# Cuprins

4. *PL/SQL* – Tipuri de date ..... 2
4.1. Tipuri de date scalare ..... 4
4.1.1. Tipuri de date *SQL* ..... 5
4.1.2. Tipuri de date *PL/SQL* ..... 14
4.1.3. Tipuri de date și subtipurile acestora ..... 15
4.1.4. Conversii între tipuri de date ..... 17
4.1.5. Atributul %*TYPE* ..... 17
4.2. Tipuri de date compuse ..... 18
4.2.1. Atributul %*ROWTYPE* ..... 18
4.2.2. Tipul de date înregistrare ..... 19
4.2.3. Tipul de date colecție ..... 20
4.2.4. Tablouri indexate ..... 22
4.2.5. Tablouri imbricate ..... 25
4.2.6. Vectori ..... 29
4.2.7. Colecții pe mai multe niveluri ..... 31
4.2.8. Compararea colecțiilor ..... 32
4.2.9. Prelucrarea colecțiilor stocate în tabele ..... 33
4.2.10. Procedeul *bulk collect* ..... 35
4.2.11. Procedeul *bulk bind* ..... 37
4.3. Vizualizări din dicționarul datelor ..... 39
Bibliografie ..... 40

# 4. *PL/SQL* – Tipuri de date

- Tipul de date este o mulțime de valori predefinită sau definită de utilizator.
- Constantele, variabilele și parametrii *PL/SQL* trebuie să aibă specificat un tip de date. Acesta va determina formatul de stocare, valorile și operațiile permise.

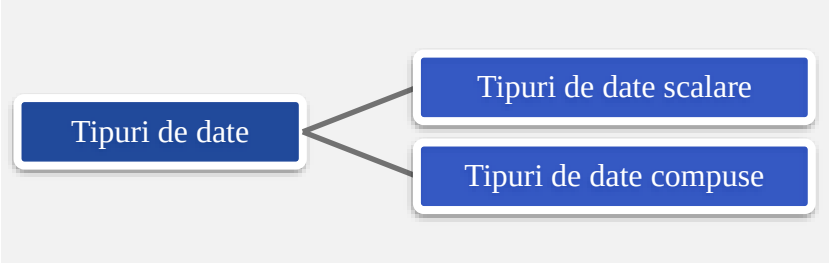


**Fig. 4.1.** Tipuri de date

- Există două categorii de tipuri de date:
  - tipuri de date scalare
    - pot stoca o singură valoare
    - valoarea stocată nu poate avea componente interne
  - tipuri de date compuse
    - pot stoca mai multe valori
    - valorile stocate pot avea componente interne ce pot fi accesate individual

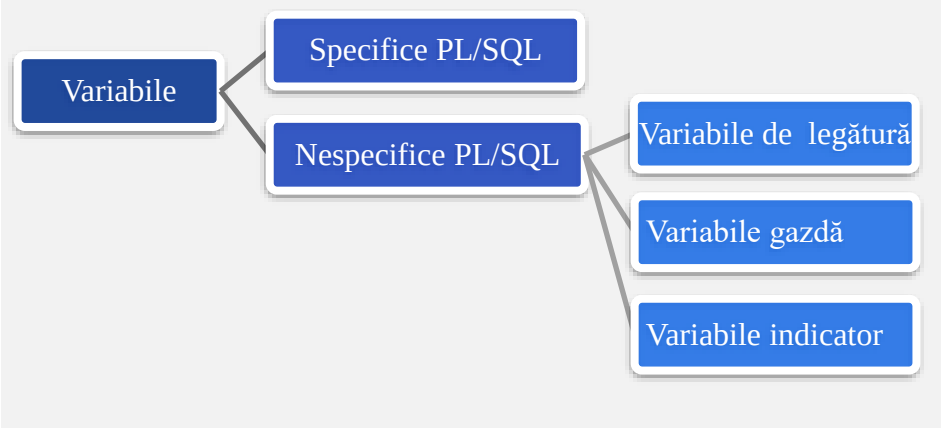


**Fig. 4.2.** Variabile utilizate de *Oracle*

- Variabilele folosite în *Oracle* pot fi:
  - specifice *PL/SQL*
  - nespecifice *PL/SQL*
    - variabile de legătură (*bind variables*)

- se declară într-un mediu gazdă și sunt folosite pentru a transfera la momentul execuției valori (numerice sau de tip caracter) din/ în unul sau mai multe programe *PL/SQL*
- în *SQL*Plus* se declară folosind comanda *VARIABLE*, iar pentru afișarea valorilor acestora se utilizează comanda *PRINT*; sunt referite prin prefixarea cu simbolul „:“, pentru a putea fi deosebite de variabilele declarate în *PL/SQL*

▪ variabile gazdă (*host variables*)
- permit transferul de valori între un mediu de programare (de exemplu, instrucțiunile *SQL* pot fi integrate în programe *C/C++*) și instrucțiunile *SQL* care comunică cu *server*-ul bazei de date *Oracle*
- în precompilatorul *Pro*C/C++* sunt declarate între directivele *EXEC SQL BEGIN DECLARE SECTION* și *EXEC SQL END DECLARE SECTION*

▪ variabile indicator (*indicator variables*)
- se asociază unei variabile gazdă și permit monitorizarea acesteia
- permit comunicarea valorii *null* între *Oracle* și un limbaj gazdă care nu are o valoare corespunzătoare pentru *null* (de exemplu, *C*)
- se utilizează folosind una dintre formele de mai jos

```
:variabilă_gazdă INDICATOR :variabilă_indicator
```

sau

```
:variabilă_gazdă :variabilă_indicator
```

- sunt de tip întreg (stocat *2 bytes*)
- *Oracle* poate atribui unei variabile indicator următoarele valori:

0, dacă operația s-a realizat cu succes

-1, dacă o valoare *null* a fost întoarsă, inserată sau actualizată

-2, dacă într-o variabilă gazdă de tip caracter s-a întors o valoare de tip *LONG* trunchiată, fără să se poată determina lungimea originală a coloanei

>0, dacă rezultatul unei comenzi SELECT sau *FETCH* într-o variabilă gazdă de tip caracter a fost trunchiat; în acest caz valoarea indicator este dimensiunea originală a coloanei.

Exemplu

```
EXEC SQL BEGIN DECLARE SECTION;
    float  pret_produs;
    short  indicator_pret;
EXEC SQL END DECLARE SECTION;
...

EXEC SQL SELECT pret
    INTO :pret_produs:indicator_pret
    FROM  produse
    WHERE id_produs = 100;

IF (indicator_pret == -1)
    PRINTF("Produsul nu are pret specificat ");

ELSE
    ...;
```

- un program poate atribui unei variabile indicator următoarele valori:

  -1, caz în care *Oracle* va atribui coloanei valoarea *null*, ignorând valoarea variabilei gazdă

  >=0, caz în care *Oracle* va atribui coloanei valoarea variabilei gazdă

Exemplu

```
…
SET v_indicator = -1;
EXEC SQL INSERT INTO clienti (id_client, status)
     VALUES (:v_cod, :v_status:v_indicator);
```

## 4.1. Tipuri de date scalare

- Un tip de date scalar stochează o singură valoare care nu poate avea componente interne.
- Tipurile de date scalare pot avea definite subtipuri.
  - Subtipul este un tip de date care reprezintă o submulțime a unui alt tip de date, denumit tip de bază.
  - Subtipul permite aceleași operații ca și tipul de bază.
- Pachetul *STANDARD* conține tipurile și subtipuri predefinite.
  - Utilizatorii pot defini propriile lor subtipuri.

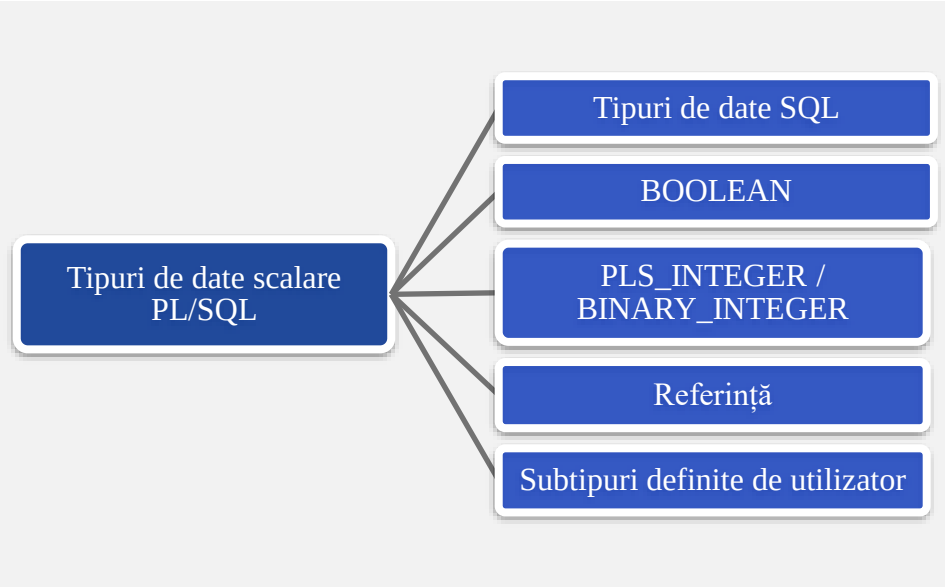


**Fig. 4.3.** Tipuri de date scalare PL/SQL

- Tipuri de date scalare *PL/SQL*:
  - tipurile de date *SQL*
  - *BOOLEAN*
  - *PLS_INTEGER / BINARY_INTEGER*
  - referință (de exemplu, *REF CURSOR* )
  - subtipuri definite de utilizator

## 4.1.1. Tipuri de date *SQL*

- Dimensiunea maximă permisă de aceste tipuri de date poate fi diferită în *PL/SQL* față de *SQL*.

