

Tipuri de date în *PL/SQL*

Fiecare variabilă sau constantă utilizată într-un bloc *PL/SQL* este de un anumit tip prin care se specifică:

- formatul său de stocare,
- constrângerile care trebuie să le verifice
- domeniul valorilor sale.

Variabilele folosite în *Oracle9i* pot fi:

- specifice *PL/SQL*;
- nespecifice *PL/SQL*.

Variabile specifice *PL/SQL* se clasifică în variabile:

- de tip scalar,
- compuse,
- referință,
- *LOB* (*large objects*),
- tipuri obiect.

Variabile nespecifice *PL/SQL* pot fi:

- variabile de legătură (*bind variables*),
- variabile gazdă (*host variables*),
- variabile indicator.

Variable specifice *PL/SQL*

Tipurile de date scalare

Nu au componente interne (conțin valori atomice). Se împart în 5 clase.

- Tipurile de date ce stochează **valori numerice** cuprind tipul *NUMBER* cu subtipurile *DEC*, *DECIMAL*, *DOUBLE PRECISION*, *FLOAT*, *INTEGER*, *INT*, *NUMERIC*, *REAL*, *SMALLINT*; tipul *BINARY_INTEGER* cu subtipurile *NATURAL*, *NATURALN*, *POSITIVE*, *POSITIVEN*, *SIGNTYPE*; tipul *PLS_INTEGER*.
- Tipurile de date ce stochează **caractere** cuprind tipul *VARCHAR2* cu subtipurile *STRING*, *VARCHAR*; tipul de date *CHAR* cu subtipul *CHARACTER*; tipurile *LONG*, *RAW*, *LONG RAW*, *ROWID*.
- Tipurile de date ce stochează **data calendaristică și ora** cuprind tipurile *DATE*, *TIMESTAMP*, *TIMESTAMP WITH TIME ZONE*, *TIMESTAMP WITH LOCAL TIME ZONE*, *INTERVAL YEAR TO MONTH*, *INTERVAL DAY TO SECOND*.
- Tipurile de date **globalizare** ce stochează date *unicode* includ tipurile *NCHAR* și *NVARCHAR2*.
- Tipul de date *BOOLEAN* stochează **valori logice** (*true*, *false* sau *null*).

Tipurile de date compuse

Au componente interne care pot fi manipulate individual. *Oracle* oferă programatorului două tipuri de date compuse:

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

Tipurile de date referință

REF CURSOR și *REF obiect* sunt tipuri de date ale căror valori, numite *pointeri*, fac referință către obiecte din program. Pointerii conțin locația de memorie (adresa) unui element și nu elementul în sine. Tipul *REF CURSOR* este folosit pentru a face referință la un cursor explicit. Tipul *REF obiect* face referință la adresa unui obiect.

Tipurile de date *LOB*

Large object sunt acele tipuri de date ale căror valori, numite locatori (*locators*) specifică localizarea unor obiecte de dimensiuni mari, adică blocuri de date nestructurate, cum ar fi texte, imagini grafice, clipuri video și sunete. Tipurile *LOB* sunt manipulate cu ajutorul pachetului *DBMS_LOB*.

Aceste tipuri sunt:

- *CLOB* (*character large object*),
- *BLOB* (*binary large object*),
- *BFILE* (*binary file*),
- *NCLOB* (*national language character large object*).

Tipurile obiect

Sunt tipuri compuse, definite de utilizator, care încapsulează structuri de date (atribute) împreună cu subprograme pentru manipularea datelor (metode).

Dintre tipurile scalare *PL/SQL*, următoarele sunt și tipuri *SQL* (adică pot fi folosite pentru coloanele tabelelor *Oracle*): *NUMBER*, *VARCHAR2*, *CHAR*, *LONG*, *RAW*, *LONG RAW*, *ROWID*, *NCHAR*, *NVARCHAR2*, *DATE*. În unele cazuri, tipurile de date *PL/SQL* diferă de corespondentele lor *SQL* prin dimensiunea maximă permisă.

Tipul *NUMBER* memorează numerele în virgulă fixă și virgulă mobilă. El are forma generală *NUMBER* (*m*, *n*), unde *m* reprezintă numărul total de cifre, iar *n* numărul de zecimale. Valoarea unei variabile de tip *NUMBER* este cuprinsă între 1.0E-129 și 9.99E125. Numărul de zecimale determină poziția în care apare rotunjirea. Valoarea sa este cuprinsă între -84 și 127, iar implicit este 0.

Tipul *NUMBER* are următoarele subtipuri, care au aceleași intervale de valori: *NUMERIC*, *REAL*, *DEC*, *DECIMAL* și *DOUBLE PRECISION* (pentru memorarea datelor numerice în virgulă fixă), *FLOAT* (pentru memorarea datelor numerice în virgulă mobilă), *SMALLINT*, *INTEGER* și *INT* (pentru memorarea numerelor întregi). Aceste subtipuri se pot utiliza pentru compatibilitate *ANSI/ISO*, *IBM SQL/DS* sau *IBM DB2*.

Tipul *BINARY_INTEGER* memorează numere întregi cu semn având valori cuprinse între $-2^{31} - 1$ și $2^{31} - 1$. Acest tip de date este utilizat frecvent pentru indecșii tabelelor, nu necesită conversii și admite mai multe subtipuri. De exemplu, pentru a restricționa domeniul variabilelor la valori întregi nenegative se utilizează tipurile *NATURAL* ($0 \dots 2^{31} - 1$) și *POSITIVE* ($1 \dots 2^{31} - 1$).

Tipul *PLS_INTEGER* este utilizat pentru stocarea numerelor întregi cu semn și are același interval de definire ca și tipul *BINARY_INTEGER*. Operațiile cu acest tip sunt efectuate mai rapid (folosesc aritmetica mașinii), decât cele cu tipurile *NUMBER* sau *BINARY_INTEGER* (folosesc librării aritmetice). Prin urmare, pentru o mai bună performanță, este preferabil să se utilizeze tipul *PLS_INTEGER*.

