PLSQL 4

De la BD

Cuprins

Colecţii şi înregistrări

Linkuri utile:

- http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/collections.htm
- http://www.orafaq.com/wiki/NESTED_TABLE
- http://cursuri.cs.pub.ro/~radulescu/bd/plsql/8.html
- http://docs.oracle.com/cd/A97630_01/appdev.920/a96624/05_colls.htm#1059

Pentru a permite lucrul cu şiruri de elemente, în SQL se vor folosi doua tipuri de date: TABLE şi VARRAY.

Colecţii - fiecare element al unei colecţii are un index ce îl identifică în mod unic în cadrul colecţiei. În PLSQL se pot face trei tipuri de colecţii:

- tablouri asociative (associative arrays)
- tabel/(ă) (nested tables)
- tablouri de dimensiune variabilă (varrays)

Pentru a lucra cu aceste tipuri de colecţii trebuie ca în prealabil să definiţi un tip după care să declaraţi variabile care să aibă tipul definit de voi. Declaraţia unui tip poate fi făcută, de exemplu, în secţiunea de declaraţii (fie a unui cod anonim, fie a unei funcţii/proceduri).

O colecţie poate fi trimisă ca parametru (pentru a transmite către o funcţie mai multe informaţii în acelaşi timp).

Tablourile asociative (associative arrays)

Tablourile asociative sunt mulţimi de cupluri de tipul cheie-valoare (cunoscute în alte limbaje de programare sub numele de tabele hash), fiecare cheie fiind unică (din cauză că pe baza ei se face accesul la valoare). Scopul acestui tip de date este de stocare temporară a informaţiilor, acestea fiind disponibile numai în cadrul aplicaţiei PL/SQL care le utilizează (deci nu sunt stocate pe disc pentru a fi regăsite ulterior, pentru aceasta trebuie să construiţi un cod special prin care să le salvaţi într-o bază de date).

Dacă se face o atribuire în tabloul asociativ pentru o cheie care nu există, aceasta este automat creată după care i se dă valoarea asignată de operaţia de atribuire.

Pentru a defini tipul ce va fi utilizat în declararea unui tablou asociativ se va utiliza comanda:

TYPE nume_tip IS TABLE OF tip_element [NOT NULL] INDEX BY [BINARY_INTEGER | PLS_INTEGER | VARCHAR2(dimendiune)];

După definirea tipului trebuie să declaraţi o variabilă de acel tip. Valoarea "nume_tip" este un identificator pentru noul tip de dată (deci veţi putea declara o variabilă tablou de tipul nume_tip), tip_element indică ce fel de elemente sunt stocate în fiecare poziţie a tabloului (se poate preciza ca această valoare trebuie să fie nenulă) iar în final este specificat tipul cheii (acesta poate fi un tip numeric sau şir de caractere).

Un exemplu de utilizare a unui tablou asociativ în care variabilele stocate sunt numere întregi iar cheile sunt şiruri de caractere:

```
DECLARE
    TYPE MyTab IS TABLE OF NUMBER INDEX BY VARCHAR2(10);
    varsta MyTab;
BEGIN
   varsta('Gigel') := 3;
   varsta('Ionel') := 4;
   DBMS_OUTPUT.PUT_LINE('Varsta lui Gigel este ' || varsta('Gigel'));
   DBMS_OUTPUT.PUT_LINE('Varsta lui Ionel este ' || varsta('Ionel'));
END;
```

În exemplul de mai sus se pot observa atât operaţia de asociere a unei chei cu o valoare cât şi modalitatea de extragere a unei valori atunci când cunoaştem cheia (în operaţia de afişare).

Dacă se doreşte adăugarea unui nou copil pe care îl cheamă (din întâmplare) tot Ionel şi care are vârsta de 7 ani, acest lucru nu va putea fi făcut deoarece cheia trebuie să fie unică (identificatorul Ionel în acest caz s-ar repeta). Încercarea de a face o atribuire de tipul varsta('Ionel') := 7; va duce la modificarea înregistrării deja existente în tablou.

