SGBD - Lab 2

Bobu Dragos Andrei

9 martie 2022

Cuprins

I Structuri de control **2**

1 Structuri de control conditionale **2**

2 Bucle **3**

II Tipuri de cursoare **6**

1 Cursoarele implicite **6**
1.1 %FOUND . . . 6
1.2 %NOTFOUND . . . 6
1.3 %ROWCOUNT . . . 6
1.4 Exemplu . . . 7

2 Cursoare explicite **7**
2.1 Declararea cursorului . . . 7
2.2 Deschiderea cursorului . . . 8
2.3 Preluarea de linii . . . 8
2.4 Inchiderea cursorului . . . 8
2.5 Exemplu . . . 8

3 Exemple - cursoare explicite **9**

4 Declararea cursoarelor explicite cu clauza FOR UPDATE **10**

Partea I
Structuri de control

Pentru ca un limbaj să fie considerat limbaj de programare, pe lângă operaţia de atribuire trebuie să permită evaluarea de condiţii (şi executarea unui cod diferit în funcţie de rezultat) şi crearea de bucle.

1 Structuri de control conditionale

În PL/SQL verificarea condiţiilor (pot fi utilizate şi funcţii care intorc o valoarea booleană - ca de exemplu between, like, etc.) este realizată de comanda IF - THEN - ELSE având următoarea structură:

```
IF condiție THEN
        secvență de instrucțiuni ce vor fi rulate în cazul în care condiția este îndeplinită;
   ELSE
        secvență de instrucțiuni ce vor fi executate în cazul în care condiția nu este îndeplinită;
END IF;
```

La fel ca şi în alte limbaje de programare, secţiunea ELSE împreună cu blocul de instrucţiuni arondat ei pot să lipsească. Iata un exemplu al utilizării instrucţiunii IF-THEN-ELSE :

```
set serveroutput on;
accept i_numar prompt "Please enter your number: ";
DECLARE
   v_numar NUMBER(5);
   i_numar NUMBER(5);
BEGIN
   v_numar := &i_numar;
   IF (v_numar MOD 2 = 0)
      THEN
        DBMS_OUTPUT.PUT_LINE('Numarul '||v_numar||' este par.');
      ELSE
        DBMS_OUTPUT.PUT_LINE('Numarul '||v_numar||' este impar.');
   END IF;
END;
```

Dacă în secţiunea ELSE se doreşte testarea unei noi condiţii se poate utiliza comanda ELSIF (care la rândul ei va fi urmată de o secţiune THEN şi apoi de una ELSE/ELSIF). Atunci când sunt utilizate mai multe secţiuni de tipul ELSIF (atenţie, nu ELSEIF), în final se va pune o singură dată comanda ce identifică încheierea ultimului bloc (END IF). Spre exemplu, dacă dorim să testăm apartenenţa unui număr la un anumit interval de numere putem recurge la codul:

```
DECLARE
   v_numar NUMBER(5) := 50;
BEGIN
   IF (v_numar < 10)
       THEN
           DBMS_OUTPUT.PUT_LINE('Numarul este mai mic decat 10');
       ELSIF (v_numar > 80) THEN
           DBMS_OUTPUT.PUT_LINE('Numarul este mai mare decat 80');
       ELSE
           DBMS_OUTPUT.PUT_LINE('Numarul este in intervalul [10,80]');
   END IF;
END;
```

În cazul în care valoarea unei variabile este comparată cu mai multe valori posibile, se poate utiliza instrucţiunea CASE. Pe lângă compararea variabilei cu valorile predefinite, instrucţiunea de control CASE permite şi executarea unui cod default în cazul în care variabila nu are nici una dintre valorile aşteptate:

Vă aminitiţi ce facea funcţia DECODE în SQL ? Această funcţie poate fi simulată cu o instrucţiune de tip CASE. Executaţi următoarea comandă SELECT:

Comanda CASE poate să nu aibă un operand iniţial caz în care valorile ce urmează după WHEN trebuie să fie de tip boolean:

```
DECLARE
   numar NUMBER := 5;
BEGIN
   CASE (numar)
       WHEN 1 THEN DBMS_OUTPUT.PUT_LINE('Primul numar natural nenul.');
       WHEN 2 THEN DBMS_OUTPUT.PUT_LINE('Primul numar natural par.');
       ELSE
           DBMS_OUTPUT.PUT_LINE('Un numar mai mare sau egal cu 3');
   END CASE;
END;
```

```
SELECT nume, prenume,
CASE bursa
   WHEN 450 THEN 'Patru sute si cinci zeci'
   WHEN 350 THEN 'Trei sute si cinci zeci'
   WHEN 250 THEN 'Doua sute si cinci zeci'
   ELSE 'Fara bursa'
END
FROM studenti;
```

```
SELECT nume, prenume,
CASE
   WHEN bursa>300 THEN 'Bogat'
   ELSE 'Sarac'
END
FROM studenti;
```

2 Bucle

Buclele în PLSQL pot fi cu condiţie iniţială, cu condiţie finală şi care se vor repeta de un anumit număr de paşi (de tip for). Bucla cu condiţie iniţială este realizată prin utilizarea WHILE(condiţie) ... END LOOP, cea cu condiţie finală poate fi obţinută prin LOOP ... END LOOP (de fapt va produce o buclă infinită din care se poate ieşi cu o instrucţiune te tip EXIT iar bucla ce se va repeta de un anumit număr de paşi se realizează cu instrucţiunea FOR contor IN initia..final LOOP ... END LOOP;. Iată câteva exemple:

Primul exemplu va fi pentru bucla de tip WHILE:

```
set serveroutput on;
DECLARE
   v_contor INTEGER := 0;
BEGIN
   WHILE (v_contor < 10) LOOP
       v_contor := v_contor + 1;
       DBMS_OUTPUT.PUT_LINE(v_contor);
   END LOOP;
END;
```

Exemplul adaptat pentru structura LOOP împreună cu comanda de ieşire din buclă (comanda EXIT poate fi aplicată pentru orice structură repetitivă şi are forma EXIT WHEN (condiţie) sau EXIT etichetă WHEN (condiţie) atunci când sunt utilizate mai multe cicluri impricate şi se doreşte

ieşirea dintr-un anumit ciclu - de exemplu cel exterior.):

```
set serveroutput on;
DECLARE
   v_contor INTEGER := 0;
BEGIN
   LOOP
       v_contor := v_contor + 1;
       DBMS_OUTPUT.PUT_LINE(v_contor);
       EXIT WHEN v_contor = 10;
   END LOOP;
END;
```

Ultima metodă de a realiza o structură repetitivă în PLSQL este utilizând comanda FOR. Un exemplu în acest sens:

```
set serveroutput on;
DECLARE
   v_contor INTEGER := 0;
BEGIN
   FOR v_contor IN 1..10 LOOP
       DBMS_OUTPUT.PUT_LINE(v_contor);
   END LOOP;
END;
```

În cazul în care se doreşte parcurgerea în sens invers a valorilor din intervalul precizat, de va utiliza cuvântul REVERSE imediat după utilizarea lui IN (e.g. FOR v_contor IN REVERSE 1..10 LOOP).

