

**Laborator 2 PL/SQL****Tipuri de date compuse (definite de utilizator)****Tipul de date înregistrare (RECORD)****Tipul de date colecție (tablouri indexate – INDEX-BY TABLES****tablouri imbricate – NESTED TABLES****vectori – VARRAYS)****Considerații legate de valoarea NULL**

- comparațiile simple ce implică NULL sunt evaluate la NULL;
- negarea unei valori NULL (NOT NULL) este NULL;
- în comenzile condiționale, dacă o condiție este evaluată la NULL, atunci secvența de comenzi asociată nu va fi executată:

```

IF condiție THEN
    -- dacă valoarea de adevăr este TRUE
    Secvența de comenzi 1;
ELSE
    -- dacă valoarea de adevăr este FALSE sau NULL
    Secvența de comenzi 2;
END IF;

```

Nr	P	Q	NOT P	P OR Q	P AND Q
1	false	false	true	false	false
2	false	true	true	true	false
3	false	Null	true	Null	false
4	true	false	false	true	false
5	true	true	false	true	true
6	true	Null	false	true	Null
7	Null	false	Null	Null	false
8	Null	true	Null	true	Null
9	Null	Null	Null	Null	Null

**1. Care este rezultatul următorului bloc PL/SQL?**

```

DECLARE
    x      NUMBER(1) := 5;
    y      x%TYPE    := NULL;
BEGIN
    IF x <> y THEN
        DBMS_OUTPUT.PUT_LINE ('valoare <> null este = true');
    ELSE
        DBMS_OUTPUT.PUT_LINE ('valoare <> null este != true');
    END IF;

    x := NULL;
    IF x = y THEN
        DBMS_OUTPUT.PUT_LINE ('null = null este = true');
    ELSE
        DBMS_OUTPUT.PUT_LINE ('null = null este != true');
    END IF;
END;
/

```

**Tipul de date RECORD**

- definește un grup de date stocate sub formă de câmpuri, fiecare cu tipul de date și numele propriu;
- numărul de câmpuri nu este limitat;
- se pot defini valori inițiale și constrângeri NOT NULL asupra câmpurilor;
- câmpurile sunt inițializate automat cu NULL;
- tipul RECORD poate fi folosit în secțiunea declarativă a unui bloc, subprogram sau pachet;
- se pot declara sau referi tipuri RECORD imbricate;
- sintaxa generală a definirii tipului RECORD este:

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

v_nume_record  nume_tip;
```

- câmpurile unei înregistrări PL/SQL sunt accesate prin prefixare cu numele înregistrării:

```
nume_record.nume_camp
```

- utilizând tipul *RECORD*:
  - se poate insera o linie într-un tabel (*INSERT*);
  - se poate actualiza o linie într-un tabel (*UPDATE* cu sintaxa *SET ROW*);
  - se poate menține informația afectată de comenzile LMD folosind clauza *RETURNING*.

2. Definiți tipul înregistrare *emp\_record* care conține câmpurile *employee\_id*, *salary* și *job\_id*. Apoi, definiți o variabilă de acest tip.

- a. Inițializați variabila definită. Afișați variabila.

```
DECLARE
  TYPE emp_record IS RECORD
    (cod employees.employee_id%TYPE,
     salariu employees.salary%TYPE,
     job employees.job_id%TYPE);
  v_ang emp_record;
BEGIN
  v_ang.cod:=700;
  v_ang.salariu:= 9000;
  v_ang.job:='SA_MAN';
  DBMS_OUTPUT.PUT_LINE ('Angajatul cu codul '|| v_ang.cod ||
    ' si jobul ' || v_ang.job || ' are salariul ' || v_ang.salariu);
END;
/
```

- b. Inițializați variabila cu valorile corespunzătoare angajatului având codul 101. Afișați variabila.

```
BEGIN
  /***** In loc de ...
  * SELECT employee_id, salary, job_id
  * INTO    v_ang.cod, v_ang.salariu, v_ang.job
  * FROM    employees
  * WHERE   employee_id = 101;
  *****/
```

```

SELECT employee_id, salary, job_id
INTO   v_ang
FROM   employees
WHERE  employee_id = 101;
DBMS_OUTPUT.PUT_LINE ('Angajatul cu codul ' || v_ang.cod ||
    ' si jobul ' || v_ang.job || ' are salariul ' || v_ang.salariu);
END;
/

```

- c. Ștergeți angajatul având codul 100 din tabelul *emp\_\*\*\** și rețineți în variabila definită anterior informații corespunzătoare acestui angajat. Anulați modificările realizate.