**Tipuri CHARACTER**

| Tip date | Descriere | Dim max PL/SQL | Dim max SQL |
|---|---|---|---|
| CHAR [(n [BYTE\|CHAR]) ] | Dimensiune fixă - *n bytes* sau caractere (un caracter poate ocupa mai mult de 1 byte). Implicit n=1 byte. | 32767 bytes | 2000 bytes |
| VARCHAR2 (n [BYTE\|CHAR]) | Dimensiune variabilă - *n bytes* sau caractere. Nu are valoare implicită. | 32767 bytes | 4000 bytes |
| NCHAR [(n)] | Dimensiune fixă - *n* caractere, aparținând setului național de caractere. Implicit n=1. | 32767 bytes | 2000 bytes |
| NVARCHAR2(n) | Dimensiune variabilă, având *n* caractere, aparținând setului național de caractere. Nu are valoare implicită. | 32767 bytes | 4000 bytes |

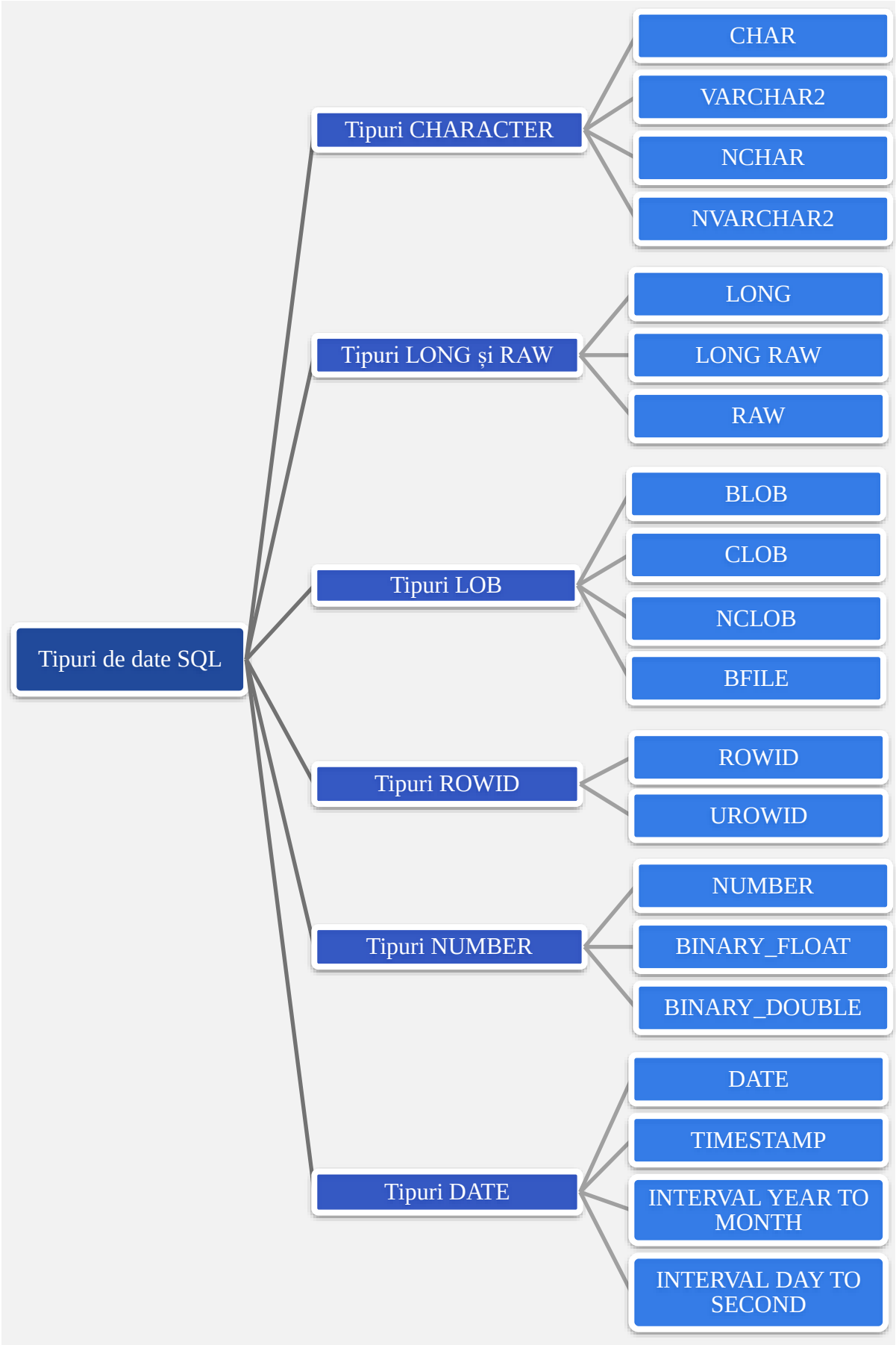


**Fig. 4.4.** Tipuri de date SQL

Exemplul 4.1

```
DECLARE
  sir_1 CHAR(10) := 'PL/SQL';
  sir_2 VARCHAR2(10) := 'PL/SQL';
BEGIN
  IF sir_1 = sir_2 THEN
      DBMS_OUTPUT.PUT_LINE (sir_1 || ' = ' || sir_2);
  ELSE
     DBMS_OUTPUT.PUT_LINE (sir_1 || ' =! ' || sir_2 );
  END IF;
END;
```

## Tipuri LONG și RAW

| Tip date | Descriere | Dim max PL/SQL | Dim max SQL |
|---|---|---|---|
| LONG | Dimensiune variabilă. Păstrat doar din motive de compatibilitate cu versiunile anterioare. În prezent se utilizează tipul LOB. | 32760 bytes | 2GB – 1 (gigabytes) |
| LONG RAW | Date în format binar. Dimensiune variabilă. Păstrat doar din motive de compatibilitate cu versiunile anterioare. În prezent se utilizează tipul LOB. | 32760 bytes | 2GB |
| RAW(n) | Date în format binar sau date care sunt prelucrate *byte* cu *byte* (grafice, fișiere audio) Dimensiune variabilă. Nu are valoare implicită. | 32767 bytes | 2000 bytes |

## Tipuri LOB

| Tip date | Descriere | Dim max PL/SQL | Dim max SQL |
|---|---|---|---|
| BLOB | Obiecte de tip binar de dimensiuni mari | 128TB (terabytes) | (4GB-1byte) * dim_bloc (dimensiune bloc date) |
| CLOB | Obiecte de tip caracter de dimensiuni mari | 128TB | (4GB-1byte) * dim_bloc |
| NCLOB | Obiecte de tip caracter de dimensiuni mari. Datele stocate corespund setului național de caractere. | 128TB | (4GB-1byte) * dim_bloc |
| BFILE | Specifică adresa unui fişier de date extern. Permite stocarea datelor binare în fișierele sistemului de operare (LOB extern). | 128TB | (4GB-1byte) * dim_bloc |

## Tipuri ROWID

| Tip date | Descriere | Dim max PL/SQL | Dim max SQL |
|---|---|---|---|
| ROWID | Adresele fizice ale liniilor. (OOOOOO.FFF.BBBBBB.LLL) | obiect.fișier. bloc.linie | obiect.fișier. bloc.linie |
| UROWID [(n)] | Adresele logice și fizice ale liniilor. Implicit 4000*bytes*. | 4000 bytes | 4000 bytes |

- *ROWID*-urile fizice stochează adresa liniilor din tabelele obișnuite (care nu sunt de tip *index-organized*), *cluster*-e, partiții și subpartiții ale tabelelor, indecși, partiții și subpartiții ale indecșilor.
- *ROWID*-urile logice stochează adresa liniilor din tabele de tip *index-organized.*

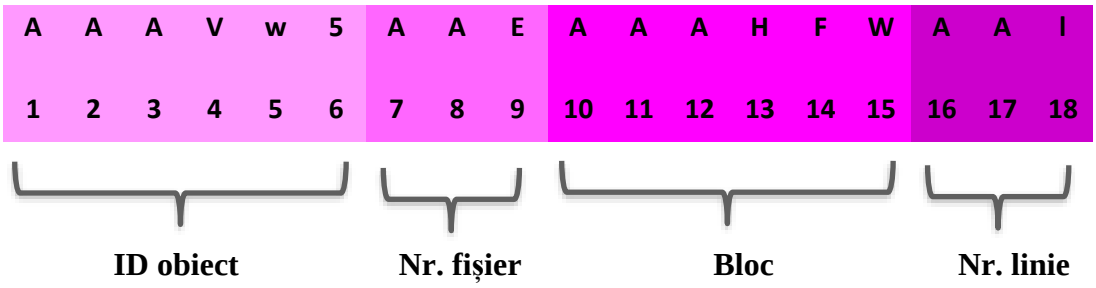


**Fig. 4.5** Componentele unui *ROWID*

Tipul de date *UROWID* (*Universal ROWID)* permite atât adresele fizice, cât și logice ale liniilor dintr-o bază de date *Oracle*, dar și adresele liniilor din tabele externe *non-Oracle*.

Tabelele relaționale obișnuite stochează datele nesortate.

Un tabel de tip *index-organized* este un tip de tabel care stochează datele într-o structură de index *B-tree*, sortate logic după cheia primară. Față de indexul normal creat automat la definirea unei chei primare, care stochează doar coloanele incluse în definiția cheii, indexul tabelului de tip *index-organized* stochează în general toate coloanele tabelului (coloanele care sunt accesate rar pot fi mutate în alte segmente față de cel principal).

