

Tipuri de date compuse

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

I. Înregistrări (RECORD)

- Declarația tipului *RECORD* se face conform următoarei sintaxe:

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

- Oracle9i introduce câteva facilități legate de acest tip de date.
 - Se poate insera (*INSERT*) o linie într-un tabel utilizând tipul *RECORD*.
 - Se poate actualiza (*UPDATE*) o linie într-un tabel utilizând tipul *RECORD* (cu sintaxa *SET ROW*)
 - se poate regăsi și returna sau șterge informația din clauza *RETURNING* a comenzilor *UPDATE* sau *DELETE*.
 - dacă în comenzile *UPDATE* sau *DELETE* se modifică mai multe linii, atunci pot fi utilizate în sintaxa *BULK COLLECT INTO*, colecții de înregistrări.

Exerciții:

1. Să se șteargă angajatul având codul 200 din tabelul *EMP_PNU*. Să se rețină într-o variabilă de tip *RECORD* codul, numele, salariul și departamentul acestui angajat (clauza *RETURNING*) . Să se afișeze înregistrarea respectivă. Rollback.

```
DECLARE
    TYPE info_ang_pnu IS RECORD (
        cod_ang NUMBER(4),
        nume     VARCHAR2(20),
        salariu  NUMBER(8),
        cod_dep  NUMBER(4));
    v_info_ang info_ang_pnu;
BEGIN
    DELETE FROM emp_pnu
        WHERE employee_id = 200
        RETURNING employee_id, last_name, salary, department_id
        INTO v_info_ang;
    DBMS_OUTPUT.PUT_LINE('A fost stearsa linia continand valorile ' ||
        v_info_ang.cod_ang || ' ' || v_info_ang.nume || ' ' || v_info_ang.salariu || ' ' ||
        v_info_ang.cod_dep);
END;
/
ROLLBACK;
```

II. Colecții

Colecțiile permit să fie prelucrate simultan mai multe variabile de același tip. Fiecare element are un indice unic, care determină poziția sa în colecție.

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

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

Obs :

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

- Singura diferență sintactică între tablourile indexate și cele imbricate este clauza *INDEX BY*. Dacă această clauză lipsește, atunci tipul este tablou imbricat.

➤ Atribute și metode ale unei colecții: (informații complete – în curs !)

Atribut sau metodă	Descriere
COUNT	numărul componentelor colecției
FIRST	Indicele primului element din tablou
LAST	Indicele ultimului element din tablou
EXISTS	întoarce TRUE dacă există în tablou componenta cu indexul specificat
NEXT	returnează indicele următoarei componente
PRIOR	returnează indicele componentei anterioare
DELETE	șterge una sau mai multe componente.
EXTEND	Adaugă elemente la sfârșit
LIMIT	Numărul maxim de elemente al unei colecții (pentru vectori), null pentru tablouri imbricate
TRIM	șterge elementele de la sfârșitul unei colecții

Ultimele 3 metode nu sunt valide pentru *index-by tables*.

➤ **bulk bind** permite ca toate liniile unei colecții să fie transferate simultan printr-o singură operație.

- este realizat cu ajutorul comenzii *FORALL*, ce poate fi folosită cu orice tip de colecție:

FORALL index *IN* lim_inf..lim_sup
comanda_sql;

Cursorul *SQL* are un atribut compus *%BULK_ROWCOUNT* care numără liniile afectate de iterațiile comenzii *FORALL*. *%BULK_ROWCOUNT(i)* reprezintă numărul de linii procesate de a *i*-a execuție a comenzii *SQL*.

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

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

➤ Clauza poate să apară în:

- comenzile *SELECT INTO* (cursoare implicite),

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

II.1. Tablouri indexate (index-by tables)

➤ Tabloul indexat *PL/SQL* are două componente:

- coloană ce cuprinde cheia primară pentru acces la liniile tabloului
- o coloană care include valoarea efectivă a elementelor tabloului.

