Programare în *PL/SQL*

3. Controlul execuţiei unui bloc *PL/SQL*

PL/SQL este un limbaj cu structură de bloc, adică programele sunt compuse din blocuri care pot fi complet separate sau imbricate. Structura unui bloc poate fi obţinută combinând subprograme, pachete, blocuri imbricate. Blocurile pot fi folosite în utilitarele *Oracle*.

Pentru modularizarea unui program este necesară:

- gruparea logică a instrucţiunilor în blocuri;
- imbricarea de subblocuri în blocuri mai mari;
- descompunerea unei probleme complexe într-o mulţime de module logice şi implementarea acestora cu ajutorul blocurilor;
- plasarea în biblioteci a codului *PL/SQL* reutilizabil, de unde poate fi folosit de aplicaţii;
- depunerea codului într-un *server Oracle*, de unde este accesibil oricărei aplicaţii care interacţionează cu baza de date *Oracle*.

Un program *PL/SQL* poate cuprinde unul sau mai multe blocuri. Un bloc poate fi anonim sau neanonim.

Blocurile anonime sunt blocuri *PL/SQL* fără nume, care sunt construite dinamic şi sunt executate o singură dată. Acest tip de bloc nu are argumente şi nu returnează un rezultat. Ele sunt declarate într-un punct al aplicaţiei, unde vor fi executate (trimise motorului *PL/SQL*). În blocurile anonime pot fi declarate proceduri şi funcţii *PL/SQL*.

Blocurile anonime pot să apară într-un program ce lucrează cu precompilator sau în *SQL*Plus*. De obicei, blocurile anonime sunt plasate într-un fişier, iar apoi fişierul este executat din *SQL*Plus*. De asemenea, declanşatorii din componentele *Developer Suite* constau din astfel de blocuri.

Blocurile neanonime sunt fie blocuri cu nume (etichetate) construite static sau dinamic şi executate o singură dată, fie subprograme, pachete sau declanşatori.

Subprogramele sunt proceduri sau funcţii depuse în baza de date. Aceste blocuri sunt executate de mai multe ori şi, în general, nu mai sunt modificate după ce au fost construite. Procedurile şi funcţiile stocate sunt depuse pe *server*-ul *Oracle*, acceptă parametri şi pot fi apelate prin nume. Procedurile şi funcţiile aplicaţie sunt depuse într-o aplicaţie *Developer Suite* sau într-o bibliotecă.

Pachetele (stocate sau aplicaţie) sunt blocuri neanonime care grupează proceduri, funcţii, cursoare, tipuri, constante, variabile într-o unitate logică, în baza de date.

Declanşatorii sunt blocuri *PL/SQL* neanonime depuse în baza de date, care pot fi asociaţi bazei, iar în acest caz sunt executaţi implicit ori de câte ori apare un anumit eveniment declanşator (de exemplu, instrucţiuni *INSERT*, *UPDATE* sau *DELETE* ce se execută asupra unui tabel al bazei de date) sau pot fi asociaţi unei aplicaţii (de exemplu, declanşator *SQL*Forms*), ceea ce presupune că se execută automat, în funcţie de anumite condiţii sistem.

Structura unui bloc *PL/SQL*

Un bloc *PL/SQL* este compus din trei secţiuni distincte.

- Secţiunea declarativă (opţională) conţine declaraţii pentru toate variabilele, constantele, cursoarele şi erorile definite de utilizator la care se face referinţă în secţiunea executabilă sau chiar în cea declarativă. De asemenea, pot fi declarate subprograme locale care sunt vizibile doar în blocul respectiv.
- Secţiunea executabilă conţine instrucţiuni neprocedurale *SQL* pentru prelucrarea datelor din baza de date şi instrucţiuni *PL/SQL* pentru prelucrarea datelor în cadrul blocului.
- Secţiunea pentru tratarea erorilor (opţională) specifică acţiunile ce vor fi efectuate atunci când în execuţia blocului apar erori sau condiţii anormale.