Comanda EXIT poate fi invocată oricând în interiorul unei structuri repetitive pentru a forţa ieşirea din aceasta. În cazul în care doua structuri repetitive sunt imbricate şi se doreşte ieşirea din ambele structuri, prima structură repetitivă va fi etichetată:

```
set serveroutput on;
DECLARE
   v_contor1 INTEGER;
   v_contor2 INTEGER;
BEGIN
   <<eticheta>>
   FOR v_contor1 IN 1..5 LOOP
       FOR v_contor2 IN 10..20 LOOP
           DBMS_OUTPUT.PUT_LINE(v_contor1||'-'||v_contor2);
           EXIT eticheta WHEN ((v_contor1=3) AND (v_contor2=17));
       END LOOP;
   END LOOP;
END;
```

Comanda CONTINUE are rolul de a sări peste restul codului rămas în buclă şi de a începe o nouă iteraţie. De exemplu, în codul de mai sus, dacă dorim să nu fie afişată linia în care apare combinaţia 3-13, putem să introducem înaintea liniei de afişare următoarea linie:

```
CONTINUE WHEN ((v_contor1=3) AND (v_contor2=13));
```

Puteţi face în aşa fel încât să nu fie afişată ultima linie ? Cea cu combinaţia 3-17 ?

```
set serveroutput on;
BEGIN
   GOTO eticheta;
   DBMS_OUTPUT.PUT_LINE('Nu se va afisa.');
   <<eticheta>>
   DBMS_OUTPUT.PUT_LINE('Se va afisa.');
END;
```

O ultimă comandă ce poate afecta o buclă este GOTO. Aceasta face saltul la o anumită etichetă. Se poate iesi forţat din buclă utilizând GOTO, urmatorul exemplu este doar un simplu salt
.

Partea II
Tipuri de cursoare

Exista 2 tipuri de cursoare in SQL:

- Cursoare implicite (alocate de catre SQL SERVER cand user-ul executa operatii DML)
- Cursoare explicite (create de catre user)

Cursoarele ar putea fi comparate cu niste pointeri care pointeaza catre o zona privata de memorie din SQL

1 Cursoarele implicite

Deşi programatorul nu are acces direct la zona de memorie în care se află un cursor implicit, acesta poate obţine anumite informaţii despre execuţia interogării prin intermediul a trei atribute specifice. Cele trei atribute ale unui cursor implicit ce pot fi utilizate de programator sunt %FOUND, %NOTFOUND si %ROWCOUNT.

1.1 %FOUND

Returneaza TRUE daca exista o interogare, in caz contrar FALSE.

1.2 %NOTFOUND

Returneaza FALSE daca exista o interogare, in caz contrar TRUE.

1.3 %ROWCOUNT

Returneaza numarul de linii selectate de SELECT si afectate de macar un INSERT, UPDATE sau DELETE

Observatii:

- Atributele cursorului implicit intodeauna vor referi cea mai recenta interogare, indiferent de pozitia acesteia.
- Daca dorim reutilizarea valorilor unui atribut, acestea ar trebui stocate intr-o variabila.
- NOTDATAFOUND este exceptia pe care o sa o primim in cazul in care SELECT INTO esueaza sa returneze vreun rand.

1.4 Exemplu

Să considerăm următorul cod PL/SQL care modifică bursa tuturor studenţilor cu o bursa având valoarea mai mare decât 300 (şi îi adaugă 10 Ron):

```
DECLARE
   v_randuri INTEGER;
BEGIN
   UPDATE studenti set bursa = bursa + 10 WHERE bursa>300;
   IF(SQL%FOUND)
      THEN
         DBMS_OUTPUT.PUT_LINE('Am marit bursa la ' || SQL%ROWCOUNT || ' studenti.');
      ELSE
         DBMS_OUTPUT.PUT_LINE('Nimanui nu i s-a marit bursa.');
    END IF;
END;
```

2 Cursoare explicite

Cursoarele explicite sunt declarate şi utilizate în scripturile PL/SQL. Ele sunt utilizate atunci când interogările pe care le efectuăm asupra bazei de date vor returna mai mult de un singur rând (altfel valorile ar putea fi reţinute în variabile şi utilizate în acest mod) şi permit procesarea informaţiilor din rezultat linie cu linie.

