

[Tipuri de date compuse:](#)

[I. Înregistrări \(RECORD\)](#)

[II. Colecții](#)

[II.1 Tablouri indexate \(index-by tables\)](#)

[II.2 Vectori \(varrays\)](#)

[II.3 Tablouri imbricate \(nested tables\)](#)

[II.4 Colecții pe mai multe niveluri](#)

[II.5 Prelucrarea colecțiilor](#)

**Tipuri de date compuse:**

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

### I. Înregistrări (RECORD)

Declararea 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),
```

## Laborator 2 PL/SQL

```

        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;
```

2. a) Folosind tipul declarat mai sus, să se adauge o linie în tabelul EMP\_PNU prin intermediul unei variabile de tip înregistrare inițializate. Efectuați modificările necesare asupra tipului de date, astfel încât această inserare să fie posibilă. La inițializarea unei variabile de tip record, țineți cont de constrângerile NOT NULL definite asupra tabelului EMP\_PNU.

b) Modificați valoarea unei componente a variabilei definite anterior și actualizați conținutul liniei introduse în tabel.

```

ALTER TABLE emp_pnu
DROP COLUMN stea;

SET SERVEROUTPUT ON
DECLARE
    TYPE info_ang_pnu IS RECORD (
        cod_ang NUMBER(4):=500,
        nume VARCHAR2(20):='abc',
        prenume VARCHAR2(20):='john',
        email emp_pnu.email%TYPE:='abc@mail',
        telefon emp_pnu.phone_number%type,
        data emp_pnu.hire_date%TYPE:=SYSDATE,
        job emp_pnu.job_id%TYPE:='SA_REP',
        salariu NUMBER(8, 2):=1000,
        comision emp_pnu.commission_pct%TYPE,
        manager emp_pnu.manager_id%TYPE,
        cod_dep NUMBER(4):=30
    );
    v_info_ang info_ang_pnu;
BEGIN
    --inserare; nu ar fi fost posibila maparea unei variabile de tip RECORD
    într-o lista
    -- explicita de coloane
    INSERT INTO emp_pnu
    VALUES v_info_ang;
    DBMS_OUTPUT.PUT_LINE('A fost introdusa linia continand valorile ' ||
    v_info_ang.cod_ang || ' ' || v_info_ang.nume || ' ' || v_info_ang.salariu || ' '

```

## Laborator 2 PL/SQL

```
|| v_info_ang.cod_dep); --actualizare
    v_info_ang.num := 'smith';
    UPDATE emp_pnu
    SET ROW=v_info_ang
    WHERE employee_id = v_info_ang.cod_ang;
    DBMS_OUTPUT.PUT_LINE('A fost actualizata linia cu valorile ' ||
    v_info_ang.cod_ang || ' ' ||v_info_ang.num|| ' ' ||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

## Laborator 2 PL/SQL

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 folosit 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 trimis 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.

### Exerciții:

3. Analizați și comentați exemplul următor. Afișați valorile variabilelor definite.

```
DECLARE  
    TYPE tab_index IS TABLE OF NUMBER  
        INDEX BY BINARY_INTEGER;  
    TYPE tab_imbri IS TABLE OF NUMBER;  
    TYPE vector IS VARRAY(15) OF NUMBER;  
    v_tab_index tab_index;  
    v_tab_imbri tab_imbri;  
    v_vector vector;  
    i INTEGER;  
  
BEGIN  
    v_tab_index(1) := 72;  
    v_tab_index(2) := 23;  
    v_tab_imbri := tab_imbri(5, 3, 2, 8, 7);  
    v_vector := vector(1, 2);  
  
    i:=v_tab_index.FIRST;  
    DBMS_OUTPUT.PUT_LINE('v_tab_index: '||v_tab_index(i));  
    LOOP
```