Blocul *PL/SQL* are următoarea structură generală:

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

Dacă blocul *PL/SQL* este executat fără erori, invariant va apărea mesajul:

PL/SQL procedure successfully completed

Compatibilitate *SQL*

Din punct de vedere al compatibilităţii dintre *PL/SQL* şi *SQL*, se remarcă următoarele reguli de bază:

- *PL/SQL* furnizează toate comenzile *LMD* ale lui *SQL*, comanda *SELECT* cu clauza *INTO*, comenzile *LCD*, funcţiile, pseudocoloanele şi operatorii *SQL*;

- *PL/SQL* nu furnizează comenzile *LDD*.

Totuşi, în ultimele sale versiuni, *Oracle* permite folosirea dinamică a comenzilor *SQL*, utilizând tehnica oferită de *SQL* dinamic. În felul acesta, orice comandă *SQL* (inclusiv comandă *LDD*) poate să fie utilizată în *PL/SQL*.

Majoritatea funcţiilor *SQL* sunt disponibile în *PL/SQL*. Există însă funcţii specifice *PL/SQL*, cum sunt funcţiile *SQLCODE* şi *SQLERRM*. De asemenea, există funcţii *SQL* care nu sunt disponibile în instrucţiuni procedurale (*DECODE*, funcţiile grup), dar care sunt disponibile în instrucţiunile *SQL* dintr-un bloc *PL/SQL*. *SQL* nu poate folosi funcţii sau atribute specifice *PL/SQL*.

Funcţiile grup trebuie folosite cu atenţie, deoarece clauza *GROUP BY* nu are sens să apară în instrucţiunea *SELECT ... INTO*. *Oracle9i* introduce clauza *OVER*, care permite ca funcţia grup căreia îi este asociată să fie considerată o funcţie analitică (poate returna mai multe linii pentru fiecare grup).

Următoarele funcţii *SQL* nu sunt permise în *PL/SQL*: *WIDTH_BUCKET*, *BIN_TO_NUM*, *COMPOSE*, *DECOMPOSE*, *TO_LOB*, *DECODE*, *DUMP*, *EXISTSNODE*, *TREAT*, *NULLIF*, *SYS_CONNECT_BY_PATH*, *SYS_DBURIGEN*, *EXTRACT*.

Instrucţiuni *PL/SQL*

Orice program poate fi scris utilizând structuri de control de bază care sunt combinate în diferite moduri pentru rezolvarea problemei propuse. *PL/SQL* dispune de comenzi ce permit controlul execuţiei unui bloc. Instrucţiunile limbajului pot fi: iterative (*LOOP*, *WHILE*, *FOR*), de atribuire (:=), condiţionale (*IF*, *CASE*), de salt (*GOTO*, *EXIT*) şi instrucţiunea vidă (*NULL*).

***Observaţii*:**

- Comentariile sunt ignorate de compilatorul *PL/SQL*. Există comentarii pe o singură linie, prefixate de simbolurile „--“, care încep în orice punct al liniei şi se termină la sfârşitul acesteia. De asemenea, există comentarii pe mai multe linii, care sunt delimitate de simbolurile „/*“ şi „*/“. Nu se admit comentarii imbricate.
- Caracterul „;“ este separator pentru instrucţiuni.
- Atât operatorii din *PL/SQL*, cât şi ordinea de execuţie a acestora, sunt identici cu cei din *SQL*. În *PL/SQL* este introdus un nou operator („**“) pentru ridicare la putere.
- Un identificator este vizibil în blocul în care este declarat şi în toate subblocurile, procedurile şi funcţiile imbricate în acesta. Dacă blocul nu găseşte identificatorul declarat local, atunci îl caută în secţiunea declarativă a blocurilor care includ blocul respectiv şi niciodată nu

caută în blocurile încuibărite în acesta.

- Comenzile *SQL*Plus* nu pot să apară într-un bloc *PL/SQL*.
- În comanda *SELECT* trebuie specificate variabilele care recuperează rezultatul acţiunii acestei comenzi. În clauza *INTO*, care este obligatorie, pot fi folosite variabile *PL/SQL* sau variabile de legătură.
- Referirea la o variabilă de legătură se face prin prefixarea acesteia cu simbolul „:“.
- Cererea dintr-o comandă *SELECT* trebuie să returneze o singură linie drept rezultat. Atunci când comanda *SELECT* întoarce mai multe linii, apare eroarea *TOO_MANY_ROWS*, iar în cazul în care comanda nu găseşte date se generează eroarea *NO_DATA_FOUND*.
- Un bloc *PL/SQL* nu este o unitate tranzacţională. Într-un bloc pot fi mai multe tranzacţii sau blocul poate face parte dintr-o tranzacţie. Acţiunile *COMMIT*, *SAVEPOINT* şi *ROLLBACK* sunt independente de blocuri, dar instrucţiunile asociate acestor acţiuni pot fi folosite într-un bloc.
- *PL/SQL* nu suportă comenzile *GRANT* şi *REVOKE*, utilizarea lor fiind posibilă doar prin *SQL* dinamic.

