

Capitolul 8. PL/SQL. Elemente generale

Limbajul SQL a fost gândit inițial ca limbaj de generația a IV-a, neprocedural, orientat pe seturi de înregistrări. Deși extrem puternic, SQL-ul nu poate acoperi toate cerințele unei aplicații, atât în ceea ce privește prelucrarea datelor, cât mai ales comunicarea cu alte module și cu utilizatorul, listarea informațiilor sub formă de rapoarte standardizate etc. Este adevărat, una din direcțiile în care SQL a evoluat enorm de la stardardul SQL-92 la SQL:1999 este procedularitatea. Totuși, conformitatea fiecărui produs cu standardul rămâne o mare problemă, și, probabil, va fi nevoie de mult timp până când vom asista la o convergență în materie de procedularitate a SGBD-urilor.

Toate serverele de baze de date, categorie în care Oracle este în primele rânduri, dispun de o extensie procedurală a SQL-ului care reprezintă, în fapt, limbajul de programare proprietar al produsului respectiv: PL/SQL (Oracle), Transact SQL sau T-SQL (SQL Server), pgPLSQL (PostgreSQL) etc. Prezentul capitol constuie o introducere frugală în programarea Oracle, în ceea ce poate fi numit “limbajul de programare de la Oracle”, și anume PL/SQL. După “părerea” producătorului, principalele atuuri ale PL/SQL sunt¹:

- suport deplin pentru SQL;
- opțiuni puternice pentru programarea orientată pe obiecte;
- performanță;
- productivitate înaltă;
- portabilitate;
- integrare strânsă cu celelalte tehnologii Oracle;
- securitate remarcabilă.

8.1. Structura unui bloc PL/SQL

Programele PL/SQL iau forma blocurilor, care pot fi *fără nume* (nenumite, anonime), blocuri ce nu fac parte de schema bazei, găsindu-se pe disc sub forma unor fișiere text (ASCII) cu extensii precum .txt, .sql etc., și a blocurilor *cu nume* (denumite), stocabile în dicționarul bazei de date. Dintre blocurile cu nume, ne vom ocupa îndeosebi de proceduri (*procedures*), funcții (*functions*), pachete (*packages*) și declanșatoare (*triggers*).

Structura unui bloc este împrumutată din limbajul ADA și prezintă trei secțiuni: declarații, zona executabilă propriu-zisă și excepții. Începem cu un prim bloc prezentat în listing 8.1 care, lansat în SQL*Plus, afișează (prin comada PUT_LINE din pachetul sistem DBMS_OUTPUT) un *Servus* politicos. Comentariile

¹ Preluare din Oracle PL/SQL. User's Guide and Reference, Release 2 (9.2), 2002, p. 1-20

pot fi introduse fie prin două cratime, caz în care lungimea comentariului nu poate depăși linia respectivă, fie delimitate prin perechile de caractere `/*` (început de comentariu) `*/` (sfârșit de comentariu), situație în care comentariul se poate întinde pe mai multe rânduri.

Listing 8.1. Primul bloc PL/SQL (mai simplu nu se poate)

```
-- acest bloc nu face aproape nimic
DECLARE
  /* prima secțiune este cea a declarațiilor (numai de variabile, cursoare, excepții) */
  prima_variabila INTEGER ;      /* o variabilă întreagă */
  a_doua_variabila VARCHAR2(50) ;      /* o alta de tip sir de caractere de
                                         lungime variabilă */
  a_treia_variabila DATE ;          /* la fel de inutilă, dar de tip dată calendaristică */
  ultima_variabila BOOLEAN ;      /* un tip de variabilă (BOOLEAN) nestocabil în tabele */

BEGIN
  -- în aceasta secțiune se scriu comenzile efective
  DBMS_OUTPUT.PUT_LINE('Servus !'); --echivalentul ardelenesc al lui Hello, World ;
  -- atenție, în SQL*Plus pentru afișare este necesară comanda SET SERVEROUTPUT ON

END;
```

Există două moduri de a crea și lansa în execuție blocul în SQL*Plus (vezi și capitolul 3). Primul constă în introducerea linie cu linie a programului “de la prompterul” SQL*Plus, apoi lansarea în execuție cu ajutorul semnului `/` (slash), ca în figura 8.1.

```
Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 9.2.0.1.0 - Production on Fri Jan 31 22:47:48 2003
Copyright (c) 1982, 2002, Oracle Corporation. All rights reserved.

Connected to:
Personal Oracle9i Release 9.2.0.1.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production

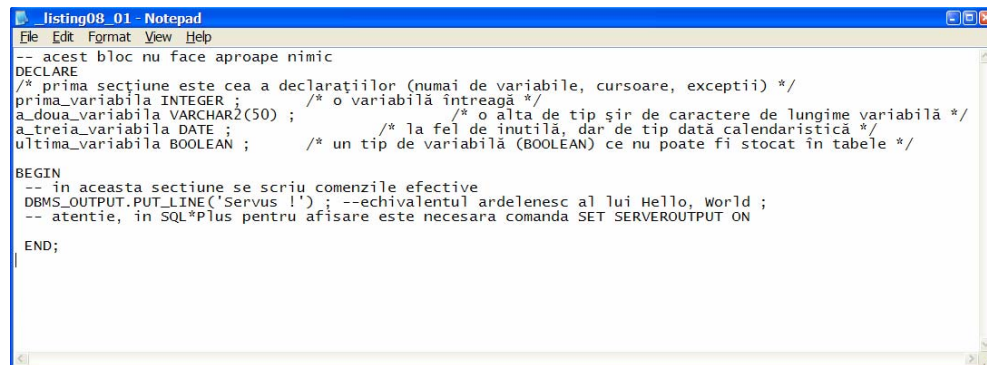
SQL> SET SERVEROUTPUT ON
SQL> -- acest bloc nu face aproape nimic
SQL> DECLARE
  2 /* prima secțiune este cea a declarațiilor (numai de variabile, cursoare, excepții) */
  3 prima_variabila INTEGER ; /* o variabilă întreagă */
  4 a_doua_variabila VARCHAR2(50) ; /* o alta de tip sir de caractere de lungime variabila */
  5 a_treia_variabila DATE ; /* la fel de inutilă, dar de tip data calendaristica */
  6 ultima_variabila BOOLEAN ; /* un tip de variabila (BOOLEAN) ce nu poate fi stocat în tabele */
  7
  8 BEGIN
  9 -- în aceasta secțiune se scriu comenzile efective
 10 DBMS_OUTPUT.PUT_LINE('Servus !'); --echivalentul ardelenesc al lui Hello, World ;
 11 -- atenție, în SQL*Plus pentru afișare este necesara comanda SET SERVEROUTPUT ON
 12
 13 END;
 14 /
Servus !

PL/SQL procedure successfully completed.

SQL> |
```

Figura 8.1. Introducerea blocului PL/SQL direct în SQL*Plus

Problemele introducerii textului în SQL*Plus țin, în primul rând, de primitivitatea interfeței. De aceea, se folosește destul de des a doua variantă de lucru: blocul se editează cu Notepad-ul (figura 8.2), salvându-se ca fișier ASCII cu numele `listing08_01` și extensia `.sql` (firește, puteam alege și o altă extensie).



```
-- acest bloc nu face aproape nimic
DECLARE
/* prima secțiune este cea a declarațiilor (numai de variabile, cursoare, excepții) */
prima_variabila INTEGER; /* o variabilă întreagă */
a_doua_variabila VARCHAR2(50); /* o alta de tip șir de caractere de lungime variabilă */
a_treia_variabila DATE; /* la fel de inutilă, dar de tip dată calendaristică */
ultima_variabila BOOLEAN; /* un tip de variabilă (BOOLEAN) ce nu poate fi stocat în tabele */

BEGIN
-- în această secțiune se scriu comenzile efective
DBMS_OUTPUT.PUT_LINE('Servus !'); --echivalentul ardelenesc al lui Hello, World ;
-- atenție, în SQL*Plus pentru afișare este necesară comanda SET SERVEROUTPUT ON

END;
```

Figura 8.2. Editarea blocului PL/SQL cu Notepad-ul

Lansarea în execuție în SQL*Plus se realizează prin comanda `START` sau `@`, ca în figura 8.3.

```
SQL> @F:\ORACLE_CARTE\CAP08_PL_SQL1\LISTING08_01.SQL
15 /
Servus !

PL/SQL procedure successfully completed.
```

Figura 8.3. Lansarea în execuție a blocului PL/SQL

Trebuie să recunoaștem că acest prim bloc este penibil de simplu. Fără a părăsi zona simplistă, luăm alte exemple pentru ilustrarea lucrului cu structuri alternative și repetitive. Dar înainte de aceasta, să trecem în revistă câteva tipuri de date și funcții sistem PL/SQL.

8.2. Tipuri de date PL/SQL. Domeniul de vizibilitate al variabilelor

Zona declarativă este, în majoritatea cazurilor și spațiului, rezervată variabilelor, deși, după cum vom vedea pe parcursul acestui capitol (și în o parte din viitoarele), tot aici sunt definite și constante, cursoare, excepții, tablouri asociative PL/SQL (*PL/SQL tables*), înregistrări (*records*), tabele încapsulate (*nested tables*), vectori cu mărime variabilă (*varrays*) etc.

Pe lângă tipurile gestionate în tabele: `CHAR`, `VARCHAR2`, `DATE`, `NUMBER`, `INTEGER` etc., PL/SQL prezintă și tipuri proprii pentru variabile, cum ar fi `BOOLEAN` sau un subtip al tipului `INTEGER`, și anume `BINARY_INTEGER`.

Figura 8.4 prezintă tipologia datelor în PL/SQL așa cum apare în Oracle 9i2 PL/SQL User's Guide and Reference (la pagina 3-2).

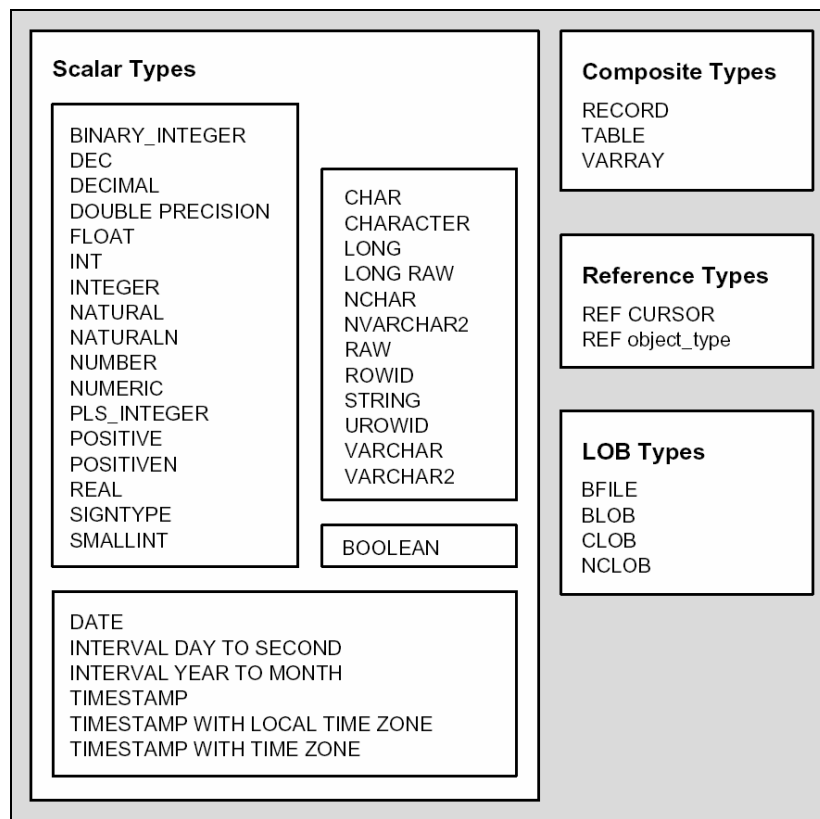


Figura 8.4. Tipuri de date PL/SQL (după documentația Oracle)

Preluarea a fost făcută cu așa mare acuratețe, încât nici traducerele nu au fost operate acolo unde ar fi fost cazul. La drept vorbind, prea multe comentarii sunt de prisos. Tipurile scalare (*scalar types*) sunt cele obișnuite, simple, cum ar fi INTEGER, NUMBER pentru numere, CHAR și VARCHAR2 pentru șiruri de caractere, DATE și TIMESTAMP pentru date calendaristice, BOOLEAN pentru date logice.

Pentru numere întregi, Oracle recomandă tipul PLS_INTEGER și renunțarea treptată la BINARY_INTEGER, din rațiuni de viteză de procesare, deși ambele pot reprezenta numere întregi în intervalul $-2^{31}..2^{31}$. Pentru șiruri de caractere de lungime fixă, tipul CHAR poate reprezenta maxim 32767 octeți, iar pentru cele de lungime variabilă tipul VARCHAR2 poate stoca maximum tot 32767 octeți. Pentru șiruri de caractere de dimensiuni mari, documentația produsului recomandă trecerea treptată de la tipul LONG la CLOB, iar pentru imagini, grafice, videoclipuri etc., trecerea de la LONG RAW la BLOB.

Tipurile compozite (*composite types*) sunt alcătuite din tipuri scalare și/sau alte tipuri compozite. Frecvent întâlnite sunt structurile de tip articol sau înregistrare (*record*), tablou (*table*) care cuprinde tablourile PL/SQL (vectori asociativi) și tabelele încapsulate (*nested tables*), vectori cu mărime variabilă (*varrays*). Tipurile referință sunt legate de variabile cursor, de lucrul cu obiecte, iar tipurile LOB permit gestionarea de date complexe (text, imagine etc.). Deoarece nu ne propunem să substituim prezenta lucrare manualului PL/SQL al firmei, vom discuta despre tipurile importante pe măsură ce vom înainta cu exemplele și problemele.

Interesant este că tipul unei variabile poate fi specificat și indirect: de exemplu, pentru o variabilă `v_marca` se poate specifica:

```
DECLARE
...
v_marca NUMBER(5);
...
```

știind că marca unui angajat este un număr din cinci cifre, dar și:

```
DECLARE
v_marca personal.marca%TYPE ;
```

Cea de-a doua soluție este mai elegantă și mai comodă. Pe de o parte, nu trebuie să ținem minte tipul și lungimea exactă a atributului `personal.marca`, iar, pe de altă parte, dacă se schimbă tipul sau lungimea câmpului în tabelă, nu mai suntem nevoiți să umblăm în toate blocurile în care `v_marca` a fost declarată `NUMBER(5)`.