## Laborator 2 PL/SQL

---

```
        i:= v_tab_index.NEXT(i);
        DBMS_OUTPUT.PUT_LINE('v_tab_index: ' || v_tab_index(i));
        EXIT WHEN (i >= v_tab_index.LAST);
    END LOOP;

    i:=v_tab_imbri.FIRST;
    WHILE (i <= v_tab_imbri.LAST) LOOP
        DBMS_OUTPUT.PUT_LINE('v_tab_imbri: ' || v_tab_imbri(i));
        i:= v_tab_imbri.NEXT(i);
    END LOOP;

    FOR i IN 1..v_vector.COUNT LOOP
        DBMS_OUTPUT.PUT_LINE('vector: ' || v_vector(i));
    END LOOP;
END;
/
```

### 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**.

## Laborator 2 PL/SQL

---

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:

4. 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;
    DBMS_OUTPUT.PUT_LINE('tabloul are ' || v_tablou.COUNT || ' elemente');
    --sau
    v_tablou := v_aux; -- metoda 2 de stergere
    DBMS_OUTPUT.PUT_LINE('tabloul are ' || v_tablou.COUNT || ' elemente');
    --sau
    v_tablou.delete; --metoda 3 de stergere
    DBMS_OUTPUT.PUT_LINE('tabloul are ' || v_tablou.COUNT || ' elemente');
END;
/
```

5. 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.

```
CREATE TABLE dept_pnu
AS SELECT *
FROM departments;

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);
DBMS_OUTPUT.PUT_LINE('tabloul are ' || dept_table.COUNT || ' elemente');
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 (varrays)

- Vectorii (varrays) 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:

6. 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;
/
```

## Laborator 2 PL/SQL

---

**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;

7. 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 coloană 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.

```
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;
/
SELECT *
FROM test_pnu;
```

8. 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
```



## Laborator 2 PL/SQL

---

```
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;
/
SELECT *
FROM emp_pnu
WHERE department_id = 50;
```

### 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 (nested tables)

- Tablourile imbricate (nested tables) sunt tablouri indexate a căror dimensiune nu este stabilită.
  - folosesc drept indici numere consecutive;
  - sunt asemenea unor tabele cu 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 într-un tablou imbricat, acesta trebuie să fie inițializat 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 parametri 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:

9. 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.

## Laborator 2 PL/SQL

```
DECLARE
    TYPE CharTab IS TABLE OF CHAR(1);
    v_Characters CharTab := CharTab('M', 'a', 'd', 'a', 'm', ',', ',', ' ', 'I',
    ',', '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;
/
```

10. 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
    TYPE NumTab IS TABLE OF NUMBER;
    TYPE IndexByNumTab IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
    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;
```

## Laborator 2 PL/SQL

---

```
END LOOP;
END;
/
```

11. Să se analizeze următorul bloc PL/SQL. Ce se obține în urma execuției acestuia ?

```
DECLARE
    TYPE alfa IS TABLE OF VARCHAR2(50);
    -- creeaza un tablou (atomic) null
    tab1 alfa ;
    /* creeaza un tablou cu un element care este null, dar
    tabloul nu este null, el este initializat, poate
    primi elemente */
    tab2 alfa := alfa();
BEGIN
    IF tab1 IS NULL THEN
        DBMS_OUTPUT.PUT_LINE('tab1 este NULL');
    ELSE
        DBMS_OUTPUT.PUT_LINE('tab1 este NOT NULL');
    END IF;
    IF tab2 IS NULL THEN
        DBMS_OUTPUT.PUT_LINE('tab2 este NULL');
    ELSE
        DBMS_OUTPUT.PUT_LINE('tab2 este NOT NULL');
    END IF;
END;
/
```

12. 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);
```

## Laborator 2 PL/SQL

---

```
IF alfa(1) = 1 THEN NULL;  
END IF; -- exceptia NO_DATA_FOUND ...  
END;  
/
```

### II.4 Colecții pe mai multe niveluri

!!! De analizat exemplele din curs!

### 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:

13. a) Să se creeze un tip LIST\_ANG\_PNU, de tip tablou imbricat, cu 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.

d) Să se utilizeze comenzi SQL pt. adaugare element în colecție, adăugare elemente obținute cu subcerere, modificare și ștergere element colecție.