Paşii pe care trebuie să îi urmăm când utilizăm un cursor sunt:

- Declararea
- Deschiderea cursorului
- Preluarea de linii
- Inchiderea cursorului

2.1 Declararea cursorului

Un cursor explicit se declară în zona de declaraţii a scriptului PL/SQL (deşi cursoare explicite pot fi utilizate şi direct în FOR). Pentru declarare se va utiliza următorul format:

```
DECLARE
   CURSOR   nume_cursor  IS  comanda_select;
   ....
```

În această construcţie, comanda_select este orice comandă de tip SELECT

2.2 Deschiderea cursorului

Deschiderea cursorului se realizează in secţiunea de cod aflată între comenzile BEGIN şi END. Acest lucru se face cu comanda OPEN urmată de numele cursorului.

```
OPEN nume_cursor
```

Urmari:

- alocarea de memorie pentru datele ce vor fi selectate de comanda SELECT
- executarea comenzii SELECT
- introducerea datelor în memorie
- poziţionarea pointerului pe primul rând returnat.

2.3 Preluarea de linii

În continuare, într-o secţiune LOOP (pentru că avem mai multe linii şi vrem ca fiecare să fie prelucrată individual), se apelează comanda FETCH urmată de numele cursorului, de cuvântul cheie INTO şi apoi de variabilele ce vor reţine valorile. Executarea comenzii FETCH va avea ca efect (pe lângă atribuirea variabilelor cu valorile din cursor) trecerea la următoarea linie returnată de comanda select. Din categoria ”Bune practici”,înainte de a intra în LOOP sau de a face primul fetch, aţi putea testa dacă selectul a returnat măcar o linie.

```
FECTH nume_cursor INTO v_var1, v_var2, v_var3;
```

2.4 Inchiderea cursorului

Pentru a testa daca s-a ajuns la ultimul rând (util pentru a ieşi din buclă) se va utiliza atributul NOTFOUND specific cursoarelor explicite. Valoarea atributului nume_cursor%NOTFOUND este aşadar true atunci când operaţia FETCH nu a mai fost capabilă să returneze informaţii din cursor. Acesta este, probabil, momentul în care se doreşte părăsirea buclei:

```
EXIT WHEN nume_cursor%NOTFOUND; --probabil va fi pus imediat după comanda FETCH
```

2.5 Exemplu

Un exemplu în acest sens (care afişează lista studenţilor bursieri) este următorul:

```
DECLARE
   CURSOR lista_studenti_bursieri  IS
       SELECT nume, prenume FROM studenti WHERE bursa IS NOT NULL;
   v_nume studenti.nume%type;
   v_prenume studenti.prenume%type;
BEGIN
    OPEN lista_studenti_bursieri;
    LOOP
        FETCH lista_studenti_bursieri INTO v_nume, v_prenume;
        EXIT WHEN lista_studenti_bursieri%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(v_nume||' '|| v_prenume);
    END LOOP;
    CLOSE lista_studenti_bursieri;
END;
```