Fluxul secvenţial de execuţie a comenzilor unui program *PL/SQL* poate fi modificat cu ajutorul structurilor de control: *IF*, *CASE*, *LOOP*, *FOR*, *WHILE*, *GOTO*, *EXIT*.

Instrucţiunea de atribuire

Instrucţiunea de atribuire se realizează cu ajutorul operatorului de asignare (:=) şi are forma generală clasică (*variabila* := *expresie*). Comanda respectă proprietăţile instrucţiunii de atribuire din clasa *LG3*. De remarcat că nu poate fi asignată valoarea *null* unei variabile care a fost declarată *NOT NULL*.

***Exemplu*:**

Următorul exemplu prezintă modul în care acţionează instrucţiunea de atribuire în cazul unor tipuri de date particulare.

```
DECLARE
  alfa  INTERVAL YEAR TO MONTH;
BEGIN
  alfa := INTERVAL '200-7' YEAR TO MONTH;
  -- alfa ia valoarea 200 de ani si 7 luni
  alfa := INTERVAL '200' YEAR;
  -- pot fi specificati numai anii
  alfa := INTERVAL '7' MONTH;
```

```
  -- pot fi specificate numai lunile
  alfa := '200-7';
  -- conversie implicita din caracter
END;
```

Instrucţiunea *IF*

Un program *PL/SQL* poate executa diferite porţiuni de cod, în funcţie de rezultatul unui test (predicat). Instrucţiunile care realizează acest lucru sunt cele condiţionale (*IF*, *CASE*).

Structura instrucţiunii *IF* în *PL/SQL* este similară instrucţiunii *IF* din alte limbaje procedurale, permiţând efectuarea unor acţiuni în mod selectiv, în funcţie de anumite condiţii. Instrucţiunea *IF-THEN-ELSIF* are următoarea formă sintactică:

IF *condiţie1* ***THEN***
  secvenţa_de_comenzi_1
[***ELSIF*** *condiţie2* ***THEN***
  secvenţa_de_comenzi_2]
...
[***ELSE***
  secvenţa_de_comenzi_n]
END IF;

O secvenţă de comenzi din *IF* este executată numai în cazul în care condiţia asociată este *TRUE*. Atunci când condiţia este *FALSE* sau *NULL*, secvenţa nu este executată. Dacă pe ramura *THEN* se doreşte verificarea unei alternative, se foloseşte ramura *ELSIF* (atenţie, nu *ELSEIF*) cu o nouă condiţie. Este permis un număr arbitrar de opţiuni *ELSIF*, dar poate apărea cel mult o clauză *ELSE*. Aceasta se referă la ultimul *ELSIF*.

***Exemplu*:**

Să se specifice dacă o sala este *mare*, *medie* sau *mica* după cum numărul fotografiilor expuse în sala respectivă este mai mare decât 200, cuprins între 100 şi 200 sau mai mic decât 100.

```
SET SERVEROUTPUT ON
DEFINE p_cod_sala = 753
DECLARE
  v_cod_sala  fotografie.cod_sala%TYPE :=
                                        &p_cod_sala;
  v_numar         NUMBER(3) := 0;
  v_comentariu    VARCHAR2(10);
BEGIN
```

```
6
  SELECT  COUNT(*)
  INTO    v_numar
  FROM    fotografie
  WHERE   cod_sala = v_cod_sala;
  IF v_numar < 100 THEN
    v_comentariu := 'mica';
  ELSIF v_numar BETWEEN 100 AND 200 THEN
    v_comentariu := 'medie';
  ELSE
    v_comentariu := 'mare';
  END IF;
  DBMS_OUTPUT.PUT_LINE('Sala avand codul '||
        v_cod_sala ||' este de tip '|| v_comentariu);
END;
/
SET SERVEROUTPUT OFF
```