Și în ceea ce privește tipologia variabilelor compozite, PL/SQL-ul este generos: articole, tablouri PL/SQL etc. constituie opțiuni folosite pe scară largă de dezvoltatorii de aplicații. Ca și în cazul variabilelor scalare (simple), și cele compozite pot fi definite prin raportarea la alte colecții. Spre exemplu, o variabilă compozită cu aceleași atribute precum cele ale tabelii `PERSONAL` poate fi definită prin:

```
DECLARE
...
TYPE t_personal IS RECORD (
    marca personal.marca%TYPE,
    numepren personal.numepren%TYPE,
    compart personal.compart%TYPE,
    datasv personal.datasv%TYPE,
    salorar personal.salorar%TYPE,
    salorarco personal.salorarco%TYPE,
    colaborator personal.colaborator%TYPE )
rec_personal t_personal ;
...
```

dar și mult mai simplu:

```
DECLARE
    ...
    rec_personal personal%ROWTYPE ;
```

Există trei moduri prin care se pot atribui valori variabilelor:

- prin atribuire directă: `a INTEGER := 12 ;` (în zona declarativă) sau `a:=12 ;` în zona executabilă;
- printr-o frază `SELECT`: `SELECT COUNT(*) INTO a FROM personal ;` sau folosind variabila compozită de mai sus: `SELECT * INTO rec_personal FROM personal WHERE marca=101 ;`
- prin specificarea unor parametri de tip `OUT` sau `IN OUT` în funcții sau proceduri.

În afara declarării numelui, tipului și lungimii, pentru o variabilă mai pot fi definite și clauzele: `DEFAULT`, pentru specificarea unei valori implicite, și `NOT NULL` pentru ca valoarea `NULL` să nu poată fi atribuită variabilei respective;

Fiecare variabilă are un domeniu de vizibilitate care reprezintă, implicit, blocul în care a fost definită. Interesant este regimul variabilelor atunci când, în blocuri sunt incluse, variabile cu același nume au tipuri și/sau valori diferite. Blocul prezentat în listing 8.2 este edificator în acest sens.

Listing 8.2. Bloc inclus. Domeniul de vizibilitate

```
-- blocul principal
DECLARE
    a INTEGER := 12 ;
    b VARCHAR2(20) ;
    c DATE ;
BEGIN
    b := 'Ana are mere' ;
    c := TO_DATE('15/05/2003', 'DD/MM/YYYY') ;

    DBMS_OUTPUT.PUT_LINE (' ');
    DBMS_OUTPUT.PUT_LINE ('La inceputul blocului principal') ;
    DBMS_OUTPUT.PUT_LINE ('a = '||a) ;
    DBMS_OUTPUT.PUT_LINE ('b = '||b) ;
    DBMS_OUTPUT.PUT_LINE ('c = '||c) ;

    -- aici începe blocul secundar
    DECLARE
        b NUMBER(12,2) ;
        c VARCHAR2(25) ;
        d DATE ;
    BEGIN
        b := 455 ;
        d := TO_DATE('11/06/2003', 'DD/MM/YYYY') ;
        DBMS_OUTPUT.PUT_LINE (' ');
        DBMS_OUTPUT.PUT_LINE (' La inceputul blocului secundar') ;
        DBMS_OUTPUT.PUT_LINE (' a = '||a) ;
        DBMS_OUTPUT.PUT_LINE (' b = '||b) ;
        DBMS_OUTPUT.PUT_LINE (' c = '||NVL(c, ' c este NULL')) ;
        DBMS_OUTPUT.PUT_LINE (' d = '||d) ;
```

```

END ;

-- revenirea în blocul principal
DBMS_OUTPUT.PUT_LINE ( ' ' );
DBMS_OUTPUT.PUT_LINE ('La revenirea in blocul principal');
DBMS_OUTPUT.PUT_LINE ('a = '||a);
DBMS_OUTPUT.PUT_LINE ('b = '||b);
DBMS_OUTPUT.PUT_LINE ('c = '||c);

-- dacă linia următoare nu ar fi comentată, s-ar declanșa eroarea din figura 8.6
-- DBMS_OUTPUT.PUT_LINE ('d = '||d);
END ;

```

Secțiunea declarativă a blocului principal definește trei variabile: *a*, *b* și *c*. *A* este de tip întreg, odată cu declararea sa atribuindu-i-se valoarea 12. Variabila *b* este de tip șir de caractere de lungime variabilă, iar *c* dată calendaristică. În secțiunea executabilă a blocului principal variabila *b* va primi valoarea preluată din Abecedar: 'Ana are mere' (șirul de caractere se scrie între două simboluri apostrof, iar *c* data de 15 mai 2003. Comenzile următoare de afișare vor servi ca valori inițiale ale variabilelor ce urmează a fi re-definite/actualizate – vezi figura 8.5.

```

SQL> START F:\ORACLE_CARTE\CAP08_PL_SQL1\LISTING08_02.SQL
43 /
La inceputul blocului principal
a = 12
b = Ana are mere
c = 15-MAY-03
La inceputul blocului secundar
a = 12
b = 455
c = c este NULL
d = 11-JUN-03
La revenirea in blocul principal
a = 12
b = Ana are mere
c = 15-MAY-03

PL/SQL procedure successfully completed.

SQL> |

```

Figura 8.5. Valorile variabilelor în diferite puncte ale blocului PL/SQL

Blocul principal conține un al doilea bloc (să-i spunem secundar) care, firește, începe cu `DECLARE` și se termină cu `END`. În acest bloc secundar se redefinesc variabilele *b* și *c*, care sunt acum numerice, respectiv șir de caractere de lungime variabilă. Apare și o nouă variabilă, *d*, de tip dată calendaristică. După atribuirea de valori pentru *b* și *d* se afișează valorile tuturor celor patru variabile. Din figura 8.5 se observă că *a* păstrează tipul și valoarea din blocul principal (12). În schimb, datorită redeclarării, *b* și *c* sunt "suprascrise". *c*, neprimind nici o valoare în blocul secundar, după declarare, va fi `NULL`. La ieșirea din blocul secundar, *b* și *c* își "recapătă" forma și conținutul din blocul principal, așa încât valorile sunt identice celor de la început. Variabila *d*, în schimb, este "expirată";

dacă se încearcă afișarea valorii sale, de-comentându-se penultima linie a blocului, mesajul va fi cel din figura 8.6.

```
SQL> START F:\ORACLE_CARTE\CAP08_PL_SQL1\LISTING08_02_VARIANTA2.SQL
43 /
  DBMS_OUTPUT.PUT_LINE ('d = '||d) ;
                                *
ERROR at line 39:
ORA-06550: line 39, column 32:
PLS-00201: identifiier 'D' must be declared
ORA-06550: line 39, column 2:
PL/SQL: Statement ignored

SQL> |
```

Figura 8.6. Eroarea datorată încercării de afișare în blocul principal a variabilei d definită în blocul secundar

8.3. Structuri alternative și repetitive

Cei mai nostalgici vor lăcrăma probabil când le vom reaminti o problemă din ciclul gimnazial - rezolvarea ecuației de gradul II: $ax^2 + bx + c = 0$. Rezolvarea scrupuloasă a ecuației presupune testarea valorii parametrilor a, b și c pentru a vedea dacă ecuația este de gradul II într-adevăr, sau de grad I, sau avem de-a face cu nedeterminare sau imposibilitate; în plus, rădăcinile x_1 și x_2 pot fi egale sau complexe. Fără a mai lungi nejustificat discuția, iată corpul blocului în listing 8.3.

Listing 8.3. Bloc PL/SQL pentru rezolvarea ecuației de gradul al II-lea

```
/* Bloc anonim pentru rezolvarea ecuației de gradul II.
Pentru cei cu anumită distanță față de perioada copilăriei, reamintim că formatul general este
ax**2 + b*x + c = 0. Să se determine x1 si x2. /
DECLARE
  a INTEGER := 0 ;
  b INTEGER := 5667 ;
  c INTEGER := 12 ;
  delta NUMBER(16,2) ;
  X1 NUMBER(16,6) ;
  X2 NUMBER(16,6) ;
BEGIN
  -- ecuația este de grad II ?
  IF a = 0 THEN
    IF b = 0 THEN
      IF c=0 THEN
        DBMS_OUTPUT.PUT_LINE('Nedeterminare !') ;
      ELSE
        DBMS_OUTPUT.PUT_LINE('Imposibil !!!') ;
      END IF ;
    ELSE
      DBMS_OUTPUT.PUT_LINE('Ecuația este de gradul I') ;
      x1 := -c / b ;
      DBMS_OUTPUT.PUT_LINE('x='||x1) ;
```



```

        END IF ;
    ELSE
        delta := b**2 - 4*a*c ;
        IF delta > 0 THEN
            x1 := (-b - SQRT(delta)) / (2 * a) ;
            x2 := (-b + SQRT(delta)) / (2 * a) ;
            DBMS_OUTPUT.PUT_LINE('x1='||x1||', x2='||x2);

        ELSE
            IF delta = 0 THEN
                x1 := -b / (2 * a) ;
                DBMS_OUTPUT.PUT_LINE('x1=x2='||x1);

            ELSE
                DBMS_OUTPUT.PUT_LINE('radacinile sunt complexe !!!') ;
            END IF ;
        END IF;
    END IF ;
END;

```

Blocul folosește variabile pentru parametrii a , b și c , pentru cele două (posibile) rădăcini $x1$ și $x2$ și o alta pentru calculul lui Δ . Variabilele PL/SQL trebuie declarate obligatoriu în prima secțiune. Orice folosire a unei variabile nedeclarate va fi prompt sancționată de Oracle. După IF și condiție, prezența lui THEN este obligatorie. Deși nu avem un asemenea caz aici, atunci când pe ramura THEN (sau ELSE, dacă apare efectiv în program) nu e nimic de executat, trebuie introdus un NULL astfel:

```

    IF abc = bcd THEN
        NULL ;
    ELSE
        cde := abc ;
    END IF ;

```

Între END și IF trebuie să apară neapărat un spațiu (END IF), iar după END IF, ca și după instrucțiunile de calcul/atribuire, afișare etc. se inserează un caracter punct-virgulă. Apelul blocului de mai sus (salvat pe disc ca fișier ASCII LISTING08_02.SQL), precum și rezultatele execuției în SQL*Plus sunt afișate în figura 8.7.

```

SQL> @F:\ORACLE_CARTE\CAP08_PL_SQL1\LISTING08_03.SQL
DOC>Pentru cei cu anumita distanta fata de perioada copilariei,
DOC> reamintim ca formatul general este
DOC>  ax**2 + b*x + c = 0
DOC>Sa se determine x1 si x2. */
Ecuatia este de gradul 1
x=-.002118

PL/SQL procedure successfully completed.

SQL>

```

Figura 8.7. Lansarea în execuție a LISTING08_03.SQL

Structura condițională multiplă este implementată în Oracle 9i2 prin comanda **CASE** cu un format simplu și asemănător altor limbaje de programare. Ilustrăm folosirea acestui gen de secvență, rescriind blocul anterior, ca în listing-ul 8.4.

Listing 8.4. Folosirea unei structuri **CASE**

```

/* Bloc anonim pentru rezolvarea ecuatiei de gradul II - varianta CASE*/
DECLARE
  a INTEGER := 34 ;
  b INTEGER := 345553 ;
  c INTEGER := 231 ;
  x1 NUMBER(16,6) ;
  x2 NUMBER(16,6) ;
BEGIN
  CASE
    WHEN a = 0 AND b = 0 AND c = 0 THEN
      DBMS_OUTPUT.PUT_LINE('Nedeterminare !') ;
    WHEN a = 0 AND b = 0 AND c <> 0 THEN
      DBMS_OUTPUT.PUT_LINE('Imposibil !!!') ;
    WHEN a = 0 AND b <> 0 THEN
      DBMS_OUTPUT.PUT_LINE('Ecuatia este de gradul I') ;
      x1 := -c / b ;
      DBMS_OUTPUT.PUT_LINE('x='||x1) ;
    WHEN a <> 0 AND b**2 - 4*a*c > 0 THEN
      x1 := (-b - SQRT(b**2 - 4*a*c)) / (2 * a) ;
      x2 := (-b + SQRT(b**2 - 4*a*c)) / (2 * a) ;
      DBMS_OUTPUT.PUT_LINE('x1='||x1||', x2='||x2) ;
    WHEN a <> 0 AND b**2 - 4*a*c = 0 THEN
      x1 := -b / (2 * a) ;
      DBMS_OUTPUT.PUT_LINE('x1=x2='||x1) ;
    ELSE
      DBMS_OUTPUT.PUT_LINE('radacinile sunt complexe !!!') ;
  END CASE;
END ;
/

```

Modul de redactare este ceva mai concentrat. Ramura **ELSE** servește la specificarea a ceea ce se execută atunci când nici una dintre condițiile specificate în clauzele **WHEN** precedente nu este îndeplinită. Ar mai fi de amintit că, deși nu aveam nimic personal cu ea, am renunțat la variabila *delta*, preferând varianta spartană a scrierii expresiei ori de câte ori este nevoie.

Pentru exemplificarea unei prime secvențe repetitive, avem o problemă mai onorabilă. Ne propunem ca, pentru o lună și un an date, în **PONTAJE** să introducem câte o înregistrare pentru fiecare angajat și zi lucrătoare, deci fără sâmbete și duminici. Blocul din listing 8.5 folosește două variabile, *an* și *luna*, care sunt inițializate la fiecare lansare în execuție pentru a se preciza pe ce lună se face popularea. Variabila *prima_zi* ia, cu ajutorul funcției **TO_DATE**, valoarea primei zile calendaristice din luna de referință. Cealaltă variabilă de tip **DATE**, *zi*, joacă rol de contorizare a ciclului, fiind, la fiecare parcurgere a buclei, incrementată cu o zi, până se ajunge la ultima zi (31, 30, respectiv 28 sau 29 ale

lunii curente). Aflarea ultimei date din luna curentă presupune folosirea funcției `LAST_DAY`.