**Tipuri NUMBER**

| Tip date | Descriere | Dim max PL/SQL | Dim max SQL |
|---|---|---|---|
| NUMBER [(p[, s]) | Număr cu precizia *p* (numărul total de cifre ale părții întregi) și scala s (numărul de cifre ale părții zecimale dacă s este pozitiv).<br><br>Implicit s=0.<br><br>$p \in [1,38]$, $s \in [-84,127]$<br><br>Dacă *p* nu este specificat, atunci se stochează valoarea dată.<br><br>Dacă *s* este pozitiv, atunci se face rotunjire a părții zecimale (de ex. dacă s=2, 12,474 devine 12,47, iar 12,476 devine 12,48).<br><br>Dacă *s* este negativ, atunci se face rotunjire a părții întregi (de ex. dacă s=-2 avem rotunjire la sute; 1245 devine 1200 și 1255 devine 1300).<br><br>Dacă s=0 se face rotunjire la întreg (de ex., numărul 3,45 devine 3, iar numărul 3,67 devine 4).<br><br>Pentru a preciza doar valoarea lui *s* se folosește NUMBER(*,s). | 38 cifre | 38 cifre |
| BINARY_FLOAT | Număr virgulă mobilă precizie simplă (32 biți) | 5 bytes (1 byte pentru lungime) | 5 bytes |
| BINARY_DOUBLE | Număr virgulă mobilă precizie dublă (64 biți) | 9 bytes (1 byte pentru lungime) | 9 bytes |

**Tipuri DATE**

| Tip de date | Descriere |
|---|---|
| DATE | Dată calendaristică între 01.01.4712 î.Hr. și 31.12.9999 d.Hr |
| TIMESTAMP [(p)] [ WITH [ LOCAL ] TIME ZONE ] | Dată calendaristică și timp, cu precizia *p* pentru milisecunde (*p* ∈ [0,9], implicit *p*=6). *WITH TIME ZONE* specifică diferența de fus orar. *LOCAL* implică transformarea datei calendaristice conform timpului regiunii care este setat la nivelul bazei de date. |
| INTERVAL YEAR [(p)] TO MONTH | Perioadă de timp specificată în ani și luni. Precizia *p* reprezintă numărul maxim de cifre al câmpului *YEAR* (*p* ∈ [0,9], implicit *p*=2). |
| INTERVAL DAY [(d)] TO SECOND [(s)] | Perioadă de timp specificată în zile, ore, minute și secunde. Precizia *d* reprezintă numărul maxim de cifre al câmpului *DAY*(*d* ∈ [0,9], implicit *d*=2). |

- *Oracle* stochează datele de tip *DATE* folosind în întotdeauna 7 *bytes*. Fiecare *byte* stochează câte un element din dată.

| Nr Byte | Descriere |
|---|---|
| 1 | Secol (înainte de stocare adaugă 100) |
| 2 | An (înainte de stocare adaugă 100) |
| 3 | Luna |
| 4 | Zi |
| 5 | Ora (înainte de stocare adaugă 1) |
| 6 | Minute (înainte de stocare adaugă 1) |
| 7 | Secunde (înainte de stocare adaugă 1) |

**Exemplul 4.2**

```
CREATE TABLE test (d DATE);

INSERT INTO test
VALUES (TO_DATE('15-OCT-2012','DD-MON-YYYY'));

INSERT INTO test
VALUES (TO_DATE('15-OCT-2012 00:00:00',
                'DD-MON-YYYY HH24:MI:SS'));

INSERT INTO test
VALUES (TO_DATE('15.10.2012  15:22:07',
                'DD.MM.YYYY HH24:MI:SS'));
```

```
INSERT INTO test VALUES (sysdate);

SELECT DUMP(d) FROM test;

Typ=12 Len=7: 120,112,10,15,1,1,1
Typ=12 Len=7: 120,112,10,15,1,1,1
Typ=12 Len=7: 120,112,10,15,16,23,8
Typ=12 Len=7: 120,112,8,24,14,34,27
```

- Dacă se utilizează direct *SYSDATE* sau *TO_DATE* formatul se modifică:
  - Typ = 13
  - Len = 8
  - *Byte*-ul 8 nu este utilizat
  - anul se poate obține cu următoarea formulă*: Byte 1 + Byte 2 * 256*

Exemplul 4.3

```
SELECT DUMP(TO_DATE('15-OCT-2012 00:00:00',
       'DD-MON-YYYY HH24:MI:SS'))
FROM   DUAL;

Typ=13 Len=8: 220,7,10,15,0,0,0,0

SELECT DUMP(SYSDATE)
FROM   DUAL;

Typ=13 Len=8: 220,7,8,24,13,35,57,0

2012 = 220+7*256
```

**Tipuri de date *SQL ANSI* sau *IBM***

- *Oracle* recunoaște numele tipurilor de date *ANSI* sau *IBM* (folosite de *SQL/DS* sau *DB2*) care diferă de numele tipurilor de date proprii.
- Atunci când este utilizat un tip de date *ANSI* sau *IBM*, acesta va fi convertit automat la tipul de date echivalent din *Oracle*.

| **Tip de date *ANSI*** | **Tip de date echivalent *ORACLE*** |
|---|---|
| CHARACTER(n)<br>CHAR(n) | CHAR(n) |
| CHARACTER VARYING(n)<br>CHAR VARYING(n) | VARCHAR2(n) |
| NATIONAL CHARACTER(n)<br>NATIONAL CHAR(n)<br>NCHAR(n) | NCHAR(n) |
| NATIONAL CHARACTER VARYING(n)<br>NATIONAL CHAR VARYING(n)<br>NCHAR VARYING(n) | NVARCHAR2(n) |
| NUMERIC[(p,s)]<br>DECIMAL[(p,s)] | NUMBER(p,s) |
| INTEGER<br>INT<br>SMALLINT | NUMBER(38) |
| FLOAT<br>DOUBLE PRECISION<br>REAL | FLOAT(126)<br>FLOAT(126)<br>FLOAT(63) |
| **Tip de date *SQL/DS* sau *DB2*** | **Tip de date echivalent *ORACLE*** |
| CHARACTER(n) | CHAR(n) |
| VARCHAR(n) | VARCHAR(n) |
| LONG VARCHAR | LONG |
| DECIMAL(p,s) | NUMBER(p,s) |
| INTEGER<br>SMALLINT | NUMBER(38) |

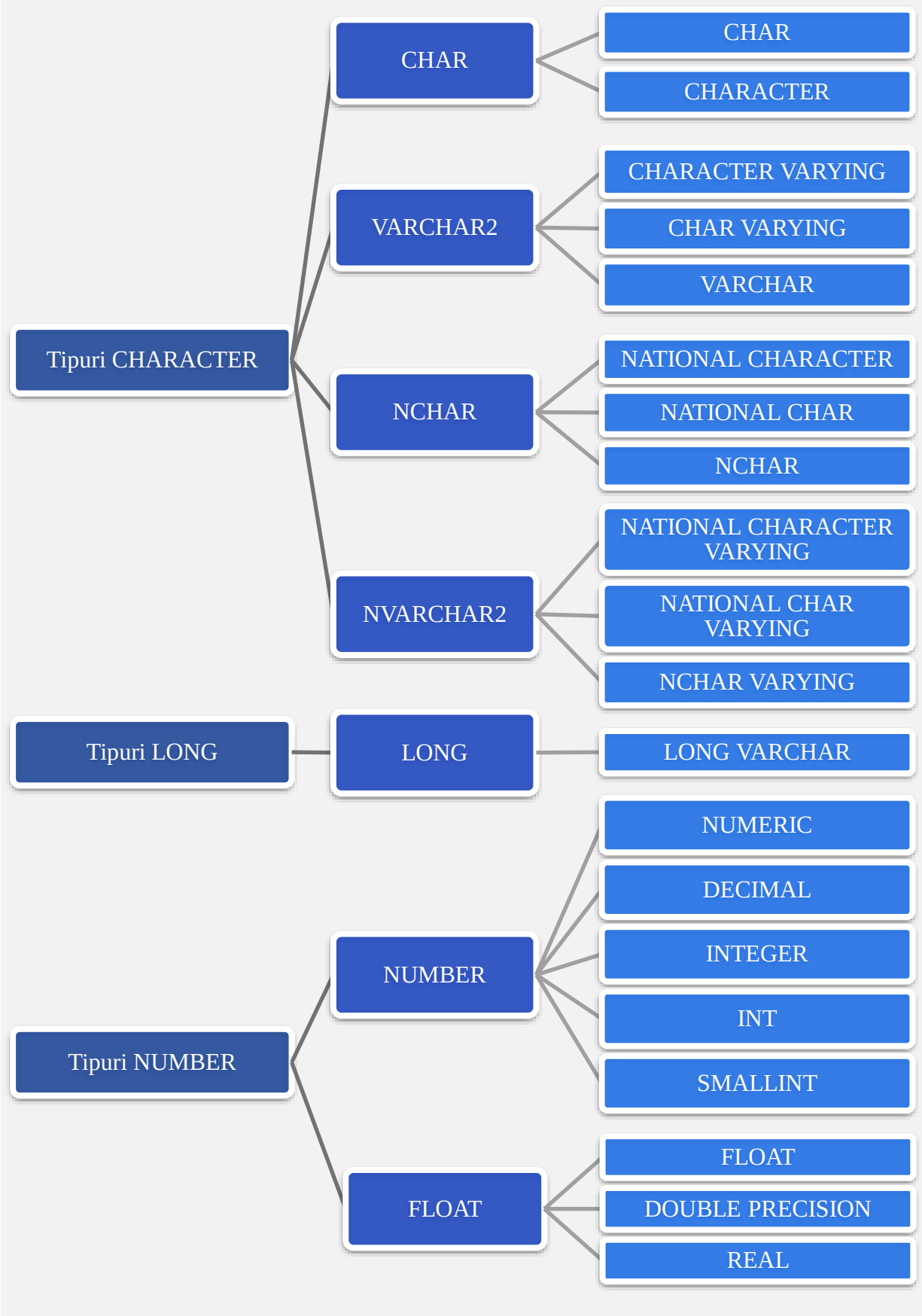


**Fig. 4.6** Tipuri de date Oracle cu tipurile ANSI/IBM echivalente

## 4.1.2. Tipuri de date *PL/SQL*

### Tipul de date *BOOLEAN*

- Stochează valorile logice *true*, *false* sau valoarea *null*
- Nu are un tip *SQL* echivalent și din acest motiv nu pot fi utilizate variabile sau parametrii de tip *boolean* în:
  - comenzi *SQL*
  - funcții *SQL* predefinite
  - funcții *PL/SQL* invocate în comenzi *SQL*

### Tipul de date *PLS_INTEGER / BINARY_INTEGER*

- Tipurile de date *PLS_INTEGER* și *BINARY_INTEGER* sunt identice.
- Stochează numere întregi cu semn reprezentate pe 32 biți cu valori cuprinse între -2.147.483.648 și 2.147.483.647 ($2^{31}$ = 2.147.483.648).