➤ Declararea tipului *TABLE* se face respectând următoarea sintaxă:

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

Observații:

- Elementele unui tablou indexat nu sunt într-o ordine particulară și pot fi inserate cu chei arbitrare.
 - Deoarece nu există constrângeri de dimensiune, dimensiunea tabloului se modifică dinamic.
 - Tabloul indexat *PL/SQL* nu poate fi inițializat în declararea sa.
 - Un tablou indexat neinițializat este vid (nu conține nici valori, nici chei).
 - Un element al tabloului este nedefinit atâta timp cât nu are atribuită o valoare efectivă.
 - Dacă se face referire la o linie care nu există, atunci se produce excepția *NO_DATA_FOUND*.
- Pentru inserarea unor valori din tablourile *PL/SQL* într-o coloană a unui tabel de date se utilizează instrucțiunea *INSERT* în cadrul unei secvențe repetitive *LOOP*.
- Pentru regăsirea unor valori dintr-o coloană a unei baze de date într-un tablou *PL/SQL* se utilizează instrucțiunea *FETCH* (cursoare) sau instrucțiunea de atribuire în cadrul unei secvențe repetitive *LOOP*.
- Pentru a șterge liniile unui tablou fie se asignează elementelor tabloului valoarea *null*, fie se declară un alt tablou *PL/SQL* (de același tip) care nu este inițializat și acest tablou vid se asignează tabloului *PL/SQL* care trebuie șters. În *PL/SQL* 2.3 ștergerea liniilor unui tabel se poate face utilizând metoda *DELETE*.

Exerciții:

2. Să se definească un tablou indexat *PL/SQL* având elemente de tipul *NUMBER*. Să se introducă 20 de elemente în acest tablou. Să se afișeze, apoi să se șteargă tabloul utilizând diverse metode.

DECLARE

```
TYPE tablou_numar IS TABLE OF NUMBER
```

```
INDEX BY PLS_INTEGER;
```

```
v_tablou tablou_numar;
```

```
v_aux    tablou_numar; -- tablou folosit pentru stergere
```

BEGIN

```
FOR i IN 1..20 LOOP
```

```
    v_tablou(i) := i*i;
```

```
    DBMS_OUTPUT.PUT_LINE(v_tablou(i));
```

```
END LOOP;
```

```
--v_tablou := NULL;
```

```
--aceasta atribuire da eroarea PLS-00382
```

```
FOR i IN v_tablou.FIRST..v_tablou.LAST LOOP -- metoda 1 de stergere
```

```
    v_tablou(i) := NULL;
```

```
END LOOP;
```

```
--sau
v_tablou := v_aux; -- metoda 2 de stergere
--sau
v_tablou.delete; --metoda 3 de stergere
DBMS_OUTPUT.PUT_LINE('tabloul are ' || v_tablou.COUNT ||
    ' elemente');
END;
/
```

3. Să se definească un tablou de înregistrări având tipul celor din tabelul *dept_pnu*. Să se inițializeze un element al tabloului și să se introducă în tabelul *dept_pnu*. Să se șteargă elementele tabloului.

```
DECLARE
    TYPE dept_pnu_table_type IS TABLE OF dept_pnu%ROWTYPE
        INDEX BY BINARY_INTEGER;
    dept_table dept_pnu_table_type;
    i          NUMBER;
BEGIN
    IF dept_table.COUNT <> 0 THEN
        i := dept_table.LAST+1;
    ELSE i:=1;
    END IF;
    dept_table(i).department_id := 92;
    dept_table(i).department_name := 'NewDep';
    dept_table(i).location_id := 2700;
    INSERT INTO dept_pnu(department_id, department_name, location_id)
    VALUES (dept_table(i).department_id,
        dept_table(i).department_name,
        dept_table(i).location_id);
    -- sau folosind noua facilitate Oracle9i
    -- INSERT INTO dept_pnu
    -- VALUES dept_table(i);
    dept_table.DELETE; -- sterge toate elementele
    DBMS_OUTPUT.PUT_LINE('Dupa aplicarea metodei DELETE
        sunt '||TO_CHAR(dept_table.COUNT)||' elemente');
END;
```

