzi se execută cel puțin o dată. Dacă nu este utilizată comanda *EXIT*, ciclarea poate continua la infinit.

***Exemplu*:**

Se presupune că a fost creată structura tabelului *org_tab*, constând din două coloane: *cod_tab* de tip *INTEGER*, ce conține un contor al înregistrărilor şi *text_tab* de tip *VARCHAR2*, ce conține un text asociat fiecărei înregistrări. Să se introducă 70 de înregistrări în tabelul *org_tab*.

```
CREATE TABLE ORG_TAB
 (cod_tab INTEGER,
   text_tab VARCHAR2(25));


SET SERVEROUTPUT ON
DECLARE
  v_contor BINARY_INTEGER := 1;
BEGIN
  LOOP
    INSERT INTO org_tab
    VALUES (v_contor, 'indicele '||  v_contor);
    v_contor := v_contor + 1;
    EXIT WHEN v_contor > 70;
  END LOOP;
COMMIT;
END;

SELECT * FROM ORG_TAB;


   COD_TAB TEXT_TAB
---------- --------------------
```

```
        67 indicele 67
        68 indicele 68
        69 indicele 69
        70 indicele 70

70 rows selected.
```

Instrucțiunea repetitivă *WHILE* permite repetarea unei secvențe de instrucțiuni, atâta timp cât o anumită condiție specificată este adevărată.

Comanda *WHILE* are următoarea sintaxă:

***WHILE** condiție **LOOP***
  *secvența_de_comenzi*;
***END LOOP***;

Dacă variabilele care apar în condiție nu se schimbă în interiorul ciclului, atunci condiția rămâne adevărată şi ciclul nu se termină.

Când condiția este evaluată ca fiind *FALSE* sau *NULL*, atunci secvența de comenzi nu este executată şi controlul trece la prima instrucțiune după *END LOOP*.

***Exemplu*:**

```
DECLARE
  v_contor BINARY_INTEGER := 1;
BEGIN
  WHILE v_contor <= 70 LOOP
    INSERT INTO org_tab
    VALUES (v_contor, 'indicele ciclului');
    v_contor := v_contor + 1;
  END LOOP;
END;
```

Instrucțiunea repetitivă *FOR* (ciclare cu pas) permite executarea unei secvențe de instrucțiuni pentru valori ale variabilei *contor* cuprinse între două limite, *lim_inf* şi *lim_sup*. Dacă este prezentă opțiunea *REVERSE*, iterația se face

(în sens invers) de la *lim_sup* la *lim_inf*.

Comanda *FOR* are sintaxa:

***FOR*** *contor_ciclu* ***IN*** [***REVERSE***] *lim_inf..lim_sup* ***LOOP***
  *secvența_de_comenzi*;
***END LOOP***;

Variabila *contor_ciclu* nu trebuie declarată. Ea este neidentificată în afara ciclului şi implicit de tip *BINARY_INTEGER*. Pasul are implicit valoarea 1 şi nu poate fi modificat. Limitele domeniului pot fi variabile sau expresii, care să poată fi convertite la întreg.

***Exemplu*:**

În structura tabelului *opera* se va introduce un nou câmp (*stea*). Să se creeze un bloc *PL/SQL* care va reactualiza acest câmp, introducând o steluță pentru fiecare 10000$ din valoarea unei opere de artă al cărei cod este specificat.

```
ALTER TABLE  opera
ADD stea     VARCHAR2(20);
DEFINE p_cod_opera = 7777
DECLARE
  v_cod_opera  opera.cod_opera%TYPE := &p_cod_opera;
  v_valoare    opera.valoare%TYPE;
  v_stea       opera.stea%TYPE := NULL;
BEGIN
  SELECT  NVL(ROUND(valoare/10000),0)
  INTO    v_valoare
  FROM    opera
  WHERE   cod_opera = v_cod_opera;
  IF v_valoare > 0 THEN
    FOR i IN 1..v_valoare LOOP
      v_stea := v_stea || '*';
    END LOOP;
  END IF;
  UPDATE  opera
  SET     stea = v_stea
  WHERE   cod_opera = v_cod_opera;
  COMMIT;
END;
```

**Exemplu:**

```
SQL> ALTER TABLE  EMPLOYEES
  2  ADD stea      VARCHAR2(20);

Table altered.
```

```
DECLARE
v_cod_ANGAJ  EMPLOYEES.EMPLOYEE_ID%TYPE :=&p_cod_ANGAJ;
v_valoare   EMPLOYEES.SALARY%TYPE;
 v_stea      EMPLOYEES.stea%TYPE := NULL;
BEGIN
  SELECT  NVL(ROUND(SALARY/10000),0)
  INTO    v_valoare
  FROM   EMPLOYEES
  WHERE   EMPLOYEE_ID = v_cod_ANGAJ;

  IF v_valoare > 0 THEN
    FOR i IN 1..v_valoare LOOP
      v_stea := v_stea || '*';
    END LOOP;
  END IF;
  UPDATE  EMPLOYEES
  SET     stea = v_stea
  WHERE   EMPLOYEE_ID = v_cod_ANGAJ;
  COMMIT;
END;
/

Enter value for p_cod_angaj: 100
old          2:    v_cod_ANGAJ        EMPLOYEES.EMPLOYEE_ID%TYPE    :=
&p_cod_ANGAJ;
new   2: v_cod_ANGAJ  EMPLOYEES.EMPLOYEE_ID%TYPE := 100;
PL/SQL procedure successfully completed.
```

```
SQL>  SELECT * FROM EMPLOYEES
  2  WHERE EMPLOYEE_ID=100;

EMPLOYEE_ID FIRST_NAME          LAST_NAME                 EMAIL
PHONE_NUMBER        HIRE_DATE JOB_ID        SALARY
----------- -------------------- ------------------------- ------------------------- --------------
COMMISSION_PCT MANAGER_ID DEPARTMENT_ID STEA
-------------- ---------- ------------- --------------------
       100 Steven               King                      SKING               515.123.4567
17-JUN-87 AD_PRES                                    24000
                                                        90       **
```

## Instrucțiuni de salt

Instrucțiunea *EXIT* permite ieşirea dintr-un ciclu. Ea are o formă necondițională (ieşire fără condiții) şi una condițională. Controlul trece fie la prima instrucțiune situată după clauza *END LOOP* corespunzătoare, fie după instrucțiunea *LOOP* având eticheta *nume_eticheta*.

***EXIT*** [*nume_eticheta*] [***WHEN*** *condiție*];

Numele etichetelor urmează aceleaşi reguli ca cele definite pentru identificatori. Eticheta se plasează înaintea comenzii, fie pe aceeaşi linie, fie pe o linie separată. În *PL/SQL* etichetele se definesc prin intercalarea numelui etichetei între caracterele „<<“ şi „>>“ (<<*eticheta*>>).

***Exemplu*:**

```
DECLARE
  v_contor    BINARY_INTEGER := 1;
  raspuns     VARCHAR2(10);
  alt_raspuns VARCHAR2(10);
BEGIN
  …
  <<exterior>>
  LOOP
    v_contor := v_contor + 1;
    EXIT WHEN v_contor > 70;
    <<interior>>
```

```
   LOOP
     …
     EXIT exterior WHEN raspuns = 'DA';
     -- se parasesc ambele cicluri
     EXIT WHEN alt_raspuns = 'DA';
     -- se paraseste ciclul interior
     …
   END LOOP interior;
   …
  END LOOP exterior;
END;
```

Instrucțiunea *GOTO* determină un salt necondiționat la o instrucțiune executabilă sau la începutul unui bloc care are eticheta specificată în comandă. Instrucțiunea are următoarea formă sintactică:

***GOTO*** *nume_eticheta*;

Nu este permis saltul:

- în interiorul unui bloc (subbloc);
- în interiorul unei comenzi *IF*, *CASE* sau *LOOP*;
- de la o clauză a comenzii *CASE*, la altă clauză a aceleaşi comenzi;
- de la tratarea unei excepții, în blocul curent;
- în exteriorul unui subprogram.

**Instrucțiunea vidă**

Instrucțiunea vidă (*NULL*) este folosită pentru o mai bună lizibilitate a programului. *NULL* este instrucțiunea care nu are nici un efect, marcând faptul că nu trebuie întreprinsă nici o acțiune. Nu trebuie confundată instrucțiunea *NULL* cu valoarea *null*!

Uneori instrucțiunea *NULL* este folosită într-o comandă *IF*, indicând faptul că pentru o anumită clauză *ELSIF* nu se execută nici o acțiune.

# 2. Tipuri de date în *PL/SQL*

Fiecare variabilă sau constantă utilizată într-un bloc *PL/SQL* este de un anumit tip prin care se specifică:

- formatul său de stocare,
- constrângerile care trebuie să le verifice
- domeniul valorilor sale.

Variabilele folosite în *Oracle9i* pot fi:

- specifice *PL/SQL;*
- nespecifice *PL/SQL*.

Variabile specifice *PL/SQL* se clasifică în variabile:

- de tip scalar,
- compuse,
- referință,
- *LOB* (*large objects*),
- tipuri obiect.

Variabile nespecifice *PL/SQL* pot fi:

- variabile de legătură (*bind variables*),
- variabile gazdă (*host variables*),
- variabile indicator.

## Variabile specifice *PL/SQL*

### Tipurile de date scalare

Nu au componente interne (conțin valori atomice). Se împart în 5 clase.

- Tipurile de date ce stochează **valori numerice** cuprind tipul *NUMBER* cu subtipurile *DEC*, *DECIMAL*, *DOUBLE PRECISION*, *FLOAT*, *INTEGER*, *INT*, *NUMERIC*, *REAL*, *SMALLINT*; tipul *BINARY_INTEGER* cu subtipurile *NATURAL*, *NATURALN*, *POSITIVE*, *POSITIVEN*, *SIGNTYPE*; tipul *PLS_INTEGER*.
- Tipurile de date ce stochează **caractere** cuprind tipul *VARCHAR2* cu subtipurile *STRING*, *VARCHAR*; tipul de date *CHAR* cu subtipul *CHARACTER*; tipurile *LONG*, *RAW*, *LONG RAW*, *ROWID*.
- Tipurile de date ce stochează **data calendaristică şi ora** cuprind tipurile *DATE*, *TIMESTAMP*, *TIMESTAMP WITH TIME ZONE*, *TIMESTAMP WITH LOCAL TIME ZONE*, *INTERVAL YEAR TO MONTH*, *INTERVAL DAY TO SECOND*.

- Tipurile de date **globalizare** ce stochează date *unicode* includ tipurile *NCHAR* şi *NVARCHAR2*.
- Tipul de date *BOOLEAN* stochează **valori logice** (*true*, *false* sau *null*).