Listing 8.5. Bloc PL/SQL pentru popularea tabeli PONTAJE pentru o lună

```
-- populare cu înregistrări pentru o lună (dintr-un an) a tabeli PONTAJE
DECLARE
    an salarii.an%TYPE := 2003;
    luna salarii.luna%TYPE := 1 ;
    prima_zi DATE ; -- variabilă care stochează data de 1 a lunii
    zi DATE ; -- variabilă folosită la ciclare
BEGIN
    prima_zi := TO_DATE('01/'||luna||'/'||an, 'DD/MM/YYYY') ;
    zi := prima_zi ;

    /* bucla se repetă pentru fiecare zi a lunii */
    WHILE zi <= LAST_DAY(prima_zi) LOOP
        IF RTRIM(TO_CHAR(zi,'DAY')) IN ('SATURDAY', 'SUNDAY') THEN
            -- e zi nelucrătoare (sâmbătă sau duminică)
            NULL ;
        ELSE
            INSERT INTO pontaje (marca, data)
            SELECT marca, zi FROM personal ;
        END IF ;
        -- se trece la ziua următoare
        zi := zi + 1 ;
    END LOOP ;
    COMMIT ;
END ;
```

Bucla este delimitată, la un capăt, de instrucțiunea `WHILE . . . LOOP`, iar la celălalt capăt de `END LOOP` (ca și la `END IF`, între cele două cuvinte spațiul este obligatoriu. Este doar un mod de a redacta o secvență repetitivă. În listing 8.6 se prezintă o altă variantă:

Listing 8.6. Un alt mod de redactare a structurii repetitive

```
-- populare cu înregistrări pentru o lună (dintr-un an) a tabeli PONTAJE
DECLARE
    ...
BEGIN
    ...
    /* bucla se repetă pentru fiecare zi a lunii */
    LOOP
        EXIT WHEN zi > LAST_DAY(prima_zi) ;
        IF RTRIM(TO_CHAR(zi,'DAY')) IN ('SATURDAY', 'SUNDAY') THEN
            ...
        END IF ;
        -- se trece la ziua următoare
        zi := zi + 1 ;
    END LOOP ;
    COMMIT ;
END ;
/
```

Cea de-a treia variantă (listing 8.7) prezentată folosește echivalentul lui FOR...NEXT (sau ENDFOR) din Visual FoxPro, Visual Basic s.a., în sensul că numărul de iterații este cunoscut la intrarea în ciclu. Deoarece variabila contor nu poate fi de tip dată calendaristică, a fost nevoie, în afara de `ultima_zi`, și de variabila `număr_ultima_zi` care trebuie să conțină una dintre valorile: 31, 30, 28 sau 29. Tot pentru diversificare, aceasta a fost aleasă de tip PLS_INTEGER, unul din tipurile recomandabile, ca viteză de prelucrare.

Listing 8.7. Schemă de ciclare de tip FOR

```
-- populare cu inregistrari pentru o luna (dintr-un an) a tabeli PONTAJE

DECLARE
  an salarii.an%TYPE := 2003;
  luna salarii.luna%TYPE := 1;
  prima_zi DATE; -- variabila care stocheaza data de 1 a lunii
  ultima_zi DATE;
  zi DATE; număr_ultima_zi PLS_INTEGER;
BEGIN
  prima_zi := TO_DATE('01/'||luna|| '/'||an, 'DD/MM/YYYY');
  ultima_zi := LAST_DAY(prima_zi);
  număr_ultima_zi = TO_NUMBER(TO_CHAR(ultima_zi, 'DD'))

  /* acum bucla se repeta pentru i de la 1 la 31 (30, 28 sau 29) */
  FOR i IN 1..număr_ultima_zi LOOP
    zi := prima_zi + i - 1;
    IF TO_CHAR(zi, 'DAY') IN ('SAT', 'SUN') THEN
      -- e zi nelucratoare (simbata sau duminica)
      NULL;
    ELSE
      INSERT INTO pontaje (marca, data)
      SELECT marca, zi
      FROM personal;
    END IF;
    -- se trece (automat) la ziua urmatoare
  END LOOP;
  COMMIT;
END;
/
```

De data aceasta, variabila de ciclare este `i` ce nu trebuie musai declarată în prealabil. Extragerea numărului (zile) corespunzător datei presupune folosirea funcției `TO_CHAR` și a șablonului `'DD'`. Cum rezultatul unei funcții `TO_CHAR` este un șir de caractere, funcția `TO_CHAR` trebuie inclusă într-o funcție `TO_NUMBER`. Comanda de iterare este un pic diferită (față de VFP, VB): `FOR variabilă_contor IN valoare_inițială..valoare_finală LOOP` (între valorile inițiale și finală se interpun două puncte (pe orizontală...)).

8.4. Excepții

Dacă ați lansat de cel cel puțin două ori măcar una dintre cele trei variante pentru aceeași lună este imposibil ca la a doua execuție să nu vă fi procopsit cu un mesaj precum cel din figura 8.8.

```
SQL> @F:\ORACLE_CARTE\CAP08_PL_SQL1\LISTING08_05.SQL
DECLARE
*
ERROR at line 1:
ORA-00001: unique constraint (FOTACHEM.PK_PONTAJE) violated
ORA-06512: at line 17

SQL>
```

Figura 8.8. Lansarea repetată (pentru aceeași lună) a LISTING08_05.SQL

Ca de obicei, SGBD-ul are dreptate. Deoarece inserarea în PONTAJE se face fără nici o precauție, la a doua lansare pentru aceeași lună se inserează încă un rând de înregistrări, cu aceleași date pentru toți angajații, ceea ce înseamnă violarea cheii primare (marca, data). Eroarea ORA-00001 are un nume predefinit, și anume DUP_VAL_ON_INDEX. Unul dintre cele mai interesante aspecte ale unui bloc PL/SQL este că prezintă o secțiune specială dedicată tratării erorilor (excepțiilor), prin care erorile pot fi captate și tratate local.

Blocul din listing 8.8 prezintă ca noutate un bloc inclus care începe pe ramura ELSE a singurului IF din program. Acest bloc încearcă să adauge în PONTAJE, pentru fiecare angajat (înregistrare din PERSONAL), o înregistrare pentru ziua curentă (variabila zi). Dacă INSERT-ul încalcă restricția de cheie primară, se declanșează excepția DUP_VAL_ON_INDEX. Aceasta este captată și tratată în zona EXCEPTION. Tratarea constă în ștergerea prealabilă a tuturor înregistrărilor din pontaje în care atributul Data are valoarea egală cu ziua curentă. Atenție ! Odată declanșată eroarea în bloc, după eventuala tratare a sa în secțiunea EXCEPTION, controlul este cedat blocului superior, altfel spus, după eroare, execuția nu se reia din locul producerii erorii ! Este motivul pentru care am apelat la blocul inclus, pentru ca la ieșirea din acesta (și reintrarea în blocul principal), eroarea să fi fost tratată.

Listing 8.8. Bloc inclus și tratarea unei excepții (DUP_VAL_ON_INDEX)

```
-- populare cu înregistrări pentru o lună (dintr-un an) a tabeli PONTAJE
-- cu un bloc inclus și folosirea excepțiilor
DECLARE
  an salarii.an%TYPE := 2003;
  luna salarii.luna%TYPE := 1 ;
  prima_zi DATE ; -- variabila care stochează data de 1 a lunii
  zi DATE ; -- variabila folosită la ciclare
BEGIN
  prima_zi := TO_DATE('01/'||luna||'/'||an, 'DD/MM/YYYY') ;
```

```

zi := prima_zi ;

/* bucla se repetă pentru fiecare zi a lunii */
WHILE zi <= LAST_DAY(prima_zi) LOOP
    IF RTRIM(TO_CHAR(zi,'DAY')) IN ('SATURDAY', 'SUNDAY') THEN
        -- e zi nelucrătoare (sâmbătă sau duminică)
        NULL ;
    ELSE
        BEGIN -- de aici începe blocul inclus
            INSERT INTO pontaje (marca, data)
            SELECT marca, zi FROM personal ;
        EXCEPTION -- se preia eventuala violare a cheii primare
        WHEN DUP_VAL_ON_INDEX THEN
            -- se șterg mai întâi înregistrările pentru ziua curentă
            DELETE FROM pontaje WHERE data = zi ;
            -- apoi se reinserează înregistrările
            INSERT INTO pontaje (marca, data)
            SELECT marca, zi FROM personal ;
        END ; -- aici se termină blocul inclus
    END IF ;
    -- se trece la ziua următoare
    zi := zi + 1 ;
END LOOP ;
COMMIT ;
END ;
/

```

Practic, obiectivul este atins. Inserările sunt corecte, oricare ar fi luna aleasă și numărul de lansări ale blocului.

Profitând de curajul prins cu tratarea excepțiilor în blocul dedicat populării tabelii PONTAJE, ne reîntoarcem la banalul exemplu dedicat ecuației de gradul II, pe care-l complicăm cât putem de mult apelând la două blocuri, ambele “dotate” cu secțiuni de tratare a excepțiilor – vezi listing 8.9. Astfel, în secțiunea executabilă a blocului principal se calculează Δ , și, dacă este pozitivă, se determină x_1 și x_2 . În caz că a este 0, în expresia de calcul a lui x_1 numitorul este zero, așa încât se declanșează excepția `ZERO_DIVIDE`. Aceasta este preluată în secțiunea dedicată excepțiilor, secțiune care este un bloc în toată regula care pornește de la premisa că, întrucât a este deja 0, ecuația poate fi cel mult de gradul I. Dacă și b este zero, atunci instrucțiunea de calcul $x_1 := -c/b$ declanșează din nou excepția `ZERO_DIVIDE` care este preluată în secțiunea `EXCEPTION` a blocului secundar. Aici, pornind de la faptul că a și b sunt zero, se testează dacă și c este zero, afișându-se după caz dacă este vorba de o nedeterminare sau imposibilitate.

Listing 8.9. Bloc inclus și tratarea a două excepții (`ZERO_DIVIDE`)

```

/* Bloc anonim pentru rezolvarea ecuației de gradul II - variantă ce folosește EXCEPȚII */
DECLARE
    a INTEGER := 5 ;
    b INTEGER := 3456 ;
    c INTEGER := 23 ;
    delta NUMBER(16,2) ;
    x1 NUMBER(16,6) ;
    x2 NUMBER(16,6) ;
BEGIN

```

```

delta := b**2 - 4*a*c ;
IF delta > 0 THEN
    x1 := (-b - SQRT(delta)) / (2 * a) ;
    x2 := (-b + SQRT(delta)) / (2 * a) ;
    DBMS_OUTPUT.PUT_LINE('x1='||x1||', x2='||x2) ;

ELSE
    IF delta = 0 THEN
        x1 := -b / (2 * a) ;
        /* acesta este locul în care, dacă a=0, se declanșează excepția ZERO_DIVIDE */
        DBMS_OUTPUT.PUT_LINE('x1 = x2 = '||x1) ; -- se execută numai dacă a<>0
    ELSE
        DBMS_OUTPUT.PUT_LINE('Radacinile sunt complexe !!!') ;
    END IF ;
END IF ;

EXCEPTION -- secțiunea de excepții a blocului principal; se știe că a=0
WHEN ZERO_DIVIDE THEN
    BEGIN -- aici începe blocul secundar
        x1 := -c / b ; -- dacă și b=0, se declanșează (din nou) excepția ZERO_DIVIDE
        DBMS_OUTPUT.PUT_LINE('Ecuatia este de gradul 1') ; -- numai dacă b <> 0
        DBMS_OUTPUT.PUT_LINE('x='||x1) ;
        EXCEPTION -- secțiunea de excepții a blocului secundar; se știe că a=0 și b=0
        WHEN ZERO_DIVIDE THEN
            IF c=0 THEN
                DBMS_OUTPUT.PUT_LINE('Nedeterminare !') ;
            ELSE
                DBMS_OUTPUT.PUT_LINE('Imposibil !!!') ;
            END IF ;
        END ; -- sfârșitul blocului secundar
    END ; -- sfârșitul blocului principal
/

```

8.5. Cursoare

Execuția comenzilor SQL și stocarea informațiilor procesate presupune folosirea de către SGBD a unor zone de lucru speciale. Cursoarele Oracle permit denumirea unor asemenea zone și accesul la informațiile lor. Există două categorii de cursoare: implicite și explicite. Cele implicite sunt create automat de sistem la execuția comenzilor DML (INSERT, UPDATE, DELETE) și a frazelor SELECT care obțin rezultate pe o singură linie. Dacă fraza SELECT extrage mai multe linii, atunci este necesară crearea și folosirea cursoarelor explicite care prezintă marea parte al posibilității prelucrării individuale a înregistrărilor.

8.5.1. Cursoare implicite

În urma execuției unei comenzi SQL de actualizare, Oracle păstrează o serie de informații despre rezultate, informații printre care: dacă a fost găsită sau prelucrată măcar o linie, numărul de linii extrase/prelucrate etc. Listingul 8.10 conține blocul PL/SQL care mărește cu 1000 de lei salariul orar, dar numai pentru angajații cu peste un număr de ani de vechime, număr specificat prin variabila

ani_etalon. Acesta a fost pretextul pentru folosirea atributelor %FOUND și %ROWCOUNT ale cursorului implicit generat cu ocazia execuției comenzii UPDATE.

Listing 8.10. Primul exemplu de cursor implicit

```
/* Primul exemplu de cursor implicit */
DECLARE
    ani_etalon PLS_INTEGER := 50 ;
    numar PLS_INTEGER ;
BEGIN
    /* se mărește cu 1000 de lei salariul orar al angajaților care au
    mai mult de un număr de ani vechime specificați la execuție prin variabila ANI_ETALON */
    UPDATE personal
    SET salorar = salorar + 1000
    WHERE MONTHS_BETWEEN (SYSDATE,datasv) / 12 >= ani_etalon ;
    IF SQL%FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Exista cel putin un angajat cu vechime de peste ' ||
            ani_etalon || ' ani ' ) ;
        numar := SQL%ROWCOUNT ;
        DBMS_OUTPUT.PUT_LINE('De fapt, numarul lor este ' || numar) ;
    ELSE
        DBMS_OUTPUT.PUT_LINE('Nu exista nici un angajat asa de matur ! ' ) ;
    END IF ;
END;
/
```

Al doilea exemplu de cursor implicit e un pic mai complex și vrea să scoată în evidență faptul că, în cazul cursorului creat printr-o frază SELECT, atributul SQL%FOUND, util la comenzile DML, nu este de mare folos, deoarece, dacă interogarea nu extrage nici o linie, se declanșează excepția NO_DATA_FOUND.

Listing 8.11. Al doilea exemplu de cursor implicit

```
/* Al doilea exemplu de cursor implicit */
DECLARE
    marca_etalon personal.marca%TYPE := 1011 ;
    v_nume personal.numepren%TYPE ;
BEGIN
    /* în cazul SELECT-ului, dacă nu există nici o înregistrare care
    să îndeplinească condiția, se declanșează excepția */
    SELECT numepren INTO v_nume FROM personal
    WHERE marca = marca_etalon ;

    -- în acest punct se ajunge numai dacă SELECT-ul extrage o înregistrare
    IF SQL%FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Exista angajatul cu marca' || marca_etalon) ;
    ELSE
        -- aceasta ramura nu se va executa niciodată, din cauza excepției NO_DATA_FOUND
        DBMS_OUTPUT.PUT_LINE('Nu exista angajatul cu marca' || marca_etalon) ;
    END IF ;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        -- aici sare execuția blocului dacă SELECT-ul nu extrage nici o linie
        DBMS_OUTPUT.PUT_LINE('Nu exista angajatul cu marca ' || marca_etalon) ;
END;
/
```


Practic, după cum indică și comentariul de după comanda `SELECT`, la `IF`-ul respectiv se ajunge numai dacă a fost extrasă o linie, prin urmare comanda de test este inutilă.

8.5.2. Cursoare explicite

SQL a fost gândit ca limbaj orientat pe seturi de înregistrări. În practică, însă, rămân suficiente probleme care își găsesc rezolvarea pe baza unei logici la nivel de linie (înregistrare). Este unul din motivele pentru care toate SGBD-urile importante prezintă opțiuni de prelucrarea individuală a liniilor, în stilul celor din limbajele de programare din generația a III-a (3GL): COBOL, FORTRAN, BASIC, C și a SGBD-urilor de tip xBase (dBase, FoxPro etc.).