Testați aceleași operații LMD pentru un vector în loc de tablou imbricat. Acestea nu pot fi efectuate pe elemente individuale din vector!

## Laborator 2 PL/SQL

---

**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;

```
DROP TABLE job_emp_pnu;
DROP TYPE list_ang_pnu;

CREATE OR REPLACE TYPE list_ang_pnu IS TABLE OF NUMBER(4);
/
CREATE TABLE job_emp_pnu (
            cod_job VARCHAR2(10),
            titlu_job VARCHAR2(25),
            info list_ang_pnu)
NESTED TABLE info STORE AS info_tab_pnu;

DECLARE
    v_list list_ang_pnu := list_ang_pnu (1002, 1003, 1004);
    v_info_list list_ang_pnu := list_ang_pnu (1005);
    v_info job_emp_pnu.info%TYPE;
    v_cod job_emp_pnu.cod_job%TYPE := 'IT_PROG';
    i INTEGER;
BEGIN
    INSERT INTO job_emp_pnu
    VALUES ('AD_PRES', 'Director', list_ang_pnu (1000, 1001));
    INSERT INTO job_emp_pnu
    VALUES ('IT_PROG', 'Programator', v_list);
    INSERT INTO job_emp_pnu
    VALUES ('SA_REP', 'Reprezentant vanzari', 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;
/

SELECT *
FROM job_emp_pnu;

SELECT *
FROM TABLE (
    SELECT info
    FROM job_emp_pnu
    WHERE cod_job = 'IT_PROG');
```

```
--adaugare element in colectie
INSERT INTO TABLE (
    SELECT info
    FROM job_emp_pnu
    WHERE cod_job = 'IT_PROG')
VALUES (0);

--adaugare elemente obtinute cu subcerere
INSERT INTO TABLE (
    SELECT info
    FROM job_emp_pnu
    WHERE cod_job = 'IT_PROG')
SELECT employee_id
FROM employees
WHERE job_id = 'IT_PROG';

--modificare element colectie
UPDATE TABLE (
    SELECT info
    FROM job_emp_pnu
    WHERE cod_job = 'IT_PROG') a
SET VALUE(a) = 1006
WHERE COLUMN_VALUE = 0;

--stergere element colectie
DELETE FROM TABLE (
    SELECT info
    FROM job_emp_pnu
    WHERE cod_job = 'IT_PROG') a
WHERE COLUMN_VALUE = 1006;
```

14. 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.

```
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;
DECLARE
    v_Dates DateTab_pnu := DateTab_pnu(
        TO_DATE('04-07-1776', 'DD-MM-YYYY'),
        TO_DATE('12-04-1861', 'DD-MM-YYYY'),
```

```
        TO_DATE('05-06-1968', 'DD-MM-YYYY'),
        TO_DATE('26-01-1986', 'DD-MM-YYYY'),
        TO_DATE('01-01-2001', 'DD-MM-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-MM-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
    DBMS_OUTPUT.PUT_LINE('Tabloul dupa DELETE(2):');
    Print(v_Dates);
    SELECT date_list
    INTO v_Dates
    FROM famous_dates_pnu
    WHERE key = 'Date importante';
    DBMS_OUTPUT.PUT_LINE('Tabloul dupa INSERT si SELECT:');
    Print(v_Dates);
END;
/
ROLLBACK;
```

15. Să se adauge o coloană info de tip tablou imbricat în tabelul DEPT\_PNU. Acest tablou are **două componente** în care pentru fiecare departament sunt depuse codul unui angajat și job-ul acestuia. Să se insereze o linie în tabelul imbricat. Să se listeze codurile departamentelor și colecția angajaților corespunzători (**tipul OBIECT**).

```
CREATE OR REPLACE TYPE rec_info_pnu
    IS OBJECT (cod_ang NUMBER(6),
              job VARCHAR2(35));
/

CREATE OR REPLACE TYPE dept_info_pnu
    IS TABLE OF rec_info_pnu;
/

ALTER TABLE dept_pnu
ADD (info dept_info_pnu)
    NESTED TABLE info STORE AS info_tab;

DESC dept_pnu
```