```

BEGIN
DELETE FROM emp_***
WHERE employee_id=100
RETURNING employee_id, salary, job_id INTO v_ang;

DBMS_OUTPUT.PUT_LINE ('Angajatul cu codul ' || v_ang.cod ||
    ' si jobul ' || v_ang.job || ' are salariul ' || v_ang.salariu);
END;
/
ROLLBACK;

```

### **Atributul %ROWTYPE**

- Este utilizat pentru a declara o variabilă de tip înregistrare cu aceeași structură ca a altei variabile de tip înregistrare, a unui tabel sau cursor.
3. Declarați două variabile cu aceeași structură ca și tabelul *emp\_\*\*\**. Ștergeți din tabelul *emp\_\*\*\** angajații 100 și 101, menținând valorile șterse în cele două variabile definite. Folosind cele două variabile, introduceți informațiile șterse în tabelul *emp\_\*\*\**.

```

DECLARE
v_ang1      employees%ROWTYPE;
v_ang2      employees%ROWTYPE;
BEGIN
-- sterg angajat 100 si mentin in variabila linia stearsa
DELETE FROM emp_***
WHERE employee_id = 100
RETURNING employee_id, first_name, last_name, email, phone_number,
        hire_date, job_id, salary, commission_pct, manager_id,
        department_id
INTO v_ang1;

-- inserez in tabel linia stearsa
INSERT INTO emp_***
VALUES v_ang1;

-- sterg angajat 101
DELETE FROM emp_***
WHERE employee_id = 101;

```

```
-- obtin datele din tabelul employees
SELECT *
INTO    v_ang2
FROM    employees
WHERE   employee_id = 101;

-- inserez o linie oarecare in emp_***
INSERT INTO emp_***
VALUES (1000, 'FN', 'LN', 'E', null, sysdate, 'AD_VP', 1000, null, 100, 90);

-- modific linia adaugata anterior cu valorile variabilei v_ang2
UPDATE emp_***
SET     ROW = v_ang2
WHERE   employee_id = 1000;
END;
/
```

### **Metode pentru colecții (tablouri indexate, tablouri imbricate, vectori)**

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

Metodele sunt apelate prin expresia:

*nume\_colecție.nume\_metodă* [ (*parametri*) ]

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

- *COUNT* întoarce numărul curent de elemente al unei colecții *PL/SQL*;
- *DELETE(n)* șterge elementul *n* dintr-o colecție *PL/SQL*; *DELETE(m, n)* șterge toate elementele având indecșii între *m* și *n*; *DELETE* șterge toate elementele unei colecții *PL/SQL* (**nu este validă pentru tipul *varrays***);
- *EXISTS(n)* întoarce *TRUE* dacă există al *n*-lea element al unei colecții *PL/SQL*; altfel, întoarce *FALSE*;
- *FIRST*, *LAST* întorc indicele primului, respectiv ultimului element din colecție;
- *NEXT(n)*, *PRIOR(n)* întorc indicele elementului următor, respectiv precedent celui de rang *n* din colecție, iar dacă nu există un astfel de element întorc valoarea *null*;
- *EXTEND* adaugă elemente la sfârșitul unei colecții: *EXTEND* adaugă un element *null* la sfârșitul colecției, *EXTEND(n)* adaugă *n* elemente *null*, *EXTEND(n, i)* adaugă *n* copii ale elementului de rang *i* (**nu este validă pentru tipul *index-by tables***);
- *LIMIT* întoarce numărul maxim de elemente al unei colecții (cel de la declarare) pentru tipul vector și *null* pentru tablouri imbricate (**nu este validă pentru tipul *index-by tables***);
- *TRIM* șterge elementele de la sfârșitul unei colecții: *TRIM* șterge ultimul element, *TRIM(n)* șterge ultimele *n* elemente (**nu este validă pentru tipul *index-by tables***). Similar metodei *EXTEND*, metoda *TRIM* operează asupra dimensiunii interne a tabloului imbricat.
  - *EXISTS* este singura metodă care poate fi aplicată unei colecții atomice *null*. Orice altă metodă declanșează excepția *COLLECTION\_IS\_NULL*.
  - *COUNT*, *EXISTS*, *FIRST*, *LAST*, *NEXT*, *PRIOR* și *LIMIT* sunt funcții, iar restul sunt proceduri *PL/SQL*.

#### Observație:

- Tipul *tablou indexat* poate fi utilizat numai în declarații *PL/SQL*. Tipurile *vector* și *tablou imbricat* 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).
- Tablourile indexate pot avea indice negativ, domeniul permis pentru index fiind

–2147483648..2147483647; pentru tablourile imbricate domeniul permis pentru index este 1..2147483647.

- Tablourile imbricate și vectorii trebuie inițializați și/sau estinși pentru a li se putea adăuga elemente noi.

### **Tablouri indexate (index-by table)**

- Sunt mulțimi de perechi cheie-valoare, în care fiecare cheie este unică și utilizată pentru a putea localiza valoarea asociată.
- Tablourile indexate pot crește în dimensiune în mod dinamic neavând specificat un număr maxim de elemente.
- Un tablou indexat nu poate fi inițializat la declarare, este necesară o comandă explicită pentru a inițializa fiecare element al său.
- Sintaxa generală pentru tabloul indexat este:

```
TYPE t_tablou_indexat IS TABLE OF {
    { tip_de_date | variabila%TYPE
      | tabel.coloana%TYPE }[NOT NULL]}
    | tabel%ROWTYPE }
INDEX BY PLS_INTEGER | BINARY_INTEGER | VARCHAR2(n);

v_tablou t_tablou_indexat;
```

4. Definiți un tablou indexat de numere. Introduceți în acest tablou primele 10 de numere naturale.
  - a. Afișați numărul de elemente al tabloului și elementele acestuia.
  - b. Setati la valoarea *null* elementele de pe pozițiile impare. Afișați numărul de elemente al tabloului și elementele acestuia.
  - c. Ștergeți primul element, elementele de pe pozițiile 5, 6 și 7, respectiv ultimul element. Afișați valoarea și indicele primului, respectiv ultimului element. Afișați elementele tabloului și numărul acestora.
  - d. Ștergeți toate elementele tabloului.

```
DECLARE
    TYPE tablou_indexat IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
    t    tablou_indexat;
BEGIN
    -- punctul a
    FOR i IN 1..10 LOOP
        t(i) := i;
    END LOOP;
    DBMS_OUTPUT.PUT('Tabloul are ' || t.COUNT || ' elemente: ');
    FOR i IN t.FIRST..t.LAST LOOP
        DBMS_OUTPUT.PUT(t(i) || ' ');
    END LOOP;
    DBMS_OUTPUT.NEW_LINE;

    -- punctul b
    FOR i IN 1..10 LOOP
        IF i mod 2 = 1 THEN t(i) := null;
        END IF;
    END LOOP;
    DBMS_OUTPUT.PUT('Tabloul are ' || t.COUNT || ' elemente: ');
```

```

FOR i IN t.FIRST..t.LAST LOOP
    DBMS_OUTPUT.PUT(nvl(t(i), 0) || ' ');
END LOOP;
DBMS_OUTPUT.NEW_LINE;

-- punctul c
t.DELETE(t.first);
t.DELETE(5,7);
t.DELETE(t.last);
DBMS_OUTPUT.PUT_LINE('Primul element are indicele ' || t.first ||
    ' si valoarea ' || nvl(t(t.first),0));
DBMS_OUTPUT.PUT_LINE('Ultimul element are indicele ' || t.last ||
    ' si valoarea ' || nvl(t(t.last),0));
DBMS_OUTPUT.PUT('Tabloul are ' || t.COUNT || ' elemente: ');
FOR i IN t.FIRST..t.LAST LOOP
    IF t.EXISTS(i) THEN
        DBMS_OUTPUT.PUT(nvl(t(i), 0) || ' ');
    END IF;
END LOOP;
DBMS_OUTPUT.NEW_LINE;

-- punctul d
t.delete;
DBMS_OUTPUT.PUT_LINE('Tabloul are ' || t.COUNT || ' elemente.');
```

END;

/

5. Definiți un tablou indexat de înregistrări având tipul celor din tabelul *emp\_\*\*\**. Ștergeți primele două linii din tabelul *emp\_\*\*\**. Afișați elementele tabloului. Folosind tabelul indexat adăugați înapoi cele două linii șterse.