Figura 8.9 schematizează un cursor explicit declarat printr-o frază `SELECT` aplicată tabelului `PERSONAL`. Cursorul este pointerul către zona de lucru în care a fost deschis setul activ de înregistrări.

Tabela

MARCA	NUMEPREN	COMPART	DATASV	SALORAR	SALORARCO	COLABORATOR
101	Angajat 1	IT	10/12/1980	56000	55000	N
102	Angajat 2	CONTA	11/12/1978	57500	56000	N
103	Angajat 3	IT	7/2/1976	67500	66000	N
104	Angajat 4	PROD	1/5/1982	67500	66000	N
105	Angajat 5	IT	11/12/1977	62500	62000	N
106	Angajat 6	CONTA	4/11/1985	71500	70000	N
107	Angajat 7	CONTA	11/21/1991	61500	60000	N
108	Angajat 8	PROD	12/30/1994	54500	52000	N

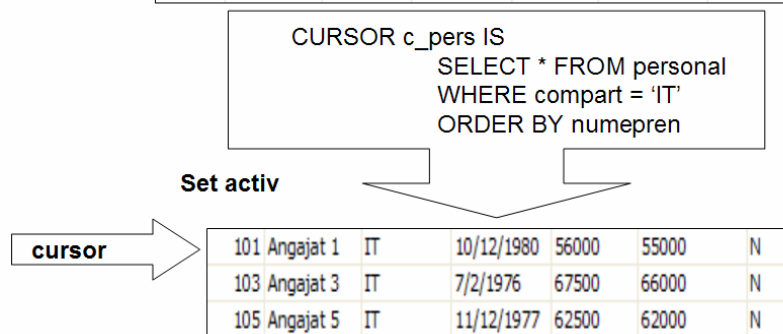


Figura 8.9. Cursor explicit

La deschiderea cursorului se execută interogarea-definiție și se determină setul activ de înregistrări. Din acest set, prin comanda de încărcare (`FETCH`) se poate încărca, pe rând fiecare linie. Încărcarea se face într-o variabilă de memorie compozită denumită de noi `rec_cursor`. După încărcare, valorile sunt citite din această variabilă ca dintr-una obișnuită. Similar 3GL, următoarea comandă de

încărcare va încerca să aducă în variabila compozită conținutul următoarei linii din setul activ. Prin intermediul atributului %FOUND (sau %NOTFOUND) al cursorului, se poate testa prin program momentul în care înregistrările din set s-au epuizat. Iată, în listing 8.12, o primă schemă de lucru cu un cursor explicit. Operațiunile principale sunt:

- declararea cursorului printr-o frază `SELECT (CURSOR nume IS SELECT...)`;
- declararea variabilei în care va fi stocată o linie a cursorului;
- deschiderea cursorului (`OPEN`);
- încărcarea următoarei linii din cursor (`FETCH`);
- structura de ciclare ce include, obligatoriu, o comandă de încărcare a următoarelor linii din cursor; altminteri bucla se repetă la infinit (sau la reboot, deși există și opțiuni mai blajine de a opri un bloc PL/SQL năraș).

Listing 8.12. Schema generală 1 de lucru cu un cursor explicit

```

DECLARE
  -- cursorul se declară printr-o frază SELECT
  CURSOR c_cursor IS
    SELECT .....

  -- se declară variabilă compusă ce va stoca o linie a cursorului
  rec_cursor c_cursor%ROWTYPE ;
...
BEGIN
  ...
  OPEN c_cursor      /* la deschidere se executa fraza SELECT-definiție
                     și se rezervă o zonă de memorie pentru liniile extrase */

  FETCH c_cursor INTO rec_cursor      /* se încarcă prima linie din cursor
                                     în variabila rec_cursor */

  WHILE c_cursor%FOUND LOOP          /* se execută bucla atâta timp cât se reușește
                                     încărcarea unei alte linii din cursor */
    ...
    ... -- corpul buclei
    ...

    FETCH c_cursor INTO rec_cursor    /* se încearcă încărcarea următoarei linii din cursor;
                                     dacă nu s-a putut, rezultă că înregistrările cursorului
                                     sunt epuizate, iar c_cursor%FOUND are
                                     valoarea logică FALSE */

  END LOOP ;
  CLOSE c_cursor      /* de multe ori, un lucru deschis trebuie închis la loc */
  ....
END ;

```

Aplicăm această schemă pentru o problemă relativ simplă. Să presupunem că în urma unor dezbateri aprinse comitetul director (board-ul, cum sună la anglofili) s-a hotărât să mărească salariile orare. Aceasta este vestea bună. Vestea proastă este că sporul (ce-i, drept, cu doar 10% din salariul orar în lei (ușori)) va fi acordat

pe compartimente, din fiecare compartiment fiind “gratulați” trei angajați, cei care au cei mai mulți ani de vechime (am uitat să vă spunem că acțiunea se înscrie într-un ciclu mai larg de combatere a emigrației tinerilor către Vest). Prin urmare, primesc câte 10% în plus la salariul orar cei mai vechi trei angajați din fiecare compartiment. Totuși, pentru a evita vărsarea de sânge, comitetul a decis că, dacă sunt și alți angajați cu același număr de ani de vechime cu al treilea (ca vechime) membru al compartimentului, să se acorde și acestora. Blocul din listing 8.13 operează mărirea de salariu atât de mult dorită (de către cei care vor beneficia de ea).

Listing 8.13. Cursor explicit pentru mărirea salariilor orare ale unor angajați

```

/* se crește cu 10% salariul orar al celor mai vechi TREI angajați din fiecare compartiment.
Atenție, dacă există mai mulți angajați cu aceeași vechime ca a unuia din cei trei,
se acordă sporul și acestora !!! */
DECLARE
    -- se declară cursorul în care se calculează numărul de ani de vechime
    -- iar liniile se ordonează pe compartimente și după anii de vechime
    CURSOR c_salariati IS
    SELECT marca, numepren, compart, salorar,
           TRUNC(MONTHS_BETWEEN(SYSDATE, datasv)/12, 0)
           AS ani_vechime
    FROM personal ORDER BY compart, 5 DESC ;

    rec_salariati c_salariati%ROWTYPE ;

    -- variabila V_COMPART va stoca, în orice moment, codul compartimentului
    -- celui mai recent angajat procesat
    v_compart personal.compart%TYPE := 'XYZ';

    -- variabila V_ANI conține, la un moment dat, anii de vechime celui mai recent
    -- angajat din compartiment căruia i s-a acordat sporul
    v_an INTEGER := 99 ;

    -- contorul numărului de angajați dintr-un compartiment care au primit deja
    -- mărirea de salariu orar
    numar INTEGER := 1 ;

BEGIN
    -- se deschide cursorul
    OPEN c_salariati ;

    -- se încarcă prima înregistrare în variabila compozită REC_SALARIATI
    FETCH c_salariati INTO rec_salariati ;

    -- se stabilesc condițiile pentru iterație
    WHILE c_salariati%FOUND LOOP
        IF rec_salariati.compart <> v_compart THEN      -- s-a schimbat compartimentu' !
            /* trebuie reinițializate variabilele NUMAR, V_COMPART și V_ANI */
            numar := 1 ;
            v_compart := rec_salariati.compart ;
            v_an := rec_salariati.ani_vechime ;

            /* fiind vorba de primul angajat din compartiment, deci cu numărul
            cel mai mare de ani de vechime, i se acordă automat sporul */
            UPDATE personal

```

```

        SET salorar = salorar + salorar * .1
        WHERE marca = rec_salariati.marca ;

ELSE -- prezentul angajat face parte din același compartiment ca
     -- și precedentul angajat

    IF numar > 3 AND rec_salariati.ani_vechime < v_ani THEN
        -- pentru acest compartiment, acordarea sporului este epuizată
        NULL ;
    ELSE
        -- se acordă sporul și acestui angajat
        UPDATE personal
        SET salorar = salorar + salorar * .1
        WHERE marca = rec_salariati.marca ;

        -- se actualizează variabilele contor și etalon
        numar := numar + 1 ;
        v_ani := rec_salariati.ani_vechime ;
    END IF ;

END IF;
-- se încearcă încărcarea următoarei înregistrări din cursor
FETCH c_salariati INTO rec_salariati ;
END LOOP ;
CLOSE c_salariati ;
END ;

```

Parcurgerea înregistrărilor are loc într-un singur sens. Dacă se dorește revenirea la o înregistrare anterioară, cursorul trebuie închis, redeschis, și re-parcurse înregistrările precedente. În afara atributului %FOUND (respectiv %NOTFOUND), precum și %ROWCOUNT care returnează numărul liniilor încărcate din cursor la un moment dat, pentru cursoarele explicite mai pot fi folosite și atributul %ISOPEN care are valoarea logică TRUE dacă respectivul cursor este deschis în momentul "citirii" atributului.

A doua schemă de folosirea a unui cursor, cea descrisă în listing 8.14, este ceva mai simplă datorită următoarelor avantaje:

- nu mai este necesară declararea explicită a variabilei în care se citește o înregistrare din cursor;
- cursorul nu mai trebuie deschis explicit, și nici închis (ambele operațiuni se realizează automat);
- încărcarea următoarei înregistrări se face automat la reluarea buclei.

Listing 8.14. A doua schemă generală de folosire a unui cursor explicit

```

DECLARE
    -- cursorul se declară printr-o frază SELECT
    CURSOR c_cursor IS
        SELECT ...

    -- nu mai este necesară declararea variabilei compozite REC_CURSOR
...
BEGIN
...
    FOR rec_cursor IN c_cursor LOOP /* automat se execută și deschiderea

```

```

...
... -- corpul buclei
...

-- citirea următoarei înregistrări se realizează automat
END LOOP ;
/* de multe ori, un lucru deschis trebuie închis la loc. NU ȘI DE DATA ACEASTA ! */
...
END ;

```

Folosind această schemă de lucru cu un cursor, rescriem blocul din listing 8.13, noua formă a programului fiind cea prezentată în listing 8.15.

Listing 8.15. O doua variantă de lucru cu un cursor explicit (C_SALARIATI)

```

...
DECLARE
... -- nu se mai declară REC_SALARIATI
...
BEGIN
  FOR rec_salariati IN c_salariati LOOP
    IF rec_salariati.compart <> v_compart THEN    -- s-a schimbat compartimentu' !
      /* trebuie reinițializate variabilele NUMAR, V_COMPART și V_ANI */
      numar := 1 ;
      v_compart := rec_salariati.compart ;
      v_anii := rec_salariati.ani_vechime ;

      /* fiind vorba de primul angajat din compartiment, deci cu numărul
         cel mai mare de ani de vechime, i se acordă automat sporul */
      UPDATE personal
      SET salorar = salorar + salorar * .1
      WHERE marca = rec_salariati.marca ;

    ELSE    -- prezentul angajat face parte din același compartiment ca
             -- și precedentul angajat

      IF numar > 3 AND rec_salariati.ani_vechime < v_anii THEN
        -- pentru acest compartiment, acordarea sporului este epuizată
        NULL ;

      ELSE
        -- se acordă sporul și acestui angajat
        UPDATE personal
        SET salorar = salorar + salorar * .1
        WHERE marca = rec_salariati.marca ;

        -- se actualizează variabilele contor si etalon
        numar := numar + 1 ;
        v_anii := rec_salariati.ani_vechime ;
      END IF ;
    END IF ;
    -- încărcarea următoarei înregistrări din cursor se face automat
  END LOOP ;
END ;

```

8.5.3. Un cursor parametrizat și o excepție-utilizator.

Modificăm un pic problema precedentă. Salariul orar crește tot cu 10% pentru (tot) cei mai vechi trei angajați din fiecare compartiment. Mâna șefilor nu este prea largă, așa că se stabilește la nivelul firmei un număr maxim de persoane care beneficiază de creșterea salarială. Dacă acest număr este depășit, se anulează toate creșterile, urmând ca procedura de alocare să fie rediscutată.

Astfel reformulată, problema constituie oportunitatea redactării unui bloc PL/SQL în care vom folosi două cursori, unul pentru compartimente (C_COMPART), iar celălalt pentru salariați (C_SALARIATI). De această dată C_SALARIATI este parametrizat. Ideea e că pentru fiecare înregistrare din C_COMPART (care corespunde unui compartiment), la deschidere, C_SALARIATI să conțină numai angajații ce fac parte din compartimentul respectiv, ordonați după anii de vechime.

Odată stabilit numărul maxim admis al celor care pot beneficia de sporul de salariu, și stocat în variabila nr_max, după fiecare angajat ce primește sporul se incrementează variabila-contor la nivelul întregii firme, nr_total, variabilă care se compară cu nr_max. În momentul în care nr_total este mai mare decât nr_max, se declanșează excepția utilizator prea_multi. Blocul din listing 8.16 realizează efectiv ceea ce tocmai a fost descris în cuvinte.

Listing 8.16. Cursor parametrizat și o excepție utilizator

```

/* se crește cu 10% salariul orar al celor mai vechi TREI angajați din fiecare compartiment.
De data aceasta se stabilește un număr maxim de angajați care pot primi sporul și se generează
o eroare când acest NR_MAX este depășit */
DECLARE
    /* folosim un cursor și pentru compartimente */
    CURSOR c_compart IS SELECT DISTINCT compart FROM personal ;

    /* cursorul C_SALARIATI se întoarce, un pic schimbat */
    CURSOR c_salariati (v_compart personal.compart%TYPE) IS
        SELECT marca, numepren, compart, salorar,
            TRUNC(MONTHS_BETWEEN(SYSDATE, datasv)/12, 0) AS ani_vechime
        FROM personal WHERE compart = v_compart ORDER BY 3 DESC ;

    v_ani INTEGER ;                -- V_ANI își păstrează semnificația

    /* NR_COMPART reprezintă câți angajați din compartimentul curent au primit deja sporul,
    în timp ce NR_TOTAL câți angajați din firmă au primit sporul la un moment dat */
    nr_compart INTEGER := 1 ;
    nr_total INTEGER := 1 ;

    nr_max INTEGER := 10 ;        -- declarăm câți angajați pot primi sporul
    prea_multi EXCEPTION ;       -- în premieră, definim și o excepție utilizator
BEGIN
    -- secvența repetitivă pentru prelucrarea înregistrărilor cursorului C_COMPART
    FOR rec_compart IN c_compart LOOP
        -- inițializarea variabilelor la nivel de compartiment
        nr_compart := 1 ;
        v_ani := 99 ;

        /* secvența iterativă pentru prelucrarea înregistrărilor cursorului C_SALARIATI,

```

```

care, spre deosebire de celălalt, este un cursor parametrizat */
FOR rec_salariati IN c_salariati (rec_compart.compart) LOOP
  IF nr_compart > 3 AND rec_salariati.ani_vechime < v_ani THEN
    -- pentru acest compartiment, acordarea sporului este epuizată
    NULL ;
  ELSE
    -- se acordă sporul și acestui angajat
    UPDATE personal
    SET salorar = salorar + salorar * .1
    WHERE marca = rec_salariati.marca ;

    -- se actualizează variabilele contor și etalon
    nr_compart := nr_compart + 1 ;
    v_ani := rec_salariati.ani_vechime ;
    nr_total := nr_total + 1 ;

    -- se verifică dacă numărul total admis nu a fost deja depășit
    IF nr_total > nr_max THEN
      -- se declanșează excepția-utilizator
      RAISE prea_multi ;
    END IF ;
  END IF ;
END LOOP;
END LOOP ;
EXCEPTION
/* tratarea excepției utilizator */
WHEN prea_multi THEN
  ROLLBACK ;
  DBMS_OUTPUT.PUT_LINE(
    'A fost depasit numarul maxim admis de angajati care pot primi sporul !' ) ;
END ;

```

În locul variabilei `nr_compart` care, la nivel de compartiment, numără câți salariați au beneficiat de creștere, ne putem folosi de atributul `%ROWCOUNT` al cursorului `C_SALARIATI`, deoarece în această ultimă versiune a blocului numărul de linii procesate din acest cursor este egal cu numărul angajaților ce au fost “parcurși”. Listing-ul 8.17 pune în evidență diferențele față de varianta din listing 8.16.