3 Exemple - cursoare explicite

```
DECLARE
   CURSOR lista_studenti_bursieri  IS
       SELECT nume, prenume FROM studenti WHERE bursa IS NOT NULL;
   v_nume studenti.nume%type;
   v_prenume studenti.prenume%type;
BEGIN
    OPEN lista_studenti_bursieri;
    LOOP
        FETCH lista_studenti_bursieri INTO v_nume, v_prenume;
        EXIT WHEN lista_studenti_bursieri%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(v_nume||' '|| v_prenume);
    END LOOP;
    CLOSE lista_studenti_bursieri;
END;
```

```
DROP TABLE indivizi;
CREATE TABLE indivizi(nume Varchar2(10), prenume varchar2(10));
BEGIN
   INSERT INTO indivizi SELECT upper(nume), prenume FROM studenti;
   INSERT INTO Indivizi SELECT upper(nume), prenume FROM profesori;
END;
```

```
DECLARE
   CURSOR lista_studenti  IS
       SELECT * FROM studenti;
   v_std_linie lista_studenti%ROWTYPE;
BEGIN
    OPEN lista_studenti;
    LOOP
        FETCH lista_studenti INTO v_std_linie;
        EXIT WHEN lista_studenti%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(v_std_linie.nume||' '|| v_std_linie.data_nastere);
    END LOOP;
    CLOSE lista_studenti;
END;
```

```
DECLARE
   CURSOR lista_studenti  IS
       SELECT * FROM studenti;
BEGIN
    FOR v_std_linie IN lista_studenti LOOP
        DBMS_OUTPUT.PUT_LINE(v_std_linie.nume||' '|| v_std_linie.data_nastere);
    END LOOP;
END;
```

```
BEGIN
    FOR v_std_linie IN  (SELECT * FROM studenti) LOOP
        DBMS_OUTPUT.PUT_LINE(v_std_linie.nume||' '|| v_std_linie.data_nastere);
    END LOOP;
END;
```

```
DECLARE
   CURSOR lista_studenti_bursieri (p_bursa studenti.bursa%type, p_an studenti.an%type) IS
       SELECT nume, prenume FROM studenti WHERE bursa > p_bursa AND an > p_an;
   v_std_linie lista_studenti_bursieri%ROWTYPE;
BEGIN
    OPEN lista_studenti_bursieri (300,2); -- aceste valori pot fi calculate de codul PLSQL
    LOOP
        FETCH lista_studenti_bursieri INTO v_std_linie;
        EXIT WHEN lista_studenti_bursieri%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(v_std_linie.nume||' '|| v_std_linie.prenume);
    END LOOP;
    CLOSE lista_studenti_bursieri;
END;
```

4 Declararea cursoarelor explicite cu clauza FOR UPDATE

Dacă în timpul procesării cursorului dorim să modificăm o linie (sau chiar să o ştergem), după declararea cursorului se vor adăuga cuvintele ”FOR UPDATE”. Acest lucru va bloca accesul altor utilizatori ce doresc să scrie în tabela (sau tabelele) selectată de cursor dar va permite totuşi citirea acestor informaţii. După cuvintele cheie FOR UPDATE poate fi specificat un anumit câmp ce va poate fi modificat prin adăugarea cuvnântului cheie OF urmat de numele coloanei ce va ar putea fi modificată (în acest caz doar acea coloană va fi blocată).

În cazul în care se doreşte modificarea prin intermediul unui cursor a bazei de date şi altcineva deja are un cursor pentru modificare deschis pe aceeaşi tabelă, se poate specifica (în definiţia cursorului) un număr de secunde ce vor fi aşteptate după care se va reîncerca deschiderea cursorului. Dacă şi a doua oară eşuează, va fi returnată o eroare. Pentru a specifica numărul de secunde se adaugă la sfârşit cuvântul cheie WAIT urmat de numărul de secunde ce poate fi aşteptat. În cazul în care nu se doreşte aşteptarea, acest lucru va fi specificat prin prezenţa cuvântului cheie NOWAIT (în loc de ”WAIT n”).

Pentru a afecta rândul curent, după comanda ce indică modifcarea coloanei (sau ştergerea întregului rând) se vor adăuga cuvintele cheie ”WHERE CURRENT OF” urmate de numele cursorului în care se face updateul.

Iată, în continuare, un script pe care toţi studenţii care au avut vreodată ”probleme” la vreo materie şi-ar dori să îl execute pe serverul în care le sunt reţinute notele:

```
DECLARE
   CURSOR update_note IS
      SELECT * FROM note FOR UPDATE OF valoare NOWAIT;
BEGIN
   FOR v_linie IN update_note LOOP
       IF (v_linie.valoare < 5)
            THEN
                 UPDATE note SET valoare=5 WHERE CURRENT OF update_note;
       END IF;
   END LOOP;
END;
```

Observaţie: când facem updateul, acesta se realizează în tabela note şi nu în cursorul update_note.