II.2 Vectori (varray)

- Vectorii (varray) sunt structuri asemănătoare vectorilor din limbajele C sau Java.
- Vectorii au o dimensiune maximă (constantă) stabilită la declarare. În special, se utilizează pentru modelarea relațiilor *one-to-many*, atunci când numărul maxim de elemente din partea „many” este cunoscut și ordinea elementelor este importantă.
- Fiecare element are un index, a cărui limită inferioară este 1.

➤ Tipul de date vector este declarat utilizând sintaxa:

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

Exerciții:

4. Analizați și comentați exemplul următor.

```
DECLARE
    TYPE secventa IS VARRAY(5) OF VARCHAR2(10);
    v_sec secventa := secventa ('alb', 'negru', 'rosu',
        'verde');
BEGIN
    v_sec (3) := 'rosu';
```

```

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

```

Obs : Pentru a putea reține și utiliza tablourile imbricate și vectorii, trebuie să declarăm în SQL tipuri de date care să îi reprezinte.

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

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

5. a) Să se declare un tip *proiect_pnu* care poate reține maxim 50 de valori de tip VARCHAR2(15).
 b) Să se creeze un tabel *test_pnu* având o coloana *cod_ang* de tip NUMBER(4) și o coloană *proiecte_alocate* de tip *proiect_pnu*. Ce relație se modelează în acest fel?
 c) Să se creeze un bloc PL/SQL care declară o variabilă (un vector) de tip *proiect_pnu*, introduce valori în aceasta iar apoi valoarea vectorului respectiv este introdusă pe una din liniile tabelului *test_pnu*.

La promptul SQL :

```

CREATE TYPE proiect_pnu AS VARRAY(50) OF VARCHAR2(15)
/
CREATE TABLE test_pnu (cod_ang NUMBER(4),
                        proiecte_alocate proiect_pnu);

```

Blocul PL/SQL:

```

DECLARE
    v_proiect proiect_pnu := proiect_pnu(); --initializare utilizând constructorul
BEGIN
    v_proiect.extend (2);
    v_proiect(1) := 'proiect 1';
    v_proiect(2) := 'proiect 2';
    INSERT INTO test_pnu VALUES (1, v_proiect);
END;
/

```

6. Să se scrie un bloc care mărește salariile angajaților din departamentul 50 cu 10%, în cazul în care salariul este mai mic decât 5000. Se va utiliza un vector corespunzător codurilor angajaților. Se cer 3 soluții.

Soluția 1 :

```

DECLARE
    TYPE t_id IS VARRAY(100) OF emp_pnu.employee_id%TYPE ;
    v_id t_id := t_id();
BEGIN
    FOR contor IN (SELECT * FROM emp_pnu) LOOP
        IF contor. Department_id = 50 AND contor.salary < 5000 THEN
            v_id.extend;
            v_id(v_id.COUNT) := contor.employee_id;
        END IF;
    END LOOP;
    FOR contor IN 1..v_id.COUNT LOOP
        UPDATE emp_pnu
        SET salary = salary *1.1
        WHERE employee_id = v_id (contor);
    END LOOP;
END;

```

```

        END LOOP;
    END;
/

```

Soluția 2 (varianta FORALL):

```

DECLARE
    TYPE t_id IS VARRAY(100) OF emp_pnu.employee_id%TYPE ;
    v_id t_id := t_id();
BEGIN
    FOR contor IN (SELECT * FROM emp_pnu) LOOP
        IF contor. Department_id =50 AND contor.salary < 5000 THEN
            V_id.extend;
            V_id(v_id.COUNT) := contor.employee_id;
        END IF;
    END LOOP;
    FORALL contor IN 1..v_id.COUNT
        UPDATE emp_pnu
        SET salary = salary *1.1
        WHERE employee_id = v_id (contor);
END;
/

```

Obs: Prin comanda FORALL sunt trimise toate datele pe server, executându-se apoi o singură comandă SELECT, UPDATE etc.

