Tipuri de date compuse

Agenda:

- Tipul de date RECORD
- Tipul de date colectie:
  - TABLOU INDEXAT
  - TABLOU IMBRICAT
  - VECTORI

Sintaxa SELECT

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.

Inregistrările trebuie definite în doi paşi:

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

Obs:

- Un *RECORD* poate sa aiba un camp de tip *RECORD*;
- Este un mod elegant de a salva o linie returnata de o cerere;
- Poate constitui (prin varianta stocata) un tip de date pentru coloana unui tabel.

Sintaxa RECORD

Definire si utilizare:

```
TYPE type_name IS RECORD
     (field_declaration[, field_declaration]…);
```

```
identifier    type_name;
```

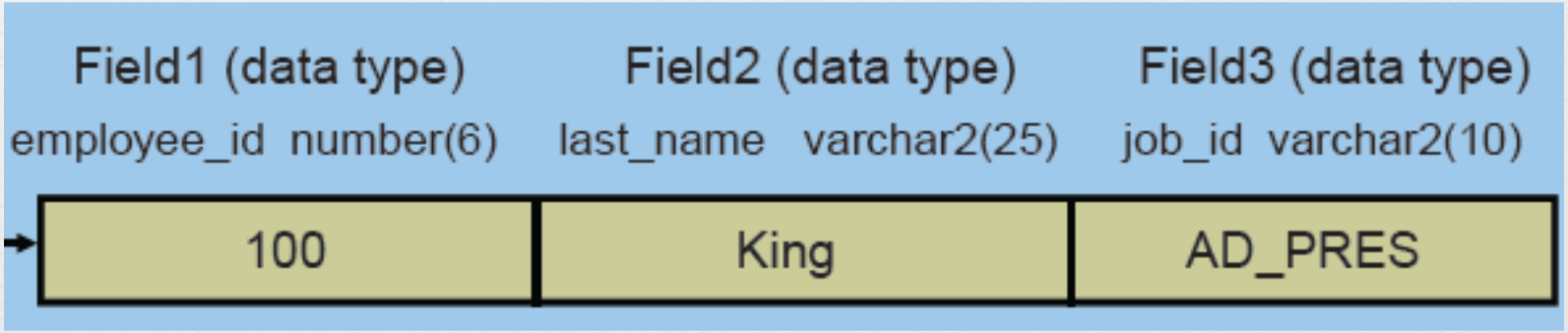
field_declaration:

```
field_name {field_type | variable%TYPE
           | table.column%TYPE | table%ROWTYPE}
           [[NOT NULL] {:= | DEFAULT} expr]
```

Sintaxa RECORD

Forma generala:

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*],...);



Observatii

- Dacă un câmp nu este iniţializat atunci implicit se consideră că are valoarea *NULL*. Dacă s-a specificat constrângerea *NOT NULL*, atunci obligatoriu câmpul trebuie iniţializat 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 instrucţiunea clasică de atribuire. De asemenea, o înregistrare poate fi asignată altei înregistrari de acelaşi tip.
- Componentele unei înregistrări pot fi de tip scalar, *RECORD*, *TABLE*, obiect, colecţie (dar, nu tipul *REF CURSOR*).
- *PL/SQL* permite declararea şi 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 şi pot să apară în clauza *RETURN* a unei funcţii.

%ROWTYPE vs RECORD

```
DECLARE
   identifier reference%ROWTYPE;
```

- tipul *RECORD* permite specificarea tipului de date pentru câmpuri şi permite declararea câmpurilor sale;
- atributul *%ROWTYPE* nu cere cunoaşterea numărului şi tipurilor coloanelor tabloului.
- 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 conţinutul unei înregistrări.
- într-o înregistrare se poate regăsi şi returna informaţia din clauza *RETURNING* a comenzilor *UPDATE* sau *DELETE*.