## Tipurile de date compuse

Au componente interne care pot fi manipulate individual. *Oracle* oferă programatorului două tipuri de date compuse:

- înregistrare (*RECORD*);
- colecție (*INDEX-BY TABLE, NESTED TABLE, VARRAY*).

## Tipurile de date referință

*REF CURSOR* si *REF obiect* sunt tipuri de date ale căror valori, numite *pointeri*, fac referință către obiecte din program. Pointerii conțin locația de memorie (adresa) unui element şi nu elementul în sine. Tipul *REF CURSOR* este folosit pentru a face referință la un cursor explicit. Tipul *REF obiect* face referință la adresa unui obiect.

## Tipurile de date *LOB*

*Large object* sunt acele tipuri de date ale căror valori, numite locatori (*locators*) specifică localizarea unor obiecte de dimensiuni mari, adică blocuri de date nestructurate, cum ar fi texte, imagini grafice, clipuri video şi sunete. Tipurile *LOB* sunt manipulate cu ajutorul pachetului *DBMS_LOB*.

Aceste tipuri sunt:

- *CLOB* (*character large object*),
- *BLOB* (*binary large object*),
- *BFILE* (*binary file*),
- *NCLOB* (*national language character large object*).

## Tipurile obiect

Sunt tipuri compuse, definite de utilizator, care încapsulează structuri de date (atribute) împreună cu subprograme pentru manipularea datelor (metode).

Dintre tipurile scalare *PL/SQL*, următoarele sunt şi tipuri *SQL* (adică pot fi folosite pentru coloanele tabelelor *Oracle*): *NUMBER*, *VARCHAR2*, *CHAR*, *LONG*, *RAW*, *LONG RAW*, *ROWID*, *NCHAR*, *NVARCHAR2*, *DATE*. În unele cazuri, tipurile de date *PL/SQL* diferă de corespondentele lor *SQL* prin dimensiunea maximă permisă.

Tipul *NUMBER* memorează numerele în virgulă fixă şi virgulă mobilă. El are forma generală *NUMBER* (*m*, *n*), unde *m* reprezintă numărul total de cifre, iar *n* numărul de zecimale. Valoarea unei variabile de tip *NUMBER* este cuprinsă între 1.0E-129 şi 9.99E125. Numărul de zecimale determină poziția în care apare rotunjirea. Valoarea sa este cuprinsă între -84 şi 127, iar implicit este 0.

Tipul *NUMBER* are următoarele subtipuri, care au aceleaşi intervale de valori: *NUMERIC*, *REAL*, *DEC*, *DECIMAL* şi *DOUBLE PRECISION* (pentru memorarea datelor numerice în virgulă fixă), *FLOAT* (pentru memorarea datelor numerice în virgulă mobilă), *SMALLINT*, *INTEGER* şi *INT* (pentru memorarea numerelor întregi). Aceste subtipuri se pot utiliza pentru compatibilitate *ANSI/ISO*, *IBM SQL/DS* sau *IBM DB2*.

Tipul *BINARY_INTEGER* memorează numere întregi cu semn având valori cuprinse între $-2^{31} - 1$ şi $2^{31} - 1$. Acest tip de date este utilizat frecvent pentru indecşii tabelelor, nu necesită conversii şi admite mai multe subtipuri. De exemplu, pentru a restricționa domeniul variabilelor la valori întregi nenegative se utilizează tipurile *NATURAL* ($0 .. 2^{31} - 1$) şi *POSITIVE* ($1 .. 2^{31} - 1$).

Tipul *PLS_INTEGER* este utilizat pentru stocarea numerelor întregi cu semn şi are acelaşi interval de definire ca şi tipul *BINARY_INTEGER*. Operațiile cu acest tip sunt efectuate mai rapid (folosesc aritmetica maşinii), decât cele cu tipurile *NUMBER* sau *BINARY_INTEGER* (folosesc librării aritmetice). Prin urmare, pentru o mai bună performanță, este preferabil să se utilizeze tipul *PLS_INTEGER*.

Variabilele alfanumerice pot fi de tip *CHAR*, *VARCHAR2*, *LONG*, *RAW* şi *LONGRAW*. Reprezentarea internă depinde de setul de caractere ales (*ASCII* sau *EBCDIC*).

Tipurile *CHAR*, *VARCHAR2* şi *RAW* pot avea un parametru pentru a preciza lungimea maximă. Dacă aceasta nu este precizată atunci, implicit, se consideră 1. Lungimea este exprimată în octeți (nu în caractere). Subtipurile acestor tipuri se pot utiliza pentru compatibilitate *ANSI/ISO*, *IBM SQL/DS* sau *IBM DB2*.

În *Oracle9i* a fost extinsă sintaxa pentru *CHAR* şi *VARCHAR2*, permițând ca variabila ce precizează lungimea maximă să fie de tip *CHAR* sau *BYTE*.

Variabilele de tip *LONG* pot memora texte, tabele de caractere sau documente, prin urmare şiruri de caractere de lungime variabilă de până la 32760 octeți. Este similar tipului *VARCHAR2*.

Tipul *RAW* permite memorarea datelor binare (biți) sau a şirurilor de octeți. De exemplu, o variabilă *RAW* poate memora o secvență de caractere grafice sau o imagine digitizată. Tipul *RAW* este similar tipului alfanumeric, cu excepția faptului

că *PL/SQL* nu interpretează datele de tip *RAW*. *Oracle* nu face conversia datelor de acest tip, atunci când se transmit de la un sistem la altul. Chiar dacă lungimea maximă a unei variabile *RAW* poate fi 32767 octeți, într-o coloană *RAW* a bazei de date nu se pot introduce decât 2000 octeți. Pentru a insera valori mai mari se foloseşte o coloană de tip *LONG RAW*, care are lungimea maximă $2^{31}$ octeți. *LONG RAW* este similar tipului *LONG*, dar datele nu mai sunt interpretate de *PL/SQL*.

Tipurile *TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE, INTERVAL YEAR TO MONTH, INTERVAL DAY TO SECOND* au fost introduse în *Oracle9i* şi permit rafinări ale tipului *DATE*. De exemplu, *TIMESTAMP* poate lua în considerare şi fracțiuni de secundă.

*PL/SQL* suportă două seturi de caractere: una specifică bazei de date care este utilizată pentru definirea identificatorilor şi a codului sursă (*database character set - DCS*) şi o mulțime de caractere naționale care este folosită pentru reprezentarea informației cu caracter național (*national character set - NCS*).

Tipurile de date *NCHAR* şi *NVARCHAR2* sunt utilizate pentru stocarea în baza de date a şirurilor de caractere ce folosesc *NCS*. Ele oferă suport pentru globalizarea datelor, astfel încât utilizatorii din toată lumea pot interacționa cu *Oracle* în limba lor națională. Aceste tipuri de date suportă numai date *Unicode*.

*Unicode* este o mulțime de caractere globale care permite stocarea de informație în orice limbă, folosind o mulțime unică de caractere.

Prin urmare, *unicode* furnizează o valoare cod unică pentru fiecare caracter, indiferent de platformă, program sau limbă.

## Variabile nespecifice *PL/SQL*

Variabila de legătură (*bind*) se declară într-un mediu gazdă şi este folosită pentru transferul (la execuție) valorilor numerice sau de tip caracter în/din unul sau mai multe programe *PL/SQL*. Variabilele declarate în mediul gazdă sau în cel apelant pot fi referite în instrucțiuni *PL/SQL*, dacă acestea nu sunt în cadrul unei proceduri, funcții sau pachet.

În *SQL*Plus*, variabilele de legătură se declară folosind comanda *VARIABLE*, iar pentru afişarea valorilor acestora se utilizează comanda *PRINT*. Ele sunt referite prin prefixarea cu simbolul „:“, pentru a putea fi deosebite de variabilele declarate în *PL/SQL*.

Deoarece instrucțiunile *SQL* pot fi integrate în programe *C*, este necesar un mecanism pentru a transfera valori între mediul de programare *C* şi instrucțiunile *SQL* care comunică cu *server*-ul bazei de date *Oracle*. În acest scop, în programul

încapsulat sunt definite variabilele gazdă (*host*). Acestea sunt declarate între directivele *BEGIN DECLARE SECTION* şi *END DECLARE SECTION* ale preprocesorului.

O valoare *null* în baza de date nu are o valoare corespunzătoare în mediul limbajului gazdă (de exemplu, limbajul *C*). Pentru a rezolva problema comunicării valorilor *null* între programul scris în limbaj gazdă şi sistemul *Oracle*, au fost definite variabilele indicator. Acestea sunt variabile speciale de tip întreg, folosite pentru a indica dacă o valoare *null* este recuperată (extrasă) din baza de date sau stocată în aceasta. Ele au următoarea formă:

:*nume_extern* [: *indicator*]

De exemplu, dacă atribuirea este făcută de limbajul gazdă, valoarea –1 a indicatorului specifică faptul că *PL/SQL* trebuie să înlocuiască valoarea variabilei prin *null*, iar o valoare a indicatorului mai mare ca zero precizează că *PL/SQL* trebuie să considere chiar valoarea variabilei.

## Declararea variabilelor

Identificatorii *PL/SQL* trebuie declarați înainte de a fi referiți în blocul *PL/SQL*. Dacă în declarația unei variabile apar referiri la alte variabile, acestea trebuie să fi fost declarate anterior. Orice variabilă declarată într-un bloc este accesibilă blocurilor conținute sintactic în acesta.

Tipurile scalare sunt predefinite în pachetul *STANDARD*. Pentru a folosi un astfel de tip într-un program este suficient să fie declarată o variabilă de tipul respectiv.

Tipurile compuse sunt definite de utilizator. Prin urmare, în acest caz trebuie definit efectiv tipul şi apoi declarată variabila de tipul respectiv.

În declararea variabilelor pot fi utilizate atributele *%TYPE* şi *%ROWTYPE*, care reprezintă tipuri de date implicite. Aceste tipuri permit declararea unei variabile în concordanță cu declarații de variabile făcute anterior.

Atributul *%TYPE* permite definirea unei variabile având tipul unei variabile declarate anterior sau tipul unei coloane dintr-un tabel.

Atributul *%ROWTYPE* permite definirea unei variabile având tipul unei înregistrări dintr-un tabel. Avantajul utilizării acestui atribut constă în faptul că nu este necesar să se cunoască numărul şi tipurile coloanelor tabelului. Elementele individuale ale acestei structuri de tip înregistrare sunt referite în maniera clasică, prefixând numele coloanei cu numele variabilei declarate.

Calitatea atributelor *%TYPE* şi *%ROWTYPE* constă în faptul că simplifică întreținerea codului *PL/SQL*. De exemplu, poate fi modificată dimensiunea unei coloane, fără să fie necesară modificarea declarației variabilelor al căror tip s-a

definit făcând referință la tipul coloanei respective.