Puteţi să construiţi un tablou asociativ în care cheile să nu mai fie de tip şir de caractere (ci un tip de date numeric) şi care să aibă ca şi valori posibile tipul liniei dintr-o tabelă:

```
DECLARE
    TYPE MyTab IS TABLE OF studenti%ROWTYPE INDEX BY PLS_INTEGER;
    linii MyTab;
BEGIN
    SELECT * INTO linii(0) FROM studenti WHERE ROWNUM = 1;
    DBMS_OUTPUT.PUT_LINE(linii(0).prenume);
END;
```

Dacă în poziţia cheii este utlizată o valoarea care nu este compatibilă (convertibilă în mod automat) de către SGBD se va semnala o eroare. În cazul în care conversia este posibilă, utilizarea valorii (de exemplu o data calendaristică ca şi cheie deşi cheile au fost declarate ca fiind de tip varchar2) nu va semnala nici o eroare şi codul va funcţiona corect:

```
DECLARE
    TYPE MyTab IS TABLE OF number INDEX BY varchar2(20);
    linii MyTab;
BEGIN
    linii(sysdate) := 123;
    DBMS_OUTPUT.PUT_LINE(linii(sysdate));
END;
```

Atenţie: în exemplul de mai sus, pentru valoarea de tip DATE ce a fost folosită ca şi cheie s-a aplicat funcţia TO_CHAR. În cazul în care un astfel de tabel asociativ ar fi salvat şi datele apoi mutate pe alt calculator, s-ar putea ca, din cauza modului în care sunt convertite datele (setările locale ale calculatorului), datele iniţiale să nu mai poată fi accesate.

Funcţiile ce pot fi utilizate pentru o colecţie de tipul tablou asociativ sunt exemplificate în următorul cod pe care vă invităm să îl rulaţi pentru a observa efectul fiecăreia. Explicaţii asupra acestor funcţii găsiţi în secţiunea "Funcţii ce pot fi utilizate pentru colecţii" din această pagină.

```
DECLARE
    TYPE MyTab IS TABLE OF NUMBER INDEX BY VARCHAR2(10);
    varsta MyTab;
BEGIN
   varsta('Gigel') := 3;
   varsta('Ionel') := 4;
   varsta('Maria') := 6;

   DBMS_OUTPUT.PUT_LINE('Numar de elemente in lista: ' || varsta.COUNT);

   DBMS_OUTPUT.PUT_LINE('Prima cheie din lista: ' || varsta.FIRST);
   DBMS_OUTPUT.PUT_LINE('Ultima cheie din lista: ' || varsta.LAST);

   DBMS_OUTPUT.PUT_LINE('Inaintea lui Ionel in lista: ' || varsta.PRIOR('Ionel'));
   DBMS_OUTPUT.PUT_LINE('Dupa Ionel in lista: ' || varsta.NEXT('Ionel'));

   varsta.DELETE('Maria');
   DBMS_OUTPUT.PUT_LINE('Dupa Ionel in lista: ' || varsta.NEXT('Ionel'));
END;
```

Tabel/(ă) (Nested tables)

Tipul tabel (sau tabelă - în www.dex.ro ambii termeni descriu acelaşi concept, indicând chiar unul spre celălalt) este un tip de date asemănător unei tabele dintr-o bază de date. Acest tip de date este utilizat, de obicei, pentru a stoca mai multe date asociate unei singure linii. La fel ca şi conceptul de tabelă din baza de date, informaţiile nu sunt organizate ci sunt mai degrabă văzute ca o mulţime: la un moment dat nu se ştie care este primul element, al doilea etc. Totuşi, la fel ca şi în baza de date, în momentul în care se doreşte iterarea elementelor, un număr provizoriu poate fi asociat fiecărei înregistrări (asemenei rownum).