Variabilele alfanumerice pot fi de tip *CHAR*, *VARCHAR2*, *LONG*, *RAW* și *LONGRAW*. Reprezentarea internă depinde de setul de caractere ales (*ASCII* sau *EBCDIC*).

Tipurile *CHAR*, *VARCHAR2* și *RAW* pot avea un parametru pentru a preciza lungimea maximă. Dacă aceasta nu este precizată atunci, implicit, se consideră 1. Lungimea este exprimată în octeți (nu în caractere). Subtipurile acestor tipuri se pot utiliza pentru compatibilitate *ANSI/ISO*, *IBM SQL/DS* sau *IBM DB2*.

În *Oracle9i* a fost extinsă sintaxa pentru *CHAR* și *VARCHAR2*, permițând ca variabila ce precizează lungimea maximă să fie de tip *CHAR* sau *BYTE*.

Variabilele de tip *LONG* pot memora texte, tabele de caractere sau documente, prin urmare șiruri de caractere de lungime variabilă de până la 32760 octeți. Este similar tipului *VARCHAR2*.

Tipul *RAW* permite memorarea datelor binare (biți) sau a șirurilor de octeți. De exemplu, o variabilă *RAW* poate memora o secvență de caractere grafice sau o imagine digitizată. Tipul *RAW* este similar tipului alfanumeric, cu excepția faptului că *PL/SQL* nu interpretează datele de tip *RAW*. *Oracle* nu face conversia datelor de acest tip, atunci când se transmit de la un sistem la altul. Chiar dacă lungimea maximă a unei variabile *RAW* poate fi 32767 octeți, într-o coloană *RAW* a bazei de date nu se pot introduce decât 2000 octeți. Pentru a insera valori mai mari se folosește o coloană de tip *LONG RAW*, care are lungimea maximă 2^{31} octeți. *LONG RAW* este similar tipului *LONG*, dar datele nu mai sunt interpretate de *PL/SQL*.

Tipurile *TIMESTAMP*, *TIMESTAMP WITH TIME ZONE*, *TIMESTAMP WITH LOCAL TIME ZONE*, *INTERVAL YEAR TO MONTH*, *INTERVAL DAY TO SECOND* au fost introduse în *Oracle9i* și permit rafinări ale tipului *DATE*. De exemplu, *TIMESTAMP* poate lua în considerare și fracțiuni de secundă.

PL/SQL suportă două seturi de caractere: una specifică bazei de date care este utilizată pentru definirea identificatorilor și a codului sursă (*database character set* - *DCS*) și o mulțime de caractere naționale care este folosită pentru reprezentarea informației cu caracter național (*national character set* - *NCS*).

Tipurile de date *NCHAR* și *NVARCHAR2* sunt utilizate pentru stocarea în baza de date a șirurilor de caractere ce folosesc *NCS*. Ele oferă suport pentru globalizarea datelor, astfel încât utilizatorii din toată lumea pot interacționa cu *Oracle* în limba lor națională. Aceste tipuri de date suportă numai date *Unicode*.

Unicode este o mulțime de caractere globale care permite stocarea de informație în orice limbă, folosind o mulțime unică de caractere.

Prin urmare, *unicode* furnizează o valoare cod unică pentru fiecare caracter, indiferent de platformă, program sau limbă.

Variabile nespecifice *PL/SQL*

Variabila de legătură (*bind*) se declară într-un mediu gazdă și este folosită pentru transferul (la execuție) valorilor numerice sau de tip caracter în/din unul sau mai multe programe *PL/SQL*. Variabilele declarate în mediul gazdă sau în cel apelant pot fi referite în instrucțiuni *PL/SQL*, dacă acestea nu sunt în cadrul unei proceduri, funcții sau pachet.

În *SQL*Plus*, variabilele de legătură se declară folosind comanda *VARIABLE*, iar pentru afișarea valorilor acestora se utilizează comanda *PRINT*. Ele sunt referite prin prefixarea cu simbolul „:”, pentru a putea fi deosebite de variabilele declarate în *PL/SQL*.

Deoarece instrucțiunile *SQL* pot fi integrate în programe *C*, este necesar un mecanism pentru a transfera valori între mediul de programare *C* și instrucțiunile *SQL* care comunică cu *server*-ul bazei de date *Oracle*. În acest scop, în programul încapsulat sunt definite variabilele gazdă (*host*). Acestea sunt declarate între directivele *BEGIN DECLARE SECTION* și *END DECLARE SECTION* ale preprocesorului.

O valoare *null* în baza de date nu are o valoare corespunzătoare în mediul limbajului gazdă (de exemplu, limbajul *C*). Pentru a rezolva problema comunicării valorilor *null* între programul scris în limbaj gazdă și sistemul *Oracle*, au fost definite variabilele indicator. Acestea sunt variabile speciale de tip întreg, folosite pentru a indica dacă o valoare *null* este recuperată (extrasă) din baza de date sau stocată în aceasta. Ele au următoarea formă:

```
:nume_extern [: indicator]
```

De exemplu, dacă atribuirea este făcută de limbajul gazdă, valoarea *-1* a indicatorului specifică faptul că *PL/SQL* trebuie să înlocuiască valoarea variabilei prin *null*, iar o valoare a indicatorului mai mare ca zero precizează că *PL/SQL* trebuie să considere chiar valoarea variabilei.

Declararea variabilelor

Identificatorii *PL/SQL* trebuie declarați înainte de a fi referiți în blocul *PL/SQL*. Dacă în declarația unei variabile apar referiri la alte variabile, acestea trebuie să fi fost declarate anterior. Orice variabilă declarată într-un bloc este accesibilă blocurilor conținute sintactic în acesta.

Tipurile scalare sunt predefinite în pachetul *STANDARD*. Pentru a folosi un astfel de tip într-un program este suficient să fie declarată o variabilă de tipul respectiv.

Tipurile compuse sunt definite de utilizator. Prin urmare, în acest caz trebuie definit efectiv tipul și apoi declarată variabila de tipul respectiv.