Sintaxa declarării unei variabile este următoarea:

*identificator* [***CONSTANT***] {*tip_de_date* | *identificator****%TYPE*** |

*identificator****%ROWTYPE***} [***NOT NULL***]

[ {:= | ***DEFAULT***} *expresie_PL/SQL*];

Se pot defini constante (valoarea stocată nu poate fi modificată) prin specificarea la declarare a cuvântului cheie *CONSTANT*.

***Exemplu*:**

```
v_valoare         NUMBER(15) NOT NULL := 0;
v_data_achizitie  DATE DEFAULT SYSDATE;
v_material        VARCHAR2(15) := 'Matase';
c_valoare         CONSTANT NUMBER := 100000;
v_stare           VARCHAR2(20) DEFAULT 'Buna';
v_clasificare     BOOLEAN  DEFAULT FALSE;
v_cod_opera       opera.cod_opera%TYPE;
v_opera           opera%ROWTYPE;
int_an_luna       INTERVAL YEAR TO MONTH :=
                    INTERVAL '3-2' YEAR TO MONTH;
```

***Observații*:**

1. Pentru a denumi o variabilă este utilizată frecvent (pentru uşurința referirii) prefixarea cu litera *v* (*v_identificator*), iar pentru o constantă este folosită prefixarea cu litera *c* (*c_identificator*).
2. Variabilele pot fi inițializate, iar dacă o variabilă nu este inițializată, valoarea implicită a acesteia este *null*. Dacă o variabilă este declarată *NOT NULL*, atunci ea va fi obligatoriu inițializată.
3. Pentru a inițializa o variabilă sau o constantă poate fi utilizată o expresie *PL/SQL* compatibilă ca tip cu variabila sau constanta respectivă.
4. Constantele trebuie inițializate când sunt declarate, altfel apare o eroare la compilare.
5. În secțiunea declarativă, pe fiecare linie, există o singură declarație de variabilă.
6. Două obiecte (variabile) pot avea acelaşi nume cu condiția să fie definite în blocuri diferite. Dacă ele coexistă, poate fi folosit doar obiectul declarat în blocul curent.
7. Atributul *%ROWTYPE* nu poate include clauze de inițializare.

## Definirea subtipurilor

Subtipurile derivă dintr-un tip de bază, la care se adaugă anumite restricții. De exemplu, *NATURAL* este un subtip predefinit *PL/SQL*, derivat din tipul de bază *BINARY_INTEGER*, cu restricția că permite prelucrarea valorilor întregi nenegative.

Prin urmare, un subtip nu reprezintă un nou tip de date, ci un tip existent asupra căruia se aplică anumite constrângeri. Subtipurile presupun acelaşi set de operații ca şi tipul de bază, dar aplicate unui subset de valori al acestui tip.

Sistemul *Oracle* permite ca utilizatorul să-şi definească propriile sale tipuri şi subtipuri de date în partea declarativă a unui bloc *PL/SQL*, subprogram sau pachet utilizând sintaxa:

***SUBTYPE*** *nume_subtip* ***IS*** *tip_de_baza* [***NOT NULL***];

În dicționarul datelor există vizualizări care furnizează informații despre tipurile de date create de utilizator (*USER_TYPES, USER_TYPE_ATTRS*).

## Conversii între tipuri de date

Există două tipuri de conversii:

implicite;

explicite.

*PL/SQL* face automat conversii implicite între caractere şi numere sau între caractere şi date calendaristice. Chiar dacă sistemul realizează automat aceste conversii, în practică se utilizează frecvent funcții de conversie explicită.

Funcțiile de conversie explicită din *SQL* sunt utilizabile şi în *PL/SQL*. *A*cestea sunt: *TO_NUMBER, TO_CHAR, TO_DATE, TO_MULTI_BYTE, TO_SINGLE_BYTE, CHARTOROWID, ROWIDTOCHAR, RAWTOHEX, HEXTORAW, TO_CLOB, TO_LOB.*

În *Oracle9i* se pot folosi următoarele funcții de conversie: *ASCIISTR, BIN_TO_NUM, NUMTODSINTERVAL, TO_TIMESTAMP, TO_YMINTERVAL, TO_NCHAR, TO_NCLOB, TO_TIMESTAMP_TZ, NUMTOYMINTERVAL, TO_DSINTERVAL, REFTOHEX, RAWTOHEX, RAWTONHEX, FROM_TZ, ROWIDTONCHAR, COMPOSE, DECOMPOSE.*

Denumirile acestor funcții reflectă posibilitățile pe care le oferă. De

exemplu, *TO_YMINTERVAL* converteşte argumentele sale la tipul *INTERVAL YEAR TO MONTH* conform unui format specificat. Funcţia *COMPOSE* converteşte un şir de caractere la un şir *unicode* (asociază o valoare cod unică pentru fiecare simbol din şir).

## Înregistrări

Tipul *RECORD* oferă un mecanism pentru prelucrarea înregistrărilor. Înregistrările au mai multe câmpuri ce pot fi de tipuri diferite, dar care sunt legate din punct de vedere logic.

Inregistrările trebuie definite în doi paşi:

- se defineşte tipul *RECORD*;
- se declară înregistrările de acest tip.

Declararea tipului *RECORD* se face conform următoarei sintaxe:

***TYPE*** *nume_tip* ***IS RECORD***
(*nume_câmp1* {*tip_câmp* | *variabilă****%TYPE*** |
*nume_tabel.coloană****%TYPE*** | *nume_tabel****%ROWTYPE***}
[ [***NOT NULL***] {:= | ***DEFAULT***} *expresie1*],
(*nume_câmp2* {*tip_câmp* | *variabilă****%TYPE*** |
*nume_tabel.coloană****%TYPE*** | *nume_tabel****%ROWTYPE***}
[ [***NOT NULL***] {:= | ***DEFAULT***} *expresie2*],…);

Identificatorul *nume_tip* reprezintă numele tipului *RECORD* care se va specifica în declararea înregistrărilor, *nume_câmp* este numele unui câmp al înregistrării, iar *tip_câmp* este tipul de date al câmpului.

***Observaţii*:**

- Dacă un câmp nu este iniţializat atunci implicit se consideră că are valoarea *NULL*. Dacă s-a specificat constrângerea *NOT NULL*, atunci obligatoriu câmpul trebuie iniţializat cu o valoare diferită de *NULL*.
- Pentru referirea câmpurilor individuale din înregistrare se prefixează numele câmpului cu numele înregistrării.
- Pot fi asignate valori unei înregistrări utilizând comenzile *SELECT*, *FETCH* sau instrucţiunea clasică de atribuire. De asemenea, o înregistrare poate fi asignată altei înregistrari de acelaşi tip.

- Componentele unei înregistrări pot fi de tip scalar, *RECORD, TABLE,* obiect, colecție (dar, nu tipul *REF CURSOR*).
- *PL/SQL* permite declararea şi referirea înregistrărilor imbricate.
- Numărul de câmpuri ale unei înregistrări nu este limitat.
- Înregistrările nu pot fi comparate (egalitate, inegalitate sau *null*).
- Înregistrările pot fi parametri în subprograme şi pot să apară în clauza *RETURN* a unei funcții.

Diferența dintre atributul *%ROWTYPE* şi tipul de date compus *RECORD:*

- tipul *RECORD* permite specificarea tipului de date pentru câmpuri şi permite declararea câmpurilor sale;
- atributul *%ROWTYPE* nu cere cunoaşterea numărului şi tipurilor coloanelor tabloului.

*Oracle9i* introduce câteva facilități legate de acest tip de date.

- Se poate insera (*INSERT*) o linie într-un tabel utilizând o înregistrare. Nu mai este necesară listarea câmpurilor individuale, ci este suficientă utilizarea numelui înregistrării.
- Se poate reactualiza (*UPDATE*) o linie a unui tabel utilizând o înregistrare. Sintaxa *SET ROW* permite să se reactualizeze întreaga linie folosind conținutul unei înregistrări.
- Într-o înregistrare se poate regăsi şi returna sau şterge informația din clauza *RETURNING* a comenzilor *UPDATE* sau *DELETE*.
- Dacă în comenzile *UPDATE* sau *DELETE* se modifică mai multe linii, atunci pot fi utilizate în sintaxa *BULK COLLECT INTO*, colecții de înregistrări.

***Exemplu*:**

Exemplul următor arată modul în care poate să fie utilizată o înregistrare în clauza *RETURNING* asociată comenzii *DELETE*.

```
DECLARE
  TYPE val_opera IS RECORD (
    cheie  NUMBER,
    val    NUMBER);
  v_info_valoare  val_opera;
```

```
BEGIN
  DELETE FROM opera
      WHERE cod_opera = 753
    RETURNING cod_opera, valoare
      INTO v_info_valoare;
  …
END;
```

## Colecţii

Uneori este preferabil să fie prelucrate simultan mai multe variabile de acelaşi tip. Tipurile de date care permit acest lucru sunt **colecţiile**. Fiecare element are un indice unic, care determină poziţia sa în colecţie.

*Oracle7* a furnizat tipul *index-by table*, iniţial numit *PL/SQL table* datorită asemănării sale cu structura tabelelor relaţionale.

*Oracle8* a introdus două tipuri colecţie, *nested table* şi *varray*. *Oracle9i* permite crearea de colecţii pe mai multe niveluri, adică colecţii de colecţii.

În *PL/SQL* există trei tipuri de colecţii:

- tablouri indexate (*index-by tables*);
- tablouri imbricate (*nested tables*);
- vectori (*varrays* sau *varying arrays*).

Tipul *index-by table* poate fi utilizat **numai** în declaraţii *PL/SQL*. Tipurile *varray* şi *nested table* pot fi utilizate atât în declaraţii *PL/SQL*, cât şi în declaraţii la nivelul schemei (de exemplu, pentru definirea tipului unei coloane a unui tabel relaţional).

***Exemplu*:**

În exemplul care urmează sunt ilustrate cele trei tipuri de colecţii.

```
DECLARE
  TYPE  tab_index IS TABLE OF NUMBER
    INDEX BY BINARY_INTEGER;
  TYPE  tab_imbri IS TABLE OF NUMBER;
  TYPE  vector IS VARRAY(15) OF NUMBER;
  v_tab_index  tab_index;
  v_tab_imbri  tab_imbri;
  v_vector     vector;
BEGIN
  v_tab_index(1) := 72;
```

```
  v_tab_index(2) := 23;
  v_tab_imbri := tab_imbri(5, 3, 2, 8, 7);
  v_vector := vector(1, 2);
END;
```

***Observații*:**

- Deoarece colecțiile nu pot fi comparate (egalitate sau inegalitate), ele nu pot să apară în clauzele *DISTINCT, GROUP BY, ORDER BY*.
- Tipul colecție poate fi definit într-un pachet.
- Tipul colecție poate să apară în clauza *RETURN* a unei funcții.
- Colecțiile pot fi parametri formali într-un subprogram.
- Accesul la elementele individuale ale unei colecții se face prin utilizarea unui indice.