La fel ca în cazul tablourilor asociative, se pot crea tabele a căror elemente să fie tot tabele (având aşadar o formă de matrice).

Pentru a accesa elementele dintr-un tabel se utilizează aceşi metodă ca şi la tablourile asociative. Există totuşi diferenţe în modul în care acestea sunt menţinute de către SGBD şi în uşurinţa cu care sunt transmise ca parametri unor funcţii sau proceduri stocate.

Spre deosebire de datele de tip tablou asociativ, datele de tip tabel pot fi stocate ca şi câmp în interiorul unui tabel din baza de date - deci se pot crea tabele (CREATE TABLE... care să aibă ca şi tip de data asociat unui câmp o variabilă de tip tabel).

Pentru a defini un nou tip ca şi tabel se va utiliza următoarea sintaxă:

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

În continuare se va declara o variabilă având ca şi tip "nume_tip" în care pot fi introduse elemente de având tipul "tip_element".

Iată un exemplu:

```
DECLARE
    TYPE prenume IS TABLE OF varchar2(10);
    student prenume;
BEGIN
    student := prenume('Gigel', 'Ionel');
    for i in student.first..student.last loop
       DBMS_OUTPUT.PUT_LINE(i||' - '||student(i));
    end loop;
END;
```

Să construim în continuare un tabel cu trei elemente, să îi adăugăm încă patru elemente copiind elementul din mijloc de încă două ori după care să ştergem elementul al doilea şi să afişăm toate poziţiile ocupate şi elementele de pe aceste poziţii:

```
DECLARE
    TYPE prenume IS TABLE OF varchar2(10);
    student prenume;
BEGIN
    student := prenume('Gigel', 'Ionel', 'Maria');
    student.EXTEND(4,2); -- copii elementul al doilea de 4 ori
    student.delete(2); -- sterg elementul al doilea
    for i in student.first..student.last loop
        if student.exists(i) then -- daca incerc sa afisez ceva ce nu exista se va produce o eroare
           DBMS_OUTPUT.PUT_LINE(i||' - '||student(i));  -- afisam pozitia si valoarea
        end if;
    end loop;
END;
```

Mai multe funcţii ce pot fi utilizate în cadrul tablourilor găsiţi în secţiunea "Funcţii ce pot fi utilizate pentru colecţii" din această pagină.

După cum se observă în exemplul anterior, ştergerea elementului al doilea din tabel nu a dus la shiftarea automată a informaţiilor de pe poziţiile 3..7 spre stânga. Atenţie: după ştergere, pe poziţia a doua nu a rămas nici un element. Încercarea de a afişa elementul de pe poziţia a doua va genera o eroare.

Valoarea lui tip_element poate fi şi altceva decât varchar2(10). Puteţi încerca să puneţi acelaşi tip cu al unei coloane existente într-un tabel (studenti.prenume%type), a unui rând dintr-o tabelă (studenti%ROWTYPE) sau a unui rând dintr-un cursor (cursor%ROWTYPE).

Nu puteti extinde siruri care sunt nule (sau neintializate):

```
set serveroutput on;
DECLARE
    TYPE prenume IS TABLE OF varchar2(10);
    student prenume := prenume();
    student_err prenume;
BEGIN
    student.EXTEND; -- merge ok
    student_err.EXTEND; -- da eroare pentru ca este null (sau nu a fost initializat)
END;
```

Iată un exemplu care încarcă într-o variabilă de tip tabel toate elementului tabelului studenti:

```
DECLARE
   CURSOR curs IS SELECT nume, prenume FROM studenti;
   -- cursorul este utilizat doar in linia urmatoare, pentru a defini tipul valorilor
   -- din nested table. Se poate folosi si un record in care definiti doar nume, prenume.
   TYPE linie_student IS TABLE OF curs%ROWTYPE;
   lista_studenti linie_student;
BEGIN
   SELECT nume, prenume BULK COLLECT INTO lista_studenti FROM studenti;
    for i in lista_studenti.first..lista_studenti.last loop
        if lista_studenti.exists(i) then -- daca incerc sa afisez ceva ce nu exista se va produce o eroare
           DBMS_OUTPUT.PUT_LINE(i||' - '||lista_studenti(i).nume);  -- afisam pozitia si valoarea
        end if;
    end loop;
    DBMS_OUTPUT.PUT_LINE('Numar studenti: '||lista_studenti.COUNT);
END;
```