```

DECLARE
    TYPE tablou_indexat IS TABLE OF emp_***%ROWTYPE
        INDEX BY BINARY_INTEGER;
    t    tablou_indexat;
BEGIN
-- stergere din tabel si salvare in tablou
DELETE FROM emp_***
WHERE ROWNUM<= 2
RETURNING employee_id, first_name, last_name, email, phone_number,
    hire_date, job_id, salary, commission_pct, manager_id,
    department_id
BULK COLLECT INTO t;

--afisare elemente tablou
DBMS_OUTPUT.PUT_LINE (t(1).employee_id ||' ' || t(1).last_name);
DBMS_OUTPUT.PUT_LINE (t(2).employee_id ||' ' || t(2).last_name);

--inserare cele 2 linii in tabel
INSERT INTO emp_*** VALUES t(1);
INSERT INTO emp_*** VALUES t(2);
END;
```

/

**Tablouri imbricate (nested table)**

Sintaxa generală pentru tabloul imbricat este:

```
[CREATE [OR REPLACE]] TYPE t_tablou_imbri IS TABLE OF
{ { tip_de_date | variabila%TYPE |
  tabel.coloana%TYPE } [NOT NULL] } | tabel%ROWTYPE };

v_tablou_imbri t_tablou_imbri;
```

- Singura diferență sintactică între tablourile indexate și cele imbricate este absența clauzei *INDEX BY*. Mai exact, dacă această clauză lipsește tipul de date declarat este tablou imbricat.
- Numărul maxim de linii al unui tablou imbricat este dat de capacitatea maximă 2 GB.
- Tablourile imbricate:
  - 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.
- 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.
- Tablourile imbricate trebuie inițializate 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, atunci când aceasta este inițializată.
  - Pentru vectori nu poate fi depășită dimensiunea maximă precizată la declarare.
  - Atunci când constructorul este apelat fără argumente se va crea o colecție fără niciun element (vidă), dar care este *not null*.

6. Rezolvați exercițiul 4 folosind tablouri imbricate.

```
DECLARE
  TYPE tablou_imbricat IS TABLE OF NUMBER;
  t      tablou_imbricat := tablou_imbricat();
BEGIN
  -- punctul a
  FOR i IN 1..10 LOOP
    t.extend;
    t(i) := i;
  END LOOP;
  DBMS_OUTPUT.PUT('Tabloul are ' || t.COUNT || ' elemente: ');

  FOR i IN t.FIRST..t.LAST LOOP
    DBMS_OUTPUT.PUT(t(i) || ' ');
  END LOOP;
  DBMS_OUTPUT.NEW_LINE;
```

```

-- punctul b
FOR i IN 1..10 LOOP
    IF i mod 2 = 1 THEN t(i):=null;
    END IF;
END LOOP;
DBMS_OUTPUT.PUT('Tabloul are ' || t.COUNT || ' elemente: ');
FOR i IN t.FIRST..t.LAST LOOP
    DBMS_OUTPUT.PUT(nvl(t(i), 0) || ' ');
END LOOP;
DBMS_OUTPUT.NEW_LINE;

-- punctul c
t.DELETE(t.first);
t.DELETE(5,7);
t.DELETE(t.last);
DBMS_OUTPUT.PUT_LINE('Primul element are indicele ' || t.first ||
    ' si valoarea ' || nvl(t(t.first),0));
DBMS_OUTPUT.PUT_LINE('Ultimul element are indicele ' || t.last ||
    ' si valoarea ' || nvl(t(t.last),0));
DBMS_OUTPUT.PUT('Tabloul are ' || t.COUNT || ' elemente: ');
FOR i IN t.FIRST..t.LAST LOOP
    IF t.EXISTS(i) THEN
        DBMS_OUTPUT.PUT(nvl(t(i), 0) || ' ');
    END IF;
END LOOP;
DBMS_OUTPUT.NEW_LINE;

-- punctul d
t.delete;
DBMS_OUTPUT.PUT_LINE('Tabloul are ' || t.COUNT || ' elemente.');
```

END;

/

7. Declarați un tip tablou imbricat de caractere și o variabilă de acest tip. Inițializați variabila cu următoarele valori: m, i, n, i, m. Afișați conținutul tabloului, de la primul la ultimul element și invers. Ștergeți elementele 2 și 4 și apoi afișați conținutul tabloului.

