Capitolul 12. Tipuri de date definite de utilizator

Oracle este un sistem de gestiune a bazelor de date obiect-relațional (SGBDOR) ([16]) adică utilizatorii pot defini tipuri de date suplimentare, specificând structura și felul de manipulare al datelor, și pot folosi aceste tipuri în modelul relațional. Aceste tipuri de date definite de utilizatori permit stocarea și manipularea datelor complexe precum imagini, secvențe audio și video, date medicale etc. Un mod eficient de a proiecta aplicații este definirea de obiecte și lucrul în mod obiect-orientat.
Tipurile de date definite de utilizator în Oracle pot fi:

- Obiecte
- Colecții de date, precum: tablouri asociative, vectori varray sau tabele imbricate

Comanda pentru definirea unui tip de date este CREATE TYPE.
Înainte de a defini tipuri de date complexe, prezentăm modul de declarare a două categorii de tipuri de date mai simple, subtipurile și tipul înregistrare (record).

12.1 Subtipuri de date

Subtipurile de date nu definesc tipuri noi de date, ci impun constrângeri asupra tipurilor existente.
Exemplu:

```
SET SERVEROUTPUT ON;
DECLARE
  SUBTYPE  nr_intreg  IS NUMBER(3);
  nr   nr_intreg;
BEGIN
  nr := 20;
  dbms_output.put_line ('nr = '||nr);
END;
/
```

12.2 Tipul înregistrare (record)

O înregistrare (record) este un tip de date compus ([17]) care permite stocarea de valori aparținând unor tipuri de date diferite. Este asemănător structurilor din C, C++ sau Java. Este foarte util pentru a înregistra date din tabele. Un caz particular care funcționează automat este specificatorul %ROWTYPE folosit în capitolele anterioare.
Ca exemplu vom construi un tip înregistrare care poate stoca datele dintr-o linie a tabelei Agenda definite în capitolul 2.

```
SET SERVEROUTPUT ON;
DECLARE
  TYPE  agenda_r  IS RECORD
  (
    id  NUMBER(3) NOT NULL := 8,
    nume        agenda.nume%TYPE,
    prenume   agenda.prenume%TYPE,
    telefon      agenda.nr_tel%TYPE
  );
  v_agenda agenda_r;
BEGIN
  v_agenda.nume := 'Marinescu';
END;
/
```

Exercițiu: Modificați programul de mai sus pentru a aloca toate câmpurile variabilei înregistrare ‚v_agenda' și afișați aceste câmpuri.

În PL/SQL putem folosi și o comandă UPDATE modificată, precum în următorul exemplu. Presupunem că avem o tabelă Angajati(idangajat, nume, prenume,salariu,...). Programul următor arată utilizarea clauzei RETURNING ... INTO care permite încărcarea variabilei de tip înregistrare ([18]).

```
SET SERVEROUTPUT ON;
DECLARE
  TYPE AngRec IS RECORD (vnume    angajati.nume%TYPE,
                                  vsalariu  angajati.salariu%TYPE);
  ang_info AngRec;
  ang_id   NUMBER := 1;
BEGIN
  UPDATE Angajati SET salariu = salariu * 1.1 WHERE idangajat = ang_id
        RETURNING nume, salariu  INTO  ang_info;
  DBMS_OUTPUT.PUT_LINE('Am marit salariul ang. ' || ang_info.vnume ||
    ', care acum castiga ' || ang_info.vsalariu);
END;
/
```

12.3 Colecții de date

O colecție (collection) este un grup de elemente ordonat, toate elementele fiind de același tip de date ([19]). Se aseamănă vectorilor din alte limbaje de programare, și sunt uni-dimensionale, indexul fiind de tip numeric sau șir de caractere.
Există trei tipuri „colecție" în Oracle, și anume:
- tablouri asociative (associative arrays)

- vectori (varrays)
- tabele imbricate (nested tables)

Există mai multe metode pentru lucrul cu colecțiile de date, precum ar fi:
- COUNT – returnează numărul de elemente
- DELETE – șterge toate elementele, sau DELETE(n) șterge elementul n dintr-un tablou asociativ sau tabelă imbricată
- EXISTS – cu sintaxa EXISTS(n) returnează true dacă elementul există
- EXTEND – se pot adăuga unul sau mai multe elemente, cu excepția tablourilor asociative
- FIRST, LAST – precum sugerează denumirea, primul sau ultimul element
- LIMIT – numărul maxim de elemente
- NEXT, PRIOR – elementul următor, sau anterior. Există și sintaxa NEXT(n) sau PRIOR(n) pentru a returna elementul care urmează celui cu indicele n, respectiv anterior acestuia
- TRIM – șterge ultimul element, cu excepția tablourilor asociative.