Exemplul anterior poate fi refăcut utilizând funcţia NEXT pentru a obţine cheia următorului element. În acest caz nu mai este nevoie de testarea existenţei elementului (se poate elimina condiţia if).

Puteţi crea tabele (cu CREATE TABLE) care să aibă un anumit câmp de tipul tabel. Iată un exemplu în acest sens:

```
GRANT CREATE TYPE TO STUDENT; -- aceasta linie se executa din "SYS as SYSDBA"

CREATE OR REPLACE TYPE lista_prenume AS TABLE OF VARCHAR2(10);
/
CREATE TABLE persoane (nume varchar2(10),
       prenume lista_prenume)
       NESTED TABLE prenume STORE AS lista;
/

INSERT INTO persoane VALUES('Popescu', lista_prenume('Ionut', 'Razvan'));
INSERT INTO persoane VALUES('Ionescu', lista_prenume('Elena', 'Madalina'));
INSERT INTO persoane VALUES('Rizea', lista_prenume('Mircea', 'Catalin'));
/
```

```
SELECT * FROM persoane;

-- mai multe operatii direct cu tabelele interne dintr-un tabel gasiti in linkurile de la inceputul acestei pagini.
```

Şi în continuare o funcţie PL/SQL care să insereze în această nouă tabelă o persoană:

```
DECLARE
    sir_prenume persoane.prenume%type;
BEGIN
    sir_prenume := lista_prenume('Cristi', 'Tudor', 'Virgil');
    INSERT INTO persoane VALUES ('Gurau', sir_prenume);
    DBMS_OUTPUT.PUT_LINE('Gata');
END;
```

Nu uitaţi să testaţi funcţiile specifice tabelelor (date într-o secţiune următoare a acestei pagini).

Tablouri cu dimensiune variabilă (Varrays)

Al doilea tip de colecţie precizat este cel al tablourilor cu dimensiune variabilă. Pentru a declara un varray se foloseşte următoarea sintaxă

```
TYPE nume_tip IS {VARRAY | VARYING ARRAY} (numar_elemente) OF tip_element [NOT NULL];
```

Iată un exemplu de utilizare a unui tablou cu dimensiune variabilă (din care se şterge un element după care se adaugă încă două elemente):

```
DECLARE
  TYPE varr IS VARRAY(5) OF varchar2(10);
  orase varr;
BEGIN
  orase := varr('Iasi', 'Bacau', 'Suceava', 'Botosani');
  DBMS_OUTPUT.PUT_LINE('Numar orase: '||orase.COUNT);
  orase.TRIM;
  FOR i IN orase.FIRST..orase.LAST LOOP
     DBMS_OUTPUT.PUT_LINE(orase(i));
  END LOOP;

  orase.EXTEND(2);
  orase(4):='Sibiu';
  orase(5):='Brasov';
  DBMS_OUTPUT.PUT_LINE('Dupa adaugare:');
  FOR i IN orase.FIRST..orase.LAST LOOP
     DBMS_OUTPUT.PUT_LINE(orase(i));
  END LOOP;
END;
```

Nu uitaţi să testaţi şi celelalte funcţii care pot fi aplicate variabilelor de tip varray (din lista cu funcţii aplicabile colecţiilor din această pagină).