Avantaje față de tipul *NUMBER* și subtipurile sale
  - necesită mai puțin spațiu de stocare
  - deoarece folosesc aritmetica mașinii operațiile cu acest tip sunt efectuate mai rapid decât operațiile cu tipurile *NUMBER* (care folosesc librării aritmetice).

### Tipul de date referință

- Are ca valoare un *pointer* care face referință către un obiect
  - *REF CURSOR* - locația din memorie (adresa) unui cursor explicit (*Informații suplimentare în cursul despre cursoare*)

### Subtipuri definite de utilizator

- Pentru a crea un subtip se utilizează comanda

```
SUBTYPE nume_subtip IS tip_de_bază [(constrângere)]
[NOT NULL];
```

  - *tip_de_bază* poate fi un tip de date scalar sau un tip definit de utilizator
  - *constrângere* se referă la precizie și scală.
- Nu se pot specifica valori implicite.

Exemplul 4.4

```
DECLARE
  SUBTYPE subtip_data IS DATE NOT NULL;
  SUBTYPE subtip_email IS CHAR(15);
  SUBTYPE subtip_descriere IS VARCHAR2(1500);
  SUBTYPE subtip_rang IS PLS_INTEGER RANGE -5..5;
  SUBTYPE subtip_test IS BOOLEAN;
  v_data subtip_data := SYSDATE;
  v_email subtip_email(10);
  v_descriere subtip_descriere;
  v_rang subtip_rang := 2;
  v_test BOOLEAN;
BEGIN
   NULL;
END;
```

## 4.1.3. Tipuri de date și subtipurile acestora

| Tip date | Subtip | Descriere |
|---|---|---|
| NUMBER | DEC \| DECIMAL \| NUMERIC | *NUMBER* cu virgulă fixă, precizie maximă 38 cifre zecimale |
| | FLOAT \| DOUBLE PRECISION | *NUMBER* cu virgulă mobilă, precizie maximă 126 cifre binare (aproximativ 38 cifre zecimale) |
| | INT \| INTEGER \| SMALLINT | Întreg, precizie maximă 38 cifre zecimale |
| | REAL | *NUMBER* cu virgulă mobilă, precizie maximă 63 cifre binare (aproximativ 18 cifre zecimale) |
| PLS_INTEGER | NATURAL | Valorile *PLS_INTEGER* nenegative |
| | NATURALN | Valorile *PLS_INTEGER* nenegative cu constrângerea *NOT NULL* |
| | POSITIVE | Valorile *PLS_INTEGER* pozitive |
| | POSITIVEN | Valorile *PLS_INTEGER* pozitive cu constrângerea *NOT NULL* |
| | SIGNTYPE | Valorile *PLS_INTEGER* -1, 0 și 1 |
| | SIMPLE_INTEGER | Valorile *PLS_INTEGER* cu constrângerea *NOT NULL* |
| CHAR | CHARACTER | Același domeniu de valori ca și *CHAR*. Este folosit din motive de compatibilitate cu tipurile *ANSI* și *IBM*. |

| | | |
|---|---|---|
| VARCHAR2 | VARCHAR \| STRING | Același domeniu de valori ca și VARCHAR2. Este folosit din motive de compatibilitate cu tipurile *ANSI* și *IBM*. |

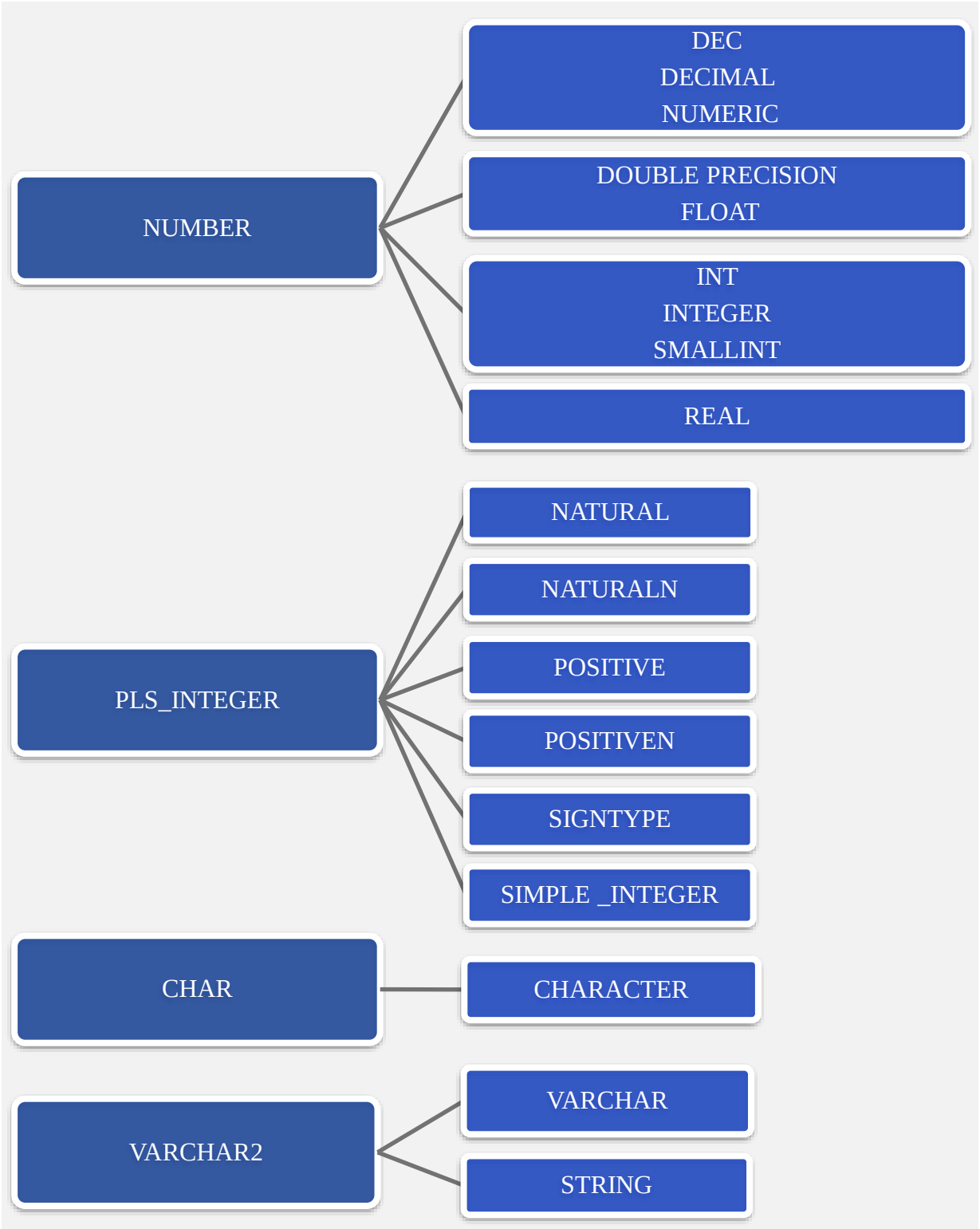


**Fig. 4.7** Tipuri de date și subtipurile lor

## 4.1.4. Conversii între tipuri de date

- Tipuri de conversii
  - implicite (realizate automat de sistem)

    *Exemple de conversii implicite*

| | DATE | NUMBER | VARCHAR2 | PLS_INTEGER |
|---|---|---|---|---|
| **DATE** | Nu se aplică | X | ✓ | X |
| **NUMBER** | X | Nu se aplică | ✓ | ✓ |
| **VARCHAR2** | ✓ | ✓ | Nu se aplică | ✓ |
| **PLS_INTEGER** | X | ✓ | ✓ | Nu se aplică |

  - explicite (realizate folosind explicit funcțiile de conversie)

    *Exemple de funcții de conversie*

    *ASCIISTR, BFILENAME, BIN_TO_NUM, CAST, CHARTOROWID, COMPOSE, CONVERT, DECOMPOSE, HEXTORAW, NUMTODSINTERVAL, NUMTOYMINTERVAL, RAWTOHEX, RAWTONHEX, REFTOHEX, ROWIDTOCHAR, ROWIDTONCHAR, SCN_TO_TIMESTAMP, TIMESTAMP_TO_SCN, TO_BINARY_DOUBLE, TO_BINARY_FLOAT, TO_CHAR, TO_CLOB, TO_DATE, TO_DSINTERVAL, TO_LOB, TO_MULTI_BYTE, TO_NCHAR, TO_NCLOB, TO_NUMBER, TO_SINGLE_BYTE, TO_TIMESTAMP, TO_TIMESTAMP_TZ, TO_YMINTERVAL, TRANSLATE USING, UNISTR*

Conversiile implicite au o serie de dezavantaje:

- pot fi lente;
- se pierde controlul asupra programului (dacă *Oracle* modifică regulile de conversie, atunci codul poate fi afectat);
- depind de mediul în care sunt utilizate (de exemplu, formatul datei calendaristice variază în funcție de setări; astfel, codul poate să nu ruleze pe *server*-e diferite);
- codul devine mai greu de înțeles.