Listing 8.17. Folosirea atributului `%ROWCOUNT` al cursorului `C_SALARIATI`

```

...
DECLARE
  ...-- nu se mai declară NR_COMPART
  ...
BEGIN
  FOR rec_compart IN c_compart LOOP
    v_ani := 99 ;
    FOR rec_salariati IN c_salariati (rec_compart.compart) LOOP
      -- se modifică IF-ul
      IF c_salariati%ROWCOUNT > 3 AND rec_salariati.ani_vechime < v_ani THEN
        NULL ;
      ELSE
        ...
      END IF ;
    END LOOP;
  END LOOP;
END LOOP ;

```

```
EXCEPTION
```

```
...
```

```
END;
```

8.5.4. Cursoare explicite, implicite, variabile de tip înregistrare și o interogare scalară

Titlul de mai sus indică ingredientele folosite în blocul PL/SQL următor care își propune o sarcină deosebit de importantă pentru aplicația SALARIZARE la a cărei bază de date trudim de câteva capitole: actualizarea tabelor SPORURI și SALARII pe baza pontajelor. Dacă în PONTAJE sunt introduse, pentru fiecare angajat, datele zilnice legate de lucru sau concediu, în SPORURI și SALARII fiecărui angajat îi corespunde o înregistrare la nivel de lună (de fapt, an și lună). Aceste ultime două tabele centralizează informații preluate din pontaje.

Astfel, în SPORURI:

- sporul de vechime (SPORURI.SpVech) se calculează prin înmulțirea venitului de bază pentru luna respectivă cu procentul sporului de vechime;
 - procentul sporului de vechime se acordă pe tranșe, în funcție de anii de vechime ai angajatului:
 - anii de vechime se calculează cu ajutorul funcției MONTHS_BETWEEN care numără lunile scurse de la data angajării sau data de la care se calculează sporul de vechime (DataSV) și data de 1 a lunii de referință;
 - data de 1 a lunii de referință se specifică cu ajutorul funcției TO_DATE: `TO_DATE('01/'||v_luna||'/'||v_an, 'DD/MM/YYYY')`;
 - rezultatul funcției MONTHS_BETWEEN se împarte la 12 pentru a afla numărul anilor;
 - procentul sporului de vechime se determină folosind numărul anilor ca argument într-o frază SELECT aplicată asupra tablei TRANSE_SV: `SELECT procent_sv INTO v_proc_sv FROM transe_sv WHERE v_aniv_echime >= ani_limita_inf AND v_aniv_echime < ani_limita_sup ;`
 - valoarea sporului de vechime se calculează ca produs între procentul sporului de vechime și venitul de bază, venit de bază care este alcătuit din venitul pentru munca prestată și venitul corespunzător concediului:
 - venitul pentru munca prestată se calculează înmulțind salariul orar (PERSONAL.SalOrar) cu numărul total de ore lucrate în luna respectivă;
 - venitul corespunzător concediului se calculează înmulțind salariul orar pentru concediu (PERSONAL.SalOrarCO) cu

numărul total de ore petrecute în concediu în luna respectivă;

- orele de noapte (SPORURI.OreNoapte) constituie suma orelor de noapte pentru luna respectivă, sumă preluată din PONTAJE.OreNoapte;
- sporul de noapte (SPORURI.SpNoapte) se calculează prin aplicarea unui procent de 15% asupra produsului dintre SPORURI.OreNoapte și SALARII.SalOrar.

Calcululele în tabele SALARII privesc următoarele câmpuri:

- orele lucrate (SALARII.OreLucrate) constituie suma orelor lucrate în luna respectivă, sumă preluată din PONTAJE.OreLucrate;
- orele petrecute în concediu de odihnă (SALARII.OreCO) constituie suma orelor de acest tip preluată din PONTAJE.OreCO;
- venitul de bază (SALARII.VenitBaza) este alcătuit (după cum am văzut mai sus) din venitul pentru munca prestată (PERSONAL.SalOrar * SALARII.OreLucrate) plus venitul corespunzător concediului de odihnă (PERSONAL.SalOrarCO * SALARII.OreCO)
- sporuri (SALARII.Sporuri) reprezintă suma: SPORURI.SpVech + SPORURI.SpNoapte + SPORURI.AlteSp.

Toate acest calcule constituie subiectul blocului PL/SQL prezentat în listing 8.18. Centralizarea orelor lucrate, de concediu și de noapte din luna curentă pentru fiecare angajat se realizează pe baza datelor din tabela PONTAJE în cursorul C_ORE, prin gruparea după atributul Marcă. Pentru angajatul corepunzător înregistrării curente din cursor (C_ORE.Marca) datele din PERSONAL (DataSV, SalOrar, SalOrarCO) sunt preluate într-o variabilă compozită (rec_personal) de tip RECORD (t_personal).

Listing 8.18. Actualizarea tabelor SPORURI și SALARII

```

/* Acest bloc actualizează, pentru o luna dată, tabelele SPORURI și SALARII
   pe baza datelor dn PONTAJE */
DECLARE
  v_an salarii.an%TYPE := 2003 ;
  v_luna salarii.luna%TYPE := 1 ;

  -- C_ORE calculează totalul orelor lucrate, de concediu și de noapte pentru luna dată
  CURSOR c_ore IS
    SELECT marca, SUM(orelucrate) AS ore_l, SUM(oreco) AS ore_co,
           SUM(orenoapte) AS ore_n
    FROM pontaje
    WHERE TO_NUMBER(TO_CHAR(data,'YYYY')) = v_an
          AND TO_NUMBER(TO_CHAR(data,'MM')) = v_luna
    GROUP BY marca ;

  /* se declară un tip RECORD pentru extragerea informațiilor necesare calculului
     venitului de bază și sporurilor */
  TYPE t_personal IS RECORD
    (datasv personal.datasv%TYPE,
     salorar personal.salorar%TYPE,
```

```

        salorarco personal.salorarco%TYPE );

rec_personal t_personal ;                -- o variabila de tipul de mai sus

-- variabile necesare calculului sporului de vechime
v_an_vechime NUMBER(4,2) ;
v_proc_sv transe_sv.procent_sv%TYPE ;
v_spvech sporuri.spvech%TYPE ;

-- variabile pentru venitul de bază, sporul de noapte și total sporuri
v_venitbaza salarii.venitbaza%TYPE ;
v_sjnoapte sporuri.sjnoapte%TYPE ;
v_sporuri salarii.sporuri%TYPE ;

BEGIN
FOR rec_ore IN c_ore LOOP
    -- se extrag datele pentru salariatul din linia curentă a cursorului C_ORE
    SELECT datasv, salorar, salorarco INTO rec_personal
    FROM personal WHERE marca = rec_ore.marca ;

    -- pentru calculul anilor de vechime se recurge la funcția MONTHS_BETWEEN
    v_an_vechime := MONTHS_BETWEEN( TO_DATE('01/'||v_luna||'/'||v_an, 'DD/MM/YYYY'),
        rec_personal.datasv) / 12 ;

    -- prin consultarea tabelii TRANSE_SV se determină procentul sporului de vechime
    SELECT procent_sv INTO v_proc_sv FROM transe_sv
    WHERE v_an_vechime >= ani_limita_inf AND v_an_vechime < ani_limita_sup ;

    /* se calculează venitul de bază, sporul de vechime și sporul de noapte;
        funcția ROUND asigură rotunjirea la ordinul sutelor */
    v_venitbaza := ROUND(rec_ore.ore_l * NVL(rec_personal.salorar,0) +
        rec_ore.ore_co * NVL(rec_personal.salorarco,0),-2) ;
    v_spvech := ROUND(v_venitbaza * v_proc_sv / 100, -3) ;
    v_sjnoapte := ROUND(rec_ore.ore_n * NVL(rec_personal.salorar,0) * .15, -3) ;

    -- se actualizează tabela SPORURI pentru angajatul curent
    UPDATE sporuri
    SET spvech = v_spvech, orenoapte = rec_ore.ore_n, spnoapte = v_sjnoapte
    WHERE marca=rec_ore.marca AND an=v_an AND luna=v_luna ;

    /* dacă UPDATE nu a prelucrat nici o linie, se inserează o înregistrare în SPORURI */
    IF SQL%NOTFOUND THEN -- reamintiți-vă discuția de la cursoare implicite
        INSERT INTO sporuri VALUES (rec_ore.marca, v_an, v_luna,
            v_spvech, rec_ore.ore_n, v_sjnoapte, 0) ;
    END IF ;

    -- se procedează analog pentru tabela SALARII
    UPDATE salarii
    SET orelucrate = rec_ore.ore_l, oreco = rec_ore.ore_co,
        venitbaza = v_venitbaza, sporuri =
            (SELECT spvech + spnoapte + altesp
             FROM sporuri
             WHERE an=v_an AND luna=v_luna AND marca = rec_ore.marca)
    WHERE marca=rec_ore.marca AND an=v_an AND luna=v_luna ;
    IF SQL%NOTFOUND THEN
        INSERT INTO salarii VALUES (rec_ore.marca, v_an, v_luna, rec_ore.ore_l,
            rec_ore.ore_co, v_venitbaza,
            (SELECT spvech + spnoapte + altesp
             FROM sporuri

```

```

WHERE an=v_an AND luna=v_luna AND marca = rec_ore.marca),
0, 0);
END IF;
END LOOP;
END;

```

Logica programului pe bază de parcurgere a liniei cu linia a cursorului C_ORE. (FOR rec_ore IN c_ore LOOP) Pentru fiecare înregistrare din cursor, ce reprezintă situația lucrului unui angajat pe luna dată, se încarcă printr-un SELECT în variabila de tip înregistrare (rec_personal) datele generale ale angajatului respectiv: data de la care se calculează sporul de vechime (DataSV), salariul orar (SalOrar) și salariul orar pentru calculul indemnizației de concediu (SalOrarCO) SELECT datasv, salorar, salorarco INTO rec_personal FROM personal WHERE marca = rec_ore.marca.

Anii de vechime se determină prin funcția MONTHS_BETWEEN și se stochează în variabila v_ani_vechime. Procentul sporului de vechime se memorează în variabila v_sp_vech în urma unui SELECT aplicat tabelii TRANSE_SV: SELECT procent_sv INTO v_proc_sv FROM transe_sv WHERE v_ani_vechime >= ani_limita_inf AND v_ani_vechime < ani_limita_sup. Variabilele care se referă la venitul de bază (v_venitbaza), sporul de vechime (v_spvech) și sporul de noapte (v_spnoapte) sunt calculate după expresii pe care le-am prezentat ceva mai sus.

Urmează secvența de folosire a primului cursor implicit. Comanda UPDATE sporuri... încearcă să modifice atributele SPORURI.SpVech, SPORURI.OreNoapte și SPORURI.SpNoapte. Modificarea operează numai dacă există deja înregistrarea pentru angajatul și luna curente în SPORURI. Dacă nu, atunci atributul SQL%NOTFOUND întoarce TRUE și linia respectivă trebuie inserată.

Al doilea cursor implicit este corespunzător celei de-a doua comenzi UPDATE care operează asupra tabelii SALARII. Pentru un plus de atractivitate, actualizarea/inserarea valorii totalului sporurilor se realizează apelând la o interogare scalară.

8.5.5. Clauzele FOR UPDATE OF și WHERE CURRENT OF

Fie și numai din cele prezentate, importanța cursorilor în blocurile PL/SQL de actualizare a tabelor este evidentă. Pentru păstrarea coerenței datelor și pentru optimizarea actualizărilor, în Oracle există două opțiuni deosebit de utile pentru aplicațiile multi-utilizator, mai ales în condițiile unui mare număr de înregistrări în tabele.

Prima clauză, FOR UPDATE OF se include în definiția cursorului, la final, și indică SGBD-ului că acel cursor servește la modificarea câmpului sau câmpurilor specificate. În virtutea acestei clauze, în blocul prezentat în listing 8.19, la deschiderea cursorului C_SALARIATI (deschidere automată realizată prin FOR rec_salariati IN c_salariati (rec_compart.compart) LOOP) înregistrările din PERSONAL corespundente celor din cursor vor fi blocate, astfel

încât nici un alt utilizator/aplicație nu le poate actualiza până la închiderea tranzacției (prin comanda COMMIT sau ROLLBACK). Problema majoră care poate să apară este ca una sau multe dintre înregistrări să fi fost deja blocate de alt utilizator/aplicație, caz în care blocul poate aștepta la nesfârșit. Pentru preîntâmpinarea unei asemenea situații se poate folosi clauza NOWAIT (CURSOR c_salariati IS SELECT... FROM... WHERE... FOR UPDATE OF... NOWAIT).