Exemple

Exemplu1:

```
DECLARE
  TYPE t_rec IS RECORD
    (v_sal number(8),
     v_minsal number(8) default 1000,
     v_hire_date employees.hire_date%type,
     v_rec1 employees%rowtype);
  v_myrec t_rec;
BEGIN
  v_myrec.v_sal := v_myrec.v_minsal + 500;
  v_myrec.v_hire_date := sysdate;
  SELECT * INTO v_myrec.v_rec1
      FROM employees WHERE employee_id = 100;
  DBMS_OUTPUT.PUT_LINE(v_myrec.v_rec1.last_name ||' '||
  to_char(v_myrec.v_hire_date) ||' '|| to_char(v_myrec.v_sal));
END;
```

Exemple

Exemplu2:

```
. . .
DECLARE
v_employee_number number:= 124;
v_emp_rec retired_emps%ROWTYPE;
BEGIN
SELECT employee_id, last_name, job_id, manager_id,
hire_date, hire_date, salary, commission_pct,
department_id INTO v_emp_rec
FROM employees
WHERE employee_id = v_employee_number;
INSERT INTO retired_emps VALUES v_emp_rec;
END;
/
```

Exemple

Exemplu3:

```
DECLARE
v_employee_number number:= 124;
v_emp_rec retired_emps%ROWTYPE;
BEGIN
SELECT * INTO v_emp_rec
FROM retired_emps
WHERE empno = v_employee_number;
v_emp_rec.leavedate:= CURRENT_DATE;
UPDATE retired_emps
SET ROW = v_emp_rec
WHERE empno=v_employee_number;
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.
- În PL/SQL există trei tipuri de colecții:
  - tablouri indexate (index-by tables);
  - tablouri imbricate (nested tables);
  - vectori (varrays sau varying arrays).

Colecții

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

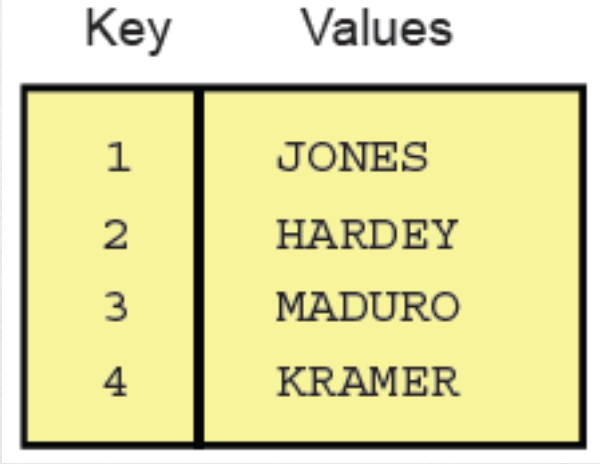
```
DECLARE
  TYPE  tab_index IS TABLE OF NUMBER
    INDEX BY BINARY_INTEGER;
  TYPE  tab_imbri IS TABLE OF NUMBER;
  TYPE  vector IS VARRAY(15) OF NUMBER;
  v_tab_index  tab_index;
  v_tab_imbri  tab_imbri;
  v_vector     vector;
BEGIN
  v_tab_index(1) := 72;
  v_tab_index(2) := 23;
  v_tab_imbri := tab_imbri(5, 3, 2, 8, 7);