Funcţii ce pot fi utilizate pentru colecţii

- FIRST - returnează valoarea cheii (sau indicele) primului element;
- LAST - returnează valoarea cheii (sau indicele) ultimului element;

- PRIOR(cheie) - returnează cheia elementului dinaintea celui dat ca parametru (cheie poate fi şi o valoare numerică dacă este vorba despre altceva decât tablouri asociative);
- NEXT(cheie) - returnează cheia elementului următor celui dat ca parametru (cheie poate fi şi o valoare numerică dacă este vorba despre altceva decât tablouri asociative);
- EXISTS(cheie) - returnează valoarea true dacă există o valoarea atribuită cheii (cheia poate fi şi poziţia într-o colecţie în cazul în care nu este vorba de tablouri asociative);
- COUNT - returnează numărul de elemente din colecţie;
- varray.LIMIT - câte elemente pot fi adăugatre în variabila de tip varray;
- EXTEND [(n[,i])] - pentru tipul tabel şi varray: pentru a adăuga n poziţii în structură (eventual toate având valoarea elementului de pe poziţia i). În cazul în care nu există nici un parametru, se extinde cu un singur element. Nu se aplică tabelelor asociative;
- TRIM [(n)] - şterge n elemente de la sfârştiul unei variabile de tip tabel sau dintr-un varray (nu este şi pentru tablouri asociative). În cazul în care n nu este dat, se şterge ultimul element;
- DELETE [(n,[m])]- şterge fie toate elementele (când nu are parametru), fie elementul de pe poziţia n, fie elementele de pe poziţiile n, n+1, ... m. Nu se aplică variabilelor de tip varray.

Tipul înregistrare (record)

Tipul înregistrare permite cumularea mai multor valori (nu neapărat de acelaşi tip) într-un singur tip - tipul înregistrării. Fiecare câmp al unei înregistrări are un nume (prin care poate fi identificat) şi un tip.

V-aţi mai întâlnit deja cu variabile de tip RECORD, atunci când le-aţi declarat ca şi ROWTYPE (în cadrul unui cursor sau pentru a prelua o linie din tabel). Să vedem care este sintaxa pentru definirea unei înregistrări în PL/SQL:

```
TYPE nume_tip IS RECORD (nume_camp tip_camp[,nume_camp tip_camp]...);
```

Exemplu:

```
CREATE TABLE minions (culoare varchar2(20), numar_ochi number(3), nume varchar2(20));
 /
DECLARE
   TYPE minion IS RECORD(
      culoare varchar2(20) := 'Galben',
      numar_ochi number(3),
      nume varchar2(20)
      );
   v_minion minion;
BEGIN
   v_minion.culoare:='Galben';
   v_minion.numar_ochi := 2;
   v_minion.nume:='Kevin';
   INSERT INTO MINIONS VALUES V_MINION;
   DBMS_OUTPUT.PUT_LINE(v_minion.culoare);
END;
```

Exercitiu (5pt)

Adaugati o coloana noua la tabelul studenti cu denumirea “lista_medii” de tip nested table in care se vor adauga mediile semestriale ale studentilor (din anul 1 sem 1, an 1 sem 2, an 2 sem 1 etc.). Campul de medii va fi extins doar cat este necesar (nu va contine decat locatii pentru mediile existente si nu 6 pentru fiecare medie posibil existenta). Mediile semestriale le puteti calcula grupand notele studentului dupa anul si semestrul in care au fost tinute curusurile la care are note. Construiti o functie care pentru un anumit student returneaza cate medii are trecute in coloana “lista_medii”.

Tema (2pt)

Construiti si generati date (cel putin 200 de randuri) pentru o tabela cu minim 5 campuri de tipuri diferite si care sa se lege intr-o manieria la alegerea voastra de tabelele existente. (de ex: masinile studentilor, filmele preferate ale unui student, referinte bibliografice pentru cursuri, etc.). Incercati sa veniti cu propriile idei. Datele generate vor parea reale, nu siruri de caractere sau numere generate aleatoriu. [la fel cum noi am generat in scriptul de creare mai multe tabele, voi veti genera doar una suplimentara]