## Laborator 2 PL/SQL

---

```
UPDATE dept_pnu
SET info = dept_info_pnu()
WHERE department_id = 90;

INSERT INTO TABLE (SELECT info
                      FROM dept_pnu
                      WHERE department_id = 90)
VALUES (100, 'MyEmp');

SELECT d.department_id, t.*
FROM dept_pnu d, TABLE (d.info) t;

DECLARE
    TYPE tip_tablou_index IS TABLE OF
        departments.department_id%type INDEX BY PLS_INTEGER;
    v_tabindex_deps tip_tablou_index;
    v_tabnest dept_info_pnu;
BEGIN
    v_tabnest:=dept_info_pnu();
    SELECT department_id
    BULK COLLECT INTO v_tabindex_deps
    FROM dept_pnu;
    FOR i IN v_tabindex_deps.FIRST..v_tabindex_deps.LAST LOOP
        IF v_tabindex_deps.EXISTS(i) THEN
            DBMS_OUTPUT.PUT_LINE('Prelucram departamentul
'||v_tabindex_deps(i));
            SELECT rec_info_pnu(e.employee_id,j.job_title)
            BULK COLLECT INTO v_tabnest
            FROM employees e, jobs j
            WHERE e.department_id = v_tabindex_deps(i)
            AND e.job_id = j.job_id;
            UPDATE dept_pnu
            SET info = v_tabnest
            WHERE department_id = v_tabindex_deps(i);
        END IF;
    END LOOP;
    COMMIT;
END;
/
SELECT d.department_id, t.*
FROM dept_pnu d, TABLE (d.info) t;
```

16. Să se creeze un tabel temporar TEMP\_TABLE\_PNU cu datele persistente la nivel de sesiune, având o coloană de tip numeric și alta de tip șir de caractere. Prin intermediul unui tablou indexat, să se adauge 500 de linii în acest tabel. Se cer două variante. Comentați diferențele între variantele propuse.

```
CREATE GLOBAL TEMPORARY TABLE temp_table_pnu
(num_col NUMBER(6),
```

## Laborator 2 PL/SQL

```
        char_col VARCHAR2(20))
ON COMMIT PRESERVE ROWS;

DECLARE
    TYPE t_Numbers IS TABLE OF temp_table_pnu.num_col%TYPE
        INDEX BY BINARY_INTEGER;
    TYPE t_Chars IS TABLE OF temp_table_pnu.char_col%TYPE
        INDEX BY BINARY_INTEGER;
    v_Numbers t_Numbers;
    v_Chars t_Chars;
BEGIN
    FOR v_Count IN 1..500 LOOP
        v_Numbers(v_Count) := v_Count;
        v_Chars(v_Count) := 'Row number ' || v_Count;
    END LOOP;
    FOR v_Count IN 1..500 LOOP
        INSERT INTO temp_table_pnu
            VALUES (v_Numbers(v_Count), v_Chars(v_Count));
    END LOOP;
END;
/
DECLARE
    TYPE t_Numbers IS TABLE OF temp_table_pnu.num_col%TYPE
        INDEX BY BINARY_INTEGER;
    TYPE t_Chars IS TABLE OF temp_table_pnu.char_col%TYPE
        INDEX BY BINARY_INTEGER;
    v_Numbers t_Numbers;
    v_Chars t_Chars;
BEGIN
    FOR v_Count IN 1..500 LOOP
        v_Numbers(v_Count) := v_Count;
        v_Chars(v_Count) := 'Row number ' || v_Count;
    END LOOP;
    FORALL v_Count IN 1..500
        INSERT INTO temp_table_pnu
            VALUES (v_Numbers(v_Count), v_Chars(v_Count));
END;
/
```