## Tablouri indexate

Tipul de date *index-by table* oferă un mecanism pentru prelucrarea tablourilor. Tabloul indexat *PL/SQL* are două componente: o coloană ce cuprinde cheia primară pentru acces la liniile tabloului şi o coloană care include valoarea efectivă a elementelor tabloului.

*Oracle7* asigură definirea tablourilor de înregistrări care pot fi declarate şi utilizate numai în programe *PL/SQL*, *Oracle8* realizează definirea tablourilor de tipuri obiect, iar *Oracle9i* permite definirea tablourilor de colecții.

În *Oracle9i* tipul *index-by table* este redenumit *associative array* pentru compatibilitate (de limbaj) cu termenul folosit în alte limbaje de programare (*C++*, *JavaScript*, *PHP, Perl*) pentru a defini această structură de date.

Tablourile indexate *PL/SQL* trebuie definite în doi paşi: se defineşte tipul *TABLE*; se declară tabloul indexat *PL/SQL* de acest tip.

Declararea tipului *TABLE* se face respectând următoarea sintaxă:

***TYPE** nume_tip **IS TABLE OF***
{*tip_coloană* | *variabilă***%TYPE** |
*nume_tabel.coloană***%TYPE** [***NOT NULL***] |
*nume_tabel***%ROWTYPE**}
***INDEX BY** tip_indexare*;

Identificatorul *nume_tip* este numele noului tip definit care va fi specificat în declararea tabloului *PL/SQL*, iar *tip_coloană* este un tip scalar simplu (de exemplu, *VARCHAR2*, *CHAR*, *DATE* sau *NUMBER*).

Până la versiunea *Oracle9i* unicul tip de indexare acceptat era *INDEX BY BINARY_INTEGER*. *Oracle9i* permite următoarele opțiuni pentru *tip_indexare*: *PLS_INTEGER, NATURAL, POSITIVE, VARCHAR2*(*n*) sau chiar indexarea după un tip declarat cu *%TYPE*. Nu sunt permise indexările *INDEX BY NUMBER, INDEX BY INTEGER, INDEX BY DATE, INDEX BY VARCHAR2, INDEX BY CHAR*(*n*) sau indexarea după un tip declarat cu *%TYPE* în care intervine unul dintre tipurile enumerate anterior.

***Observații*:**

- Elementele unui tablou indexat nu sunt într-o ordine particulară şi pot fi inserate cu chei arbitrare.
- Deoarece nu există constrângeri de dimensiune, dimensiunea tabloului se modifică dinamic.
- Tabloul indexat *PL/SQL* nu poate fi inițializat în declararea sa.
- Un tablou indexat neinițializat este vid (nu conține nici valori, nici chei).
- Un element al tabloului este nedefinit atâta timp cât nu are atribuită o valoare efectivă.
- Inițial, un tablou indexat este nedens. După declararea unui tablou se poate face referire la liniile lui prin precizarea valorii cheii primare.
- Dacă se face referire la o linie care nu există, atunci se produce excepția *NO_DATA_FOUND*.
- Dacă se doreşte contorizarea numărului de linii, trebuie declarată o variabilă în acest scop sau poate fi utilizată o metodă asociată tabloului.
- Deoarece numărul de linii nu este limitat, operația de adăugare de linii este restricționată doar de dimensiunea memoriei alocate.
- Tablourile pot să apară ca argumente într-o procedură.

Pentru inserarea unor valori din tablourile *PL/SQL* într-o coloană a unui tabel de date se utilizează instrucțiunea ***INSERT*** în cadrul unei secvențe repetitive *LOOP*.

Asemănător, pentru regăsirea unor valori dintr-o coloană a unei baze de date într-un tablou *PL/SQL* se utilizează instrucțiunea ***FETCH*** (cursoare) sau instrucțiunea de atribuire în cadrul unei secvențe repetitive *LOOP*.

Pentru a şterge liniile unui tablou fie se asignează elementelor tabloului valoarea *null*, fie se declară un alt tablou *PL/SQL* (de acelaşi tip) care nu este inițializat şi acest tablou vid se asignează tabloului *PL/SQL* care trebuie şters. În *PL/SQL* 2.3 ştergerea liniilor unui tabel se poate face utilizând metoda *DELETE*.

***Exemplu*:**
Să se definească un tablou indexat *PL/SQL* având elemente de tipul *NUMBER*. Să se introducă 20 de elemente în acest tablou. Să se şteargă tabloul.

```
DECLARE
  TYPE  tablou_numar IS TABLE OF NUMBER
    INDEX BY PLS_INTEGER;
  v_tablou  tablou_numar;
BEGIN
  FOR i IN 1..20 LOOP
    v_tablou(i) := i*i;
    DBMS_OUTPUT.PUT_LINE(v_tablou(i));
  END LOOP;
  --v_tablou := NULL;
  --aceasta atribuire da eroarea PLS-00382
  FOR i IN v_tablou.FIRST..v_tablou.LAST LOOP
    v_tablou(i) := NULL;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('tabloul are ' || v_tablou.COUNT ||
                       ' elemente');
END;
```

În *PL/SQL* este folosit frecvent tipul tablou de înregistrări. Referirea la un element al tabloului se face prin forma clasică: *tabel*(*index*).*câmp*.

***Exemplu*:**
Să se definească un tablou de înregistrări având tipul celor din tabelul *organizator*. Să se iniţializeze un element al tabloului şi să se introducă în tabelul *organizator*. Să se şteargă elementele tabloului.

```
DECLARE
  TYPE org_table_type IS TABLE OF organizator%ROWTYPE
    INDEX BY BINARY INTEGER;
  org_table  org_table_type;
  i          NUMBER;
BEGIN
  IF org_table.COUNT <>0 THEN
    i := org_table.LAST+1;
    ELSE i:=1;
  END IF;
  org_table(i).cod_org := 752;
  org_table(i).nume := 'Grigore Ion';
  org_table(i).adresa := 'Calea Plevnei 18 Sibiu';
  org_table(i).tip := 'persoana fizica';
  INSERT INTO organizator
  VALUES (org_table(i).cod_org, org_table(i).nume,
```

```
          org_table(i).adresa, org_table(i).tip);
  -- sau folosind noua facilitate Oracle9i
  -- INSERT INTO organizator
  -- VALUES (org_table(i));
  org_table.DELETE; -- sterge toate elementele
  DBMS_OUTPUT.PUT_LINE('Dupa aplicarea metodei DELETE
      sunt '||TO_CHAR(org_table.COUNT)||' elemente');
END;
```

## Vectori

Vectorii (*varray*) sunt structuri asemănătoare vectorilor din limbajele *C* sau *Java*. Spre deosebire de tablourile indexate, vectorii au o dimensiune maximă (constantă) stabilită la declarare. În special, se utilizează pentru modelarea relațiilor *one-to-many*, atunci când numărul maxim de elemente din partea „*many*“ este cunoscut şi ordinea elementelor este importantă.

Vectorii reprezintă **structuri dense**. Fiecare element are un index care dă poziția sa în vector şi care este folosit pentru accesarea elementelor particulare. Limita inferioară a indicelui este 1. Vectorul poate conține un număr variabil de elemente, de la 0 (vid) la numărul maxim specificat obligatoriu în definiția sa.

Tipul de date vector este declarat utilizând sintaxa:

***TYPE*** *nume_tip* ***IS***
{***VARRAY*** | ***VARYING ARRAY***} (*lungime_maximă*)
***OF*** *tip_elemente* [***NOT NULL***];

Identificatorul *nume_tip* este numele tipului de date vector, iar *lungime_maximă* reprezintă numărul maxim de elemente din vector. *Tip_elemente* este un tip scalar *PL/SQL*, tip înregistrare sau tip obiect. De asemenea, acest tip poate fi definit utilizând atributele *%TYPE* sau *%ROWTYPE*.

În *Oracle9i* sunt permise (pentru *tip_elemente*) tipurile *TABLE* sau alt tip *VARRAY*. Există restricții referitoare la tipul elementelor, în sensul că acesta nu poate să fie *BOOLEAN*, *NCHAR*, *NCLOB*, *NVARCHAR2*, *REF CURSOR*, *PLS_INTEGER*, *LONG*, *LONG RAW*, *NATURAL*, *NATURALN*, *POSITIVE*, *POSITIVEN*, *BINARY_INTEGER*, *SIGNTYPE*, *STRING*, tip obiect cu atribute *TABLE* sau *VARRAY*, *BLOB*, *CLOB*, tip obiect cu atribute *BLOB* sau *CLOB*.

***Exemplu*:**

```
DECLARE
  TYPE secventa IS VARRAY(5) OF VARCHAR2(10);
  v_sec  secventa := secventa ('alb', 'negru', 'rosu',
                               'verde');
BEGIN
```

```
  v_sec (3) := 'rosu';
  v_sec.EXTEND; -- adauga un element null
  v_sec(5) := 'albastru';
  -- extinderea la 6 elemente va genera eroarea ORA-06532
  v_sec.EXTEND;
END;
```

## Tablouri imbricate

Tablourile imbricate (*nested table*) sunt tablouri indexate a căror dimensiune nu este stabilită. Numărul maxim de linii ale unui tablou imbricat este dat de capacitatea maximă 2 *GB.*

Un tablou imbricat este o mulțime neordonată de elemente de acelaşi tip. Valorile de acest tip:

-- pot fi stocate în baza de date,

-- pot fi prelucrate direct în instrucțiuni *SQL*

-- au excepții predefinite proprii.

Sistemul *Oracle* nu stochează liniile unui tablou imbricat într-o ordine particulară. Dar, când se regăseşte tabloul în variabile *PL/SQL*, liniile vor avea indici consecutivi începând cu valoarea 1. Inițial, aceste tablouri sunt structuri dense, dar se poate ca în urma prelucrării să nu mai aibă indici consecutivi.

Comanda de declarare a tipului de date tablou imbricat are sintaxa:

***TYPE*** *nume_tip* ***IS TABLE OF*** *tip_ elemente* [***NOT NULL***];

Identificatorul *nume_tip* reprezintă numele noului tip de date tablou imbricat, iar *tip_elemente* este tipul fiecărui element din tabloul imbricat, care poate fi un tip definit de utilizator sau o expresie cu *%TYPE*, respectiv *%ROWTYPE*.

În *Oracle9i* sunt permise (pentru *tip_elemente*) tipurile *TABLE* sau alt tip *VARRAY*. Există restricții referitoare la tipul elementelor, în sensul că acesta nu poate să fie *BOOLEAN*, *STRING*, *NCHAR*, *NCLOB*, *NVARCHAR2*, *REF CURSOR*, *BINARY_INTEGER*, *PLS_INTEGER*, *LONG*, *LONG RAW*, *NATURAL*, *NATURALN*, *POSITIVE*, *POSITIVEN*, *SIGNTYPE*, tip obiect cu atributele *TABLE* sau *VARRAY.*

Tabloul imbricat are o singură coloană, iar dacă aceasta este de tip obiect, tabloul poate fi vizualizat ca un tabel multicoloană, având câte o coloană pentru fiecare atribut al tipului obiect.