  v_vector := vector(1, 2);
END;
```

Observatii colecții

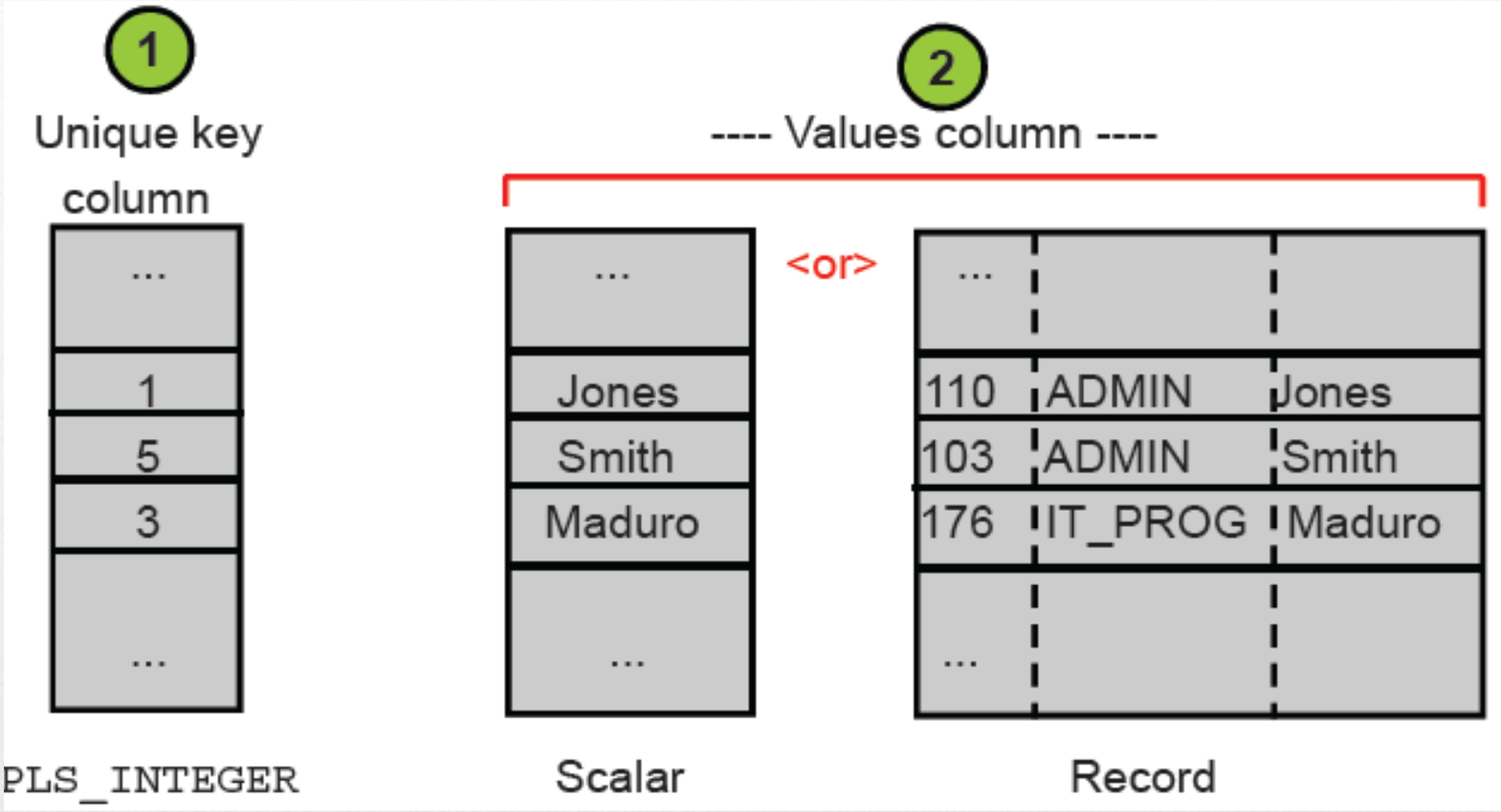
- Deoarece colecţiile nu pot fi comparate (egalitate sau inegalitate), ele nu pot să apară în clauzele *DISTINCT*, *GROUP BY*, *ORDER BY*.
- Tipul colecţie poate fi definit într-un pachet.
- Tipul colecţie poate să apară în clauza *RETURN* a unei funcţii.
- Colecţiile pot fi parametri formali într-un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

Tablouri indexate - *associative array*

- Tabloul indexat *PL/SQL* are două componente: o coloană ce cuprinde cheia primară (integer/string) pentru acces la liniile tabloului şi o coloană care include valoarea efectivă a elementelor tabloului



Structura tablou indexat



Declarare tip date tablou indexat

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 (sau alt tip definit anterior).
Initial unicul tip de indexare acceptat era *INDEX BY BINARY_INTEGER*, ulterior au fost adaugate următoarele opţiuni 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

Observatii:

- Elementele unui tablou indexat **nu sunt într-o ordine particulară** şi pot fi inserate cu chei arbitrare.
- Deoarece nu există constrângeri de dimensiune, **dimensiunea tabloului se modifică dinamic**.
- Tabloul indexat *PL/SQL* **nu poate fi iniţializat** în declararea sa.
- Un tablou indexat **neiniţializat este vid** (nu conţine nici valori, nici chei).
- Un element al tabloului este nedefinit atâta timp cât nu are atribuită o valoare efectivă.
- Iniţial, **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 excepţia *NO_DATA_FOUND*.
- Dacă se doreşte 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, operaţia de adăugare de linii este restricţionată doar de dimensiunea memoriei alocate.

Exemple

