Baze de date, IESC – Îndrumar de Laborator

Capitolul 5 – Elemente ale limbajului PL/SQL

Limbajul SQL a devenit limbajul standard pentru lucrul cu baze de date deoarece este flexibil, puternic şi uşor de învăţat. Datele stocate pot fi manipulate cu uşurinţă folosind câteva comenzi construite pe sintaxa limbii engleze, precum SELECT, INSERT, UPDATE, DELETE etc. [O_Doc].

PL/SQL reprezintă extensia procedurală a SQL, şi a fost introdus de Oracle prima dată în 1991, odată cu SGBD-ul Oracle 6.0, evoluând continuu cu apariţia versiunilor ulterioare.

Un limbaj procedural semnifică un limbaj de programare care permite controlul fluxului de execuţie prin instrucţiuni decizionale precum IF, cicluri (FOR), permite să declarăm variabile, să definim funcţii şi proceduri, să prelucrăm erorile de execuţie etc. PL/SQL permite folosirea tuturor comenzilor de manipulare a datelor din SQL şi utilizarea aceloraşi tipuri de date (NUMBER, VARCHAR2 etc.).

PL/SQL este un limbaj cu structura de bloc, adică programele pot fi împărţite în blocuri logice.

Un **bloc PL/SQL** este compus din trei secţiuni [IP]:

- **secţiunea declarativă** (opţională) conţine toate variabilele, constantele, cursoarele etc. la care se face referinţă în secţiunea executabilă sau chiar în secţiunea declarativă;
- **secţiunea executabilă** conţine instrucţiuni SQL pentru manipularea datelor din baza de date şi instrucţiuni PL/SQL pentru manipularea variabilelor, constantelor etc.;
- **secţiunea pentru tratarea erorilor** (opţională) specifică acţiunile ce vor fi efectuate când în secţiunea executabilă apar erori.

Un program PL/SQL trebuie să conţină cel puţin un bloc. Acesta are următoarea structură generală:

```
[ nume_bloc ]
 [ DECLARE
        instrucţiuni de declarare ]
 BEGIN
        instrucţiuni executabile  (SQL sau PL/SQL)
 [ EXCEPTION
        tratarea erorilor ]
 END [ nume_bloc ];
```

Blocurile pot fi: anonime (fără nume), neanonime, proceduri, funcţii, triggere etc. De asemenea, un bloc poate conţine mai multe sub-blocuri.

Cel mai simplu program PL/SQL:

```
SET  SERVEROUTPUT  ON  -- seteaza afisarea rezultatului pe ecran
BEGIN
  /* functia de afisare a textului 'Hello world !'  */
  DBMS_OUTPUT.PUT_LINE('Hello world !');
END;
/
```

Prima comandă, SET SERVEROUTPUT ON, setează afişarea rezultatului pe ecran, şi nu trebuie omisă dacă dorim această afişare. Afişarea se realizează cu funcţia PUT_LINE() aparţinând bibliotecii DBMS_OUTPUT.

Remarcaţi utilizarea comentariilor în codul de mai sus: un comentariu pe o singură linie este precedat de --, iar comentariile pe una sau mai multe linii se pot grupa cu /* …. */ ca în C. De asemenea, ultimul caracter special, /, marchează sfârşitul blocului, şi ne va permite ca să avem mai multe blocuri sau alte instrucţiuni SQL în aceeaşi fereastră de editare a SQL Developer.

Blocul poate fi rulat direct în SQL Developer sau SQL * Plus, sau se poate salva într-un fişier text cu extensia .SQL, de exemplu: *hello.sql*

Apoi, se poate rula în două moduri:

SQL> *start hello /*

Sau:

SQL> *@hello /*

Declararea unei variabile are următoarea sintaxă:

Nume_variabila [CONSTANT] Tip_de_date [NOT NULL] [:= | DEFAULT expresie]

Exemple:

```
v_nume   VARCHAR2(20);
-- sau, cu iniţializarea unei valori:
v_nume   VARCHAR2(20)  := ‘Popescu’;
v_data_nasterii   DATE;
v_CNP    NUMBER(13);
PI  CONSTANT  NUMBER(4,3) := 3.14;  -- constanta PI
```