***Exemplu*:**

```
DECLARE
  TYPE numartab IS TABLE OF NUMBER;
  -- se creeaza un tablou cu un singur element
```

```
  v_tab_1  numartab := numartab(-7);
  -- se creeaza un tablou cu 4 elemente
  v_tab_2  numartab := numartab(7,9,4,5);
  -- se creeaza un tablou fara nici un element
  v_tab_3  numartab := numartab();
BEGIN
  v_tab_1(1) := 57;
  FOR j IN 1..4 LOOP
    DBMS_OUTPUT.PUT_LINE (v_tab_2(j) || ' ');
  END LOOP;
END;
```

Se observă că singura diferență sintactică între tablourile indexate şi cele imbricate este absența clauzei *INDEX BY BINARY_INTEGER*. Mai exact, dacă această clauză lipseşte, tipul este tablou imbricat.

***Observații*:**

- Spre deosebire de tablourile indexate, vectorii şi tablourile imbricate pot să apară în definirea tabelelor bazei de date.
- Tablourile indexate pot avea indice negativ, domeniul permis pentru index fiind –2147483647..2147483647, iar pentru tabele imbricate domeniul indexului este 1..2147483647.
- Tablourile imbricate, spre deosebire de tablourile indexate, pot fi prelucrate prin comenzi *SQL*.
- Tablourile imbricate trebuie inițializate şi/sau extinse pentru a li se adăuga elemente.

Când este creat un tablou indexat care nu are încă elemente, el este vid. Dacă un tablou imbricat (sau un vector) este declarat, dar nu are încă nici un element (nu este inițializat), el este automat inițializat (atomic) *null*. Adică, colecția este *null*, nu elementele sale. Prin urmare, pentru tablouri imbricate poate fi utilizat operatorul *IS NULL*. Dacă se încearcă **să se adauge** un element la un tablou imbricat *null*, se va genera eroarea „*ORA - 06531: reference to uninitialized collection*“ care corespunde excepției predefinite *COLLECTION_IS_NULL*.

Prin urmare, cum poate fi inițializat un tablou imbricat? Ca şi obiectele, vectorii şi tablourile imbricate sunt inițializate cu ajutorul **constructorului**. Acesta are acelaşi nume ca şi tipul colecției referite. *PL/SQL* apelează un constructor numai în mod explicit. Tabelele indexate nu au constructori.

Constructorul primeşte ca argumente o listă de valori de tip *tip_elemente*. Elementele sunt numerotate în ordine, de la 1 la numărul de valori date ca

parametrii constructorului. Dimensiunea inițială a colecției este egală cu numărul de argumente date în constructor, când aceasta este inițializată. Pentru vectori nu poate fi depăşită dimensiunea maximă precizată la declarare. Atunci când constructorul este fără argumente, va crea o colectie fără nici un element (vida), dar care are valoarea *not null*. Exemplul următor este concludent în acest sens.

***Exemplu*:**

```
DECLARE
  TYPE  alfa IS TABLE OF VARCHAR2(50);
  -- creeaza un tablou (atomic) null
  tab1  alfa ;
  /* creeaza un tablou cu un element care este null, dar
     tabloul nu este null, el este initializat, poate
     primi elemente */
  tab2  alfa := alfa() ;
BEGIN

  IF tab1 IS NULL THEN
    DBMS_OUTPUT.PUT_LINE('tab1 este NULL');
  ELSE
    DBMS_OUTPUT.PUT_LINE('tab1 este NOT NULL');
  END IF;
  IF tab2 IS NULL THEN
    DBMS_OUTPUT.PUT_LINE('tab2 este NULL');
  ELSE
    DBMS_OUTPUT.PUT_LINE('tab2 este NOT NULL');
  END IF;
END;
```

În urma execuției acestui bloc se obține următorul rezultat:

```
tab1 este NULL
tab2 este NOT NULL
```

Excepțiile semnificative care apar în cazul utilizării incorecte a colecțiilor:

***Exemplu:***

```
DECLARE
  TYPE  numar IS TABLE OF INTEGER;
  alfa  numar;
BEGIN
  alfa(1) := 77;
  -- declanseaza exceptia COLLECTION_IS_NULL
  alfa := numar(15, 26, 37);
```

```
  alfa(1) := ASCII('X');
  alfa(2) := 10*alfa(1);
  alfa('P') := 77;
  /* declanseaza exceptia VALUE_ERROR deoarece indicele
     nu este convertibil la intreg */
  alfa(4) := 47;
  /* declanseaza exceptia SUBSCRIPT_BEYOND_COUNT deoarece
     indicele se refera la un element neinitializat */
  alfa(null) := 7; -- declanseaza exceptia VALUE_ERROR
  alfa(0) := 7; -- exceptia SUBSCRIPT_OUTSIDE_LIMIT
  alfa.DELETE(1);
  IF alfa(1) = 1 THEN … -- exceptia NO_DATA_FOUND
  …
END;
```

Tablourile imbricate şi vectorii pot fi utilizați drept câmpuri în tabelele bazei. Aceasta presupune că fiecare înregistrare din tabelul respectiv conține un obiect de tip colecție. Înainte de utilizare, tipul trebuie stocat în dicționarul datelor, deci trebuie declarat prin comanda:

***CREATE TYPE*** *nume_tip* ***AS*** {***TABLE*** | ***VARRAY***} ***OF*** *tip_elemente*;

După crearea tabelului (prin comanda *CREATE TABLE*), pentru fiecare câmp de tip tablou imbricat din tabel este necesară clauza de stocare:

***NESTED TABLE*** *nume_câmp* ***STORE AS*** *nume_tabel*;

## Colecții pe mai multe niveluri

În *Oracle9i* se pot construi colecții pe mai multe niveluri (*multilevel collections*), prin urmare colecții ale căror elemente sunt, în mod direct sau indirect, colecții. În felul acesta pot fi definite structuri complexe: vectori de vectori, vectori de tablouri imbricate, tablou imbricat de vectori, tablou imbricat de tablouri imbricate, tablou imbricat sau vector de un tip definit de utilizator care are un atribut de tip tablou imbricat sau vector.

Aceste structuri complexe pot fi utilizate ca tipuri de date pentru definirea:

- coloanelor unui tabel relațional,
- atributelor unui obiect într-un tabel obiect,
- variabilelor *PL/SQL*.

***Observații:***

- Numărul nivelurilor de imbricare este limitat doar de capacitatea de stocare a sistemului.
- Pentru a accesa un element al colecţiei incluse sunt utilizate două seturi de paranteze.
- Obiectele de tipul colecţie pe mai multe niveluri nu pot fi comparate.

***Exemplu*:**
În exemplele care urmează sunt definite trei structuri complexe şi sunt prezentate câteva modalităţi de utilizare ale acestora. Exemplele se referă la vectori pe mai multe niveluri, tablouri imbricate pe mai multe niveluri şi tablouri indexate pe mai multe niveluri.

```
DECLARE
  TYPE  alfa IS VARRAY(10) OF INTEGER;
  TYPE  beta IS VARRAY(10) OF alfa;
  valf  alfa := alfa(12,31,5); --initializare
  vbet  beta := beta(valf,alfa(55,6,77),alfa(2,4),valf);
  i     integer;
  var1  alfa;
BEGIN
  i := vbet(2)(3); -- i va lua valoarea 77
  vbet.EXTEND; -- se adauga un element de tip vector la vbet
  vbet(5) := alfa(56,33);
  vbet(4) := alfa(44,66,77,4321);
  vbet(4)(4) := 7; -- 4321 este inlocuit cu 7
  vbet(4).EXTEND; -- se adauga un element la al 4-lea element
  vbet(4)(5) := 777; -- acest nou element adaugat va fi 777
END;
/
DECLARE
  TYPE  gama IS TABLE OF VARCHAR2(20);
  TYPE  delta IS TABLE OF gama;
  TYPE  teta IS VARRAY(10) OF INTEGER;
  TYPE  epsi IS TABLE OF teta;
  var1  gama := gama('alb','negru');
  var2  delta := delta(var1);
  var3  epsi := epsi(teta(31,15),teta(1,3,5));
BEGIN
  var2.EXTEND;
  var2(2) := var2(1);
  var2.DELETE(1); -- sterge primul element din var2
  /* sterge primul sir de caractere din al doilea
```

```
     tablou al tabloului imbricat */
  var2(2).DELETE(1);
END;
/
DECLARE
  TYPE  alfa IS TABLE OF INTEGER INDEX BY BINARY_INTEGER;
  TYPE  beta IS TABLE OF alfa INDEX BY BINARY_INTEGER;
  TYPE  gama IS VARRAY(10) OF VARCHAR2(30);
  TYPE  delt IS TABLE OF gama INDEX BY BINARY_INTEGER;
  var1  gama := gama('alb','negru');
  var2  beta;
  var3  delt;
  var4  alfa;
  var5  alfa; -- tablou vid
BEGIN
  var4(1) := 324;
  var4(2) := 222;
  var4(42) := 333;
  var2(27) := var4;
  var3(39) := gama(77,76,89,908);
  -- var2(40)(3) := 55; eroare nu exista element 40 in var2
  var2(40) := var5; -- asignez un tablou null
  var2(40)(3) := 55; -- corect
END;
/
```

**Prelucrarea colecțiilor**

O colecție poate fi exploatată fie în întregime (atomic) utilizând comenzi *LMD*, fie pot fi prelucrate elemente individuale dintr-o colecție (*piecewise updates*) utilizând operatori *SQL* sau anumite facilități oferite de *PL/SQL*.

- Comanda *INSERT* permite inserarea unei colecții într-o linie a unui tabel. Colecția trebuie să fie creată şi inițializată anterior.
- Comanda *UPDATE* este folosită pentru modificarea unei colecții stocate.
- Comanda *DELETE* poate şterge o linie ce conține o colecție.
- Colecțiile din baza de date pot fi regăsite în variabile *PL/SQL*, utilizând comanda *SELECT*.