În declararea variabilelor pot fi utilizate atributele *%TYPE* și *%ROWTYPE*, care reprezintă tipuri de date implicite. Aceste tipuri permit declararea unei variabile în concordanță cu declarații de variabile făcute anterior.

Atributul *%TYPE* permite definirea unei variabile având tipul unei variabile declarate anterior sau tipul unei coloane dintr-un tabel.

Atributul *%ROWTYPE* permite definirea unei variabile având tipul unei înregistrări dintr-un tabel. Avantajul utilizării acestui atribut constă în faptul că nu este necesar să se cunoască numărul și tipurile coloanelor tabelului. Elementele individuale ale acestei structuri de tip înregistrare sunt referite în maniera clasică, prefixând numele coloanei cu numele variabilei declarate.

Calitatea atributelor *%TYPE* și *%ROWTYPE* constă în faptul că simplifică întreținerea codului *PL/SQL*. De exemplu, poate fi modificată dimensiunea unei coloane, fără să fie necesară modificarea declarației variabilelor al căror tip s-a definit făcând referință la tipul coloanei respective.

Sintaxa declarării unei variabile este următoarea:

```
identificator [CONSTANT] {tip_de_date / identificator%TYPE /
                          identificator%ROWTYPE} [NOT NULL]
[ {:= / DEFAULT} expresie_PL/SQL];
```

Se pot defini constante (valoarea stocată nu poate fi modificată) prin specificarea la declarare a cuvântului cheie *CONSTANT*.

Exemplu:

```
v_valoare          NUMBER(15) NOT NULL := 0;
v_data_achizitie   DATE DEFAULT SYSDATE;
v_material         VARCHAR2(15) := 'Matase';
c_valoare          CONSTANT NUMBER := 100000;
v_stare            VARCHAR2(20) DEFAULT 'Buna';
v_clasificare      BOOLEAN DEFAULT FALSE;
v_cod_opera        opera.cod_opera%TYPE;
v_opera            opera%ROWTYPE;
int_an_luna        INTERVAL YEAR TO MONTH :=
                   INTERVAL '3-2' YEAR TO MONTH;
```

Observații:

1. Pentru a denumi o variabilă este utilizată frecvent (pentru ușurința referirii) prefixarea cu litera *v* (*v_identificator*), iar pentru o constantă este folosită prefixarea cu litera *c* (*c_identificator*).

2. Variabilele pot fi inițializate, iar dacă o variabilă nu este inițializată, valoarea implicită a acesteia este *null*. Dacă o variabilă este declarată *NOT NULL*, atunci ea va fi obligatoriu inițializată.
3. Pentru a inițializa o variabilă sau o constantă poate fi utilizată o expresie *PL/SQL* compatibilă ca tip cu variabila sau constanta respectivă.
4. Constantele trebuie inițializate când sunt declarate, altfel apare o eroare la compilare.
5. În secțiunea declarativă, pe fiecare linie, există o singură declarație de variabilă.
6. Două obiecte (variabile) pot avea același nume cu condiția să fie definite în blocuri diferite. Dacă ele coexistă, poate fi folosit doar obiectul declarat în blocul curent.
7. Atributul *%ROWTYPE* nu poate include clauze de inițializare.

Definirea subtipurilor

Subtipurile derivă dintr-un tip de bază, la care se adaugă anumite restricții. De exemplu, *NATURAL* este un subtip predefinit *PL/SQL*, derivat din tipul de bază *BINARY_INTEGER*, cu restricția că permite prelucrarea valorilor întregi nenegative.

Prin urmare, un subtip nu reprezintă un nou tip de date, ci un tip existent asupra căruia se aplică anumite constrângeri. Subtipurile presupun același set de operații ca și tipul de bază, dar aplicate unui subset de valori al acestui tip.

Sistemul *Oracle* permite ca utilizatorul să-și definească propriile sale tipuri și subtipuri de date în partea declarativă a unui bloc *PL/SQL*, subprogram sau pachet utilizând sintaxa:

SUBTYPE *nume_subtip* ***IS*** *tip_de_baza* [***NOT NULL***];

În dicționarul datelor există vizualizări care furnizează informații despre tipurile de date create de utilizator (*USER_TYPES*, *USER_TYPE_ATTRS*).

Conversii între tipuri de date

Există două tipuri de conversii:

- implicite;
- explicite.

PL/SQL face automat conversii implicite între caractere și numere sau între caractere și date calendaristice. Chiar dacă sistemul realizează automat aceste conversii, în practică se utilizează frecvent funcții de conversie explicită.

Funcțiile de conversie explicită din *SQL* sunt utilizabile și în *PL/SQL*. Acestea sunt: *TO_NUMBER*, *TO_CHAR*, *TO_DATE*, *TO_MULTI_BYTE*, *TO_SINGLE_BYTE*, *CHARTOROWID*, *ROWIDTOCHAR*, *RAWTOHEX*, *HEXTORAW*, *TO_CLOB*, *TO_LOB*.

În *Oracle9i* se pot folosi următoarele funcții de conversie: *ASCIISTR*, *BIN_TO_NUM*, *NUMTODSINTERVAL*, *TO_TIMESTAMP*, *TO_YMINTERVAL*, *TO_NCHAR*, *TO_NCLOB*, *TO_TIMESTAMP_TZ*, *NUMTOYMINTERVAL*, *TO_DSINTERVAL*, *REFTOHEX*, *RAWTOHEX*, *RAWTONHEX*, *FROM_TZ*, *ROWIDTONCHAR*, *COMPOSE*, *DECOMPOSE*.

Denumirile acestor funcții reflectă posibilitățile pe care le oferă. De exemplu, *TO_YMINTERVAL* convertește argumentele sale la tipul *INTERVAL YEAR TO MONTH* conform unui format specificat. Funcția *COMPOSE* convertește un șir de caractere la un șir *unicode* (asociază o valoare cod unică pentru fiecare simbol din șir).

Înregistrări

Tipul *RECORD* oferă un mecanism pentru prelucrarea înregistrărilor. Înregistrările au mai multe câmpuri ce pot fi de tipuri diferite, dar care sunt legate din punct de vedere logic.