17. Să se insereze o linie nouă în tabelul EMP\_PNU, obținându-se rowid-ul acesteia. Să se afișeze valoarea obținută. Măriți cu 30% salariul angajatului cu rowid-ul respectiv și obțineți numele și prenumele acestuia. Ștergeți apoi linia corespunzătoare acelui rowid și afișați informațiile corespunzătoare.

```
DECLARE
    v_NewRowid ROWID;
    v_FirstName employees.first_name%TYPE;
    v_LastName employees.last_name%TYPE;
    v_id employees.employee_id%TYPE;
BEGIN
```

## Laborator 2 PL/SQL

```
-- Insereaza o linie noua in tabelul emp_pnu si obtine rowid-ul acesteia
INSERT INTO emp_pnu
(employee_id, first_name, last_name, email, job_id, salary, hire_date)
VALUES
(employeees_seq.NEXTVAL, 'Xavier', 'Juan','jxav', 'MK_MAN', 2000,
SYSDATE) RETURNING rowid INTO v_NewRowid;
DBMS_OUTPUT.PUT_LINE(' rowid-ul noii linii este ' || v_NewRowid);
UPDATE emp_pnu
SET salary = salary * 1.3
WHERE rowid = v_NewRowid
RETURNING first_name, last_name INTO v_FirstName, v_LastName;
DBMS_OUTPUT.PUT_LINE('Name: ' || v_FirstName || ' ' || v_LastName);
DELETE FROM emp_pnu
WHERE rowid = v_NewRowid
RETURNING employee_id INTO v_id;
DBMS_OUTPUT.PUT_LINE('ID-ul noii linii a fost ' || v_id);

END;
/
```

18. Să se declare un tip tablou indexat de numere T\_NUMBERS și un tip tablou indexat de elemente de tip T\_NUMBERS, numit T\_MULTINUMBERS. Să se declare un TIP T\_MULTIVARRAY de elemente T\_NUMBERS și un tip tablou imbricat de elemente T\_NUMBERS, numit T\_MULTINESTED. Declarați o variabilă de fiecare din aceste tipuri, inițializați-o și apoi afișați.

```
--acesta este doar codul partial!
DECLARE
    TYPE t_Numbers IS TABLE OF NUMBER
        INDEX BY BINARY_INTEGER;
    TYPE t_MultiNumbers IS TABLE OF t_Numbers
        INDEX BY BINARY_INTEGER;
    TYPE t_MultiVarray IS VARRAY(10) OF t_Numbers;
    TYPE t_MultiNested IS TABLE OF t_Numbers;
    v_MultiNumbers t_MultiNumbers;
BEGIN
    v_MultiNumbers(1)(1) := 12345;
END;
/
```

19. Definiți un tip tablou imbricat EmpTab cu elemente de tipul liniilor tabelului EMP\_PNU. Definiți o variabilă de tip EmpTab și apoi inserați linia corespunzătoare în tabelul EMP\_PNU.

```
--acesta este doar codul partial!
DECLARE
    TYPE EmpTab IS TABLE OF emp_pnu%ROWTYPE;
    v_EmpList EmpTab;
BEGIN
    v_EmpList(1) := EmpTab( ... );
    ...
```

## Laborator 2 PL/SQL

---

```
END;  
/
```

20. Declarați un tip EmpTab de tip tablou indexat de tablouri imbricate de linii din tabelul EMPLOYEES. Declarați o variabilă de acest tip. Inserați într-unul din elementele tabloului informațiile corespunzătoare angajatului având codul 200. Atribuiți valori pentru câmpurile last\_name și first\_name ale altui element. Afișați.

```
--acesta este doar codul partial!  
DECLARE  
    TYPE EmpTab IS TABLE OF employees%ROWTYPE  
        INDEX BY BINARY_INTEGER;  
    /* Fiecare element al lui v_Emp este o înregistrare */  
    v_Emp EmpTab;  
BEGIN  
    SELECT *  
    INTO v_Emp(200)  
    FROM employees  
    WHERE employee_id = 200;  
    v_Emp(1).first_name := 'Larry';  
    v_Emp(1).last_name := 'Lemon';  
    --afisare ...  
END;  
/
```