Soluția 3 (varianta BULK COLLECT):

```

DECLARE
    TYPE t_id IS VARRAY(100) OF emp_pnu.employee_id%TYPE ;
    v_id t_id := t_id();
BEGIN
    SELECT employee_id
    BULK COLLECT INTO v_id
    FROM emp_pnu
    WHERE department_id =50 AND salary < 5000;
    FORALL contor IN 1..v_id.COUNT
        UPDATE emp_pnu
        SET salary = salary *1.1
        WHERE employee_id = v_id (contor);
END;
/

```

II.3 Tablouri imbricate

- Tablourile imbricate (*nested table*) sunt tablouri indexate a căror dimensiune nu este stabilită.
 - folosesc drept indici numere consecutive ;
 - sunt asemenea unor tabele cu o singură coloană;
 - nu au dimensiune limitată, ele cresc dinamic;
 - inițial, un tablou imbricat este dens (are elementele pe poziții consecutive) dar pot apărea spații goale prin ștergere ;
 - metoda NEXT ne permite să ajungem la următorul element ;
 - pentru a insera un element nou, tabloul trebuie extins cu metoda EXTEND(nr_comp) ;
 - Un tablou imbricat este o mulțime neordonată de elemente de același tip. Valorile de acest tip:
 - pot fi stocate în baza de date,
 - pot fi prelucrate direct în instrucțiuni SQL
 - au excepții predefinite proprii.
- Comanda de declarare a tipului de date tablou imbricat are sintaxa:

TYPE nume_tip **IS TABLE OF** tip_elemente [**NOT NULL**];

- Pentru adaugarea de linii intr-un tablou imbricat, acesta trebuie sa fie initializat cu ajutorul **constructorului**.
 - *PL/SQL* apelează un constructor numai în mod explicit.
 - Tabelele indexate nu au constructori.
 - Constructorul primește ca argumente o listă de valori numerotate în ordine, de la 1 la numărul de valori date ca parametrii constructorului.
 - Dimensiunea inițială a colecției este egală cu numărul de argumente date în constructor, când aceasta este inițializată.
 - Pentru vectori nu poate fi depășită dimensiunea maximă precizată la declarare.
 - Atunci când constructorul este fără argumente, va crea o colecție fără nici un element (vida), dar care are valoarea *not null*.

Exerciții:

7. Să se declare un tip tablou imbricat și o variabilă de acest tip. Inițializați variabila și afișați conținutul tabloului, de la primul la ultimul element și invers.

```
DECLARE
  TYPE CharTab IS TABLE OF CHAR(1);
  v_Characters CharTab :=
    CharTab('M', 'a', 'd', 'a', 'm', ',', ', ',
            'l', '"', 'm', ' ', 'A', 'd', 'a', 'm');
  v_Index INTEGER;
BEGIN
  v_Index := v_Characters.FIRST;
  WHILE v_Index <= v_Characters.LAST LOOP
    DBMS_OUTPUT.PUT(v_Characters(v_Index));
    v_Index := v_Characters.NEXT(v_Index);
  END LOOP;
  DBMS_OUTPUT.NEW_LINE;
  v_Index := v_Characters.LAST;
  WHILE v_Index >= v_Characters.FIRST LOOP
    DBMS_OUTPUT.PUT(v_Characters(v_Index));
    v_Index := v_Characters.PRIOR(v_Index);
  END LOOP;
  DBMS_OUTPUT.NEW_LINE;
END;
/
```