Înregistrările trebuie definite în doi pași:

- se definește tipul *RECORD*;
- se declară înregistrările de acest tip.

Declararea tipului *RECORD* se face conform următoarei sintaxe:

***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],...);

Identificatorul *nume_tip* reprezintă numele tipului *RECORD* care se va specifica în declararea înregistrărilor, *nume_câmp* este numele unui câmp al înregistrării, iar *tip_câmp* este tipul de date al câmpului.

Observații:

- Dacă un câmp nu este inițializat atunci implicit se consideră că are valoarea *NULL*. Dacă s-a specificat constrângerea *NOT NULL*, atunci obligatoriu câmpul trebuie inițializat cu o valoare diferită de *NULL*.
- Pentru referirea câmpurilor individuale din înregistrare se prefixează numele câmpului cu numele înregistrării.
- Pot fi asignate valori unei înregistrări utilizând comenzile *SELECT*, *FETCH* sau instrucțiunea clasică de atribuire. De asemenea, o înregistrare poate fi asignată altei înregistrări de același tip.

- Componentele unei înregistrări pot fi de tip scalar, *RECORD*, *TABLE*, obiect, colecție (dar, nu tipul *REF CURSOR*).
- *PL/SQL* permite declararea și referirea înregistrărilor imbricate.
- Numărul de câmpuri ale unei înregistrări nu este limitat.
- Înregistrările nu pot fi comparate (egalitate, inegalitate sau *null*).
- Înregistrările pot fi parametri în subprograme și pot să apară în clauza *RETURN* a unei funcții.

Diferența dintre atributul *%ROWTYPE* și tipul de date compus *RECORD*:

- tipul *RECORD* permite specificarea tipului de date pentru câmpuri și permite declararea câmpurilor sale;
- atributul *%ROWTYPE* nu cere cunoașterea numărului și tipurilor coloanelor tabloului.

Oracle9i introduce câteva facilități legate de acest tip de date.

- Se poate insera (*INSERT*) o linie într-un tabel utilizând o înregistrare. Nu mai este necesară listarea câmpurilor individuale, ci este suficientă utilizarea numelui înregistrării.
- Se poate reactualiza (*UPDATE*) o linie a unui tabel utilizând o înregistrare. Sintaxa *SET ROW* permite să se reactualizeze întreaga linie folosind conținutul unei înregistrări.
- Într-o înregistrare 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.

Exemplu:

Exemplul următor arată modul în care poate să fie utilizată o înregistrare în clauza *RETURNING* asociată comenzii *DELETE*.

```
DECLARE
  TYPE val_opera IS RECORD (
    cheie  NUMBER,
    val    NUMBER);
  v_info_valoare  val_opera;
BEGIN
  DELETE FROM opera
    WHERE cod_opera = 753
    RETURNING cod_opera, valoare
      INTO v_info_valoare;
  ...
END;
```

Colecții

Uneori este preferabil să fie prelucrate simultan mai multe variabile de același tip. Tipurile de date care permit acest lucru sunt colecțiile. Fiecare element are un indice unic, care determină poziția sa în colecție.

Oracle7 a furnizat tipul *index-by table*, inițial numit *PL/SQL table* datorită asemănării sale cu structura tabelelor relaționale.

Oracle8 a introdus două tipuri colecție, *nested table* și *varray*. *Oracle9i* permite crearea de colecții pe mai multe niveluri, adică colecții de colecții.

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

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

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).

Exemplu:

În exemplul care urmează sunt ilustrate cele trei tipuri de colecții.

```
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;
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);
END;
```

Observații:

- Deoarece colecțiile nu pot fi comparate (egalitate sau inegalitate), ele nu pot să apară în clauzele *DISTINCT*, *GROUP BY*, *ORDER BY*.
- Tipul colecție poate fi definit într-un pachet.
- Tipul colecție poate să apară în clauza *RETURN* a unei funcții.
- Colecțiile pot fi parametri formali într-un subprogram.
- Accesul la elementele individuale ale unei colecții se face prin utilizarea unui indice.

Tablouri indexate

Tipul de date *index-by table* oferă un mecanism pentru prelucrarea tablourilor. Tabloul indexat *PL/SQL* are două componente: o coloană ce cuprinde cheia primară pentru acces la liniile tabloului și o coloană care include valoarea efectivă a elementelor tabloului.

Oracle7 asigură definirea tablourilor de înregistrări care pot fi declarate și utilizate numai în programe *PL/SQL*, *Oracle8* realizează definirea tablourilor de tipuri obiect, iar *Oracle9i* permite definirea tablourilor de colecții.

În *Oracle9i* tipul *index-by table* este redenumit *associative array* pentru compatibilitate (de limbaj) cu termenul folosit în alte limbaje de programare (*C++*, *JavaScript*, *Perl*) pentru a defini această structură de date.

Tablourile indexate *PL/SQL* trebuie definite în doi pași: se definește tipul *TABLE*; se declară tabloul indexat *PL/SQL* de acest tip.