Alt exemplu de bloc anonim, ceva mai complex:

```
SET  SERVEROUTPUT  ON
DECLARE
     salariu_lunar       NUMBER(4);
     zile_lucrate         NUMBER(2);
     plata_zilnica        NUMBER(6,2);
BEGIN
     salariu_lunar := 2000;
     zile_lucrate := 21;
     plata_zilnica := salariu_lunar / zile_lucrate;

     DBMS_OUTPUT.PUT_LINE('Salariul pe zi este ' || TO_CHAR(plata_zilnica)  || ' lei.');
EXCEPTION
     WHEN ZERO_DIVIDE THEN
        plata_zilnica := 0;
END;
/
```

Programul de mai sus conţine toate cele trei secţiunile specifice unui bloc PL/SQL (declarativă, executabilă şi de tratare a erorilor).

Remarcaţi utilizarea operatorului de atribuire :=

Atenţie:

Ca în toate limbajele de programare (Pascal, C, Java etc.), şi în PL/SQL există următoarele convenţii simple, după cum urmează:

- un program este compus din mai multe categorii **distincte** de „obiecte”:

  - *instrucţiuni:* cuvinte cheie ale limbajului (de exemplu: IF, FOR, PUT_LINE), care trebuie reţinute pe de rost. Acestea pot fi: comenzi (IF, FOR) sau funcţii cu argumente (PUT_LINE(…)).

  - *variabile:* un fel de containere sau „sertărașe” care pot stoca date. Atenţie: datele stocate depind de tipul de date declarat al variabilei (NUMBER, VARCHAR2 etc.). Remarcaţi că şi numele tipului de date este un cuvânt cheie al limbajului. Variabilele pot avea orice nume dorim (dar să înceapă cu un caracter sau cu underscore _).

  - *constante:* variabile care nu vor permite schimbarea valorii stocate, sau pur şi simplu valori numerice (ex: 1000) sau şiruri de caractere între apostrof (ex: **'**Ionescu**'**).

  E de la sine înţeles că pentru a evita orice confuzie, nu trebuie niciodată să denumim variabile cu nume de cuvinte cheie rezervate ale limbajului, precum: IF, FOR, TABLE, CREATE etc.

  În PL/SQL trebuie să avem grijă dacă folosim o variabilă care va stoca la un moment o valoare a unui atribut dintr-o linie a unei tabele, să nu denumim variabila tot cu numele coloanei respective. Putem aplica convenţia: v_Nume pentru a stoca un nume din coloana Nume.

- *ordinea de execuţie a instrucţiunilor este cea naturală*: de la stânga la dreapta şi de sus în jos, ca şi când am parcurge un text scris. Ordinea naturală poate fi schimbată prin instrucţiuni de decizie (IF) sau prin instrucţiuni repetitive (FOR).

Astfel, în exemplul de mai sus, variabila *plata_zilnica* poate fi calculată doar după ce mai sus am alocat valori (printr-o atribuire) pentru *salariu_lunar* şi *zile_lucrate*. La rândul lor, aceste atribuiri nu puteau fi executate dacă mai sus nu am fi declarat cele două variabile.

Exerciţii:

- verificaţi modul de funcţionare al tratării excepţiei prin atribuirea *zile_lucrate* = 0.

- scrieţi un program (bloc anonim) PL/SQL care să calculeze media aritmetică a trei variabile numerice *a* , *b* şi *c*. Valorile vor fi setate prin atribuiri. Afişaţi media cu 2 zecimale.

Instrucţiunea SELECT în PL/SQL

Sintaxa instrucţiunii SELECT în PL/SQL este:

```
SELECT lista_coloane INTO lista_variabile
        FROM lista_tabele
        [WHERE condiție]
        [alte clauze]
```

Clauza INTO precizează lista variabilelor care vor primi valori după executarea instrucţiunii.

Nu se pot prelua decât valori dintr-o singură linie rezultat la un moment dat (putem folosi SELECT …. INTO doar dacă instrucțiunea returnează o singură linie).

Numărul de variabile din *lista_variabile* trebuie să fie egal cu numărul de coloane care apar în *lista_coloane*.