8. Creați un tip tablou imbricat, numit NumTab. Afișați conținutul acestuia, utilizând metoda EXISTS. Atribuiți valorile tabloului unui tablou index-by. Afișați și acest tablou, în ordine inversă.

```
DECLARE -- cod partial, nu sunt declarate tipurile!
  v_NestedTable NumTab := NumTab(-7, 14.3, 3.14159, NULL, 0);
  v_Count BINARY_INTEGER := 1;
  v_IndexByTable IndexByNumTab;
BEGIN
  LOOP
    IF v_NestedTable.EXISTS(v_Count) THEN
      DBMS_OUTPUT.PUT_LINE(
        'v_NestedTable(' || v_Count || '): ' ||
        v_NestedTable(v_Count));
      v_IndexByTable(v_count) := v_NestedTable(v_count);
      v_Count := v_Count + 1;
    ELSE
```

```

    EXIT;
  END IF;
END LOOP;
-- atribuire invalida
-- v_IndexByTable := v_NestedTable;
v_Count := v_IndexByTable.COUNT;
LOOP
  IF v_IndexByTable.EXISTS(v_Count) THEN
    DBMS_OUTPUT.PUT_LINE(
      'v_IndexByTable(' || v_Count || '): ' ||
      v_IndexByTable(v_Count));
    v_Count := v_Count - 1;
  ELSE
    EXIT;
  END IF;
END LOOP;
END;
END;
/

```

9. Analizați următorul exemplu, urmărind excepțiile semnificative care apar în cazul utilizării incorecte a colecțiilor:

```

DECLARE
  TYPE numar IS TABLE OF INTEGER;
  alfa numar;
BEGIN
  alfa(1) := 77;
  -- declanseaza exceptia COLLECTION_IS_NULL
  alfa := numar(15, 26, 37);
  alfa(1) := ASCII('X');
  alfa(2) := 10*alfa(1);
  alfa('P') := 77;
  /* declanseaza exceptia VALUE_ERROR deoarece indicele
    nu este convertibil la intreg */
  alfa(4) := 47;
  /* declanseaza exceptia SUBSCRIPT_BEYOND_COUNT deoarece
    indicele se refera la un element neinitializat */
  alfa(null) := 7; -- declanseaza exceptia VALUE_ERROR
  alfa(0) := 7; -- exceptia SUBSCRIPT_OUTSIDE_LIMIT
  alfa.DELETE(1);
  IF alfa(1) = 1 THEN ... -- exceptia NO_DATA_FOUND
  ...
END;
/

```

II.5 Prelucrarea colecțiilor

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

- operatorul *TABLE* permite prelucrarea elementelor unui tablou imbricat care este stocat într-un tabel. Operatorul permite interogarea unei colecții în clauza *FROM* (la fel ca un tabel).
- Pentru tablouri imbricate pe mai multe niveluri, operațiile *LMD* pot fi făcute atomic sau pe elemente individuale, iar pentru vectori pe mai multe niveluri, operațiile pot fi făcute numai atomic.
- Pentru prelucrarea unei colecții locale se poate folosi și operatorul *CAST*. *CAST* are forma sintactică:

CAST (*nume_colecție AS tip_colecție*)

Exerciții:

10. a) Să se creeze un tip *LIST_ANG_PNU*, de tip vector, cu maxim 10 componente de tip *NUMBER(4)*.

b) Să se creeze un tabel *JOB_EMP_PNU*, având coloanele: *cod_job* de tip *NUMBER(3)*, *titlu_job* de tip *VARCHAR2(25)* și *info* de tip *LIST_ANG_PNU*.

c) Să se creeze un bloc PL/SQL care declară și inițializează două variabile de tip *LIST_ANG_PNU*, o variabilă de tipul coloanei *info* din tabelul *JOB_EMP_PNU* și o variabilă de tipul codului job-ului. Să se insereze prin diverse metode 3 înregistrări în tabelul *JOB_EMP_PNU*.