## 4.1.5. Atributul %*TYPE*

- Este utilizat pentru a declara o variabilă cu același tip de date ca al altei variabile sau al unei coloane dintr-un tabel.

```
variabilă_2 variabilă_1%TYPE;
variabilă nume_tabel.nume_coloană%TYPE;
```

Avantaje:

- nu este necesar să se cunoască exact tipul de date al coloanei din tabel
- anumite modificări realizate asupra tipului de date al coloanei (de exemplu, se mărește dimensiunea), nu vor afecta programul

## 4.2. Tipuri de date compuse

- Un tip de date compus stochează mai multe valori care pot avea componente interne ce pot fi accesate individual.

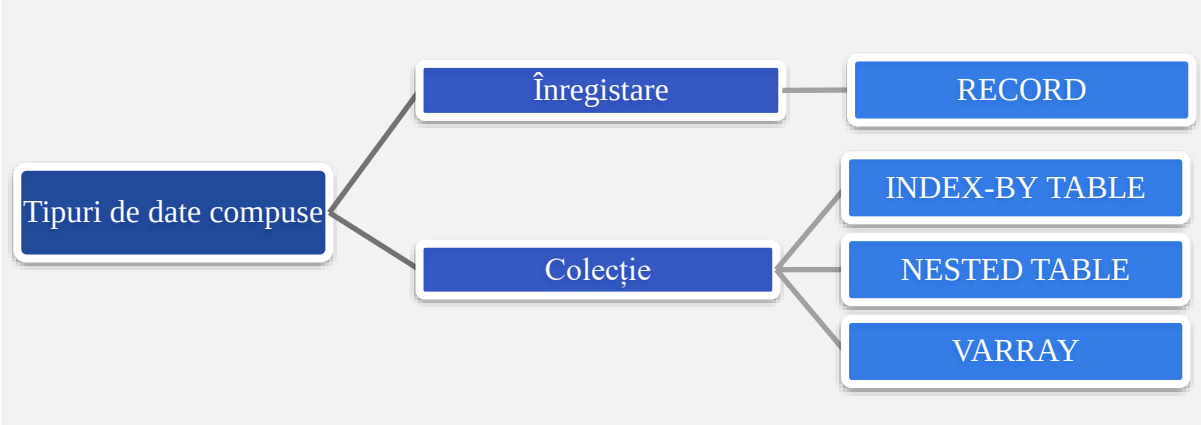


**Fig. 4.8.** Tipuri de date compuse

- Tipurile de date compuse
  - înregistrare (*RECORD*)
    - componentele interne pot avea tipuri de date diferite și sunt denumite câmpuri
  - colecție (*INDEX-BY TABLE*, *NESTED TABLE*, *VARRAY*)
    - componentele interne au același tip de date și sunt denumite elemente
    - fiecare element poate fi accesat folosind indexul său

### 4.2.1. 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.

```
variabilă_2 variabilă_1%ROWTYPE;
variabilă nume_tabel%ROWTYPE;
variabilă nume_cursor%ROWTYPE;
```

## 4.2.2. Tipul de date înregistrare

- Înregistrările se definesc în doi pași:
  - se definește un tip *RECORD*;
  - se declară variabile de acest tip.

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

variabilă nume_tip;
```

- Câmpurile unei înregistrări
  - Au implicit valoarea *null*.
  - Numărul lor nu este limitat.
  - Se referă prin prefixare cu numele înregistrării.
  - Pot fi tip scalar, *RECORD*, obiect, colecție.
  - Nu pot fi de tip *REF CURSOR*.
- Atribuirea de valori unei înregistrări se poate realiza cu
  - instrucțiunea de atribuire
  - comenzile *SELECT* sau *FETCH*
- Înregistrările
  - nu pot fi comparate (egalitate, inegalitate sau *null*).
  - pot fi parametri în subprograme.
  - pot să apară în clauza *RETURN* a unei funcții.
- Folosind direct numele înregistrării (fără a accesa individual câmpurile) se poate:
  - insera o linie în tabel (*INSERT*);
  - actualiza o linie (*UPDATE …SET ROW*);
  - capta o linie inserată, modificată sau ștearsă (*RETURNING*);
  - regăsi o linie (*SELECT … INTO*).

Tipul *RECORD* nu poate fi definit decât local (într-un bloc *PL/SQL* sau pachet).

Exemplul 4.5

```
DECLARE
  TYPE rec IS RECORD
        (id   categorii.id_categorie%TYPE,
         den  categorii.denumire%TYPE,
         niv  categorii.nivel%TYPE);
  v_categ  rec;
  v_categ2 rec;
BEGIN
  v_categ.den := 'Categorie noua';
  v_categ.niv :=1;
  SELECT MAX(id_categorie)+1 INTO v_categ.id
  FROM   categorii;

  -- eroare
  -- INSERT INTO categorii(id_categorie, denumire, nivel)
  -- VALUES v_categ;
  INSERT INTO categorii(id_categorie, denumire, nivel)
  VALUES (v_categ.id, v_categ.den, v_categ.niv);

  SELECT id_categorie, denumire, nivel INTO v_categ2
  FROM   categorii
  WHERE  id_categorie= v_categ.id;

  DBMS_OUTPUT.PUT_LINE ('Ati inserat: '|| v_categ2.id ||
    ' ' || v_categ2.den || '  '|| v_categ2.niv);
END;
```

Exemplul 4.6 **– vezi curs**

### 4.2.3. Tipul de date colecție

- Există 3 tipuri de colecții:
  - tablouri indexate (*index-by tables*), care sunt denumite și vectori asociativi (*associative arrays*)
    - sunt similare cu tabelele de dispersie (*hash tables*) din alte limbaje de programare
  - tablouri imbricate (*nested tables*)
    - sunt similare cu mulțimile (*sets*, *multisets*) din alte limbaje de programare
  - vectori cu dimensiune variabilă (*varrays*, prescurtare de la *variable-size arrays*)
    - sunt similari cu vectorii din alte limbaje de programare
    - din motive de simplificare vor fi referiți în continuare ca *vectori*
- Declararea unei colecții se realizează în 2 pași:
  - se definește un tip colecție
  - se declară o variabilă de acel tip

- Caracteristicile tipurilor colecție

| Tip colecție | Număr maxim elemente | Tip index | Dens sau Împrăștiat | Loc definire |
|---|---|---|---|---|
| **Tablouri indexate** | nefixat | întreg ∈ [-2147483648, 2147483647] sau șir de caractere $2^{31}-1 = 2147483647$ | ambele | doar în blocuri PL/SQL |
| **Tablouri imbricate** | nefixat | întreg ∈ [1,2147483647] | inițial dense, dar pot deveni împrăștiate | în blocuri PL/SQL sau la nivel de schemă |
| **Vectori** | fixat (n dat) | întreg ∈ [1,n] | dense | în blocuri PL/SQL sau la nivel de schemă |

- Metodele asociate colecțiilor
  - sunt subprograme *PL/SQL* predefinite (funcții sau proceduri)
  - întorc informații despre o colecție sau operează asupra acesteia
  - pot fi apelate numai din comenzi procedurale (nu pot fi apelate în comenzi *SQL*)
  - pot fi invocate folosind forma următoare

```
nume_colecție.nume_metodă[(parametri)]
```

- Metodele disponibile pentru colecții sunt date în tabelul următor
  - Notațiile utilizate
    - *Tab ind* – tablou indexat
    - *Tab imb* – tablou imbricat
    - *Vec* – vector

| Metodă | Descriere | Validitate | | |
|---|---|---|---|---|
| | | **Tab ind** | **Tab imb** | **Vec** |
| **COUNT** | Întoarce numărul curent de elemente | ✓ | ✓ | ✓ |
| **DELETE** | Șterge toate elementele | ✓ | ✓ | ✓ |
| **DELETE(n)** | Șterge elementul *n* | ✓ | ✓ | |
| **DELETE(n,m)** | Șterge toate elementele care au indexul cuprins între *n* și *m* | ✓ | ✓ | |

| | | | | |
|---|---|---|---|---|
| **EXISTS(n)** | Întoarce *TRUE* dacă există al *n*-lea element, altfel întoarce *FALSE* (în locul excepției *SUBSCRIPT_OUTSIDE_LIMIT*) | ✓ | ✓ | ✓ |
| **FIRST** | Întoarce indexul primului element (cel mai mic index) | ✓ | ✓ | ✓ |
| **LAST** | Întoarce indexul ultimului element (cel mai mare index) | ✓ | ✓ | ✓ |
| **NEXT(n)** | Întoarce indexul elementului care urmează după elementul cu indexul *n*. Dacă nu există, întoarce *null*. | ✓ | ✓ | ✓ |
| **PRIOR(n)** | Întoarce indexul elementului care precede elementul cu indexul *n*. Dacă nu există, întoarce *null*. | ✓ | ✓ | ✓ |
| **EXTEND** | Adaugă un element *null* la sfârșit | | ✓ | ✓ |
| **EXTEND(n)** | Adaugă *n* elemente *null* la sfârșit | | ✓ | ✓ |
| **EXTEND(n,i)** | Adaugă *n* copii ale elementului de rang *i* la sfârșit | | ✓ | ✓ |
| **LIMIT** | Întoarce numărul maxim de elemente specificat la declarare în cazul vectorilor, respectiv valoarea *null* în cazul tablourilor imbricate | | ✓ | ✓ |
| **TRIM** | Șterge ultimul element | | ✓ | ✓ |
| **TRIM(n)** | Șterge ultimele *n* elemente. Dacă *n* este mai mare decât numărul curent de elemente, atunci apare excepția *SUBSCRIPT_BEYOND_COUNT* | | ✓ | ✓ |

- *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*.

## 4.2.4. Tablouri indexate

- Sunt mulțimi de perechi cheie-valoare, în care fiecare cheie este unică și utilizată pentru a putea localiza valoarea asociată.
- Atunci când este creat un tablou indexat care nu are încă elemente, acesta este vid. Nu este inițializat automat (atomic) *null*, ca în cazul celorlalte tipuri de colecții.
- Atunci când o valoare este asociată pentru prima oară unei chei, cheia este adăugată în tablou.

- Sintaxă declarare tip