***Exemplu*:**

```
CREATE OR REPLACE TYPE operalist AS VARRAY(10) OF
                                    NUMBER(4);

CREATE TABLE gal_ope (
```

```
  cod_galerie   NUMBER(10),
  nume_galerie  VARCHAR2(20),
  info          operalist);
DECLARE
  v_opera    operalist := operalist (777, 888, 999);
  v_info_op  operalist := operalist (7007);
  v_info     gal_ope.info%TYPE;
  v_cod      gal_ope.cod_galerie%TYPE := 2345;
BEGIN
  INSERT INTO gal_ope
  VALUES (4567, 'Impresionisti', operalist(4567,4987));
  INSERT INTO gal_ope
  VALUES (2345, 'Cubism', v_opera);
  INSERT INTO gal_ope
  VALUES (123, 'Alfa', v_info_op);
  SELECT info
  INTO   v_info
  FROM   gal_ope
  WHERE  cod_galerie = v_cod;
END;
```

Un vector stocat într-un tabel este prelucrat ca un întreg (nu pot fi modificate elemente individuale). Prin urmare, elementele individuale ale unui vector nu pot fi referite în comenzile *INSERT*, *UPDATE* sau *DELETE*. Pentru referirea acestora trebuie utilizate comenzi procedurale *PL/SQL*. Pentru a modifica un vector, el trebuie selectat într-o variabilă *PL/SQL* a cărei valoare poate fi modificată şi apoi reinserată în tabel.

Tablourile imbricate depuse în baza de date sunt mai flexibile, deoarece pot fi prelucrate fie în întregime, fie ca elemente individuale. În fiecare caz pot fi utilizate **numai** comenzi *SQL*.

Se pot face reactualizări sau inserări asupra tablourilor imbricate care dau o valoare nouă pentru întreaga colecție sau se pot face inserări, ştergeri, reactualizări de elemente particulare din colecție.

O colecție poate fi asignată altei colecții prin comenzile *INSERT*, *UPDATE*, *FETCH*, *SELECT*, instrucțiunea de atribuire sau prin apelul unui subprogram, dar colecțiile trebuie să fie de acelaşi tip. Dacă unei colecții i se asignează o colecție atomic *null*, aceasta devine atomic *null* şi trebuie reinițializată.

În *Oracle8i* a fost introdus operatorul *TABLE*, ce permite **prelucrarea elementelor unui tablou imbricat care este stocat într-un tabel**. Operatorul permite interogarea unei colecții în clauza *FROM* (la fel ca un tabel).

Operandul lui *TABLE* este:

- fie numele unei colecții şi atunci rezultatul operatorului este tot o colecție,
- fie este o subinterogare referitoare la o colecție, iar în acest caz, operatorul *TABLE* returnează o singură valoare (coloană) care este un tablou imbricat sau un vector. Prin urmare, lista din clauza *SELECT* a subcererii trebuie să aibă un singur articol.

***Exemplu*:**

Se presupune că tabelul *opera* are o coloană *info* de tip tablou imbricat. Acest tablou are două componente în care pentru fiecare operă de artă sunt depuse numele articolului referitor la opera respectivă şi revista în care a apărut. Să se insereze o linie în tabelul imbricat.

```
INSERT INTO TABLE (SELECT info
                   FROM   opera
                   WHERE  titlu = 'Primavara')
VALUES ('Pictura moderna', 'Orizonturi');
```

Listarea codului fiecărei opere de artă şi a colecției articolelor referitoare la aceste opere de artă se face prin comanda:

```
SELECT  a.cod_opera, b.*
FROM    opera a, TABLE (a.info) b;
```

Pentru tablouri imbricate pe mai multe niveluri, operațiile *LMD* pot fi făcute atomic sau pe elemente individuale, iar pentru vectori pe mai multe niveluri, operațiile pot fi făcute numai atomic.

Pentru prelucrarea unei colecții locale se poate folosi şi operatorul *CAST*. *CAST* are forma sintactică:

***CAST*** (*nume_colecție* ***AS*** *tip_colecție*)

Operanzii lui *CAST* sunt o colecție declarată local (de exemplu, într-un bloc *PL/SQL* anonim) şi un tip colecție *SQL*. *CAST* converteşte colecția locală la tipul specificat. În felul acesta, o colecție poate fi prelucrată ca şi cum ar fi un tabel *SQL* al bazei de date.

## Metodele unei colecții

*PL/SQL* oferă subprograme numite metode (*methods*), care operează asupra unei colecții. Acestea pot fi apelate **numai din comenzi procedurale**, şi nu din *SQL*.

Metodele sunt apelate prin expresia:

*nume_colecţie.nume_metodă* [ (*parametri*) ]

Metodele care se pot aplica colecţiilor *PL/SQL* sunt următoarele:

***COUNT*** returnează numărul curent de elemente ale unei colecţii *PL/SQL*;

***DELETE*(*n*)** şterge elementul *n* dintr-o colecţie *PL/SQL*; *DELETE*(*m*, *n*) şterge toate elementele având indecşii între *m* şi *n*; *DELETE* şterge toate elementele unei colecţii *PL/SQL* (nu este validă pentru tipul *varrays*);

***EXISTS*(*n*)** returnează *TRUE* dacă există al *n*-lea element al unei colecţii *PL/SQL* (altfel, returnează *FALSE*, chiar dacă elementul este *null*);

***FIRST*, *LAST*** returnează indicele primului, respectiv ultimului element din colecţie;

***NEXT*(*n*)**, ***PRIOR*(*n*)** returnează indicele elementului următor, respectiv precedent celui de rang *n* din colecţie, iar dacă nu există un astfel de element returnează valoarea *null*;

***EXTEND*** adaugă elemente la sfârşitul unei colecţii: *EXTEND* adaugă un element *null* la sfârşitul colecţiei, *EXTEND*(*n*) adaugă *n* elemente *null*, *EXTEND*(*n*, *i*) adaugă *n* copii ale elementului de rang *i* (nu este validă pentru tipul *index-by tables*); nu poate fi utilizată pentru a iniţializa o colecţie atomic *null*;

***LIMIT*** returnează numărul maxim de elemente ale unei colecţii (cel de la declarare) pentru tipul vector şi *null* pentru tablouri imbricate (nu este validă pentru tipul *index-by tables*);

***TRIM*** şterge elementele de la sfârşitul unei colecţii: *TRIM* şterge ultimul element, *TRIM*(*n*) şterge ultimele *n* elemente (nu este validă pentru tipul *index-by tables*). Similar metodei *EXTEND*, metoda *TRIM* operează asupra dimensiunii interne a tabloului imbricat.

*EXISTS* este singura metodă care poate fi aplicată unei colecţii atomice *null*. Orice altă metodă declanşează excepţia *COLLECTION_IS_NULL*.

## *Bulk bind*

În exemplul care urmează, comanda *DELETE* este trimisă motorului *SQL* pentru fiecare iteraţie a comenzii *FOR*.

***Exemplu:***

```
DECLARE
  TYPE nume IS VARRAY(20) OF NUMBER;
  alfa nume := nume(10,20,70); -- coduri ale galeriilor
BEGIN
  FOR j IN alfa.FIRST..alfa.LAST
```

```
    DELETE FROM opera
    WHERE  cod_galerie = alfa (j);
  END LOOP;
END;
```

Pentru a realiza mai rapid această operație, ar trebui să existe posibilitatea de a şterge (prelucra) întreaga colecție şi nu elemente individuale. Tehnica care permite acest lucru este cunoscută sub numele ***bulk bind***.

În timpul compilării, compilatorul *PL/SQL* asociază identificatorii cu o adresă, un tip de date şi o valoare. Acest proces este numit **binding.**

Comenzile SQL din blocurile PL/SQL sunt trimise motorului SQL pentru a fi executate. Motorul SQL poate trimite înapoi date motorului PL/SQL (de exemplu, ca rezultat al unei interogări). De multe ori, datele care trebuie manipulate aparțin unei colecții, iar colecția este iterată printr-un ciclu FOR. Prin urmare, transferul (în ambele sensuri) între SQL şi PL/SQL are loc pentru fiecare linie a colecției.

Începând cu Oracle8i există posibilitatea ca toate liniile unei colecții să fie transferate simultan printr-o singură operație. Procedeul este numit bulk bind şi este realizat cu ajutorul comenzii FORALL, ce poate fi folosită cu orice tip de colecție.

**Comanda FORALL are sintaxa:**
**FORALL index IN lim_inf..lim_sup**
**comanda_sql;**

Motorul SQL execută comanda_sql o singură dată pentru toate valorile indexului. Comanda_sql este una din comenzile INSERT, UPDATE, DELETE care referă elementele uneia sau mai multor colecții. Variabila index poate fi referită numai în comanda FORALL şi numai ca indice de colecție.

În exemplul care urmează este optimizată problema anterioară, în sensul că instrucțiunea DELETE este trimisă motorului SQL o singură dată, pentru toate liniile colecției.

***Exemplu:***

```
DECLARE
  TYPE nume IS VARRAY(20) OF NUMBER;
  alfa nume := nume(10,20,70); -- coduri ale galeriilor
BEGIN
  …
  FORALL j IN alfa.FIRST..alfa.LAST
    DELETE FROM opera
```

```
   WHERE  cod_galerie = alfa (j);
END;
```

Pentru utilizarea comenzii *FORALL* este necesară respectarea următoarelor restricții:

- comanda poate fi folosită numai în programe *server-side*, altfel apare eroarea "*this feature is not supported in client-side programs*";
- comenziile *INSERT*, *UPDATE*, *DELETE* trebuie să refere cel puțin o colecție;
- toate elementele colecției din domeniul precizat trebuie să existe (dacă, de exemplu, un element a fost şters, atunci este semnalată o eroare);
- indicii colecțiilor nu pot să fie expresii şi trebuie să aibă valori continue.

***Exemplu:***

```
CREATE TABLE exemplu (x NUMBER, y NUMBER);
DECLARE
  TYPE  nume IS TABLE OF NUMBER;
  ttt   nume:= nume(8,10,12);
BEGIN
  FORALL i IN ttt.FIRST..ttt.LAST
    INSERT INTO exemplu
    VALUES(ttt(i), 100);  -- corect
  FORALL i IN 1..3
    INSERT INTO exemplu
    VALUES(7, 9);         -- exceptie nu e colectie
END;

  /
PL/SQL procedure successfully completed.
SQL> select * from exemplu;

         X          Y
---------- ----------
         8        100
        10        100
        12        100
```

```
 FORALL i IN gama.FIRST..gama.LAST
    DELETE FROM carte
    WHERE codel = gama(i+1);
  -- eroare dubla (expresie si >LAST)


DECLARE
  TYPE alfa IS TABLE OF NUMBER;
  xx   alfa := alfa(8,10,12);
BEGIN
  FORALL i IN xx.FIRST..xx.LAST
    DELETE FROM exemplu
    WHERE x = xx(i);  -- eroare
END;


PL/SQL procedure successfully completed.

SQL> select * from exemplu;

no rows selected
```

Dacă există o eroare în procesarea unei linii printr-o operație *LMD* de tip *bulk*, numai acea linie va fi *rollback*.

**Regăsirea rezultatului unei interogări în colecții** (înainte de a fi trimisă motorului *PL/SQL*) se poate obține cu ajutorul clauzei *BULK COLLECT*.