Listing 8.19. Clauzele FOR UPDATE OF... și WHERE CURRENT OF

```

...
/* ... acelasi enunț ca în Listing 8.16 și 8.17. ... */
DECLARE
    -- nu mai declarăm cursorul pentru compartimente */
    /* pentru cursorul (parametrizat) C_SALARIATI se folosește clauza FOR UPDATE OF */
    CURSOR c_salariati (v_compart personal.compart%TYPE) IS
        SELECT marca, numepren, compart, salorar,
            TRUNC(MONTHS_BETWEEN(SYSDATE, datasv)/12, 0) AS ani_vechime
        FROM personal WHERE compart = v_compart ORDER BY 3 DESC
        FOR UPDATE OF salorar ;

    v_anii INTEGER ; /* anii de vechime ai celui mai recent angajat cu mărire de salariu */
    nr_total INTEGER := 7 ; -- nr. total salariaților carora li s-a mărit salariul orar
    nr_max INTEGER := 15 ; -- nr maxim de salariați ce pot beneficia de mărirea salariului

    prea_multi EXCEPTION ; -- excepție

BEGIN
    -- de data aceasta cursorul este definit ad-hoc, prin subconsultare
    FOR rec_compart IN (SELECT DISTINCT compart FROM personal ) LOOP
        v_anii := 99 ;

        /* secvența pentru prelucrarea cursorului C_SALARIATI. La deschiderea cursorului
           se blochează înregistrările corespondente din tabela PERSONAL */
        FOR rec_salariati IN c_salariati (rec_compart.compart) LOOP
            IF c_salariati%ROWCOUNT > 3 AND rec_salariati.ani_vechime < v_anii THEN
                NULL ;
            ELSE
                UPDATE personal SET salorar = salorar + salorar * .1
                WHERE CURRENT OF c_salariati ;

                v_anii := rec_salariati.ani_vechime ;
                nr_total := nr_total + 1 ;
                IF nr_total > nr_max THEN
                    RAISE prea_multi ;
                END IF ;
            END IF ;
        END LOOP ;
    COMMIT ;
    END LOOP ;
EXCEPTION
    ...
END ;

```

A doua clauză discutată este WHERE CURRENT OF și “lucrează” împreună cu FOR UPDATE OF. În lipsa acesteia, precizarea liniei ce urmează a fi modificată în PERSONAL se realizează printr-o clauză WHERE de genul WHERE marca =

`rec_salariati.marca`. La crearea cursorului, Oracle crează legătura dintre înregistrările din setul activ și cele din tabelă. Astfel încât, specificând `WHERE CURRENT OF c_salariati` se obține un important câștig de timp.

Ar mai fi de adăugat, tot ca premieră, clauza `FOR` care folosește, în loc de cursor explicit, o subconsultare care crează cursorul chiar la execuție.

8.6. Colecții în PL/SQL

O colecție poate fi definită frugal drept un grup ordonat de elemente de același tip ce pot fi referite printr-un indice care arată poziția fiecărui element în cadrul grupului. Nu e limbaj întreg cel care nu are opțiuni pentru gestionarea vectorilor, listelor etc. PL/SQL nu putea face opinie separată, deși orientarea sa a fost, de la început, pentru lucrul cu date. Mult timp, singurul tip din categoria colecțiilor a fost tabloul PL/SQL (*index-by table*). Între timp, lucrurile au evoluat, așa că, în prezent, Oracle 9i2 dispune, pe lângă “clasicele” tablouri, numite și vectori asociativi, de tabele încapsulate (*nested tables*) și vectori cu mărime variabilă (*varrays*). Vectorii asociativi și tabelele încapsulate desemnează tablourile sau tabelele PL/SQL.

8.6.1. Vectori asociativi

Vectorii asociativi sunt seturi de perechi cheie (indice)-valoare. Au apărut în PL/SQL 2.0 (Oracle 7) și ameliorați în PL/SQL 2.3 (Oracle 7.3). Accesul la o anumită valoare se realizează cunoscându-i cheia (indicele). Ceea ce distinge această categorie de masiv este că indicele pentru precizarea poziției unui element în cadrul masivului poate fi un număr întreg (deci și negativ), și chiar un șir de caractere, ceea ce în cazul vectorilor “tradiționali” nu este posibil nicicum. În plus, vectorii asociativi nu pot fi stocați “nativ” în baza de date, ci numai în sesiunea curentă (în blocuri anonime, proceduri, funcții și pachete).

Componentele nu sunt obligatoriu consecutive. Atunci când are loc o primă operațiune de atribuire, de exemplu `un_vector (-3) := -10`, se adaugă în tablou cheia (-3) asociată valorii (-10) respective. Următoarele referiri ale componentei folosind cheia respectivă vor modifica valoarea din tablou. Componentele care ar avea cheia -2, -1, 0 sunt neinițializate. Orice tentativă de a le citi/afișa declanșează excepția `NO_DATA_FOUND`.

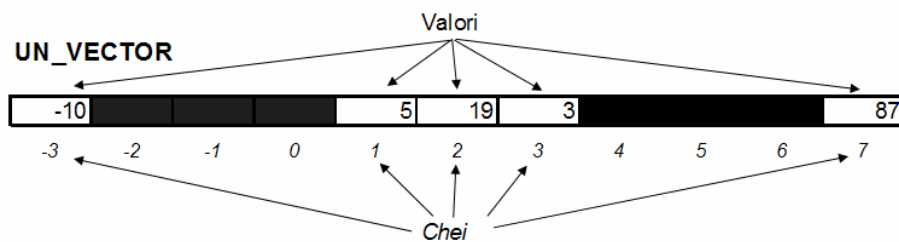


Figura 8.10. Tablou asociativ (INDEX-BY)

Ca și celelalte colecții, tablourile asociative au câteva atribute și metode:

- COUNT indică numărul componentelor inițializate (în cazul nostru 5);
- FIRST furnizează indicele (cheia) primului element din tablou (-3);
- LAST furnizează indicele (cheia) ultimului element din tablou (7);
- EXISTS întoarce valoarea logică TRUE dacă există în tablou componenta cu indexul specificat.
- NEXT – metodă prin care se realizează poziționarea pe următoarea componentă inițializată a tabloului, relativ la componenta curentă;
- PRIOR – metodă prin care se realizează poziționarea pe componenta precedentă a tabloului, relativ la componenta curentă;
- DELETE – metodă care șterge una, mai multe sau toate componentele tabloului.

Blocul prezentat în listing 8.20 ilustrează modalitatea de declarare și manipulare a unui tablou asociativ. În secțiunea declarativă se definește mai întâi un tip de tablou care va conține variabile de tip NUMBER(14, 2) . Prezența clauzei INDEX BY BINARY_INTEGER face diferența “declarativă” dintre tablouri asociative și tabele (tablouri) încapsulate. Variabila un_vector este tabloul propriu-zis. Zona executabilă a blocului ilustrează un mod eronat și un altul corect de parcurgere a componentelor.

Listing 8.20. Un tablou asociativ

```

/* tablouri asociative - vol. 1 */
DECLARE
  -- declararea tipului de tablouri asociative ce pot conține numere reale
  TYPE t_vector IS TABLE OF NUMBER(14,2) INDEX BY BINARY_INTEGER ;

  -- iată și tabloul propriu-zis
  un_vector t_vector ;

BEGIN
  -- inițializarea a câteva dintre componente
  un_vector(-3) := -10 ;
  un_vector(1) := 5 ;
  un_vector(2) := 19 ;
  un_vector(3) := 3 ;
  un_vector(7) := 87 ;

  DBMS_OUTPUT.PUT_LINE('Tabloul are '||un_vector.COUNT||' componente ');
  DBMS_OUTPUT.PUT_LINE('Indexul primeia este '||un_vector.FIRST);
  DBMS_OUTPUT.PUT_LINE('Indexul ultimei este '||un_vector.LAST);
  DBMS_OUTPUT.PUT_LINE('-----');

  /* acesta e un mod eronat de a parcurge vectorul, deoarece pot
  exista și indecși negativi, iar valorile nu sunt consecutive */
  BEGIN
    FOR i IN 1..un_vector.COUNT LOOP
      /* la prima componenta (dupa 1) neinițializată, se declanșează excepția
      NO_DATA_FOUND */
      DBMS_OUTPUT.PUT_LINE('Componenta '||i||' este '|| un_vector(i));
    END LOOP ;
  
```

```

EXCEPTION
-- preluăm eroarea (acesta e motivul pentru care am folosit un bloc inclus)
WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Componenta neinitializata ');
WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Alta eroare');
END;
DBMS_OUTPUT.PUT_LINE('-----');
/* acesta este modul indicat de parcurgere a tabloului, bazat pe proprietățile FIRST și LAST */
FOR i IN un_vector.FIRST..un_vector.LAST LOOP
    -- folosind clauza EXISTS evităm declanșarea excepției
    IF un_vector.EXISTS(i) THEN
        DBMS_OUTPUT.PUT_LINE('Componenta '||i||' este '|| un_vector(i));
    ELSE
        DBMS_OUTPUT.PUT_LINE('Componenta '||i||' nu este initializata');
    END IF;
END LOOP;
END;

```

Rezultatele execuției blocului, cele din figura 8.11, sunt cât se poate de explicite.

```

SQL> @F:\ORACLE_CARTE\CAP08_PL_SQL1\LISTING08_20.SQL
52 /
Tabloul are 5 componente
Indexul primeia este -3
Indexul ultimea este 7
-----
Componenta 1 este 5
Componenta 2 este 19
Componenta 3 este 3
Componenta neinitializata
-----
Componenta -3 este -10
Componenta -2 nu este initializata
Componenta -1 nu este initializata
Componenta 0 nu este initializata
Componenta 1 este 5
Componenta 2 este 19
Componenta 3 este 3
Componenta 4 nu este initializata
Componenta 5 nu este initializata
Componenta 6 nu este initializata
Componenta 7 este 87

PL/SQL procedure successfully completed.

SQL> |

```

Figura 8.11. Rezultatele execuției blocului din listing 8.20

Pentru a ilustra folosirea atributelor și metodelor aferente unei colecții PL/SQL, ne propunem să rezolvăm următoarea problemă: dat fiind tabloul PL/SQL din blocul precedent, să se aranjeze consecutiv componentele inițializate ale tabloului, indicii având valori de la 1 la COUNT.

O primă variantă, cea din listing 8.21, folosește două tablouri, unul sursă și un altul destinație, aranjat după cum cere enunțul problemei. Cheile primeia și ultimei componente a tabloului se determină cu proprietățile FIRST și LAST.

Avansarea la următoarea componentă inițializată presupune folosirea metodei NEXT, iar ștergerea tabloului sursă se realizează cu metoda DELETE.

Listing 8.21. Dispunerea consecutivă componentelor unui tablou PL/SQL

```

/* tablouri PL/SQL - vol. II.                Tema: Reorganizarea tabloului PL/SQL, astfel încât
   prima componentă să aibă indexul 1 (sa nu existe indecși (chei) negativi(e)),
   iar componentele să fie consecutive (fără spații între ele) */
DECLARE
  TYPE t_vector IS TABLE OF NUMBER(14,2) INDEX BY BINARY_INTEGER ;
  vector_sursa t_vector ; -- tabloul inițial
  vector_destinatie t_vector ; -- tabloul după prelucrare

  /* câte un index pentru fiecare tablou */
  i_sursa BINARY_INTEGER ;
  i_destinatie BINARY_INTEGER := 1 ;
BEGIN
  -- inițializare "manuală"
  vector_sursa(-3) := -10 ;
  vector_sursa(1) := 5 ;
  vector_sursa(2) := 19 ;
  vector_sursa(3) := 3 ;
  vector_sursa(7) := 87 ;

  DBMS_OUTPUT.PUT_LINE('INITIAL ');
  DBMS_OUTPUT.PUT_LINE('Tabloul are ' || vector_sursa.COUNT || ' componente ');
  DBMS_OUTPUT.PUT_LINE('Indexul primeia este ' || vector_sursa.FIRST);
  DBMS_OUTPUT.PUT_LINE('Indexul ultimei este ' || vector_sursa.LAST);

  -- se inițializează indexul vectorului sursă
  i_sursa := vector_sursa.FIRST ;

  WHILE i_sursa <= vector_sursa.LAST LOOP
    vector_destinatie(i_destinatie) := vector_sursa(i_sursa) ;
    i_destinatie := i_destinatie + 1 ;

    -- deplasarea, în vectorul sursă, pe următoarea componentă inițializată
    i_sursa := vector_sursa.NEXT(i_sursa) ;
  END LOOP ;

  DBMS_OUTPUT.PUT_LINE(' ');
  DBMS_OUTPUT.PUT_LINE('DUPA PRELUCRARE ');
  DBMS_OUTPUT.PUT_LINE('Tabloul are ' || vector_destinatie.COUNT || ' componente ');
  DBMS_OUTPUT.PUT_LINE('Indexul primeia este ' || vector_destinatie.FIRST);
  DBMS_OUTPUT.PUT_LINE('Indexul ultimei este ' || vector_destinatie.LAST);

  -- în fine, se șterge tabloul sursă
  vector_sursa.DELETE ;
END ;

```

Blocul din listingul 8.22 pune în practică o altă idee de reorganizare: nu se mai recurge la alt vector, ci se mută ultima componentă pe primul spațiu liber (nefolosit) dintre două elemente, iar apoi se șterge ultima componentă, tabeloul fiind scurtat.

Listing 8.22. Dispunerea consecutivă componentelor unui tablou PL/SQL – varianta 2

```

/* tablouri PL/SQL - vol. III. Reorganizarea tabloului PL/SQL - varianta 2 */

```



```

DECLARE
    TYPE t_vector IS TABLE OF NUMBER(14,2) INDEX BY BINARY_INTEGER ;
    vector_sursa t_vector; -- tabloul inițial
    v_reia BOOLEAN := TRUE ; -- variabilă folosită la ciclare
BEGIN
    -- inițializare "manuală" și afișarea situației inițiale se păstrează
    ...
    WHILE v_reia LOOP
        v_reia := FALSE ;
        FOR i IN vector_sursa.FIRST..vector_sursa.LAST LOOP
            IF vector_sursa.EXISTS(i) THEN
                -- se testează dacă cheia elementului curent este negativă
                IF i <= 0 THEN
                    -- se mută componenta curentă după ultima
                    vector_sursa (vector_sursa.COUNT + 1) := vector_sursa (i) ;
                    vector_sursa.DELETE(i) ;
                    v_reia := TRUE ;
                    EXIT ;
                ELSE
                    -- elementul i există, iar cheia sa este pozitivă
                    NULL ;
                END IF ;
            ELSE
                -- elementul i nu e inițializat, așa că se preia valoarea ultimului din tablou
                vector_sursa (i) := vector_sursa.LAST ;
                -- se șterge ultimul element
                vector_sursa.DELETE(vector_sursa.LAST) ;
                v_reia := TRUE ;
                EXIT ;
            END IF ;
        END LOOP ;
    END LOOP ;
    -- afișarea vectorului vector_sursa după prelucrare
    ...
END ;

```

8.6.2. Tabele încapsulate

Acest tip de date a apărut în Oracle 8. Un tabel încapsulat (*nested table*) stochează un număr oarecare de componente, folosind numere secvențiale drept indici. Avantajul major al acestui gen de colecție ține de posibilitatea de a fi stocat în tabelele bazei și interogat/exploatat prin comenzi SQL. În cadrul unei tabele a bazei, un tabel încapsulat poate fi asimilat unei tabele relaționale cu un singur atribut. La încărcarea unui tabel încapsulat într-o variabilă PL/SQL fiecare linie a acestuia va avea un indice crescător (prima linie va fi prima componentă a variabilei, a doua linie va avea indicele 2 în variabilă s.a.m.d.).