12.3.1 Tablouri asociative (associative array)

Un vector asociativ sau tablou asociativ (associative array), numit și „index-by-table" ([20]) este o mulțime de perechi de tipul cheie-valoare. Fiecare cheie este unică, și se folosește pentru a găsi valoarea stocată în vector. Cheile pot fi numerice sau de tip șir de caractere.

Următorul bloc PL/SQL conține un exemplu de vector asociativ [20]. Mai întâi este declarat tipul vectorului, apoi se declară o variabilă de acest tip. În secțiunea executabilă se alocă trei elemente vectorului, ulterior se modifică unul din elemente, iar apoi urmează afișarea vectorului într-o buclă de tip while.

```
SET SERVEROUTPUT ON;
DECLARE
   TYPE nr_locuitori IS TABLE OF NUMBER  -- Tipul vectorului
     INDEX BY VARCHAR2(64);

   locuitori_oras  nr_locuitori;        -- O variabila
   i  VARCHAR2(64);

 BEGIN
   -- Adaugam elemente noi la vector:

   locuitori_oras('Arad') := 234000;
   locuitori_oras('Cluj') := 461000;
   locuitori_oras('Iasi') := 361000;
```

```
  -- Modificam pentru un oras:

  locuitori_oras('Arad') := 234001;

  -- Urmeaza afisarea vectorului:

  i := locuitori_oras.FIRST;

  WHILE i IS NOT NULL LOOP
   DBMS_Output.PUT_LINE
    ('Nr. de locuitori ai orasului ' || i || ' este ' || TO_CHAR(locuitori_oras(i)));
   i := locuitori_oras.NEXT(i);
  END LOOP;
 END;
/
```

Vectorii asociativi sunt niște structuri de date cu caracter temporar ([20]); dacă dorim ca vectorul să persiste pe toată durata sesiunii de lucru cu baza de date, acesta trebuie declarat în secțiunea de specificații a unui pachet PL/SQL, și inițializat în corpul pachetului respectiv.

12.3.2 Tipul VARRAY

Vectorii (varray) în Oracle sunt structuri similare vectorilor din C, Java etc. ([21]). Se declară ca un tip de date (cu TYPE) și se specifică o dimensiune (număr de elemente) maximă la declarare. Primul indice începe de la 1. Elementele sunt de același tip de date și nu se permit spații între elemente, nici ștergerea unui element din interiorul vectorului. Se poate șterge ultimul element prin metoda TRIM. Vectorul se inițializează de obicei cu constructorul tipului de date, ca în exemplul următor.

```
SET SERVEROUTPUT ON;
DECLARE
 TYPE V_ORASE IS VARRAY(7) OF VARCHAR2(30);  -- tipul varray

 -- inițializarea unei variabile de tip v_orase:
 orase V_ORASE := V_ORASE('Cluj', 'Arad', 'Iasi', 'Bacau');

 PROCEDURE afiseaza  IS
 BEGIN
   DBMS_OUTPUT.PUT_LINE('Orasele sunt:');

   FOR i IN 1..orase.COUNT LOOP
     DBMS_OUTPUT.PUT_LINE(i || '.' || orase(i));
   END LOOP;
```

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

BEGIN
   afiseaza;
   -- schimb un oras
   orase(2) := 'Oradea';  -- Schimb valoarea elementului nr. 2
   afiseaza;
   -- extindem vectorul daca dorim adaugarea de elemente:
   orase.EXTEND;
   -- apoi putem adauga un element
   orase(5) := 'Buzau';
   afiseaza;
   -- daca dorim stergerea ultimului element
   orase.TRIM;
   afiseaza;
END;
/
```

Observați în exemplul anterior folosirea metodei COUNT pentru a determina numărul de elemente al vectorului, precum și metoda EXTEND pentru a adăuga un nou element.
Dacă se dorește definirea unui tip de date varray vizibil global (nu doar într-un bloc PL-SQL) se poate folosi comanda CREATE OR REPLACE TYPE.

Folosirea vectorilor în interogări în PL/SQL

Reluăm interogarea de la capitolul 5 pentru a exemplifica o utilizare a comenzii SELECT în PL/SQL care permite preluarea mai multor linii în variabile de tip VARRAY. Pentru aceasta înainte de clauza INTO trebuie scrisă și specificarea „BULK COLLECT" care va permite preluarea mai multor linii. În exemplul următor am înlocuit variabilele simple nume și prenume de la capitolul 5 cu 2 vectori v_nume și v_prenume, afișarea lor fiind efectuată într-o buclă FOR.