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;
```

Identificatorul *nume_tip* este numele noului tip definit care va fi specificat în declararea tabloului *PL/SQL*, iar *tip_coloană* este un tip scalar simplu (de exemplu, *VARCHAR2*, *CHAR*, *DATE* sau *NUMBER*).

Până la versiunea *Oracle9i* unicul tip de indexare acceptat era *INDEX BY BINARY_INTEGER*. *Oracle9i* permite următoarele opțiuni pentru *tip_indexare*: *PLS_INTEGER*, *NATURAL*, *POSITIVE*, *VARCHAR2(n)* sau chiar indexarea după un tip declarat cu *%TYPE*. Nu sunt permise indexările *INDEX BY NUMBER*, *INDEX BY INTEGER*, *INDEX BY DATE*, *INDEX BY VARCHAR2*, *INDEX BY CHAR(n)* sau indexarea după un tip declarat cu *%TYPE* în care intervine unul dintre tipurile enumerate anterior.

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.
- Inițial, un tablou indexat este nedens. După declararea unui tablou se poate face referire la liniile lui prin precizarea valorii cheii primare.
- Dacă se face referire la o linie care nu există, atunci se produce excepția *NO_DATA_FOUND*.
- Dacă se dorește contorizarea numărului de linii, trebuie declarată o variabilă în acest scop sau poate fi utilizată o metodă asociată tabloului.
- Deoarece numărul de linii nu este limitat, operația de adăugare de linii este restricționată doar de dimensiunea memoriei alocate.
- Tablourile pot să apară ca argumente într-o procedură.

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*.

Asemănător, pentru regăsirea unor valori dintr-o coloană a unei baze de date într-un tablou *PL/SQL* se utilizează instrucțiunea *FETCH* (cursoare) sau instrucțiunea de atribuire în cadrul unei secvențe repetitive *LOOP*.

Pentru a șterge liniile unui tablou fie se asignează elementelor tabloului valoarea *null*, fie se declară un alt tablou *PL/SQL* (de același tip) care nu este inițializat și acest tablou vid se asignează tabloului *PL/SQL* care trebuie șters. În *PL/SQL* 2.3 ștergerea liniilor unui tabel se poate face utilizând metoda *DELETE*.

Exemplu:

Să se definească un tablou indexat *PL/SQL* având elemente de tipul *NUMBER*. Să se introducă 20 de elemente în acest tablou. Să se șteargă tabloul.

```
DECLARE
  TYPE tablou_numar IS TABLE OF NUMBER
    INDEX BY PLS_INTEGER;
  v_tablou  tablou_numar;
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
    v_tablou(i) := NULL;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('tabloul are ' || v_tablou.COUNT ||
    ' elemente');
END;
```

În *PL/SQL* este folosit frecvent tipul tablou de înregistrări. Referirea la un element al tabloului se face prin forma clasică: *tabel(index).câmp*.

Exemplu:

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

```
DECLARE
    TYPE org_table_type IS TABLE OF organizator%ROWTYPE
        INDEX BY BINARY_INTEGER;
    org_table    org_table_type;
    i            NUMBER;
BEGIN
    IF org_table.COUNT <> 0 THEN
        i := org_table.LAST+1;
    ELSE i:=1;
    END IF;
    org_table(i).cod_org := 752;
    org_table(i).nume := 'Grigore Ion';
    org_table(i).adresa := 'Calea Plevnei 18 Sibiu';
    org_table(i).tip := 'persoana fizica';
    INSERT INTO organizator
    VALUES (org_table(i).cod_org, org_table(i).nume,
            org_table(i).adresa, org_table(i).tip);
    -- sau folosind noua facilitate Oracle9i
    -- INSERT INTO organizator
    -- VALUES (org_table(i));
    org_table.DELETE; -- sterge toate elementele
    DBMS_OUTPUT.PUT_LINE('Dupa aplicarea metodei DELETE
        sunt '||TO_CHAR(org_table.COUNT)||' elemente');
END;
```

Vectori

Vectorii (*varray*) sunt structuri asemănătoare vectorilor din limbajele *C* sau *Java*. Spre deosebire de tablourile indexate, 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.

Vectorii reprezintă **structuri dense**. Fiecare element are un index care dă poziția sa în vector și care este folosit pentru accesarea elementelor particulare. Limita inferioară a indicelui este 1. Vectorul poate conține un număr variabil de elemente, de la 0 (vid) la numărul maxim specificat obligatoriu în definiția sa.

Tipul de date vector este declarat utilizând sintaxa:

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

Identificatorul *nume_tip* este numele tipului de date vector, iar *lungime_maximă* reprezintă numărul maxim de elemente din vector. *Tip_elemente* este un tip scalar *PL/SQL*, tip înregistrare sau tip obiect. De asemenea, acest tip poate fi definit utilizând attributele *%TYPE* sau *%ROWTYPE*.

În *Oracle9i* sunt permise (pentru *tip_elemente*) tipurile *TABLE* sau alt tip *VARRAY*. Există restricții referitoare la tipul elementelor, în sensul că acesta nu poate să fie *BOOLEAN*, *NCHAR*, *NCLOB*, *NVARCHAR2*, *REF CURSOR*, *PLS_INTEGER*, *LONG*, *LONG RAW*, *NATURAL*, *NATURALN*, *POSITIVE*, *POSITIVEN*, *BINARY_INTEGER*, *SIGNTYPE*, *STRING*, tip obiect cu attribute *TABLE* sau *VARRAY*, *BLOB*, *CLOB*, tip obiect cu attribute *BLOB* sau *CLOB*.

Exemplu:

```
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;
```

Tablouri imbricate

Tablourile imbricate (*nested table*) sunt tablouri indexate a căror dimensiune nu este stabilită. Numărul maxim de linii ale unui tablou imbricat este dat de capacitatea maximă 2 GB.

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.

Sistemul *Oracle* nu stochează liniile unui tablou imbricat într-o ordine particulară. Dar, când se regăsește tabloul în variabile *PL/SQL*, liniile vor avea indici consecutivi începând cu valoarea 1. Inițial, aceste tablouri sunt structuri dense, dar se poate ca în urma prelucrării să nu mai aibă indici consecutivi.

Comanda de declarare a tipului de date tablou imbricat are sintaxa:

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

Identificatorul *nume_tip* reprezintă numele noului tip de date tablou imbricat, iar *tip_elemente* este tipul fiecărui element din tabloul imbricat, care poate fi un tip definit de utilizator sau o expresie cu *%TYPE*, respectiv *%ROWTYPE*.

În *Oracle9i* sunt permise (pentru *tip_elemente*) tipurile *TABLE* sau alt tip *VARRAY*. Există restricții referitoare la tipul elementelor, în sensul că acesta nu poate să fie *BOOLEAN*, *STRING*, *NCHAR*, *NCLOB*, *NVARCHAR2*, *REF CURSOR*, *BINARY_INTEGER*, *PLS_INTEGER*, *LONG*, *LONG RAW*, *NATURAL*, *NATURALN*, *POSITIVE*, *POSITIVEN*, *SIGNTYPE*, tip obiect cu attributele *TABLE* sau *VARRAY*.

Tabloul imbricat are o singură coloană, iar dacă aceasta este de tip obiect, tabloul poate fi vizualizat ca un tabel multicoloană, având câte o coloană pentru fiecare atribut al tipului obiect.

Exemplu:

```
DECLARE
  TYPE numartab IS TABLE OF NUMBER;
  -- se creeaza un tablou cu un singur element
  v_tab_1  numartab := numartab(-7);
  -- se creeaza un tablou cu 4 elemente
  v_tab_2  numartab := numartab(7,9,4,5);
  -- se creeaza un tablou fara nici un element
  v_tab_3  numartab := numartab();
BEGIN
  v_tab_1(1) := 57;
  FOR j IN 1..4 LOOP
    DBMS_OUTPUT.PUT_LINE (v_tab_2(j) || ' ');
  END LOOP;
END;
```

Se observă că singura diferență sintactică între tablourile indexate și cele imbricate este absența clauzei *INDEX BY BINARY_INTEGER*. Mai exact, dacă această clauză lipsește, tipul este tablou imbricat.

Observații:

- Spre deosebire de tablourile indexate, vectorii și tablourile imbricate pot să apară în definirea tabelor bazei de date.
- Tablourile indexate pot avea indice negativ, domeniul permis pentru *index* fiind *-2147483647..2147483647*, iar pentru tabele imbricate domeniul indexului este *1..2147483647*.
- Tablourile imbricate, spre deosebire de tablourile indexate, pot fi prelucrate prin comenzi *SQL*.

- Tablourile imbricate trebuie inițializate și/sau extinse pentru a li se adăuga elemente.

Când este creat un tablou indexat care nu are încă elemente, el este vid. Dacă un tablou imbricat (sau un vector) este declarat, dar nu are încă nici un element (nu este inițializat), el este automat inițializat (atomic) *null*. Adică, colecția este *null*, nu elementele sale. Prin urmare, pentru tablouri imbricate poate fi utilizat operatorul *IS NULL*. Dacă se încearcă **să se adauge** un element la un tablou imbricat *null*, se va genera eroarea „*ORA - 06531: reference to uninitialized collection*“ care corespunde excepției predefinite *COLLECTION_IS_NULL*.

Prin urmare, cum poate fi inițializat un tablou imbricat? Ca și obiectele, vectorii și tablourile imbricate sunt inițializate cu ajutorul **constructorului**. Acesta are același nume ca și tipul colecției referite. *PL/SQL* apelează un constructor numai în mod explicit. Tabelele indexate nu au constructori.

Constructorul primește ca argumente o listă de valori de tip *tip_elemente*. Elementele sunt numerotate în ordine, de la 1 la numărul de valori date ca parametrii constructorului. Dimensiunea inițială a colecției este egală cu numărul de argumente date în constructor, când aceasta este inițializată. Pentru vectori nu poate fi depășită dimensiunea maximă precizată la declarare. Atunci când constructorul este fără argumente, va crea o colecție fără nici un element (vida), dar care are valoarea *not null*. Exemplul următor este concludent în acest sens.

Exemplu:

```
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;
```

În urma execuției acestui bloc se obține următorul rezultat:

```

tab1 este NULL
tab2 este NOT NULL
```

Excepțiile semnificative care apar în cazul utilizării incorecte a colecțiilor:

Exemplu:

```

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

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*;

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*;

Colecții pe mai multe niveluri

În *Oracle9i* se pot construi colecții pe mai multe niveluri (*multilevel collections*), prin urmare colecții ale căror elemente sunt, în mod direct sau indirect, colecții. În felul acesta pot fi definite structuri complexe: vectori de vectori, vectori de tablouri imbricate, tablou imbricat de vectori, tablou imbricat de tablouri imbricate, tablou imbricat sau vector de un tip definit de utilizator care are un atribut de tip tablou imbricat sau vector.

Aceste structuri complexe pot fi utilizate ca tipuri de date pentru definirea:

- coloanelor unui tabel relațional,
- atributelor unui obiect într-un tabel obiect,
- variabilelor *PL/SQL*.

Observații:

- Numărul nivelurilor de imbricare este limitat doar de capacitatea de stocare a sistemului.
- Pentru a accesa un element al colecției incluse sunt utilizate două seturi de paranteze.
- Obiectele de tipul colecție pe mai multe niveluri nu pot fi comparate.

Exemplu:

În exemplele care urmează sunt definite trei structuri complexe și sunt prezentate câteva modalități de utilizare ale acestora. Exemplele se referă la vectori pe mai multe niveluri, tablouri imbricate pe mai multe niveluri și tablouri indexate pe mai multe niveluri.

```
DECLARE
  TYPE  alfa IS VARRAY(10) OF INTEGER;
  TYPE  beta IS VARRAY(10) OF alfa;
  valf  alfa := alfa(12,31,5); --initializare
  vbet  beta := beta(valf,alfa(55,6,77),alfa(2,4),valf);
  i      integer;
  var1   alfa;
BEGIN
  i := vbet(2)(3); -- i va lua valoarea 77
  vbet.EXTEND; -- se adauga un element de tip vector la vbet
  vbet(5) := alfa(56,33);
  vbet(4) := alfa(44,66,77,4321);
  vbet(4)(4) := 7; -- 4321 este inlocuit cu 7
  vbet(4).EXTEND; -- se adauga un element la al 4-lea element
  vbet(4)(5) := 777; -- acest nou element adaugat va fi 777
END;
```

```

/
DECLARE
    TYPE gama IS TABLE OF VARCHAR2(20);
    TYPE delta IS TABLE OF gama;
    TYPE teta IS VARRAY(10) OF INTEGER;
    TYPE epsi IS TABLE OF teta;
    var1 gama := gama('alb','negru');
    var2 delta := delta(var1);
    var3 epsi := epsi(teta(31,15),teta(1,3,5));
BEGIN
    var2.EXTEND;
    var2(2) := var2(1);
    var2.DELETE(1); -- sterge primul element din var2
    /* sterge primul sir de caractere din al doilea
       tablou al tabloului imbricat */
    var2(2).DELETE(1);
END;
/
DECLARE
    TYPE alfa IS TABLE OF INTEGER INDEX BY BINARY_INTEGER;
    TYPE beta IS TABLE OF alfa INDEX BY BINARY_INTEGER;
    TYPE gama IS VARRAY(10) OF VARCHAR2(30);
    TYPE delt IS TABLE OF gama INDEX BY BINARY_INTEGER;
    var1 gama := gama('alb','negru');
    var2 beta;
    var3 delt;
    var4 alfa;
    var5 alfa; -- tablou vid
BEGIN
    var4(1) := 324;
    var4(2) := 222;
    var4(42) := 333;
    var2(27) := var4;
    var3(39) := gama(77,76,89,908);
    -- var2(40)(3) := 55; eroare nu exista element 40 in var2
    var2(40) := var5; -- asigneaza un tablou null
    var2(40)(3) := 55; -- corect
END;
/