```
...
DECLARE
TYPE ename_table_type IS TABLE OF
employees.last_name%TYPE
INDEX BY PLS_INTEGER;
TYPE hiredate_table_type IS TABLE OF DATE
INDEX BY PLS_INTEGER;
ename_table ename_table_type;
hiredate_table hiredate_table_type;
BEGIN
ename_table(1) := 'CAMERON';
hiredate_table(8) := SYSDATE + 7;
IF ename_table.EXISTS(1) THEN
INSERT INTO ...
...
END;
/
...
```

Exemple

```
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));
  org_table.DELETE; -- sterge toate elementele
  DBMS_OUTPUT.PUT_LINE('Dupa aplicarea metodei DELETE
      sunt '||TO_CHAR(org_table.COUNT)||' elemente');
END;
```

Metode asociate

- EXISTS
- COUNT
- FIRST
- LAST
- PRIOR
- NEXT
- DELETE

Descriere

- `EXISTS(n)` – boolean; verifica daca exista componenta n;
- `COUNT` – numarul de elemente din colectie;
- `FIRST – NULL` sau indicele primei componente;
- `LAST – NULL` sau indicele ultimei componente ;
- `PRIOR(n)` – indicele componentei anterioare;
- `NEXT(n)` – indicele componentei urmatoare;
- `DELETE, DELETE(n), DELETE(m, n)` – indiceleutilizata pentru stergerea de componente;
- `EXTEND(n), TRIM(n)` – adauga/sterge n componente;
- `LIMIT` – intoarce numarul maxim de elemente al unei colectii;

Exemple

```
DECLARE
TYPE dept_table_type
IS