Clauza poate să apară în:

- comenzile *SELECT INTO* (cursoare implicite*)*,
- comenzile *FETCH INTO* (cursoare explicite),
- clauza *RETURNING INTO* a comenzilor *INSERT*, *UPDATE*, *DELETE*.

Clauza are următoarea sintaxă:

*...**BULK COLLECT INTO** nume_colecție* [,*nume_colecție*]...

```
DECLARE
 TYPE tip1 IS TABLE OF opera.cod_opera%TYPE;
 TYPE tip2 IS TABLE OF opera.titlu%TYPE;
```

```
 alfa  tip1;
 beta  tip2;
BEGIN
 …
 /* motorul SQL incarca in intregime coloanele
    cod_opera si titlu in tabelele imbricate,
    inainte de a returna
    tabelele motorului PL/SQL */
 SELECT cod_opera, titlu BULK COLLECT INTO alfa,beta
 FROM   opera;
 …
 /* daca exista n opere de arta in stare buna,
    atunci alfa va contine codurile celor n opere */
 DELETE FROM opera WHERE stare = 'buna'
 RETURNING cod_opera BULK COLLECT INTO alfa;
 …
END;
```

Acelasi exemplu pentru tabelul exemplu creat mai sus.
Inseram sase inregistrari (toate inreg. au fost sterse prin exemplul de mai sus)

```
DECLARE
TYPE  nume IS TABLE OF NUMBER;
    ttt   nume:= nume(8,10,12,14,12,8,12);
  BEGIN
    FORALL i IN ttt.FIRST..ttt.LAST
 INSERT INTO exemplu
VALUES(ttt(i), 100);
 end;
/
PL/SQL procedure successfully completed.

SQL> select * from exemplu;

         X          Y
---------- ----------
         8        100
        10        100
        12        100
        14        100
        12        100
```

```
         8        100
        12        100

7 rows selected.

SQL> DECLARE
  2    TYPE tip1 IS TABLE OF exemplu.x%TYPE;
  3    alfa  tip1;
  4  BEGIN
  5    SELECT x BULK COLLECT INTO alfa
  6    FROM  exemplu;
  7    DELETE FROM exemplu WHERE x=12
  8    RETURNING x BULK COLLECT INTO alfa;
  9  For I in alfa.first .. alfa.last loop
 10    Dbms_output.put_line('alfa('||I  ||  ')  =  '  ||
alfa(i));
 11  End loop;
 12  END;
 13  /
alfa(1) = 12
alfa(2) = 12
alfa(3) = 12
PL/SQL procedure successfully completed.
```

Comanda *FORALL* se poate combina cu clauza *BULK COLLECT*. Totuşi, trebuie subliniat că ele nu pot fi folosite simultan în comanda *SELECT*.

Motorul SQL incarca toate liniile unei coloane. Cum se poate limita numarul de linii procesate?

```
SQL> DECLARE
  2    TYPE alfa IS TABLE OF employees.salary%TYPE;
  3    xx   alfa;
  4  BEGIN
  5    SELECT salary BULK COLLECT INTO xx
  6    FROM   employees WHERE  ROWNUM <= 25;
  7  For I in xx.first .. xx.last LOOP
  8  DBMS_OUTPUT.PUT_LINE('XX('||I||')=' || xx(i));
  9  End loop;
 10  END;
 11  /
```

```
XX(1)=24000
XX(2)=17000
XX(3)=17000
XX(4)=9000
XX(5)=6000
XX(6)=4800
XX(7)=4800
XX(8)=4200
XX(9)=12000
XX(10)=9000
XX(11)=8200
XX(12)=7700
XX(13)=7800
XX(14)=6900
XX(15)=11000
XX(16)=3100
XX(17)=2900
XX(18)=2800
XX(19)=2600
XX(20)=2500
XX(21)=8000
XX(22)=8200
XX(23)=7900
XX(24)=6500
XX(25)=5800

PL/SQL procedure successfully completed.
```

# 4. Gestiunea cursoarelor în *PL/SQL*

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

Zona conţine informaţii necesare procesării comenzii, cum ar fi:

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

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

Există două tipuri de cursoare:

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

Atât cursoarele implicite cât şi 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 şi reprezintă numărul liniilor încărcate de cursor;
- %*FOUND*, care este de tip boolean şi ia valoarea *TRUE* dacă ultima operaţie de încărcare (*FETCH*) dintr-un cursor a avut succes (în cazul cursoarelor explicite) sau dacă instrucţiunea *SQL* a întors cel puţin o linie (în cazul cursoarelor implicite);
- %*NOTFOUND*, care este de tip boolean şi are semnificaţie opusă faţă de cea a atributului %*FOUND*;
- %*ISOPEN*, care este de tip boolean şi indică dacă un cursor este deschis (în cazul cursoarelor implicite, acest atribut are întotdeauna valoarea *FALSE*, deoarece un cursor implicit este închis de sistem imediat după executarea instrucţiunii *SQL* asociate).

Atributele pot fi referite prin expresia *SQL%nume_atribut*, în cazul cursoarelor implicite, sau prin *nume_cursor%nume_atribut*, în cazul unui cursor explicit. Ele pot să apară în comenzi *PL/SQL*, în funcţii, în secţiunea 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* şi atributul *%BULK_ROWCOUNT* au aceiaşi indici, deci folosesc acelaşi 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 cursoarelor explicite sunt necesare următoarele etape:

- declararea cursorului (atribuirea unui nume şi asocierea cu o comandă *SELECT*);
- deschiderea cursorului pentru cerere (executarea interogării asociate şi 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* şi *CLOSE*.

## Declararea unui cursor explicit

Prin declarația *CURSOR* în cadrul comenzii *DECLARE* este definit un cursor explicit şi 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 şi căruia nu i se poate atribui o valoare.

**Deschiderea unui cursor explicit**

Comanda *OPEN* execută cererea asociată cursorului, identifică mulțimea liniilor rezultat şi 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* regăseşte 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);
- citeşte datele liniei curente în variabile *PL/SQL*;
- dacă *pointer*-ul este poziționat la sfârşitul 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 şi deschis anterior. Variabila sau lista de variabile din clauza *INTO* trebuie să fie compatibilă (ca ordine şi tip) cu lista selectată din cererea asociată cursorului.

La un moment dat, comanda *FETCH* regăseşte o singură linie. Totuşi, 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 şi resursele asociate acestuia pot fi eliberate. Aceste resurse includ spațiul utilizat pentru memorarea mulțimii active şi 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 toţi artiştii care au opere de artă expuse în muzeu să se insereze în tabelul *temp* informaţii referitoare la numele acestora şi anul naşterii.

```
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:

| | | %FOUND | %ISOPEN | %NOTFOUND | %ROWCOUNT |
|---|---|---|---|---|---|
| ***OPEN*** | Înainte | Excepție | *False* | Excepție | Excepție |
| | După | *Null* | *True* | *Null* | 0 |
| **Prima** | Înainte | *Null* | *True* | *Null* | 0 |
| **Încărcare** | După | *True* | *True* | *False* | 1 |
| **Următoarea** | Înainte | *True* | *True* | *False* | 1 |
| **încărcare** | După | *True* | *True* | *False* | Depinde de date |
| **Ultima** | Înainte | *True* | *True* | *False* | Depinde de date |
| **încărcare** | După | *False* | *True* | *True* | Depinde de date |
| ***CLOSE*** | Înainte | *False* | *True* | *True* | Depinde de date |
| | După | Excepție | *False* | 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 foloseşte operația de ciclare (*LOOP*, *WHILE*, *FOR*), prin care la fiecare iterație se va încărca o nouă linie. Comanda *EXIT* poate fi utilizată pentru ieşirea din ciclu, iar valoarea atributului *%ROWCOUNT* pentru terminarea ciclului.

Procesarea liniilor unui cursor explicit se poate realiza şi cu ajutorul unui ciclu *FOR* special, numit ciclu cursor. Pentru acest ciclu este necesară doar declararea cursorului, operațiile de deschidere, încărcare şi 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 nişte cursoare ale căror comenzi *SELECT* ataşate depind de parametri ce pot fi modificaţi 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* ataşată 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 uşor de interpretat şi de întreținut, oferind şi 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* şi, 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 şi cei actuali se face prin:

- poziție – parametrii formali şi 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 şi 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 şterse 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 ştergere). În felul acesta este realizată consistența la citire a sistemului. De exemplu, această situație este utilă când se reactualizează o valoare a unei linii şi 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 aştepta (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 aşteptare. Dacă liniile nu sunt deblocate în *n* secunde, atunci se declanşează 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 aşteaptă până ce linia este deblocată şi atunci returnează rezultatul comenzii *SELECT*.

Dacă un cursor este declarat cu clauza *FOR UPDATE*, atunci comenzile *DELETE* şi *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 şi ştergerile să se efectueze asupra acestei linii, fără referirea explicită a cheii primare sau pseudocoloanei *ROWID*. De subliniat că instrucțiunile *UPDATE* şi *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 nişte 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 eşua. În cazul în care cursorul nu este definit prin *SELECT...FOR UPDATE*, nu sunt probleme în acest sens şi, 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 şi 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 şi diferiți clienți. De exemplu, un *client OCI*, o aplicație *Oracle Forms* şi *server*-ul *Oracle* pot referi aceeaşi zonă de lucru (care conține mulțimea rezultat). Pentru a reduce traficul în rețea, o variabilă cursor poate fi declarată pe stația *client*, deschisă şi 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 lipseşte 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*) şi 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 dispare 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 şi poziționează cursorul la prima linie din mulțimea rezultat. Sintaxa comenzii este:

***OPEN*** {*variabila_cursor* | :*variabila_cursor_host*}
***FOR*** {*cerere_select* |
*şir_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 *şir_dinamic* este o secvență de caractere care reprezintă cererea multilinie.

Opțiunea *şir_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 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.

***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* şi 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 şi variabilele cursor nu sunt interoperabile. Nu poate fi folosită una din ele, când este aşteptată 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 şi cursoare generate de subcereri. *PL/SQL* acceptă cereri care au expresii cursor în cadrul unei declarații cursor, declarații *REF CURSOR* şi 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 defineşte 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*) şi care se desfăşoară într-o localitate precizată (*val_oras*). Să se afişeze 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 şi pentru fiecare galerie să se afişeze 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.nume_galerie);
    FOR sal IN (SELECT cod_sala, nume_sala
                FROM   sala
                WHERE  cod_galerie = gal.cod.galerie)
    LOOP
      DBMS_OUTPUT.PUT_LINE (sal.nume_sala);
    END LOOP;
  END LOOP;
END;
```

*Varianta 2*:

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

# 4. Gestiunea cursoarelor în *PL/SQL*

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

Zona conţine informaţii necesare procesării comenzii, cum ar fi:

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

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

Există două tipuri de cursoare:

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

Atât cursoarele implicite cât şi 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 şi reprezintă numărul liniilor încărcate de cursor;
- %*FOUND*, care este de tip boolean şi ia valoarea *TRUE* dacă ultima operaţie de încărcare (*FETCH*) dintr-un cursor a avut succes (în cazul cursoarelor explicite) sau dacă instrucţiunea *SQL* a întors cel puţin o linie (în cazul cursoarelor implicite);
- %*NOTFOUND*, care este de tip boolean şi are semnificaţie opusă faţă de cea a atributului %*FOUND*;
- %*ISOPEN*, care este de tip boolean şi indică dacă un cursor este deschis (în cazul cursoarelor implicite, acest atribut are întotdeauna valoarea *FALSE*, deoarece un cursor implicit este închis de sistem imediat după executarea instrucţiunii *SQL* asociate).