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

```
/
```

```
CREATE TABLE job_emp_pnu (
    cod_job NUMBER(3),
    titlu_job VARCHAR2(25),
    info list_ang_pnu);
```

```
DECLARE
```

```
    v_list list_ang_pnu := list_ang_pnu (123, 124, 125);
    v_info_list list_ang_pnu := list_ang_pnu (700);
    v_info job_emp_pnu.info%TYPE;
    v_cod job_emp_pnu.cod_job%TYPE := 7;
    i INTEGER;
```

```
BEGIN
```

```
    INSERT INTO job_emp_pnu
    VALUES (5, 'Analist', list_ang_pnu (456, 457));
```

```
    INSERT INTO job_emp_pnu
    VALUES (7, 'Programator', v_list);
```

```
    INSERT INTO job_emp_pnu
    VALUES (10, 'Inginer', v_info_list);
```

```
    SELECT info
```

```
    INTO v_info
```

```
    FROM job_emp_pnu
```

```
    WHERE cod_job = v_cod;
```

```
    --afisare v_info
```

```
    DBMS_OUTPUT.PUT_LINE('v_info:');
```

```
    i := v_info.FIRST;
```

```
    while (i <= v_info.last) loop
```

```
        DBMS_OUTPUT.PUT_LINE(v_info(i));
```

```
        i := v_info.next(i);
```

```
    end loop;
```

```
END;
```

```
/
```

```
ROLLBACK;
```

11. Creați un tip de date tablou imbricat *DateTab_pnu* cu elemente de tip *DATE*. Creați un tabel *FAMOUS_DATES_PNU* având o coloană de acest tip. Declarați o variabilă de tip *DateTab_pnu* și adăugați-i 5 date calendaristice. Ștergeți al doilea element și apoi introduceți tabloul în tabelul *FAMOUS_DATES_PNU*. Selectați-l din tabel. Afișați la fiecare pas.

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

NESTED TABLE nume_câmp **STORE AS** nume_tabel;

În SQL*Plus:

```
DROP TABLE famous_dates_pnu;
DROP TYPE DateTab_pnu;
CREATE OR REPLACE TYPE DateTab_pnu AS
  TABLE OF DATE;
/
CREATE TABLE famous_dates_pnu (
  key      VARCHAR2(100) PRIMARY KEY,
  date_list DateTab_pnu)
NESTED TABLE date_list STORE AS dates_tab;
```

Blocul PL/SQL:

DECLARE

```
v_Dates DateTab_pnu := DateTab_pnu(TO_DATE('04-JUL-1776', 'DD-MON-YYYY'),
                                     TO_DATE('12-APR-1861', 'DD-MON-YYYY'),
                                     TO_DATE('05-JUN-1968', 'DD-MON-YYYY'),
                                     TO_DATE('26-JAN-1986', 'DD-MON-YYYY'),
                                     TO_DATE('01-JAN-2001', 'DD-MON-YYYY'));
```

-- Procedura locala pentru afisarea unui DateTab.

```
PROCEDURE Print(p_Dates IN DateTab_pnu) IS
  v_Index BINARY_INTEGER := p_Dates.FIRST;
BEGIN
  WHILE v_Index <= p_Dates.LAST LOOP
    DBMS_OUTPUT.PUT(' ' || v_Index || ': ');
    DBMS_OUTPUT.PUT_LINE(TO_CHAR(p_Dates(v_Index),
                                'DD-MON-YYYY'));
    v_Index := p_Dates.NEXT(v_Index);
  END LOOP;
END Print;
```

BEGIN

```
DBMS_OUTPUT.PUT_LINE('Valoarea initiala a tabloului');
Print(v_Dates);
INSERT INTO famous_dates_pnu (key, date_list)
  VALUES ('Date importante', v_Dates);
v_Dates.DELETE(2); -- tabloul va avea numai 4 elemente
SELECT date_list
  INTO v_Dates
  FROM famous_dates
  WHERE key = 'Date importante';
DBMS_OUTPUT.PUT_LINE('Tabloul dupa INSERT si SELECT:');
Print(v_Dates);
END;
```

/

ROLLBACK;