```
TYPE nume_tip IS TABLE OF tip_element [NOT NULL]
   [INDEX BY {   PLS_INTEGER
               | BINARY_INTEGER
               | VARCHAR2(n)
             }
   ];

unde tip_element poate fi orice tip PL/SQL mai puțin REF CURSOR

{ nume_cursor%ROWTYPE
  | nume_tabel{%ROWTYPE | .nume_coloană%TYPE}
  | nume_obiect%TYPE
  | [REF] nume_tip_obiect
  | nume_record[.nume_câmp]%TYPE
  | nume_tip_record
  | nume_tip_date_scalar
  | variabilă%TYPE
}
```

Pentru indexare se pot utiliza și subtipurile *VARCHAR*, *STRING* sau *LONG*.

Tablourile indexate folosesc spațiu de stocare temporar.

Pentru a deveni persistente pe perioada sesiunii trebuie declarate într-un pachet (atât tipul, cât și variabilele de acel tip), iar valorile elementelor trebuie asignate în corpul pachetului.

Tablourile indexate

- ❖ nu au constrângeri legate de dimensiune, deci dimensiunea acestora se modifică dinamic
- ❖ nu sunt inițializate la declarare
- ❖ neinițializate sunt vide (nu au chei sau valori)
- ❖ au elemente definite doar dacă acestor elemente li se atribuie valori (dacă se încearcă utilizarea unui element căreia nu i s-a atribuit nicio valoare, se declanșează excepția *NO_DATA_FOUND*)
- ❖ permit inserarea de elemente cu chei arbitrare (nu într-o ordine prestabilită)
- ❖ nu au memorie restricționată relativă la numărul de elemente, ci la dimensiunea de memorie utilizată
- ❖ pot să apară ca parametrii în proceduri.

Exemplul 4.7

```
DECLARE
  TYPE tab_ind IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
  t    tab_ind;
BEGIN
  -- atribuire valori
  FOR i IN 1..10 LOOP
    t(i):=i;
  END LOOP;
  --parcurgere
  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;

  -- numar elemente
  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;

  -- stergere elemente
  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;

  t.DELETE;
  DBMS_OUTPUT.PUT_LINE('Tabloul are ' || t.COUNT
     ||' elemente.');
END;
```

Exemplul 4.8

```
DECLARE
  TYPE tab_ind IS TABLE OF produse%ROWTYPE
       INDEX BY PLS_INTEGER;
  t    tab_ind;
BEGIN
  -- atribuire valori
SELECT * BULK COLLECT INTO t
FROM   produse
WHERE  ROWNUM<=10;

  --parcurgere
  DBMS_OUTPUT.PUT_LINE('Tabloul are ' || t.COUNT ||
           ' elemente: ');
  FOR i IN t.FIRST..t.LAST LOOP
      DBMS_OUTPUT.PUT_LINE(t(i).id_produs || ' '||
            t(i).denumire);
  END LOOP;
END;
```

Exemplul 4.9 **– vezi curs**

Exemplul 4.10 **– vezi curs**

Exemplul 4.11 **– vezi curs**

### 4.2.5. Tablouri imbricate

- În baza de date, tabloul imbricat este tip al unei coloane care stochează un număr nespecificat de linii, în nicio ordine particulară (stochează o mulțime de valori).
  - Atunci când tabloul imbricat din baza de date este preluat de o variabilă *PL/SQL*, sistemul atribuie liniilor indecși consecutivi (începând cu valoarea 1). Astfel, se permite accesarea liniilor folosind indecșii, în mod asemănător cu vectorii.
  - Indecșii și ordinea liniilor dintr-un tablou imbricat ar putea să nu rămână stabile în timp ce tabloul este stocat sau regăsit din baza de date.
- Numărul maxim de linii este dat de capacitatea maximă 2 GB.
- Inițial, tabloul este dens, dar în urma prelucrării este posibil să nu mai aibă indici consecutivi.

- Tablourile imbricate:
  - pot fi stocate în baza de date;
  - pot fi prelucrate direct în instrucțiuni *SQL*;
  - trebuie inițializate și extinse pentru a li se adăuga elemente.
- Sintaxă declarare tip

```
[CREATE [OR REPLACE]] TYPE nume_tip
IS TABLE OF tip_element [NOT NULL];
```

```
tip_element poate fi orice tip PL/SQL mai puțin REF CURSOR

{ nume_cursor%ROWTYPE
  | nume_tabel{%ROWTYPE | .nume_coloană%TYPE}
  | nume_obiect%TYPE
  | [REF] nume_tip_obiect
  | nume_record[.nume_câmp]%TYPE
  | nume_tip_record
  | nume_tip_date_scalar
  | variabilă%TYPE
}
```

- Un tablou imbricat/vector declarat, dar neinițializat, este automat inițializat (atomic) *null*.
  - Astfel, pentru verificare poate fi utilizat operatorul *IS NULL*.
  - Dacă se încearcă să se adauge un element într-un tablou imbricat/vector neinițializat (atomic *null*), se declanșează eroarea „*ORA-06531: reference to uninitialized collection*“ care corespunde excepției predefinite *COLLECTION_IS_NULL*.
- Inițializarea se realizează cu ajutorul unui constructor.
  - tabelele indexate nu au constructori
- Constructorul unei colecții
  - este o funcție sistem predefinită, cu același nume ca și numele tipului colecție referite
  - întoarce o colecție de acel tip
  - se invocă folosind sintaxa

    `nume_tip_colecție([valoare [, valoare] … ]);`
  - dacă pentru parametrii nu sunt specificate valori, atunci întoarce o colecție vidă (nu are elemente, dar nu este atomic *null*); altfel, întoarce o colecție care conține valorile specificate

- Dimensiunea inițială a colecției este egală cu numărul de valori specificate în constructor la inițializare.

Exemplul 4.12_a

```
DECLARE
  TYPE tab_imb IS TABLE OF NUMBER;
  t    tab_imb := tab_imb();
BEGIN
  -- atribuire valori
  FOR i IN 1..10 LOOP
    t.EXTEND;
    t(i):=i;
  END LOOP;
  --parcurgere
  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;

  -- numar elemente
  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;

  -- stergere elemente
  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;

  t.DELETE;
  DBMS_OUTPUT.PUT_LINE('Tabloul are ' || t.COUNT
     ||' elemente.');
END;
```

**Exemplul 4.12_b** **– vezi curs**

- Definirea tabelelor cu coloane de tip tablou imbricat presupune
  - definirea unui tip tablou imbricat

    ```
    CREATE TYPE nume_tip IS TABLE OF tip_element [NOT NULL];
    ```

  - definirea tabelului precizând pentru coloană tipul creat
    - pentru fiecare coloană de tip tablou imbricat din tabel este necesară clauza de stocare:

    ```
    NESTED TABLE nume_coloană STORE AS nume_tabel;
    ```

**Exemplul 4.13**

```
CREATE TYPE t_imb_categ IS TABLE OF VARCHAR2(40);
/

CREATE TABLE raion_grupe_imb
( id_categorie NUMBER(4) PRIMARY KEY,
  denumire     VARCHAR2(40),
  grupe        t_imb_categ)
NESTED TABLE grupe STORE AS tab_imb_grupe;

INSERT INTO raion_grupe_imb
VALUES (1, 'r1', t_imb_categ('r11','r12'));

INSERT INTO raion_grupe_imb
VALUES (2, 'r2', t_imb_categ('r21'));
INSERT INTO raion_grupe_imb(id_categorie, denumire)
VALUES (3,'r3');

UPDATE raion_grupe_imb
SET    grupe = t_imb_categ('r31','r32')
WHERE  id_categorie =3;

SELECT * FROM raion_grupe_imb;

SELECT id_categorie, denumire, b.*
FROM   raion_grupe_imb  a, TABLE(a.grupe) b;

SELECT grupe
FROM   raion_grupe_imb
WHERE  id_categorie = 1;