```

DECLARE
    TYPE tablou_imbricat IS TABLE OF CHAR(1);
    t tablou_imbricat := tablou_imbricat('m', 'i', 'n', 'i', 'm');
    i INTEGER;
BEGIN
    i := t.FIRST;
    WHILE i <= t.LAST LOOP
        DBMS_OUTPUT.PUT(t(i));
        i := t.NEXT(i);
    END LOOP;
    DBMS_OUTPUT.NEW_LINE;

    i := t.LAST;
    WHILE i >= t.FIRST LOOP
        DBMS_OUTPUT.PUT(t(i));
        i := t.PRIOR(i);
    END LOOP;
END;
```



```

END LOOP;
DBMS_OUTPUT.NEW_LINE;

t.delete(2);
t.delete(4);

i := t.FIRST;
WHILE i <= t.LAST LOOP
    DBMS_OUTPUT.PUT(t(i));
    i := t.NEXT(i);
END LOOP;
DBMS_OUTPUT.NEW_LINE;

i := t.LAST;
WHILE i >= t.FIRST LOOP
    DBMS_OUTPUT.PUT(t(i));
    i := t.PRIOR(i);
END LOOP;
DBMS_OUTPUT.NEW_LINE;

END;
/

```

## Vectori

- Sintaxa generală pentru declararea vectorilor:

```

[CREATE [OR REPLACE]] TYPE t_vector IS VARRAY(limita) OF
    { { tip_de_date | variabila%TYPE |
      Tabel.coloana%TYPE } [NOT NULL] }
    | tabel%ROWTYPE };

v_vector t_vector;

```

- Spre deosebire de tablourile imbricate, 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.

### 8. Rezolvați exercițiul 4 folosind vectori.

```

DECLARE
    TYPE vector IS VARRAY(20) OF NUMBER;
    t    vector:= vector();
BEGIN
    -- punctul a
    FOR i IN 1..10 LOOP
        t.extend; t(i):=i;
    END LOOP;
    DBMS_OUTPUT.PUT('Tabloul are ' || t.COUNT || ' elemente: ');
    FOR i IN t.FIRST..t.LAST LOOP
        DBMS_OUTPUT.PUT(t(i) || ' ');
    END LOOP;
    DBMS_OUTPUT.NEW_LINE;
    -- punctul b
    FOR i IN 1..10 LOOP
        IF i mod 2 = 1 THEN t(i):=null;

```

```

        END IF;
    END LOOP;
    DBMS_OUTPUT.PUT('Tabloul are ' || t.COUNT || ' elemente: ');
    FOR i IN t.FIRST..t.LAST LOOP
        DBMS_OUTPUT.PUT(nvl(t(i), 0) || ' ');
    END LOOP;
    DBMS_OUTPUT.NEW_LINE;

-- punctul c
-- metodele DELETE(n), DELETE(m,n) nu sunt valabile pentru vectori!!!
-- din vectori nu se pot sterge elemente individuale!!!

-- punctul d
t.delete;
DBMS_OUTPUT.PUT_LINE('Tabloul are ' || t.COUNT || ' elemente.');
```

END;

/

9. Definiți tipul *subordonati\_\*\*\** (vector, dimensiune maximă 10, menține numere). Creați tabelul *manageri\_\*\*\** cu următoarele câmpuri: *cod\_mgr* NUMBER(10), *nume* VARCHAR2(20), *lista subordonati\_\*\*\**. Introduceți 3 linii în tabel. Afișați informațiile din tabel. Ștergeți tabelul creat, apoi tipul.

```

CREATE OR REPLACE TYPE subordonati_*** AS VARRAY(10) OF NUMBER(4);
/

CREATE TABLE manageri_*** (cod_mgr NUMBER(10),
                           nume VARCHAR2(20),
                           lista subordonati_***);

DECLARE
    v_sub    subordonati_***:= subordonati_***(100,200,300);
    v_lista  manageri_***.lista%TYPE;
BEGIN
    INSERT INTO manageri_***
    VALUES (1, 'Mgr 1', v_sub);

    INSERT INTO manageri_***
    VALUES (2, 'Mgr 2', null);

    INSERT INTO manageri_***
    VALUES (3, 'Mgr 3', subordonati_***(400,500));

    SELECT lista
    INTO    v_lista
    FROM    manageri_***
    WHERE   cod_mgr=1;

    FOR j IN v_lista.FIRST..v_lista.LAST loop
        DBMS_OUTPUT.PUT_LINE(v_lista(j));
    END LOOP;

END;

/

SELECT * FROM manageri_***;
```

```
DROP TABLE manageri_***;
DROP TYPE subordonati_***;
```

- 10.** Creați tabelul *emp\_test\_\*\*\** cu coloanele *employee\_id* și *last\_name* din tabelul *employees*. Adăugați în acest tabel un nou câmp numit *telefon* de tip tablou imbricat. Acest tablou va menține pentru fiecare salariat toate numerele de telefon la care poate fi contactat. Inserați o linie nouă în tabel. Actualizați o linie din tabel. Afișați informațiile din tabel. Ștergeți tabelul și tipul.