```

Prelucrarea colecțiilor

O colecție poate fi exploatată fie în întregime (atomic) utilizând comenzi *LMD*, fie pot fi prelucrate elemente individuale dintr-o colecție (*piecewise updates*) utilizând operatori *SQL* sau anumite facilități oferite de *PL/SQL*.

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

Exemplu:

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

CREATE TABLE gal_ope (
    cod_galerie    NUMBER(10),
    nume_galerie   VARCHAR2(20),
    info           operalist);

DECLARE
    v_opera        operalist := operalist (777, 888, 999);
    v_info_op      operalist := operalist (7007);
    v_info         gal_ope.info%TYPE;
    v_cod          gal_ope.cod_galerie%TYPE := 2345;
BEGIN
    INSERT INTO gal_ope
    VALUES (4567, 'Impresionisti', operalist(4567,4987));
    INSERT INTO gal_ope
    VALUES (2345, 'Cubism', v_opera);
    INSERT INTO gal_ope
    VALUES (123, 'Alfa', v_info_op);
    SELECT info
    INTO    v_info
    FROM    gal_ope
    WHERE   cod_galerie = v_cod;
END;
```

Un vector stocat într-un tabel este prelucrat ca un întreg (nu pot fi modificate elemente individuale). Prin urmare, elementele individuale ale unui vector nu pot fi referite în comenzile *INSERT*, *UPDATE* sau *DELETE*. Pentru referirea acestora trebuie utilizate comenzi procedurale *PL/SQL*. Pentru a modifica un vector, el trebuie selectat într-o variabilă *PL/SQL* a cărei valoare poate fi modificată și apoi reinserată în tabel.

Tablourile imbricate depuse în baza de date sunt mai flexibile, deoarece pot fi prelucrate fie în întregime, fie ca elemente individuale. În fiecare caz pot fi utilizate **numai** comenzi *SQL*.

Se pot face reactualizări sau inserări asupra tablourilor imbricate care dau o valoare nouă pentru întreaga colecție sau se pot face inserări, ștergeri, reactualizări de elemente particulare din colecție.

O colecție poate fi asignată altei colecții prin comenzile *INSERT*, *UPDATE*, *FETCH*, *SELECT*, instrucțiunea de atribuire sau prin apelul unui subprogram, dar colecțiile trebuie să fie de același tip. Dacă unei colecții i se asignează o colecție atomic *null*, aceasta devine atomic *null* și trebuie reinițializată.

În *Oracle8i* a fost introdus operatorul *TABLE*, ce 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).

Operandul lui *TABLE* este:

- fie numele unei colecții și atunci rezultatul operatorului este tot o colecție,
- fie este o subinterogare referitoare la o colecție, iar în acest caz, operatorul *TABLE* returnează o singură valoare (coloană) care este un tablou imbricat sau un vector. Prin urmare, lista din clauza *SELECT* a subcererii trebuie să aibă un singur articol.

Exemplu:

Se presupune că tabelul *opera* are o coloană *info* de tip tablou imbricat. Acest tablou are două componente în care pentru fiecare operă de artă sunt depuse numele articolului referitor la opera respectivă și revista în care a apărut. Să se insereze o linie în tabelul imbricat.

```
INSERT INTO TABLE (SELECT info
                      FROM   opera
                      WHERE  titlu = 'Primavara')
VALUES ('Pictura moderna', 'Orizonturi');
```

Listarea codului fiecărei opere de artă și a colecției articolelor referitoare la aceste opere de artă se face prin comanda:

```
SELECT  a.cod_opera, b.*
FROM    opera a, TABLE (a.info) b;
```

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)**

Operanzii lui *CAST* sunt o colecție declarată local (de exemplu, într-un bloc *PL/SQL* anonim) și un tip colecție *SQL*. *CAST* convertește colecția locală la tipul specificat. În felul acesta, o colecție poate fi prelucrată ca și cum ar fi un tabel *SQL* al bazei de date.

Metodele unei colecții

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

Metodele sunt apelate prin expresia:

nume_colecție.nume_metodă [(parametri)]

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

COUNT returnează numărul curent de elemente ale 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*) returnează *TRUE* dacă există al *n*-lea element al unei colecții *PL/SQL* (altfel, returnează *FALSE*, chiar dacă elementul este *null*);

FIRST, **LAST** returnează indicele primului, respectiv ultimului element din colecție;

NEXT(*n*), **PRIOR(*n*)** returnează indicele elementului următor, respectiv precedent celui de rang *n* din colecție, iar dacă nu există un astfel de element returnează 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*); nu poate fi utilizată pentru a inițializa o colecție atomic *null*;

LIMIT returnează numărul maxim de elemente ale 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*.

Bulk bind

În exemplul care urmează, comanda *DELETE* este trimisă motorului *SQL* pentru fiecare iterație a comenzii *FOR*.

Exemplu:

```
DECLARE
  TYPE nume IS VARRAY(20) OF NUMBER;
  alfa nume := nume(10,20,70); -- coduri ale galeriilor
BEGIN
  FOR j IN alfa.FIRST..alfa.LAST
    DELETE FROM opera
    WHERE cod_galerie = alfa (j);
  END LOOP;
END;
```

Pentru a realiza mai rapid această operație, ar trebui să existe posibilitatea de a șterge (prelucra) întreaga colecție și nu elemente individuale. Tehnica care permite acest lucru este cunoscută sub numele ***bulk bind***.

În timpul compilării, compilatorul *PL/SQL* asociază identificatorii cu o adresă, un tip de date și o valoare. Acest proces este numit *binding*.

Comenzile *SQL* din blocurile *PL/SQL* sunt trimise motorului *SQL* pentru a fi executate. Motorul *SQL* poate trimite înapoi date motorului *PL/SQL* (de exemplu, ca rezultat al unei interogări). De multe ori, datele care trebuie manipulate aparțin unei colecții, iar colecția este iterată printr-un ciclu *FOR*. Prin urmare, transferul (în ambele sensuri) între *SQL* și *PL/SQL* are loc pentru fiecare linie a colecției.

Începând cu *Oracle8i* există posibilitatea ca toate liniile unei colecții să fie transferate simultan printr-o singură operație. Procedeu este numit ***bulk bind*** și este realizat cu ajutorul comenzii *FORALL*, ce poate fi folosită cu orice tip de colecție.

Comanda *FORALL* are sintaxa:

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

Motorul *SQL* execută *comanda_sql* o singură dată pentru toate valorile indexului. *Comanda_sql* este una din comenzile *INSERT*, *UPDATE*, *DELETE* care referă elementele uneia sau mai multor colecții. Variabila *index* poate fi referită numai în comanda *FORALL* și numai ca indice de colecție.

În exemplul care urmează este optimizată problema anterioară, în sensul că instrucțiunea *DELETE* este trimisă motorului *SQL* o singură dată, pentru toate liniile colecției.

Exemplu:

```

DECLARE
  TYPE nume IS VARRAY(20) OF NUMBER;
  alfa nume := nume(10,20,70); -- coduri ale galeriilor
BEGIN
  ...
  FORALL j IN alfa.FIRST..alfa.LAST
    DELETE FROM opera
    WHERE cod_galerie = alfa (j);
END;
```

Pentru utilizarea comenzii *FORALL* este necesară respectarea următoarelor restricții:

- comanda poate fi folosită numai în programe *server-side*, altfel apare eroarea “*this feature is not supported in client-side programs*”;
- comenzile *INSERT*, *UPDATE*, *DELETE* trebuie să refere cel puțin o colecție;
- toate elementele colecției din domeniul precizat trebuie să existe (dacă, de exemplu, un element a fost șters, atunci este semnalată o eroare);
- indicii colecțiilor nu pot să fie expresii și trebuie să aibă valori continue.

Exemplu:

```

CREATE TABLE exemplu (x NUMBER, y NUMBER);
DECLARE
  TYPE nume IS TABLE OF NUMBER;
  ttt nume:= nume(1,2,3);
BEGIN
  FORALL i IN ttt.FIRST..ttt.LAST
    INSERT INTO exemplu
    VALUES(ttt(i), 10); -- corect
  FORALL i IN 1..3
    INSERT INTO exemplu
    VALUES(7, 9);      -- exceptie
END;
```

```

FORALL i IN gama.FIRST..gama.LAST
  DELETE FROM carte
  WHERE codel = gama(i+1);
-- eroare dubla (expresie si >LAST)
```



```

DECLARE
    TYPE alfa IS TABLE OF NUMBER;
    xx    alfa := alfa(10,20,30,40);
BEGIN
    xx.DELETE(3);
    FORALL i IN xx.FIRST..xx.LAST
        DELETE FROM carte
        WHERE codel = xx(i);    -- eroare
END;

```

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

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*. Dacă nu este afectată nici o linie, valoarea atributului este 0. *%BULK_ROWCOUNT* nu poate să fie parametru în subprograme și nu poate fi asignat altor colecții.

Începând cu *Oracle9i* este utilizabilă o nouă clauză în comanda *FORALL*. Clauza, numită *SAVE EXCEPTIONS*, permite ca toate excepțiile care apar în timpul execuției comenzii *FORALL* să fie salvate și astfel procesarea poate să continue. În acest context, poate fi utilizat atributul cursor *%BULK_EXCEPTIONS* 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.

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

Clauza poate să apară în:

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

Clauza are următoarea sintaxă:

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

```

DECLARE
    TYPE tip1 IS TABLE OF opera.cod_opera%TYPE;
    TYPE tip2 IS TABLE OF opera.titlu%TYPE;
    alfa tip1;
    beta tip2;
BEGIN
    ...
    /* motorul SQL incarca in intregime coloanele
       cod_opera si titlu in tabelele imbricate,
       inainte de a returna
       tabelele motorului PL/SQL */
    SELECT cod_opera, titlu BULK COLLECT INTO alfa,beta
    FROM    opera;

    ...
    /* daca exista n opere de arta in stare buna,
       atunci alfa va contine codurile celor n opere */
    DELETE FROM opera WHERE stare = 'buna'
    RETURNING cod_opera BULK COLLECT INTO alfa;

    ...
END;

```

Comanda *FORALL* se poate combina cu clauza *BULK COLLECT*. Totuși, trebuie subliniat că ele nu pot fi folosite simultan în comanda *SELECT*.

Motorul SQL incarca toate liniile unei coloane. Cum se poate limita numarul de linii procesate? Exemplelele urmatoare dau 2 variante de rezolvare.

```

DECLARE
    TYPE alfa IS TABLE OF carte.pret%TYPE;
    xx alfa;
BEGIN
    SELECT pret BULK COLLECT INTO xx
    FROM    carte WHERE ROWNUM <= 500;
END;

DECLARE
    ...
    zz NATURAL := 10;
BEGIN
    OPEN c1;
    LOOP
        FETCH c1 BULK COLLECT INTO xx, yy LIMIT zz;
        EXIT WHEN c1%NOTFOUND;
    ...
END;

```