TABLE OF departments%ROWTYPE INDEX BY VARCHAR2(20);
dept_table dept_table_type;
-- Each element of dept_table is a record
BEGIN
SELECT * INTO dept_table(1) FROM departments
WHERE department_id = 10;
DBMS_OUTPUT.PUT_LINE(dept_table(1).department_id ||'
'||
dept_table(1).department_name ||' '||
dept_table(1).manager_id);
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 şi ordinea elementelor este importantă.

Vectorii reprezintă **structuri dense**. Fiecare element are un index care dă poziția sa în vector şi 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***];

Vectori

```
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 mulţime neordonată de elemente de acelaşi tip. Valorile de acest tip:

- pot fi stocate în baza de date,
- pot fi prelucrate direct în instrucţiuni *SQL*
- au excepţii predefinite proprii.

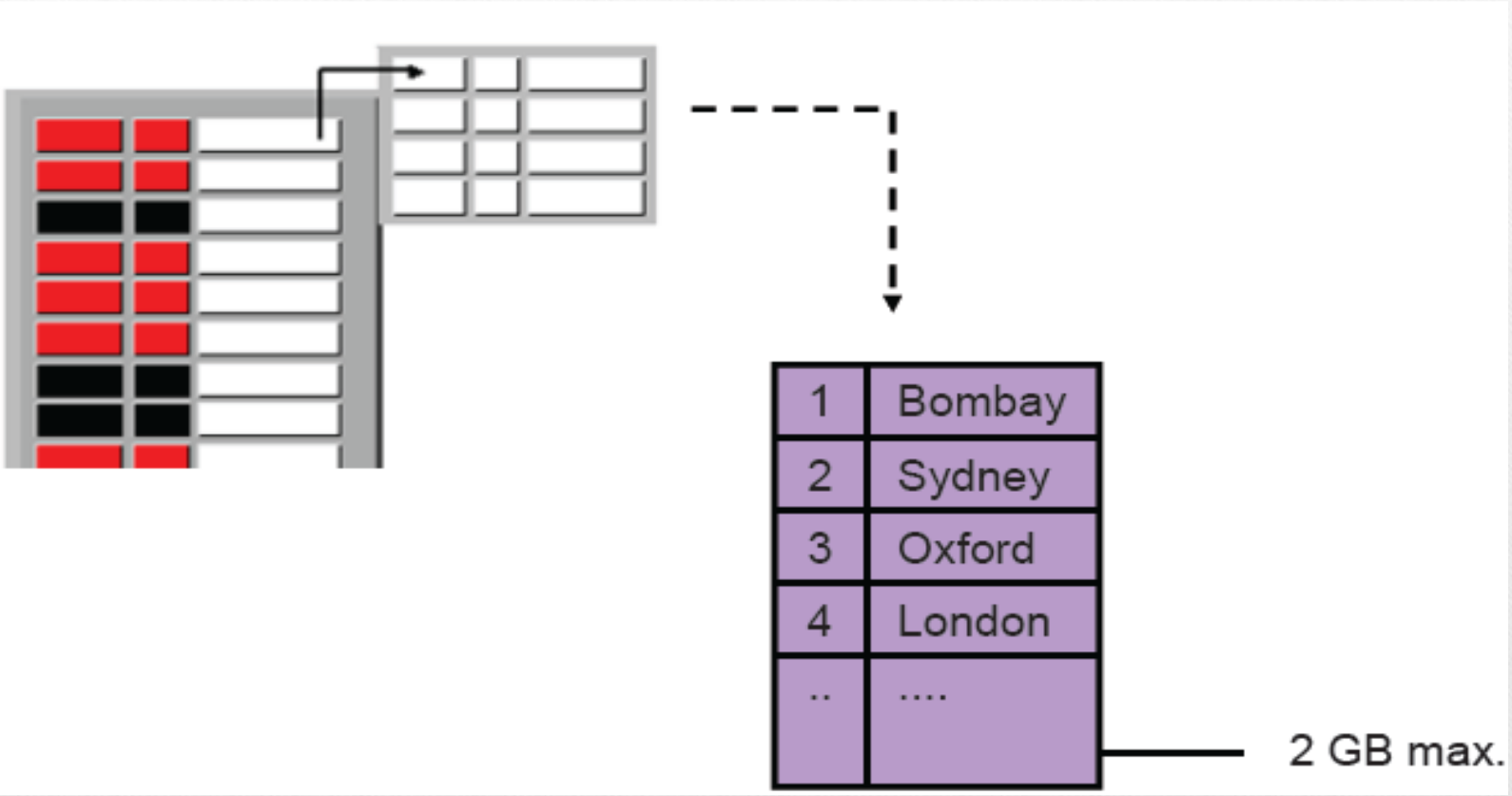
Sistemul *Oracle* nu stochează liniile unui tablou imbricat într-o ordine particulară. Dar, când se regăseşte tabloul în variabile *PL/SQL*, liniile vor avea indici consecutivi începând cu valoarea 1. Iniţial, aceste tablouri sunt structuri dense, dar se poate ca în urma prelucrării să nu mai aibă indici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

Tablouri imbricate



Exemple