Exemplu:

```
SET SERVEROUTPUT ON

DECLARE
   Prenume     VARCHAR2(20);
   Nume        VARCHAR2(10);
```

```
BEGIN
    SELECT first_name, last_name     INTO  Prenume,  Nume
          FROM Employees
          WHERE Employee_ID = 200;
    DBMS_OUTPUT.PUT_LINE('Nume si prenume: ' || Nume || '  ' ||  Prenume);
    EXCEPTION
       WHEN   NO_DATA_FOUND   THEN
              DBMS_OUTPUT.PUT_LINE('Nu exista date');
       WHEN   TOO_MANY_ROWS   THEN
              DBMS_OUTPUT.PUT_LINE('Mai multe linii');
       WHEN   OTHERS   THEN
               DBMS_OUTPUT.PUT_LINE('Alta exceptie');
END;
/
```

Notă: Observaţi construcţia blocului de tratare a excepţiilor, şi testaţi funcţionarea declanşând erori (prin setarea unui ID inexistent, sau înlocuirea egalităţii cu > 200).

Putem afişa numele unei erori (dacă nu cunoaştem identificatorul erorii ca în exemplul de mai sus) utilizând variabila Oracle *SQLERRM*, de exemplu puteţi înlocui blocul de tratare a excepţiilor de mai sus cu:

```
EXCEPTION
   WHEN   OTHERS   THEN
         DBMS_OUTPUT.PUT_LINE(SQLERRM);
```

Atributul %TYPE

Dacă dorim să declarăm variabile care să preia valori din tabele existente, în loc de a specifica tipul de date al coloanei respective, putem scrie doar numele coloanei şi %TYPE. De exemplu:

```
v_Nume   Employees.Last_name%TYPE;
v_Salariu  Employees.Salary%TYPE;
```

Astfel, tipul de date al noii variabile va fi identic cu cel al coloanei specificate.

De asemenea, putem declara şi noi variabile folosind tipul de date al unor variabile deja declarate:

```
v_numar        NUMBER(5,2);
v_altnumar     v_numar%TYPE;
```

Instrucţiuni condiţionale

Sintaxa instrucţiunii IF este următoarea:

```
IF  conditie1  THEN
        instructiuni1;
[ELSIF  conditie2  THEN
        instructiuni2;]
. . . . . .
[ELSE
        instructiuni_else;]
END IF;
```

Exemplu: Fie următorul program care rezolvă o ecuaţie a*x + b = 0. Dacă a ≠ 0 ecuaţia are soluţie, dacă a şi b sunt 0 ecuaţia e nedeterminată, iar dacă a = 0 şi b ≠ 0 ecuaţia nu are soluţie.

Pentru a scrie codul programelor care conţin instrucţiuni condiţionale este utilă conceperea mai întâi a schemei logice a programului (Fig. 1).

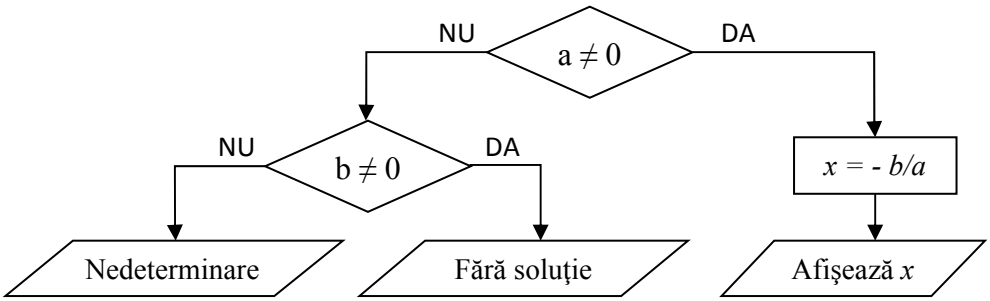

Fig. 1. Schema logică a programului

Din schema logică ne putem da seama că avem nevoie de două IF-uri (romburile din schemă), şi vedem care sunt instrucţiunile care se vor executa pe ramurile TRUE / FALSE ale acestor IF-uri.