Dimensiunea unui tabel încapsulat este nelimitată și variabilă. Ca și în cazul vectorilor asociativi, oricare componentă poate fi ștearsă (prin metoda `DELETE`), caz în care elementele tabelului nu mai sunt consecutive; metoda `NEXT` asigură, însă, saltul pe următoarea componentă inițializată. La inițializare, cheile (indicii) sunt obligatoriu secvențiali și pozitivi. Pentru ilustrarea lucrului cu un tablou încapsulat, creăm în prealabil un tip obiect numit `SCOLARITATE` destinat stocării traseului școlarității unei persoane: liceu, facultate, studii de perfecționare, școli

post-liceale, studii post-universitare, doctorale etc. Pentru aceasta în SQL*Plus lansăm comanda CREATE TYPE:

```
CREATE OR REPLACE TYPE scolaritate AS OBJECT (
    an_inceput NUMBER(4),
    an_final NUMBER (4),
    institutie VARCHAR2(50),
    specializare_sectie VARCHAR2(100) )
/
```

Blocul din listing 8.23 crează un tablou încapsulat de tipul obiectului scolaritate. La declarare, tabloul trebuie inițializat obligatoriu folosind constructorul ce are, automat, nume identic cu tipul (t_scolaritate). Astfel, tabloul va fi inițializat, deși nu are nici o componentă inițializată.

Listing 8.23. Un exemplu de folosire a tabelor încapsulate

```
/* Tablouri încapsulate (NESTED TABLES) - partea I */

/* Nu uitați ca, în prealabil, să lansați în SQL*Plus comanda pentru crearea
tipului SCOLARITATE */
DECLARE
    -- tipul de tabel încapsulat
    TYPE t_scolaritate IS TABLE OF scolaritate ;

    v_scolaritate t_scolaritate := t_scolaritate() ; /* inițializarea unui tabel încapsulat, chiar fără
                                                         elemente, este obligatorie ; altminteri, s-ar declanșa
                                                         excepția COLLECTION_IS_NULL.
                                                         T_SCOLARITATE() reprezintă constructorul
                                                         (are același nume ca și tipul însuși) */

BEGIN
    -- lungimea inițială a tabelului încapsulat era zero, așa că îl mărim un pic
    v_scolaritate.EXTEND ;

    -- prima perioadă de școlarizare
    v_scolaritate(1) := scolaritate (1990, 1994, 'Liceul de Informatica Iasi', 'Informatica') ;

    /* am prins curaj, așa că inițializăm și a doua componentă, ce reprezintă
    a doua perioadă de școlarizare */
    v_scolaritate.EXTEND ;
    v_scolaritate(2) := scolaritate (1994, 1998, 'Univ. Al.I.Cuza Iasi',
    'Facult. de Economie si Administrarea Afacerilor - spec. Informatica Economica') ;

    -- mai adăugăm trei componente la tablou
    v_scolaritate.EXTEND (3);

    -- lăsăm două componente neocupate, întrucât o vom inițializa numai pe a 4-a
    v_scolaritate(4) := scolaritate (2001, NULL, 'Univ. Al.I.Cuza Iasi',
    'FEAA - ELITEC - Master Sisteme Informationale (MIS)') ;

    DBMS_OUTPUT.PUT_LINE('Tabloul încapsulat are ' || v_scolaritate.COUNT || ' componente') ;
    DBMS_OUTPUT.PUT_LINE('Indexul primeia este ' || v_scolaritate.FIRST) ;
    DBMS_OUTPUT.PUT_LINE('Indexul ultimei este ' || v_scolaritate.LAST) ;

    FOR i IN v_scolaritate.FIRST..v_scolaritate.LAST LOOP
        DBMS_OUTPUT.PUT_LINE('-----');
    END LOOP;
```

```

DBMS_OUTPUT.PUT_LINE(i);
IF v_scolaritate(i) IS NOT NULL THEN
    DBMS_OUTPUT.PUT_LINE(v_scolaritate(i).an_inceput);
    DBMS_OUTPUT.PUT_LINE(v_scolaritate(i).an_final);
    DBMS_OUTPUT.PUT_LINE(v_scolaritate(i).institutie);
    DBMS_OUTPUT.PUT_LINE(v_scolaritate(i).specializare_sectie);
ELSE
    DBMS_OUTPUT.PUT_LINE('Aceasta componenta nu e initializata');
END IF;
END LOOP;
END;

```

La adăugarea unei componente în tablou, este necesară folosirea metodei `EXTEND` care îl mărește cu un element. Proprietățile și metodele `COUNT`, `FIRST`, `LAST`, `NEXT`, `PRIOR` etc. funcționează asemănător celorlalte tipuri de colecții. Dacă în bloc s-ar folosi un indice mai mare decât cel al ultimei componente (`LAST`), s-ar declanșa eroarea `ORA-06533: Subscript beyond count`. Pentru blocul de mai sus, rezultatul execuției sale în `SQL*Plus` este cel din figura 8.12.

```

SQL> @F:\ORACLE_CARTE\CAP08_PL_SQL1\LISTING08_23.SQL
51 /
Tabloul încapsulat are 5 componente
Indexul primeia este 1
Indexul ultimei este 5
-----
1
1990
1994
Liceul de Informatica Iasi
Informatica
-----
2
1994
1998
Univ. Al.I.Cuza Iasi
Facult. de Economie si Administrarea Afacerilor - spec. Informatica Economica
-----
3
Aceasta componenta nu e initializata
-----
4
2001
Univ. Al.I.Cuza Iasi
FEAA - ELITEC - Master Sisteme Informationale (MIS)
-----
5
Aceasta componenta nu e initializata

PL/SQL procedure successfully completed.

SQL>

```

Figura 8.12. Componentele tabloului încapsulat `V_SCOLARITATE`

În capitolul 12 vom discuta despre modul în care poate fi gestionat un tablou încapsulat într-o tabelă a bazei, inclusiv modul în care poate fi invocat în interogări `SQL`.

8.6.3. Vectori de mărime variabilă

Acest tip de colecție a apărut tot din versiunea 8 a Oracle și este cel mai apropiat de noțiunea de vectori din C sau Java. Deși similari vectorilor asociativi ca mod de accesare, vectorii cu mărime variabilă (*varray*) au o limită ce trebuie declarată, în timp ce tablourile asociative sunt nelimitate. Indicii sunt obligatoriu numere pozitive (mai mari ca zero), consecutive. Similar celorlalte colecții, într-un VARRAY un element poate fi scalar sau compozit.

La declararea unui vector cu mărime variabilă trebuie specificat atât tipul componentelor, cât și numărul maxim al acestora: `TYPE t_telefoane IS VARRAY(10) OF CHAR(10)` declară un tip de vectori de mărime variabilă cu maximum 10 componente. Inițializarea presupune folosirea unui constructor, iar numărul de argumente transmise constructorului devine mărimea inițială a vectorului – vezi listing 8.24.

Listing 8.24. Exemplu de utilizare a unui vector cu mărime variabilă

```
DECLARE
-- declarăm un tip VARRAY de obiecte SCOLARITATE
TYPE t_v_scolaritate IS VARRAY(8) OF scolaritate ;
v_scolaritate t_v_scolaritate := t_v_scolaritate();
BEGIN
-- rezervăm cinci componente
v_scolaritate.EXTEND(5);
-- initializăm trei
v_scolaritate(1) := scolaritate (1990, 1994, 'X1', 'Y1');
v_scolaritate(2) := scolaritate (1995, null, 'X2', 'Y2');
v_scolaritate(4) := scolaritate (NULL, NULL, NULL, NULL);

-- un prim bloc inclus pentru a prezenta o eroare
BEGIN
/* prima eroare - se inițializează o componentă pentru care nu s-a făcut "activarea" */
DBMS_OUTPUT.PUT_LINE('Incercam initializarea componentei 7');
v_scolaritate(7) := scolaritate (2005, NULL, 'TEST', NULL);
EXCEPTION
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('Ecce eroarea:');
DBMS_OUTPUT.PUT_LINE('Codul sau este '||SQLCODE||', iar mesajul '||SQLERRM);
END ;

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

-- al doilea bloc inclus pentru un alt tip de eroare
BEGIN
/*incercăm să extindem vectorul cu încă 8 componente, deși l-am declarat cu max. 10 */
DBMS_OUTPUT.PUT_LINE('Incercam sa marim vectorul cu 8 componente');
v_scolaritate.EXTEND(8);
EXCEPTION
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('Iata si a doua eroare:');
DBMS_OUTPUT.PUT_LINE('Codul sau este '||SQLCODE||', iar mesajul '||SQLERRM);
END ;
END;
```

Blocul de mai sus conține două blocuri incluse în vederea prezentării a două erori tipice în lucrul de vectori de mărime variabilă: declararea dimensiunii maxime a vectorului nu implică automat și declararea celor n componente, ci indică doar cât se poate “întinde” vectorul. După declarare urmează un soi de activare a componentelor, realizată fie prin folosirea constructorului la declarare, fie prin extinderea colecției (metoda `EXTEND`). Tentativa de a referi o componentă neactivată se soldează cu primul mesaj de eroare din figura 8.13.

```
SQL> @F:\ORACLE_CARTE\CAP08_PL_SQL1\LISTING08_24.SQL
42 /
Incercam initializarea componentei 7
Ecce eroarea:
Codul sau este -6533, iar mesajul ORA-06533: Subscript beyond count
-----
Incercam sa marim vectorul cu 8 componente
Iata si a doua eroare:
Codul sau este -6532, iar mesajul ORA-06532: Subscript outside of limit

PL/SQL procedure successfully completed.

SQL> |
```

Figura 8.13. Execuția în SQL*Plus a blocului din listing 8.24

A doua eroare ilustrată în figura 8.13 se declanșează atunci când se încearcă activarea unei componente cu indice peste limita maximă indicată la declararea vectorului. Pentru claritate, au fost folosite două funcții, `SQLCODE` și `SQLERRM` cu ajutorul cărora se afișează în secțiunea `EXCEPTION` codul și mesajul erorii declanșate.

Încercând o mică sistematizare, elementele comparative ale celor trei tipuri de colecții sunt puse în evidență de tabelul 8.1².

Tabel 8.1. Câteva similarități și diferențieri între tipurile de colecții sistem

Vectori asociativi	Tabele încapsulate	Vectori cu mărime variabilă
Introduși în Oracle 7.0 și ameliorați în 7.3	Introduși în Oracle 8	Introduși în Oracle 8
Nu pot fi stocați în tabelele bazei de date	Pot fi stocați în tabelele bazei de date	Pot fi stocați în tabelele bazei de date
Cheile pot fi pozitive sau negative	Cheile pot fi numai pozitive	Cheile pot fi numai pozitive
Nu prezintă dimensiune maximă explicită	Nu prezintă dimensiune maximă explicită	La declararea precizarea dimensiunii maxime este obligatorie
Pot fi mapați vectorilor-gazdă	Nu pot fi mapați vectorilor-gazdă	Nu pot fi mapați vectorilor-gazdă
Componentele pot fi nonsecvențiale (pot	Componentele pot fi nonsecvențiale (pot exista	Componentele sunt secvențiale (nu pot exista componente ne-

² Preluare din [Urman2002], p.318

exista componente ne-alocate)	componente ne-alocate)	alocate)
Referirea unei componente inexistente declașează eroarea NO_DATA_FOUND	Referirea unei componente inexistente declașează eroarea SUBSCRIPT_BEYOND_COUNT	Referirea unei componente inexistente declașează eroarea SUBSCRIPT_BEYOND_COUNT
Pot fi declarați numai în blocuri PL/SQL	Pot fi declarați în blocuri PL/SQL, dar și în afara blocurilor prin comanda CREATE TYPE	Pot fi declarați în blocuri PL/SQL, dar și în afara blocurilor prin comanda CREATE TYPE
Componentele pot fi accesate și modificate direct, fără inițializare	Trebuie inițializate și extinse înainte ca elementele (componentele) să primească valori	Trebuie inițializate și extinse înainte ca elementele să primească valori

Pe baza celor acumulate în actualul paragraf, ne propunem să rescriem programul de actualizare a tabelelor SPORURI și SALARII pe baza datelor din PONTAJE, pentru o lună dată. Noutatea constă în folosirea variabilelor de tip colecție pentru a mări viteza de execuție. Astfel, ținând seama că aproape toți angajații (liniile din PERSONAL) au cel puțin un pontaj pe lună, oricare ar fi ea (luna), pentru a nu interoga tabela PERSONAL la fiecare linie din cursorul C_ORE, creăm un tablou asociativ, `v_personal`, în care cheia (indexul) este chiar marca angajatului. Astfel, pentru angajatul din linia cursorului C_ORE, marca, data sporului de vechime, salariul orar și salariul pentru calculul concediului vor fi preluate din tablou: `v_personal (c_ore.marca).datasv`, `v_personal (c_ore.marca).salorar` și `v_personal (c_ore.marca).salorarco` - vezi listing 8.25.