Instrucţiunea *CASE*

Oracle9i furnizează o nouă comandă (*CASE*) care permite implementarea unor condiţii multiple. Instrucţiunea are următoarea formă sintactică:

[<<*eticheta*>>]
CASE *test_var*
  WHEN *valoare_1* ***THEN*** *secvenţa_de_comenzi_1*;
  WHEN *valoare_2* ***THEN*** *secvenţa_de_comenzi_2*;
  ...
  WHEN *valoare_k* ***THEN*** *secvenţa_de_comenzi_k*;
  [***ELSE*** *altă_secvenţă*;]
END CASE [*eticheta*];

Se va executa *secvenţa_de_comenzi_p*, dacă valoarea selectorului *test_var* este *valoare_p*. După ce este executată secvenţa de comenzi, controlul va trece la următoarea instrucţiune după *CASE*. Selectorul *test_var* poate fi o variabilă sau o expresie complexă care poate conţine chiar şi apeluri de funcţii.

Clauza *ELSE* este opţională. Dacă această clauză este necesară în implementarea unei probleme, dar totuşi lipseşte, iar *test_var* nu ia nici una dintre valorile ce apar în clauzele *WHEN*, atunci se declanşează eroarea predefinită *CASE_NOT_FOUND* (*ORA - 06592*).

Comanda *CASE* poate fi etichetată şi, în acest caz, eticheta poate să apară la sfârşitul clauzei *END CASE*. De remarcat că eticheta după *END CASE* este permisă numai în cazul în care comanda *CASE* este etichetată.

Selectorul *test_var* poate să lipsească din structura comenzii *CASE*, care în acest caz va avea următoarea formă sintactică:

[<<*eticheta*>>]
CASE
  WHEN *condiţie_1* ***THEN*** *secvenţa_de_comenzi_1*;
  WHEN *condiţie_2* ***THEN*** *secvenţa_de_comenzi_2*;
  ...
  WHEN *condiţie_k* ***THEN*** *secvenţa_de_comenzi_k*;
  [***ELSE*** *altă_secvenţă*;]
END CASE [*eticheta*];

Fiecare clauză *WHEN* conţine o expresie booleană. Dacă valoarea lui *condiţie_p* este *TRUE*, atunci este executată *secvenţa_de_comenzi_p*.

***Exemplu*:**

În funcţie de o valoare introdusă de utilizator, care reprezintă abrevierea zilelor unei săptămâni, să se afişeze (în cele două variante) un mesaj prin care este specificată ziua săptămânii corespunzătoare abrevierii respective.

Varianta 1:

```
SET SERVEROUTPUT ON
DEFINE p_zi = x
DECLARE
  v_zi  CHAR(2) := UPPER('&p_zi');
BEGIN
  CASE v_zi
    WHEN 'L'  THEN DBMS_OUTPUT.PUT_LINE('Luni');
    WHEN 'M'  THEN DBMS_OUTPUT.PUT_LINE('Marti');
    WHEN 'MI' THEN DBMS_OUTPUT.PUT_LINE('Miercuri');
    WHEN 'J'  THEN DBMS_OUTPUT.PUT_LINE('Joi');
    WHEN 'V'  THEN DBMS_OUTPUT.PUT_LINE('Vineri');
    WHEN 'S'  THEN DBMS_OUTPUT.PUT_LINE('Sambata');
    WHEN 'D'  THEN DBMS_OUTPUT.PUT_LINE('Duminica');
    ELSE DBMS_OUTPUT.PUT_LINE('este o eroare!');
  END CASE;
END;
/
SET SERVEROUTPUT OFF
```