Codul programului este următorul:

```
SET SERVEROUTPUT ON;

DECLARE
   a NUMBER  := 5;   b NUMBER  := 7;   x NUMBER; -- Schimbati aici valorile pentru a si b

BEGIN
  IF (a<>0) THEN
        x := -b/a;
        DBMS_OUTPUT.PUT_LINE(' x = ' || TO_CHAR(x));
  ELSE
        IF (b<>0) THEN
              DBMS_OUTPUT.PUT_LINE('Fara solutie');
        ELSE
               DBMS_OUTPUT.PUT_LINE('Nedeterminare');
        END IF;
  END IF;

END;
/
```

Sintaxa instrucţiunii CASE este următoarea:

```
CASE expresie
 WHEN 'val1' THEN
        Instructiuni1;
 WHEN 'val2' THEN
        Instructiuni2;
 ……..
 [ELSE
        Instructiuni_else;]
 END CASE;
```

Exemplu de utilizare:

```
SET SERVEROUTPUT ON;
DECLARE
   Nota NUMBER(2);
BEGIN
   Nota := 8;
   CASE Nota
      WHEN 10 THEN DBMS_OUTPUT.PUT_LINE('Excelent');
      WHEN 9   THEN DBMS_OUTPUT.PUT_LINE('Foarte bine');
      WHEN 8   THEN DBMS_OUTPUT.PUT_LINE('Bine');
      WHEN 7   THEN DBMS_OUTPUT.PUT_LINE('Satisfacator');
      WHEN 6   THEN DBMS_OUTPUT.PUT_LINE('Slab');
      WHEN 5   THEN DBMS_OUTPUT.PUT_LINE('De trecere');
      ELSE    DBMS_OUTPUT.PUT_LINE('Nota prea mica (<5) sau Inexistenta (>10)');
   END CASE;
END;
/
```

Instrucţiuni repetitive

Instrucţiunea LOOP are următoarea sintaxă:

```
LOOP
    Instructiuni;
    EXIT [WHEN conditie]; -- ieșire din buclă când condiţie = TRUE
    Instructiuni;
END LOOP;
```

Exemplu: Afişarea pătratelor numerelor de la 1 la 10:

```
DECLARE
   i NUMBER(2) :=1;
BEGIN
   LOOP
      DBMS_OUTPUT.PUT_LINE('i=' || TO_CHAR(i*i));
      EXIT WHEN i = 10;
      i := i+1;
  END LOOP;
END;
/
```

Sintaxa instrucţiunii WHILE:

```
WHILE  conditie  LOOP  -- testare iniţială a condiţiei de ieşire din ciclu
    Instructiuni;
END LOOP;
```

Ciclul din programul de mai sus devine:

```
WHILE (i <= 10) LOOP
    DBMS_OUTPUT.PUT_LINE('i=' || TO_CHAR(i*i));
    i := i+1;
END LOOP;
```

Sintaxa instrucţiunii FOR:

```
FOR  contor  IN [REVERSE]  valoare_initiala .. valoare_finala   LOOP
     Instructiuni;
END LOOP;
```

Clauza REVERSE e folosită dacă se doreşte parcurgerea inversă (de la valoarea finală a contorului la valoarea iniţială).

Ciclul de mai sus devine:

```
FOR i  IN  1 .. 10  LOOP
    DBMS_OUTPUT.PUT_LINE('i=' || TO_CHAR(i*i));
END LOOP;
```

Ca o particularitate a PL/SQL, contorul pentru FOR poate să nu fie declarat, fiind considerat de tip întreg. De asemenea, pasul este 1 şi nu poate fi schimbat.

Probleme propuse

1) Scrieţi un program PL/SQL care să calculeze salariul mediu al angajaţilor programatori (IT_PROG) din tabela Employees, apoi să-l compare cu salariul unui angajat (de exemplu cu ID = 200). Programul să afişeze: Salariul angajatului cu numele … este mai mare / mai mic decât salariul mediu.
2) Scrieţi un program PL/SQL în care introducând (prin atribuire) un cod angajat din tabela Employees, să afişeze anotimpul în care s-a angajat respectiva persoană. Puteţi utiliza funcţia: TO_CHAR(Data_ang,'MM') pentru a extrage luna ca un număr.
3) Creaţi (înaintea programului PL/SQL) o tabelă BONUS(v_id NUMBER, v_salariu NUMBER). Apoi, realizaţi un program PL/SQL care să insereze în tabela BONUS ID-ul şi salariul majorat cu 10% al angajaţilor din tabela Employees, pentru ID-urile: 101,102,…,110.
4) Realizaţi un program care să actualizeze tabela BONUS în felul următor: salariile mai mari de 9000 să fie diminuate cu 25%, iar salariile mai mici decât 6000 să fie majorate cu 10%.