Atributele pot fi referite prin expresia ***SQL%nume_atribut,*** în cazul cursoarelor implicite, sau prin ***nume_cursor%nume_atribut***, în cazul unui cursor explicit. Ele pot să apară în comenzi *PL/SQL*, în funcţii, în secţiunea 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* şi atributul *%BULK_ROWCOUNT* au aceiaşi indici, deci folosesc acelaşi 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

**Exemplu:**
Sa se creeze tabelul JOB_ANG cu cimputile employee_id si job_id. Sa se numere citi angajati sunt pe fiecare job si citi angajati in total.

```
CREATE TABLE    JOB_ANG
  (EMPLOYEE_ID  NUMBER(6)    NOT NULL  ,
    JOB_ID  VARCHAR2(10)   NOT NULL  );
```

```
SQL> CREATE TABLE JOB_ANG
  2    (EMPLOYEE_ID  NUMBER(6) NOT NULL  ,
  3     JOB_ID  VARCHAR2(10) NOT NULL  );
```

Table created.

```
SET SERVEROUTPUT ON
DECLARE
  TYPE alfa  IS TABLE OF  JOBS.JOB_ID%TYPE;
  beta    alfa;
BEGIN
  SELECT JOB_ID BULK COLLECT INTO beta FROM JOBS;
  FORALL  j IN 1..beta.COUNT
    INSERT INTO JOB_ANG
      SELECT EMPLOYEE_ID, JOB_ID
      FROM   EMPLOYEES
      WHERE  JOB_ID = beta(j);
  FOR j IN 1..beta.COUNT LOOP
   DBMS_OUTPUT.PUT_LINE ('Pentru jobul  ' ||  beta(j)  || '   au fost inserate   ' ||
      SQL%BULK_ROWCOUNT(j)  || '  inregistrari (angajati)');
  END LOOP;
  DBMS_OUTPUT.PUT_LINE ('Numarul total de inregistrari inserate este ' ||
     SQL%ROWCOUNT);
END;
/
Pentru jobul  AD_PRES   au fost inserate   1  inregistrari (angajati)
Pentru jobul  AD_VP   au fost inserate   2  inregistrari (angajati)
Pentru jobul  AD_ASST   au fost inserate   1  inregistrari (angajati)
Pentru jobul  FI_MGR   au fost inserate   1  inregistrari (angajati)
```

Pentru jobul FI_ACCOUNT au fost inserate 5 inregistrari (angajati)
Pentru jobul AC_MGR au fost inserate 1 inregistrari (angajati)
Pentru jobul AC_ACCOUNT au fost inserate 1 inregistrari (angajati)
Pentru jobul SA_MAN au fost inserate 5 inregistrari (angajati)
Pentru jobul SA_REP au fost inserate 30 inregistrari (angajati)
Pentru jobul PU_MAN au fost inserate 1 inregistrari (angajati)
Pentru jobul PU_CLERK au fost inserate 5 inregistrari (angajati)
Pentru jobul ST_MAN au fost inserate 5 inregistrari (angajati)
Pentru jobul ST_CLERK au fost inserate 20 inregistrari (angajati)
Pentru jobul SH_CLERK au fost inserate 20 inregistrari (angajati)
Pentru jobul IT_PROG au fost inserate 5 inregistrari (angajati)
Pentru jobul MK_MAN au fost inserate 1 inregistrari (angajati)
Pentru jobul MK_REP au fost inserate 1 inregistrari (angajati)
Pentru jobul HR_REP au fost inserate 1 inregistrari (angajati)
Pentru jobul PR_REP au fost inserate 1 inregistrari (angajati)
Numarul total de inregistrari inserate este 107

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

**Cursoare explicite**

Pentru gestiunea cursoarelor explicite sunt necesare următoarele etape:

- declararea cursorului (atribuirea unui nume şi asocierea cu o comandă *SELECT*);
- deschiderea cursorului pentru cerere (executarea interogării asociate şi 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* şi *CLOSE*.

**Declararea unui cursor explicit**

Prin declarația *CURSOR* în cadrul comenzii *DECLARE* este definit un cursor explicit şi 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 şi căruia **nu** i se poate atribui o valoare.

**Deschiderea unui cursor explicit**

Comanda *OPEN* execută cererea asociată cursorului, identifică mulțimea liniilor rezultat şi 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* regăseşte 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);
- citeşte datele liniei curente în variabile *PL/SQL*;
- dacă *pointer*-ul este poziționat la sfârşitul 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 şi deschis anterior. Variabila sau lista de variabile din clauza *INTO* trebuie să fie compatibilă (ca ordine şi tip) cu lista selectată din cererea asociată cursorului.

La un moment dat, comanda *FETCH* regăseşte o singură linie. Totuşi, 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 şi resursele asociate acestuia pot fi eliberate. Aceste resurse includ spațiul utilizat pentru memorarea mulțimii active şi 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 artiştii care au opere de artă expuse în muzeu să se insereze în tabelul *temp* informații referitoare la numele acestora şi anul naşterii.

```
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:

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

**Procesarea liniilor unui cursor explicit**

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

Procesarea liniilor unui cursor explicit se poate realiza şi cu ajutorul unui ciclu ***FOR* special, numit ciclu cursor.** Pentru acest ciclu este necesară doar declararea cursorului, operațiile de deschidere, încărcare şi 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 nişte cursoare ale căror comenzi *SELECT* ataşate depind de parametri ce pot fi modificaţi la momentul execuţiei, în *PL/SQL* s-a introdus noţiunea de **cursor parametrizat**. Prin urmare, un cursor parametrizat este un cursor în care comanda *SELECT* ataşată depinde de unul sau mai mulţi parametri.

Transmiterea de parametri unui cursor parametrizat se face în mod similar procedurilor stocate. Un astfel de cursor este mult mai uşor de interpretat şi de întreţinut, oferind şi 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* şi, 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 şi cei actuali se face prin:

- poziție – parametrii formali şi 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*:**

Să se declare un **cursor parametrizat** (parametrii fiind var_salary şi var_dept) prin care să se afişeze in ordine alfabetica numele, salariul si codul salariaților pentru care salary<var_salary şi department_id=var_dept. Rezolvarea se va face în trei moduri (cursor explicit, ciclu cursor, ciclu cursor cu subcereri).

```
DECLARE
  v_nume     employees.last_name%TYPE;
  v_sal      employees.salary%TYPE;
  v_cod      employees.employee_id %TYPE;
  CURSOR ang_cursor (var_salary NUMBER, var_dept NUMBER) IS
    SELECT employee_id, last_name, salary
    FROM employees
    WHERE salary<var_salary AND department_id=var_dept
    ORDER BY last_name;
BEGIN
  DBMS_OUTPUT.PUT_LINE('---Cursor explicit---');
```

```
 OPEN ang_cursor(10000,80);
   LOOP
     FETCH ang_cursor  INTO v_cod, v_nume,v_sal;
     EXIT WHEN ang_cursor%NOTFOUND;
     DBMS_OUTPUT.PUT_LINE('Salariatul '|| v_nume||
              ' are salariul ' ||v_sal||' cod '||v_cod);
   END LOOP;
  CLOSE ang_cursor;

 DBMS_OUTPUT.PUT_LINE('---Ciclu cursor---');
 FOR v_ang_cursor IN ang_cursor(10000,80) LOOP

    DBMS_OUTPUT.PUT_LINE('Salariatul '|| v_ang_cursor.last_name ||
          ' are salariul ' || v_ang_cursor.salary ||' cod '||
      v_ang_cursor.employee_id);
  END LOOP;

 DBMS_OUTPUT.PUT_LINE('---Ciclu cursor cu subcereri---');
 FOR vv_ang_cursor IN (SELECT employee_id, last_name, salary
     FROM employees
         WHERE salary<10000 AND department_id=80 ORDER BY
          last_name)
 LOOP
    DBMS_OUTPUT.PUT_LINE('Salariatul '|| vv_ang_cursor.last_name||
         ' are salariul ' ||vv_ang_cursor.salary||' cod '||
     vv_ang_cursor.employee_id);
  END LOOP;
 END;
 /
```

***Exemplu*:**

Utilizând un cursor parametrizat să se obţină codurile operelor de artă din fiecare sală, identificatorul sălii şi 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 şterse 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 ştergere). În felul acesta este realizată consistența la citire a sistemului. De exemplu, această situație este utilă când se reactualizează o valoare a unei linii şi 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 aştepta (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 aşteptare. Dacă liniile nu sunt deblocate în *n* secunde, atunci se declanşează 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 aşteaptă până ce linia este deblocată şi atunci returnează rezultatul comenzii *SELECT*.

Dacă un cursor este declarat cu clauza *FOR UPDATE*, atunci comenzile *DELETE* şi *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 şi ştergerile să se efectueze asupra acestei linii, fără referirea explicită a cheii primare sau pseudocoloanei *ROWID*. De subliniat că instrucțiunile *UPDATE* şi *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 nişte 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 eşua. În cazul în care cursorul nu este definit prin *SELECT...FOR UPDATE*, nu sunt probleme în acest sens şi, prin urmare, în interiorul ciclului unde se fac schimbări ale datelor poate fi utilizat un *COMMIT*.

**Exemplu:**

Să se mărească cu 1000 salariile angajaților care au fost angajați în 2000 din tabelul emp_***. Se va folosi un cursor SELECT FOR UPDATE.

```
SELECT last_name, hire_date, salary
FROM emp_***
WHERE TO_CHAR(hire_date, 'yyyy') = 2000;
DECLARE
  CURSOR emp_cursor IS
    SELECT *
    FROM   emp_***
    WHERE  TO_CHAR(hire_date, 'YYYY') = 2000
    FOR UPDATE OF salary NOWAIT;
BEGIN
  FOR v_emp_cursor IN emp_cursor  LOOP
    UPDATE  emp_***
    SET     salary= salary+1000
    WHERE CURRENT OF emp_cursor;
  END LOOP;
END;
/
SELECT last_name, hire_date, salary
FROM emp_***
WHERE TO_CHAR(hire_date, 'yyyy') = 2000;
ROLLBACK;
```

***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;
```