Varianta 2:

```
SET SERVEROUTPUT ON
DEFINE p_zi = x
DECLARE
  v_zi  CHAR(2) := UPPER('&p_zi');
BEGIN
  CASE
    WHEN v_zi = 'L'  THEN
         DBMS_OUTPUT.PUT_LINE('Luni');
    WHEN v_zi = 'M'  THEN
         DBMS_OUTPUT.PUT_LINE('Marti');
    WHEN v_zi = 'MI' THEN
         DBMS_OUTPUT.PUT_LINE('Miercuri');
    WHEN v_zi = 'J'  THEN
         DBMS_OUTPUT.PUT_LINE('Joi');
    WHEN v_zi = 'V'  THEN
         DBMS_OUTPUT.PUT_LINE('Vineri');
    WHEN v_zi = 'S'  THEN
         DBMS_OUTPUT.PUT_LINE('Sambata');
    WHEN v_zi = 'D'  THEN
         DBMS_OUTPUT.PUT_LINE('Duminica');
    ELSE DBMS_OUTPUT.PUT_LINE('Este o eroare!');
  END CASE;
END;
/
SET SERVEROUTPUT OFF
```

Oracle8i a implementat suportul pentru expresii *CASE* care sunt permise numai în comenzi *SQL*. În *Oracle9i* poate fi utilizată o construcţie *CASE* într-o comandă *SQL* a unui bloc *PL/SQL*.

Expresia *CASE* are sintaxa similară comenzii *CASE*, dar clauzele *WHEN* nu se termină prin caracterul „;“, clauza *END* nu include cuvântul cheie *CASE* şi nu se fac atribuiri în clauza *WHEN*.

***Exemplu*:**

```
BEGIN
  FOR j IN (SELECT
              CASE valoare
                WHEN 1000 THEN 1100
                WHEN 10000 THEN 11000
```

```
                WHEN 100000 THEN 110000
                ELSE valoare
             END val
           FROM fotografie) LOOP
    DBMS_OUTPUT.PUT_LINE(j.val);
  END LOOP;
END;
```

Instrucţiuni iterative

Există trei tipuri de comenzi iterative: ciclarea simplă *LOOP*, ciclarea *WHILE* şi ciclarea *FOR*.

Acestea permit repetarea (condiţionată sau necondiţionată) execuţiei uneia sau mai multor instrucţiuni. Ciclurile pot fi imbricate pe mai multe niveluri. Ele pot fi etichetate, iar ieşirea din ciclu se poate realiza cu ajutorul comenzii *EXIT*.

Se utilizează:

- comanda *LOOP*, dacă instrucţiunile din cadrul ciclului trebuie să se execute cel puţin o dată;
- comanda *WHILE*, în cazul în care condiţia trebuie evaluată la începutul fiecărei iteraţii;
- comanda *FOR*, dacă numărul de iteraţii este cunoscut.

Instrucţiunea *LOOP* are următoarea formă sintactică:

LOOP
secvenţa_de_comenzi;
END LOOP;

Ciclarea simplă cuprinde o mulţime de comenzi incluse între cuvintele cheie *LOOP* şi *END LOOP*. Aceste comenzi se execută cel puţin o dată. Dacă nu este utilizată comanda *EXIT*, ciclarea poate continua la infinit.

***Exemplu*:**

Se presupune că a fost creată structura tabelului *org_tab*, constând din două coloane: *cod_tab* de tip *INTEGER*, ce conţine un contor al înregistrărilor şi *text_tab* de tip *VARCHAR2*, ce conţine un text asociat fiecărei înregistrări. Să se introducă 70 de înregistrări în tabelul *org_tab*.

```
CREATE TABLE org_tab
  (cod_tab INTEGER,
   text_tab VARCHAR2(50));


DECLARE
  v_contor BINARY_INTEGER := 1;
```

```
BEGIN
  LOOP
    INSERT INTO org_tab
    VALUES (v_contor, 'indicele ciclului');
    v_contor := v_contor + 1;
    EXIT WHEN v_contor > 70;
  END LOOP;
END;
```

Instrucţiunea repetitivă *WHILE* permite repetarea unei secvenţe de instrucţiuni, atâta timp cât o anumită condiţie specificată este adevărată.

Comanda *WHILE* are următoarea sintaxă:

WHILE *condiţie* ***LOOP***
  secvenţa_de_comenzi;
END LOOP;

Dacă variabilele care apar în condiţie nu se schimbă în interiorul ciclului, atunci condiţia rămâne adevărată şi ciclul nu se termină.

Când condiţia este evaluată ca fiind *FALSE* sau *NULL*, atunci secvenţa de comenzi nu este executată şi controlul trece la prima instrucţiune după *END LOOP*.