```
CREATE TABLE emp_test_*** AS
  SELECT employee_id, last_name FROM employees
  WHERE ROWNUM <= 2;

CREATE OR REPLACE TYPE tip_telefon_*** IS TABLE OF VARCHAR(12);
/

ALTER TABLE emp_test_***
ADD (telefon tip_telefon_***)
NESTED TABLE telefon STORE AS tabel_telefon_***;

INSERT INTO emp_test_***
VALUES (500, 'XYZ', tip_telefon_***('074XXX', '0213XXX', '037XXX'));

UPDATE emp_test_***
SET   telefon = tip_telefon_***('073XXX', '0214XXX')
WHERE employee_id=100;

SELECT  a.employee_id, b.*
FROM    emp_test_*** a, TABLE (a.telefon) b;

DROP TABLE emp_test_***;
DROP TYPE   tip_telefon_***;
```

- 11.** Ștergeți din tabelul *emp\_\*\*\** salariații având codurile menținute într-un vector.

*Obs.* Comanda *FORALL* permite ca toate liniile unei colecții să fie transferate simultan printr-o singură operație. Procedul este numit **bulk bind**.

```
FORALL index IN lim_inf..lim_sup
  comanda_sql;
```

#### *Varianta 1*

```
DECLARE
  TYPE tip_cod IS VARRAY(5) OF NUMBER(3);
  coduri tip_cod := tip_cod(205,206);
BEGIN
  FOR i IN coduri.FIRST..coduri.LAST LOOP
    DELETE FROM emp_***
    WHERE employee_id = coduri (i);
  END LOOP;
END;
/
SELECT employee_id FROM emp_***;
ROLLBACK;
```

*Varianta 2*

```
DECLARE
  TYPE tip_cod IS VARRAY(20) OF NUMBER;
  coduri tip_cod := tip_cod(205,206);
BEGIN
  FORALL i IN coduri.FIRST..coduri.LAST
    DELETE FROM emp_***
      WHERE employee_id = coduri (i);
END;
/
SELECT employee_id FROM emp_***;
ROLLBACK;
```

**Exerciții**

- E1.** Mențineți într-o colecție codurile celor mai prost plătiți 5 angajați care nu câștigă comision. Folosind această colecție măriți cu 5% salariul acestor angajați. Afișați valoarea veche a salariului, respectiv valoarea nouă a salariului.
- E2.** Definiți un tip colecție denumit `tip_orase_***`. Creați tabelul `excursie_***` cu următoarea structură: `cod_excursie` NUMBER(4), `denumire` VARCHAR2(20), `orase` tip\_orase\_\*\*\* (ce va conține lista orașelor care se vizitează într-o excursie, într-o ordine stabilită; de exemplu, primul oraș din listă va fi primul oraș vizitat), `status` (disponibilă sau anulată).
- Inserați 5 înregistrări în tabel.
  - Actualizați coloana `orase` pentru o excursie specificată:
    - adăugați un oraș nou în listă, ce va fi ultimul vizitat în excursia respectivă;
    - adăugați un oraș nou în listă, ce va fi al doilea oraș vizitat în excursia respectivă;
    - inversați ordinea de vizitare a două dintre orașe al căror nume este specificat;
    - eliminați din listă un oraș al cărui nume este specificat.
  - Pentru o excursie al cărui cod este dat, afișați numărul de orașe vizitate, respectiv numele orașelor.
  - Pentru fiecare excursie afișați lista orașelor vizitate.
  - Anulați excursiile cu cele mai puține orașe vizitate.
- E3.** Rezolvați problema anterioară folosind un alt tip de colecție studiat.
- E4.** Adaptați cerințele exercițiilor 9 și 10 pentru diagrama proiectului prezentată la materia Baze de Date din anul I. Rezolvați aceste exerciții în PL/SQL, folosind baza de date proprie.