Listing 8.25.Actualizare SPORURI și SALARII folosind colecții PL/SQL

```

/* Refacerea blocului din listing 8.18, cu folosirea unui tablou asociativ și a unui vector
cu mărime variabilă*/
DECLARE
  v_an salarii.an%TYPE := 2003 ;
  v_luna salarii.luna%TYPE := 1 ;

  -- C_ORE calculează totalul orelor lucrate, de concediu și de noapte pentru luna dată
  CURSOR c_ore IS
    SELECT marca, SUM(orelucrate) AS ore_l, SUM(oreco) AS ore_co, SUM(orenoapte)
      AS ore_n
    FROM pontaje
   WHERE TO_NUMBER(TO_CHAR(data,'YYYY')) = v_an AND
         TO_NUMBER(TO_CHAR(data,'MM')) = v_luna
   GROUP BY marca ;

  /* se declară un tip RECORD pentru extragerea informațiilor necesare calculului
  venitului de bază și sporurilor */
  TYPE t_personal IS RECORD (datasv personal.datasv%TYPE,
    salorar personal.salorar%TYPE, salorarco personal.salorarco%TYPE ) ;

  -- tipul și variabila vector asociativ
  TYPE t_personal IS TABLE OF t_personal INDEX BY BINARY_INTEGER ;
  v_personal t_personal ;

```

```

-- tipul și variabila vector cu mărime variabilă
TYPE t_sporv IS VARRAY(6) OF transe_sv%ROWTYPE ; --sunt 6 tranșe de vechime
-- (în TRANSE_SV)

v_sporv t_sporv := t_sporv();

-- variabile necesare calculului sporului de vechime
v_ani_vechime NUMBER(4,2) ;
v_proc_sv transe_sv.procent_sv%TYPE ;
v_spvech sporuri.spvech%TYPE ;

-- variabile pentru venitul de bază, sporul de noapte și total sporuri
v_venitbaza salarii.venitbaza%TYPE ;
v_sпноapte sporuri.spnoapte%TYPE ;
v_sporuri salarii.sporuri%TYPE ;

-- o variabilă contoar
i_vechime PLS_INTEGER := 1 ;

BEGIN

/* încă de la început inițializăm vectorul asociativ cu DATA_SV, SALORAR și SALORARCO
ale tuturor angajaților. Cheia tabloului e chiar Marca */
FOR rec_personal IN (SELECT * FROM personal) LOOP
    v_personal(rec_personal.marca).datasv := rec_personal.datasv ;
    v_personal(rec_personal.marca).salorar := rec_personal.salorar ;
    v_personal(rec_personal.marca).salorarco := rec_personal.salorarco ;
END LOOP ;

-- inițializăm vectorul cu mărime variabilă V_SPORV care conține tabela TRANSE_SV
FOR rec_transe_sv IN (SELECT * FROM transe_sv ORDER BY ani_limita_inf) LOOP
    v_sporv.EXTEND ;
    v_sporv(v_sporv.COUNT).ani_limita_inf := rec_transe_sv.ani_limita_inf ;
    v_sporv(v_sporv.COUNT).ani_limita_sup := rec_transe_sv.ani_limita_sup ;
    v_sporv(v_sporv.COUNT).procent_sv := rec_transe_sv.procent_sv ;
END LOOP ;

-- secvența iterativă principală
FOR rec_ore IN c_ore LOOP
    v_ani_vechime := MONTHS_BETWEEN( TO_DATE('01/'||v_luna||'/'||v_an, 'DD/MM/YYYY'),
        v_personal(rec_ore.marca).datasv) / 12 ;

    -- în loc de consultarea tabelii TRANSE_SV, procentul se va afla din VARRAY
    FOR i IN 1..v_sporv.COUNT LOOP
        IF v_ani_vechime >= v_sporv(i).ani_limita_inf AND
            v_ani_vechime < v_sporv(i).ani_limita_sup THEN
            -- componenta curentă a VARRAY-ului este care conține % corect
            i_vechime := i ;
            EXIT ;
        END IF ;
    END LOOP ;
    v_proc_sv := v_sporv(i_vechime).procent ;

/* se calculează venitul de bază, sporul de vechime și sporul de noapte; indexul vectorului
asociativ este chiar marca curentă din C_ORE*/
v_venitbaza := ROUND(rec_ore.ore_l * v_personal(rec_ore.marca).salorar +
    rec_ore.ore_co * v_personal(rec_ore.marca).salorarco,-2) ;
v_spvech := ROUND(v_venitbaza * v_proc_sv / 100, -3) ;
v_sпноapte := ROUND(rec_ore.ore_n * v_personal(rec_ore.marca).salorar * .15, -3) ;

```

```

-- se actualizează tabela SPORURI pentru angajatul curent
UPDATE sporuri
SET spvech = v_spvech, orenoapte = rec_ore.ore_n, spnoapte = v_spnoapte
WHERE marca=rec_ore.marca AND an=v_an AND luna=v_luna ;

/* dacă UPDATE nu a prelucrat nici o linie, se inserează o înregistrare în SPORURI */
IF SQL%NOTFOUND THEN
    INSERT INTO sporuri VALUES (rec_ore.marca, v_an, v_luna, v_spvech,
                                rec_ore.ore_n, v_spnoapte, 0) ;
END IF ;

-- se procedează analog pentru tabela SALARII
UPDATE salarii
SET orelucrate = rec_ore.ore_l, oreco = rec_ore.ore_co, venitbaza = v_venitbaza,
sporuri = (SELECT spvech + spnoapte + altesp
            FROM sporuri
            WHERE an=v_an AND luna=v_luna AND marca = rec_ore.marca)
WHERE marca=rec_ore.marca AND an=v_an AND luna=v_luna ;

IF SQL%NOTFOUND THEN
    INSERT INTO salarii VALUES (rec_ore.marca, v_an, v_luna, rec_ore.ore_l,
                                rec_ore.ore_co, v_venitbaza,
                                (SELECT spvech + spnoapte + altesp
                                 FROM sporuri
                                 WHERE an=v_an AND luna=v_luna AND marca = rec_ore.marca),
                                0, 0) ;
END IF ;
END LOOP ;
END ;

```

Cele șase tranșe pentru determinarea procentului de spor de vechime sunt stocate în vectorul cu mărime variabilă `v_sporv` care are șase componente. Fiecare componentă este compozită (tipul unei înregistrări din `TRANSE_SV` – `transe_sv%ROWTYPE`).

Secțiunea executabilă a blocului începe cu încărcarea celor două tablouri și continuă pe aceeași logică a scriptului din listing 8.18. Practic, câștigul de viteză ar trebui să provină din cele două `SELECT`-uri care au fost înocuite cu referințe la variabile de tip tablou. Soluția este rezonabilă atunci când tabela `PERSONAL` nu are un număr uriaș de înregistrări, premisă cât se poate de reală.

8.6.4. Clauzele **BULK COLLECT** și **FORALL**

Blocurile PL/SQL sunt executate, firește, de motorul PL/SQL aflat pe server. Interesant este că toate comezile SQL dintr-un bloc PL/SQL sunt transmise motorului SQL care întoarce rezultatele motorului PL/SQL. Prezentate grafic, lucrurile sunt mai clare –vezi figura 8.14.

Cei de la Oracle s-au gândit ca, atunci când comenzi de inserare/modificare/ștergere sunt executate pe baza înregistrărilor din colecții, în loc de `n` apeluri ale motorului SQL, să fie posibilă “înămănunchierea” tuturor apelurilor într-unul

singur. Lucrurile sunt valabile și la inițializarea colecțiilor pe baza unei consultări (SELECT). Astfel, timpul pierdut prin comutarea între motoarele PL/SQL și SQL (*context switches* – comutări de context) este mult diminuat. Mecanismul apare în Oracle 8i și se bazează pe două clauze: BULK BIND și FORALL.

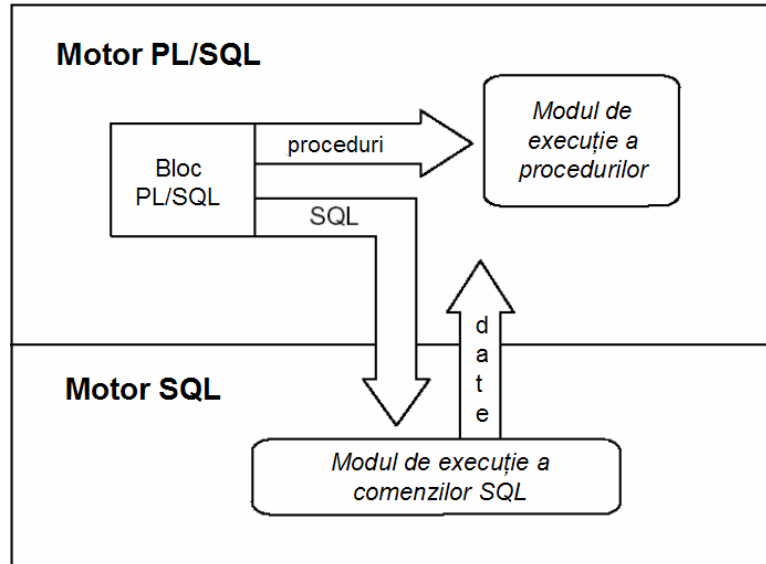


Figura 8.14. Schema de execuție a unui bloc PL/SQL ce conține comenzi SQL

Prima exemplificare reia programul de modificare cu 10% a salariului orar al celor mai vechi trei angajați din fiecare compartiment. Blocul PL/SQL din listing 8.26 nu mai actualizează imediat tabela PERSONAL, pe baza mărcii din cursorul C_ORE, ci stochează într-un VARRAY toate mărcile “meritoșilor” iar, în final, include comanda UPDATE într-o comandă FORALL.

Listing 8.26. Primul exemplu de folosire FORALL

```

/* ... același enunț ca în Listing 8.16 și 8.17. ...; renunțăm la excepție. Schimbăm logica
programului: se folosește un vector cu mărime variabilă pentru a stoca mărcile angajaților
pentru care se operează modificarea salariului*/
DECLARE
    -- declararea tipului VARRAY și a vectorului propriu-zis
    TYPE t_marca IS VARRAY(1000) OF personal.marca%TYPE ;
    v_marca t_marca := t_marca() ;

    CURSOR c_salariati (v_compart personal.compart%TYPE) IS
        SELECT marca, salorar, TRUNC(MONTHS_BETWEEN(SYSDATE, datasv)/12, 0)
           AS ani_vechime
        FROM personal WHERE compart = v_compart ORDER BY 3 DESC ;

    v_anî INTEGER ; /* anii de vechime ai celui mai recent angajat cu mărire de salariu */
BEGIN
    FOR rec_compart IN (SELECT DISTINCT compart FROM personal ) LOOP
        v_anî := 99 ;
        FORALL rec_salariati IN c_salariati (rec_compart.compart) LOOP

```

```

        IF c_salariati%ROWCOUNT > 3 AND rec_salariati.ani_vechime < v_ani THEN
            NULL ;
        ELSE
            /* prezentul angajat beneficiază de sporul de salariu, așa că marca
               sa va fi stocată în următoarea componentă a vectorului */
            v_marca.EXTEND ;
            v_marca(v_marca.COUNT) := rec_salariati.marca ;
            v_ani := rec_salariati.ani_vechime ;
        END IF ;
    END LOOP ;
END LOOP ;

/* Acesta este modul "clasic" de parcurgere a vectorului și actualizare a tabeli PERSONAL
FOR i IN 1..v_marca.COUNT LOOP
    UPDATE personal SET salorar = salorar + salorar * .1 WHERE marca = v_marca (i) ;
END LOOP ; */

-- noi vom folosi o delicată - "BULK BIND"
FORALL i IN 1..v_marca.COUNT
    UPDATE personal SET salorar = salorar + salorar * .1 WHERE marca = v_marca (i) ;
    COMMIT ;
END ;

```

Pentru comparație, este comentată și varianta "clasică" de parcurgere secvențială a tabloului. Prin recursul la `FORALL` în loc de `v_marca.COUNT` de apeluri ale motorului SQL, blocul PL/SQL va efectua doar unul.

Continuăm prin a rescrie blocul de populare a tabeli pontaje pentru un an și lună date (cel din listing 8.8) - vezi listing 8.27. Primul dintre tablouri - `v_marca` - conține toate mărcile angajaților (extrase din tabela `PERSONAL`). Pentru optimizare, comanda `SELECT marca BULK COLLECT INTO v_marca FROM PERSONAL` asigură citirea tabeli și inițializarea tabloului "dintr-o singură mișcare. Al doilea și al treilea tablou vor conține câte o componentă pentru fiecare zi lucrătoare și angajat. Astfel, în bucla care se repetă de la 1 la ultima zi a luni, se testează dacă ziua curentă este lucrătoare și, dacă da, atunci se parcurge o a doua secvență repetitivă, `FOR j IN 1..v_marca.COUNT LOOP`, prin care, dată fiind ziua curentă lucrătoare (zi), în `v_pont_marca (k)` și `v_pont_data (k)` se vor stoca toate mărcile și data curentă.

Listing 8.27. Popularea tabeli PONTAJE cu folosirea `BULK COLLECT` și `FORALL`

```

-- populare tabeli PONTAJE - folosirea colecțiilor și clauzelor BULK COLLECT și FORALL
DECLARE
    an salarii.an%TYPE := 2003;
    luna salarii.luna%TYPE := 3 ;
    zi DATE ; -- variabila folosită la ciclare
    k PLS_INTEGER := 1 ; -- altă variabilă necesară lucrului cu vectorii

    TYPE t_marca IS TABLE OF personal.marca%TYPE INDEX BY PLS_INTEGER ;
    v_marca t_marca ;

    TYPE t_pont_marca IS TABLE OF pontaje.marca%TYPE INDEX BY PLS_INTEGER ;
    v_pont_marca t_pont_marca ;

```

```

TYPE t_pont_data IS TABLE OF pontaje.data%TYPE INDEX BY PLS_INTEGER ;
v_pont_data t_pont_data ;

BEGIN
/* Se inițializează vectorul v_marca folosindu-se clauza BULK COLLECT */
SELECT marca BULK COLLECT INTO v_marca FROM PERSONAL ;

/* tablourile V_PONT_MARCA și V_PONT_DATA conțin elemente
pentru toate zilele lucrătoare și angajații */
FOR i IN 1..TO_NUMBER(TO_CHAR(LAST_DAY(TO_DATE('01/' || luna || '/' || an,
'DD/MM/YYYY')), 'DD')) LOOP
    zi := TO_DATE( TO_CHAR(i,'99') || '/' || luna || '/' || an, 'DD/MM/YYYY') ;
    IF RTRIM(TO_CHAR(zi, 'DAY')) IN ('SATURDAY', 'SUNDAY') THEN
        -- e zi nelucrătoare (sâmbătă sau duminică)
        NULL ;
    ELSE
        FOR j IN 1..v_marca.COUNT LOOP
            v_pont_marca(k) := v_marca(j) ;
            v_pont_data(k) := zi ;
            k := k + 1 ;
        END LOOP ;
    END IF ;
END LOOP ;
/* inserare cu folosirea comenzii FORALL. Se folosește un bloc inclus pentru
tratarea eventualei erori de violare a cheii primare */
BEGIN
    FORALL i IN 1..k-1
        INSERT INTO pontaje (marca, data) VALUES (v_pont_marca(i), v_pont_data(i)) ;
EXCEPTION -- se preia eventuala violare a cheii primare
    WHEN DUP_VAL_ON_INDEX THEN
        -- se șterg mai întâi înregistrările pentru ziua curentă
        DELETE FROM pontaje WHERE TO_NUMBER(TO_CHAR(data, 'YYYY')) = an
            AND TO_NUMBER(TO_CHAR(data, 'MM')) = luna ;
        COMMIT ;
        -- apoi se reinserează înregistrările
        DBMS_OUTPUT.PUT_LINE ('Se reinsereaza ');
        FORALL i IN 1..k-1
            INSERT INTO pontaje (marca, data) VALUES (v_pont_marca(i), v_pont_data(i)) ;
END ; -- aici se termină blocul inclus
COMMIT ;
END ;

```

După “umplerea” celor două tablouri se încearcă inserarea printr-un apel singur la motorul SQL a tuturor elementelor, drept pentru care se apelează la o comandă `FORALL`. Pentru a preîntâmpina eșecul datorat violării cheii primare, structura `FORALL` și comanda `INSERT` fac parte dintr-un bloc inclus ce prezintă o secțiune de tratare a excepției.

Ar mai fi de adăugat că, într-o primă versiune a blocului, am încercat să definim un singur tabloul asociativ cu elemente de tip `RECORD` în care să fie incluse atributele `marca` și `data`. `FORALL`-ul, însă, ne-a refuzat politicos, așa că am recurs la două tablouri cu elemente “scalare”...