```
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 diferenţă sintactică între tablourile indexate şi cele imbricate este absenţa clauzei *INDEX BY BINARY_INTEGER*. Mai exact, dacă această clauză lipseşte, tipul este tablou imbricat.

Observatii:

- Spre deosebire de tablourile indexate, vectorii şi tablourile imbricate pot să apară în definirea tabelelor 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 iniţializate şi/sau extinse pentru a li se adăuga elemente.

Initializare sau nu?

```
DECLARE
TYPE  alfa IS TABLE OF VARCHAR2(50);
tab1  alfa; -- creeaza un tablou (atomic) null
tab2  alfa := alfa(); --tabloul nu este null, e initializat
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 execuţiei acestui bloc se obţine următorul rezultat:

```
tab1 este NULL
tab2 este NOT NULL
```

Tipuri de date stocate

Tipurile de date tablou imbricat, vector si object(record) pot fi utilizate drept câmpuri în tabelele bazei. Aceasta presupune că fiecare înregistrare din tabelul respectiv conţine un obiect de tip colecţie. Înainte de utilizare, tipul trebuie stocat în dicţionarul datelor, deci trebuie declarat prin comanda:

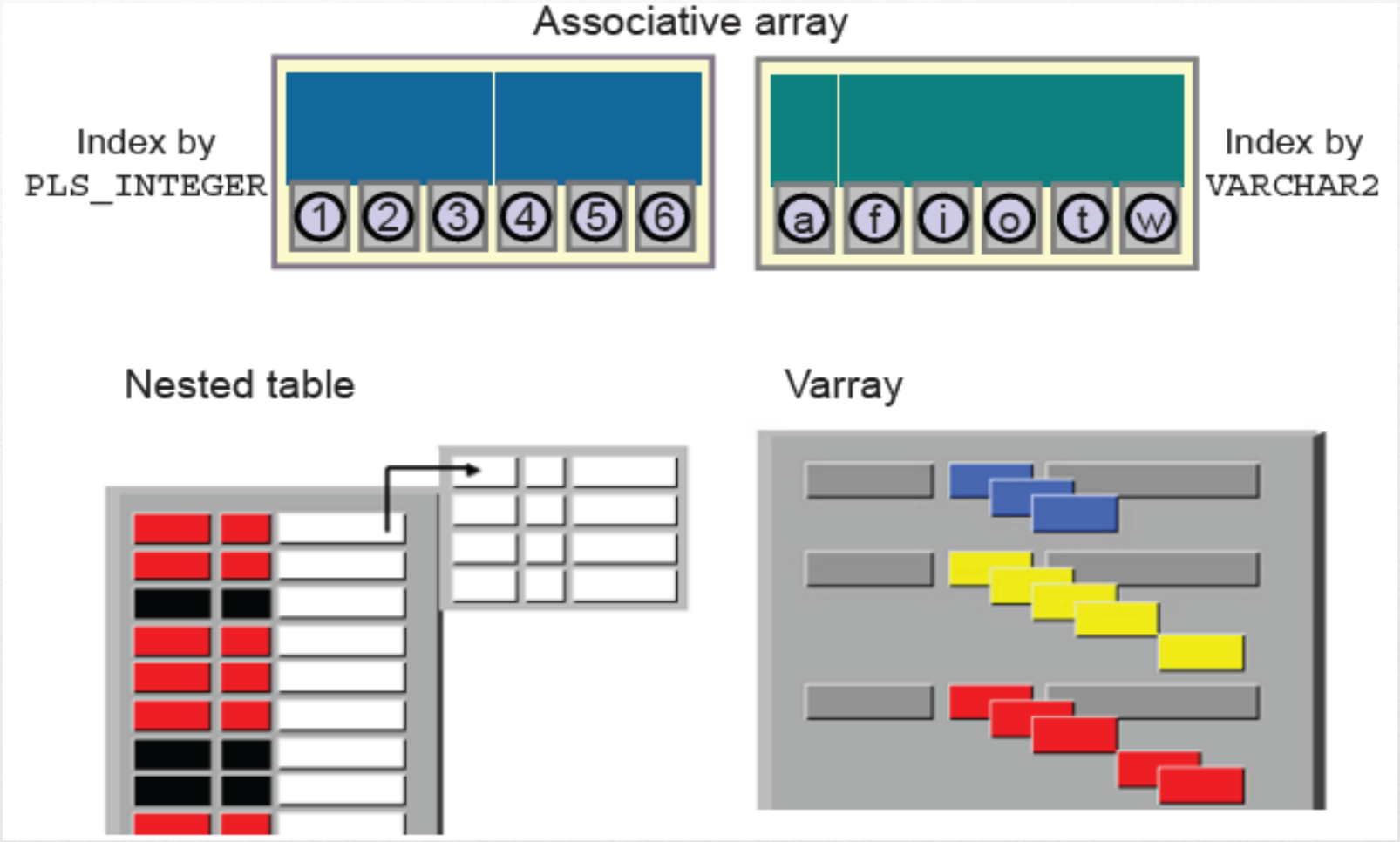
CREATE TYPE *nume_tip* ***AS*** {***TABLE*** | ***VARRAY***} ***OF*** *tip_elemente*;

CREATE TYPE *nume_tip* ***AS OBJECT(camp1 type1, …)***;

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*;

Imagine generala



Colecţii pe mai multe niveluri

Se pot construi colecţii pe mai multe niveluri (*multilevel collections*), prin urmare colecţii ale căror elemente sunt, în mod direct sau indirect, colecţii. 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 relaţional,
- atributelor unui obiect într-un tabel obiect,
- variabilelor *PL/SQL*.

Observaţii:

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

Exercitii

1. Definiți un bloc anonim in cadrul căruia trebuie sa definiți 3 tipuri de date (tablou, tablou imbricat, vector). Populați cu ajutorul clauzelor repetitive studiate 3 variabile de acest tip (minim 100 de numere) – clauza diferita pentru fiecare variabila. Ulterior, pentru fiecare variabila ștergeți componentele care contin numere impare. Parcurgeți cele 3 variabile de la prima la ultima componenta si invers.
2. Definiți un tip de date care sa permită sa stocați o matrice pătratică de dimensiune n. Definiți si inițializați o astfel de variabila cu valori care apar pe coloana *employees.employee_id*. Afișați conținutul acestei variabile parcurgând matricea in spirala.