SELECT *
FROM   TABLE(SELECT grupe
             FROM raion_grupe_imb
             WHERE id_categorie=1);
```

**Exemplul 4.14** **– vezi curs**

## 4.2.6. Vectori

- Se utilizează în special pentru modelarea relațiilor *one-to-many*, atunci când numărul maxim de elemente *copil* este cunoscut și ordinea elementelor este importantă.
- Reprezintă structuri dense.
  - Fiecare element are un index care precizează poziția sa în vector (primul index are valoarea 1).
  - Indexul este utilizat pentru accesarea elementelor din vector.
- Vectorii:
  - față de tablourile imbricate au o dimensiune maximă specificată la declarare;
  - pot fi stocați în baza de date;
  - pot fi prelucrați direct în instrucțiuni *SQL*;
  - trebuie inițializați și extinși pentru a li se adăuga elemente.
- Sintaxă declarare tip

```
[CREATE [OR REPLACE]] TYPE nume_tip
IS {VARRAY | VARYING ARRAY}(lungime_maximă) OF tip_element
[NOT NULL];
```

```
tip_element poate fi orice tip PL/SQL mai puțin REF CURSOR

{ nume_cursor%ROWTYPE
  | nume_tabel{%ROWTYPE | .nume_coloană%TYPE}
  | nume_obiect%TYPE
  | [REF] nume_tip_obiect
  | nume_record[.nume_câmp]%TYPE
  | nume_tip_record
  | nume_tip_date_scalar
  | variabilă%TYPE
}
```

**Exemplul 4.15**

```
DECLARE
  TYPE tab_vec IS VARRAY(10) OF NUMBER;
  t    tab_vec := tab_vec();
BEGIN
  -- atribuire valori
  FOR i IN 1..10 LOOP
    t.EXTEND;
    t(i):=i;
  END LOOP;
  --parcurgere
  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;

  -- numar elemente
  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;
  -- stergere elemente
  t.DELETE;
  DBMS_OUTPUT.PUT_LINE('Tabloul are ' || t.COUNT
     ||' elemente.');
END;
```

Exemplul 4.16

```
CREATE TYPE t_vect_categ IS VARRAY(10) OF VARCHAR2(40);
/

CREATE TABLE raion_grupe_vect
( id_categorie NUMBER(4) PRIMARY KEY,
  denumire     VARCHAR2(40),
  grupe        t_vect_categ);

INSERT INTO raion_grupe_vect
VALUES (1, 'r1', t_vect_categ('r11','r12'));

INSERT INTO raion_grupe_vect
VALUES (2, 'r2', t_vect_categ('r21'));

INSERT INTO raion_grupe_vect (id_categorie, denumire)
VALUES (3,'r3');

UPDATE raion_grupe_vect
SET    grupe = t_vect_categ('r31','r32')
WHERE  id_categorie =3;

SELECT * FROM raion_grupe_vect;

SELECT id_categorie, denumire, b.*
FROM   raion_grupe_vect  a, TABLE(a.grupe) b;

SELECT grupe
FROM   raion_grupe_vect
WHERE  id_categorie = 1;

SELECT *
FROM   TABLE(SELECT grupe
             FROM raion_grupe_vect
             WHERE id_categorie=1);
```

## 4.2.7. Colecții pe mai multe niveluri

- O colecție are o singură dimensiune. Pentru a modela o colecție multidimensională se definește o colecție ale cărei elemente sunt direct sau indirect colecții (*multilevel collections*).
  - Numărul nivelurilor de imbricare este limitat doar de capacitatea de stocare a sistemului.
- Colecții pe mai multe niveluri permise:
  - vectori de vectori;
  - vectori de tablouri imbricate;
  - tablouri imbricate de tablouri imbricate;
  - tablouri imbricate de vectori;
  - tablouri imbricate sau vectori de un tip definit de utilizator care are un atribut de tip tablou imbricat sau vector.
- Pot fi utilizate ca tipuri de date pentru definirea:
  - coloanelor unui tabel relațional;
  - variabilelor *PL/SQL*;
  - atributelor unui obiect într-un tabel obiect.

Exemplul 4.17

```
DECLARE
  type t_linie is VARRAY(3) OF INTEGER;
  type matrice IS VARRAY(3) OF t_linie;
  v_linie t_linie := t_linie(4,5,6);
  a    matrice := matrice(t_linie(1,2,3), v_linie);
BEGIN
  -- se adauga un element de tip vector matricei a (linie noua)
  a.EXTEND;
  -- se adauga valori elementului nou
  a(3) := t_linie(7,8);
  -- se extinde elementul nou
  a(3).EXTEND;
  -- se adauga valoare elementului nou
  a(3)(3) := 9;

 FOR i IN 1..3 LOOP
    FOR j IN 1..3 LOOP
      DBMS_OUTPUT.PUT(a(i)(j)||' ');
    END LOOP;
    DBMS_OUTPUT.NEW_LINE;
  END LOOP;
 END;
```

## 4.2.8. Compararea colecțiilor

- Două variabile de tip colecție nu pot fi comparate nativ, utilizând operatorii relaționali (< , <=, =, <>, >=, >).
  - De exemplu, pentru a determina dacă o variabilă de tip colecție este mai mică decât alta se poate defini o funcție *PL/SQL* și utiliza în locul operatorului „<”.
- Variabilele de tip tablou indexat
  - nu pot fi comparate între ele sau cu valoarea *null*
- Variabilele de tip vector
  - pot fi comparate cu valoarea *null*
- Variabilele de tip tablou imbricat
  - pot fi comparate cu valoarea *null*
  - pot fi comparate între ele (doar dacă sunt egale sau diferite) utilizând funcții și operatori *SQL multiset*
- Funcții și operatori *SQL multiset*:
  - Funcția *CARDINALITY*

    ```
    CARDINALITY(tablou_imbricat)
    ```

    - Întoarce numărul de elemente al unui tablou imbricat.
    - Dacă tabloul este *null* sau nu are elemente, atunci întoarce *null*.
  - Funcția *SET*

    ```
    SET(tablou_imbricat)
    ```

    - Întoarce un tablou imbricat (de același tip cu argumentul său) în care păstrează doar elementele distincte (elimină duplicatele).
  - Operatorul *MULTISET EXCEPT*

    ```
    tablou_imbricat_1
    MULTISET EXCEPT [ ALL | DISTINCT ]
    tablou_imbricat_2
    ```

    - Întoarce un tablou imbricat ale cărui elemente sunt în *tablou_imbricat_1*, dar nu și în *tablou_imbricat_2*.
    - Cele două tablouri trebuie să aibă același tip.
  - Operatorul *MULTISET UNION*

    ```
    tablou_imbricat_1
    MULTISET UNION [ ALL | DISTINCT ]
    tablou_imbricat_2
    ```

    - Întoarce un tablou imbricat ale cărui elemente apar în *tablou_imbricat_1* sau în *tablou_imbricat_2*.
    - Cele două tablouri trebuie să aibă același tip.

- Operatorul *MULTISET INTERSECT*

```
tablou_imbricat_1
MULTISET INTERSECT [ ALL | DISTINCT ]
tablou_imbricat_2
```

  - Întoarce un tablou imbricat ale cărui elemente apar atât în *tablou_imbricat_1*, cât și în *tablou_imbricat_2*.
  - Cele două tablouri trebuie să aibă același tip.
- Alți operatori:
  - =, <>
  - IN, NOT IN
  - IS [NOT] A SET
  - IS [NOT] EMPTY
  - MEMBER OF
  - [NOT] SUBMULTISET OF

`Exemplul 4.18` **– vezi curs**

### 4.2.9. Prelucrarea colecțiilor stocate în tabele

- O colecție poate fi exploatată fie în întregime (atomic) utilizând comenzi *LMD*, fie pot fi prelucrate elemente individuale dintr-o colecție (*piecewise updates*) utilizând funcții/operatori *SQL* sau anumite facilități oferite de *PL/SQL*.
- Așa cum s-a observat în exemplele anterioare, se poate utiliza:
  - comanda *INSERT* pentru a insera o colecție într-o linie a unui tabel;.
  - comanda *UPDATE* pentru a modifica o colecție stocată într-un tabel;
  - comanda *DELETE* pentru a șterge o linie a unui tabel ce conține o colecție;
  - comanda *SELECT* pentru a afișa sau a regăsi în variabile *PL/SQL* o colecție stocată într-un tabel.

- Vector stocat într-un tabel
  - este prelucrat ca un întreg (nu pot fi modificate elemente individuale)
  - elementele individuale nu pot fi referite de comenzile *INSERT*, *UPDATE*, *DELETE*
  - modificarea unui element individual se poate realiza doar din *PL/SQL*
    - se selectează vectorul într-o variabilă *PL/SQL*
    - se modifică valoarea variabilei
    - se inserează înapoi în tabel

- Tablou imbricat stocat într-un tabel
  - poate fi prelucrat ca întreg
    - inserări și actualizări asupra întregii colecții
  - poate fi prelucrat la nivel de elemente individuale
    - inserarea unor elemente noi în colecție
    - ștergerea unor elemente din colecție
    - actualizarea elementelor din colecție
- Pentru a putea prelucra elementele individuale ale unui tablou imbricat stocat într-un tabel se utilizează funcția *TABLE*.
- Funcția *TABLE*
  - Poate fi aplicată:
    - unei colecții
    - unei subcereri referitoare la o colecție (lista *SELECT* din subcerere trebuie să conțină o singură coloană de tip colecție și să întoarcă o singură linie din tabel).
  - Dacă este utilizată în clauză *FROM*, atunci permite interogarea colecției în mod asemănător unui tabel (exemplele 4.13 și 4.16) .

Exemplul 4.19 – continuare exemplu 4.13

```
-- selectie elemente colectie
SELECT *
FROM TABLE (SELECT grupe
            FROM   raion_grupe_imb
            WHERE  id_categorie = 1);

--adaugare element in colectie
INSERT INTO TABLE (SELECT grupe
                   FROM   raion_grupe_imb
                   WHERE  id_categorie = 1)
VALUES ('r13');