```
SET SERVEROUTPUT ON;
DECLARE
    TYPE  tip_v_nume  IS VARRAY(107)  OF VARCHAR2(30);
    TYPE  tip_v_prenume  IS VARRAY(107)  OF VARCHAR2(30);
    v_nume    tip_v_nume;
    v_prenume   tip_v_prenume;
BEGIN
  SELECT first_name, last_name  BULK COLLECT   INTO  v_prenume,  v_nume
          FROM Employees
          WHERE Employee_ID > 200;
```

```
   FOR i in  1 .. v_nume.COUNT   LOOP
      DBMS_OUTPUT.PUT_LINE('Nume si prenume: ' || v_nume(i) || '  ' ||
                                          v_prenume(i));
   END LOOP;
END;
/
```

12.3.3 Tabele imbricate

Tabelele imbricate (nested tables) reprezintă un set de valori ([18]). Pot fi considerate un vector mono-dimensional de elemente de același tip de date. Se pot defini și tabele imbricate multi-dimensionale dacă se definesc tabele de tabele.
Numărul de elemente al unei tabele imbricate nu este limitat, și pot să existe elemente lipsă, după cum se poate observa în figura următoare, care prezintă comparativ un vector de tip varray și o tabelă imbricată echivalentă.

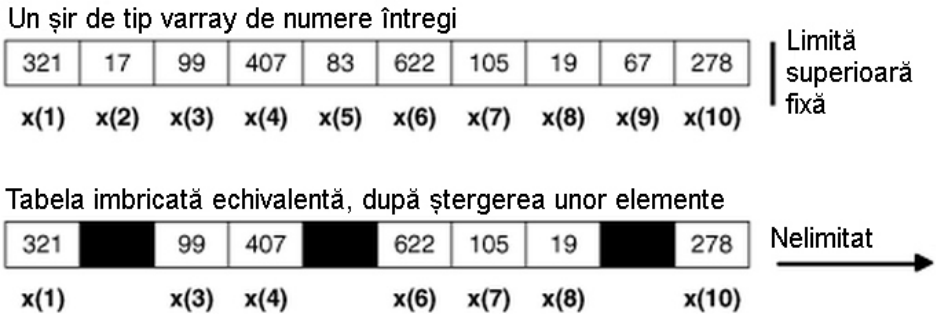

Figura 12.1 Comparație între varray și nested table ([18])

În programul următor se poate observa declararea unei tabele imbricate cu elemente de tip șir de caractere. Întâi se declară tipul de date, apoi o variabilă care este inițializată în aceeași linie cu declararea. Observați de asemenea alocarea a 7 elemente pentru vector, iar apoi afișarea elementelor cu o buclă for.

```
SET SERVEROUTPUT ON;
DECLARE
  TYPE  tip_zile  IS TABLE OF  Varchar2(15);
  var_zile  tip_zile  := tip_zile();   --initializare cu metoda constructor
  i Number;
BEGIN
 var_zile.EXTEND(7);  -- aloc 7 elemente
 var_zile(1) := 'Luni';
 var_zile(2) := 'Marti';
 var_zile(3) := 'Miercuri';
 var_zile(4) := 'Joi';
 var_zile(5) := 'Vineri';
 var_zile(6) := 'Sambata';
```

```
  var_zile(7) := 'Duminica';
  FOR i IN 1 .. var_zile.COUNT LOOP
      DBMS_OUTPUT.PUT_LINE(var_zile(i));
  END LOOP;
END;
/
```

După cum s-a precizat mai sus, tabelele imbricate permit și elemente lipsă. Astfel, în următorul exemplu ștergem un element dintr-un tabel imbricat, iar apoi afișăm tabelul. Observați utilizarea metodelor FIRST și NEXT în acest caz.

```
SET SERVEROUTPUT ON;
DECLARE
   TYPE tip_numere IS TABLE OF NUMBER;
   v_numere   tip_numere :=  tip_numere(1,2,3,4,5);
   i NUMBER;
BEGIN
   v_numere.DELETE(2); -- sterg elementul al doilea
   i := v_numere.FIRST;
   WHILE i IS NOT NULL LOOP
      DBMS_OUTPUT.PUT_LINE('v_numere(' || i || ') = ' || v_numere(i));
      i := v_numere.NEXT(i);
   END LOOP;
END;
/
```

Tabelele imbricate, ca și vectorii de tip varray, pot fi declarați și la nivelul bazei de date (nu doar în interiorul unui bloc PL/SQL), prin comanda CREATE TYPE. Astfel datele devin persistente. Un exemplu de astfel de declarare a unei tabele imbricate este prezentat în următorul cod SQL. Observați utilizarea comenzilor SQL clasice în acest caz (SELECT, INSERT etc.). Specificatorul STORE AS determină denumirea tabelei auxiliare în care va fi stocată tabela imbricată.

```
CREATE OR REPLACE TYPE tip_imbricat AS TABLE OF VARCHAR2(30);
/
CREATE TABLE tabela_n (id NUMBER, col1 tip_imbricat)
      NESTED TABLE col1 STORE AS denumire1;

INSERT INTO tabela_n VALUES (1, tip_imbricat('A'));
INSERT INTO tabela_n VALUES (2, tip_imbricat('B', 'C'));
INSERT INTO tabela_n VALUES (3, tip_imbricat('D', 'E', 'F'));
COMMIT;
SELECT * FROM tabela_n;
```