-- adaugare elemente obtinute cu subcerere
INSERT INTO TABLE (SELECT grupe
                   FROM   raion_grupe_imb
                   WHERE  id_categorie = 1)
SELECT denumire
FROM   categorii
WHERE  id_parinte = 1;

-- modificare element colectie
UPDATE TABLE (SELECT grupe
              FROM   raion_grupe_imb
              WHERE  id_categorie = 1) a
SET VALUE(a) =  'r1333'
WHERE COLUMN_VALUE = 'r13';
```

```
--stergere element colectie
DELETE FROM TABLE (SELECT grupe
                   FROM raion_grupe_imb
                   WHERE id_categorie = 1) a
WHERE COLUMN_VALUE = 'r1333';
```

- Pentru prelucrarea unei colecții locale se poate folosi și funcția *CAST*.
- Funcția *CAST*
  - Convertește o colecție locală la tipul colecție specificat.
  - Sintaxa

    ```
    CAST({nume_colecție | MULTISET (subcerere) }
         AS tip_colecție)
    ```

    - *nume_colecție* este o colecție declarată local (de exemplu, într-un bloc *PL/SQL*)
    - *subcerere* este o cerere *SQL* al cărui rezultat este transformat în colecție
    - *tip_colecție* este un tip colecție *SQL*
- Funcția *COLLECT*
  - Are ca argument o coloană de orice tip și întoarce un tablou imbricat format din liniile selectate.

    ```
    COLLECT(coloană)
    ```
  - Trebuie utilizată împreună cu funcția *CAST*.

Exemplul 4.20

```
SELECT CAST (COLLECT(denumire) AS t_imb_categ)
FROM   categorii
WHERE  id_parinte = 1;

SELECT CAST(MULTISET(SELECT denumire
                     FROM   categorii
                     WHERE  id_parinte=1)
            AS t_imb_categ)
FROM  DUAL;
```

## 4.2.10. Procedeul *bulk collect*

- Execuția comenzilor *SQL* specificate în programe determină comutări ale controlului între motorul *PL/SQL* și motorul *SQL*. Prea multe astfel de schimbări de context afectează performanța.

- Pentru a reduce numărul de comutări între cele două motoare se utilizează procedeul *bulk collect*, care permite transferul liniilor între *SQL* și *PL/SQL* prin intermediul colecțiilor.
- Procedeul *bulk collect* implică doar două comutări între cele două motoare:
  - motorul *PL/SQL* comunică motorului *SQL* să obțină mai multe linii odată și să le plaseze într-o colecție;
  - motorul *SQL* regăsește toate liniile și le încarcă în colecție, după care predă controlul motorului *PL/SQL*.
- Sintaxa:

```
comandă_clauză BULK COLLECT INTO nume_colecție
                                 [,nume_colecție]…
```

unde *comandă_clauză* poate fi

  - comanda *SELECT* (cursoare implicite);
  - comanda *FETCH* (cursoare explicite);
  - clauza *RETURNING* a comenzilor *INSERT*, *UPDATE*, *DELETE*.

Exemplul 4.21

```
DECLARE
   TYPE t_ind IS TABLE OF categorii.id_categorie%TYPE
         INDEX BY PLS_INTEGER;
   TYPE t_imb IS TABLE OF categorii.denumire%TYPE;
   TYPE t_vec IS VARRAY(10) OF categorii.nivel%TYPE;
   v_ind t_ind;
   v_imb t_imb;
   v_vec t_vec;

BEGIN
   SELECT id_categorie, denumire, nivel
   BULK COLLECT INTO v_ind, v_imb, v_vec
   FROM  categorii
   WHERE ROWNUM <= 10;

   FOR i IN 1..10 LOOP
     DBMS_OUTPUT.PUT(v_ind(i) || ' ' || v_imb(i) ||
                     ' ' ||v_vec(i));
     DBMS_OUTPUT.NEW_LINE;
  END LOOP;
END;
```

## 4.2.11. Procedeul *bulk bind*

- În exemplul următor datele menținute într-o colecție sunt inserate în tabel.
  - Colecția este parcursă folosind comanda *FOR*.
  - La fiecare iterație o comandă *INSERT* este transmisă motorului *SQL*.

Exemplul 4.22

```
DECLARE
  TYPE tab_ind IS TABLE OF tip_plata%ROWTYPE
       INDEX BY PLS_INTEGER;
  t    tab_ind;
BEGIN
  -- atribuire valori
   DELETE FROM tip_plata
   WHERE  id_tip_plata NOT IN (SELECT id_tip_plata
                               FROM facturi)
   RETURNING id_tip_plata, cod, descriere BULK COLLECT INTO t;

  -- insert in tabel
  FOR i IN t.FIRST..t.LAST LOOP
      INSERT INTO tip_plata VALUES t(i);
  END LOOP;
END;
```

- Procedeul *bulk bind* permite transferul liniilor din colecție printr-o singură operație.
  - Este realizat cu ajutorul comenzii *FORALL*

```
FORALL index IN lim_inf..lim_sup [SAVE EXCEPTIONS]
  comandă_sql;
```

    - *comandă_sql* poate fi o comandă *INSERT*, *UPDATE* sau *DELETE* care referă elementele unei colecții (de orice tip)
    - variabila *index* poate fi referită numai ca indice de colecție
  - Restricții de utilizare a comenzii *FORALL*
    - comanda poate fi folosită numai în programe *server-side*
    - *comandă_sql* trebuie să refere cel puțin o colecție
    - toate elementele colecției din domeniul precizat trebuie să existe
    - indicii colecțiilor nu pot fi expresii și trebuie să aibă valori continue

Exemplul 4.23

```
DECLARE
  TYPE tab_ind IS TABLE OF tip_plata%ROWTYPE
       INDEX BY PLS_INTEGER;
  t    tab_ind;
BEGIN
  -- atribuire valori
   DELETE FROM tip_plata
   WHERE  id_tip_plata NOT IN (SELECT id_tip_plata
                               FROM facturi)
   RETURNING id_tip_plata, cod, descriere BULK COLLECT INTO t;

  -- insert in tabel
  FORALL i IN t.FIRST..t.LAST
      INSERT INTO tip_plata VALUES t(i);
END;
```

- Cursorul *SQL* are un atribut compus %BULK_ROWCOUNT care numără liniile afectate de iterațiile comenzii *FORALL*.
  - SQL%BULK_ROWCOUNT(i) reprezintă numărul de linii procesate de a i-a execuție a comenzii *SQL*.
  - Dacă nu este afectată nici o linie, valoarea atributului este 0.
  - %BULK_ROWCOUNT nu poate fi parametru în subprograme și nu poate fi asignat altor colecții.

Exemplul 4.24

```
CREATE TABLE produse_copie AS SELECT * FROM PRODUSE;

DECLARE
   TYPE tip_vec IS VARRAY(3) OF NUMBER(4);
   v tip_vec := tip_vec(800, 900, 1000);
BEGIN
   FORALL i IN 1..3
      DELETE FROM produse_copie
      WHERE id_categorie = v(i);

  FOR j in 1..v.LAST LOOP
  DBMS_OUTPUT.PUT_LINE( 'Numar linii procesate la pasul ' ||
        j || ': ' || SQL%BULK_ROWCOUNT(j));
  END LOOP;
END;
/
ROLLBACK;
```

- Dacă există o eroare în procesarea unei linii printr-o operație *LMD* de tip *bulk*, numai acea linie va fi *rollback*.

- Clauza *SAVE EXCEPTIONS*, permite ca toate excepțiile care apar în timpul execuției comenzii *FORALL* să fie salvate și astfel procesarea poate să continue.
  - Atributul cursor %BULK_EXCEPTIONS poate fi utilizat pentru a vizualiza informații despre aceste excepții.
  - Atributul acționează ca un tablou *PL/SQL* și are două câmpuri:
    - *%BULK_EXCEPTIONS(i).ERROR_INDEX*, reprezentând iterația în timpul căreia s-a declanșat excepția;
    - *%BULK_EXCEPTIONS(i).ERROR_CODE*, reprezentând codul *Oracle* al erorii respective.

`Exemplul 4.25` **`– vezi curs`**

## 4.3. Vizualizări din dicționarul datelor

- Vizualizări care conțin informații despre tipurile de date create de utilizatori:
  - *USER_TYPES*
  - *USER_TYPE_ATTRS*

# Bibliografie


1. Connolly T.M., Begg C.E., Database Systems: *A Practical Approach to Design, Implementation and Management*, 5th edition, Pearson Education, 2005
2. Dollinger R., Andron L., *Baze de date şi gestiunea tranzacţiilor*, Editura Albastră, Cluj-Napoca, 2004
3. Oracle and/or its affiliates, *Oracle Database Concepts*, 1993, 2017
4. Oracle and/or its affiliates, *Oracle Database Performance Tuning Guide*, 2013, 2017
5. Oracle and/or its affiliates, *Oracle Database SQL Language Reference*, 1996, 2017
6. Oracle and/or its affiliates, *Oracle Database PL/SQL Language Reference*, 1996, 2017
7. Oracle and/or its affiliates, *Oracle Database Administrator's Guide*, 2001, 2010
8. Oracle and/or its affiliates, *Pro*C/C++ Programmer's Guide*, 1996, 2014
9. Oracle University, *Oracle Database 11g*: *PL/SQL Fundamentals*, *Student Guide*, 2009
10. Popescu I., Alecu A., Velcescu L., Florea (Mihai) G., *Programare avansată în Oracle9i*, Ed. Tehnică, 2004
11. Microsoft Online Documentation
    *http://msdn.microsoft.com*
12. MySQL Online Documentation
    *http://dev.mysql.com*