***Exemplu*:**

Exemplul precedent poate fi implementat utilizând comanda *WHILE* în următoarea manieră:

```
DECLARE
  v_contor BINARY_INTEGER := 1;
BEGIN
  WHILE v_contor <= 70 LOOP
    INSERT INTO org_tab
    VALUES (v_contor, 'indicele ciclului');
    v_contor := v_contor + 1;
  END LOOP;
END;
```

Instrucţiunea repetitivă *FOR* (ciclare cu pas) permite executarea unei secvenţe de instrucţiuni pentru valori ale variabilei *contor* cuprinse între două limite, *lim_inf* şi *lim_sup*. Dacă este prezentă opţiunea *REVERSE*, iteraţia se face (în sens invers) de la *lim_sup* la *lim_inf*.

Comanda *FOR* are sintaxa:

FOR *contor_ciclu* ***IN*** [***REVERSE***] *lim_inf..lim_sup* ***LOOP***
  secvenţa_de_comenzi;
END LOOP;

Variabila *contor_ciclu* nu trebuie declarată. Ea este neidentificată în afara ciclului şi implicit de tip *BINARY_INTEGER*. Pasul are implicit valoarea 1 şi nu poate fi modificat. Limitele domeniului pot fi variabile sau expresii, care să poată fi convertite la întreg.

***Exemplu*:**

În structura tabelului *fotografie* se va introduce un nou câmp (*stea*). Să se creeze un bloc *PL/SQL* care va reactualiza acest câmp, introducând o steluţă pentru fiecare 100000$ din valoarea unei fotografii al cărei cod este specificat.

```
ALTER TABLE  fotografie
ADD stea     VARCHAR2(2000);

DEFINE p_cod_fotografie = 7777
DECLARE
  v_cod_fotografie fotografie.cod_fotografie%TYPE :=
                                      &p_cod_fotografie;
  v_valoare    fotografie.valoare%TYPE;
  v_stea       fotografie.stea%TYPE := NULL;
BEGIN
  SELECT  NVL(ROUND(valoare/100000),0)
  INTO    v_valoare
  FROM    fotografie
  WHERE   cod_fotografie = v_cod_fotografie;
  DBMS_OUTPUT.PUT_LINE(v_valoare);
  IF v_valoare > 0 THEN
    FOR i IN 1..v_valoare LOOP
      v_stea := v_stea || '*';
    END LOOP;
  END IF;
  UPDATE  fotografie
  SET     stea = v_stea
  WHERE   cod_fotografie = v_cod_fotografie;
  COMMIT;
END;
```

Instrucţiuni de salt

Instrucţiunea *EXIT* permite ieşirea dintr-un ciclu. Ea are o formă necondiţională (ieşire fără condiţii) şi una condiţională. Controlul trece fie la prima instrucţiune situată după clauza *END LOOP* corespunzătoare, fie după instrucţiunea *LOOP* având eticheta *nume_eticheta*.

EXIT [*nume_eticheta*] [***WHEN*** *condiţie*];

Numele etichetelor urmează aceleaşi reguli ca cele definite pentru identificatori. Eticheta se plasează înaintea comenzii, fie pe aceeaşi linie, fie pe o linie separată. În *PL/SQL* etichetele se definesc prin intercalarea numelui etichetei între caracterele „<<“ şi „>>“ (<<*eticheta*>>).

***Exemplu*:**

```
DECLARE
  v_contor    BINARY_INTEGER := 1;
  raspuns     VARCHAR2(10);
  alt_raspuns VARCHAR2(10);
BEGIN
  …
  <<exterior>>
  LOOP
    v_contor := v_contor + 1;
    EXIT WHEN v_contor > 70;
    <<interior>>
    LOOP
      …
      EXIT exterior WHEN raspuns = 'DA';
      -- se parasesc ambele cicluri
      EXIT WHEN alt_raspuns = 'DA';
      -- se paraseste ciclul interior
      …
    END LOOP interior;
    …
  END LOOP exterior;
END;
```

Instrucţiunea *GOTO* determină un salt necondiţionat la o instrucţiune executabilă sau la începutul unui bloc care are eticheta specificată în comandă. Instrucţiunea are următoarea formă sintactică:

GOTO *nume_eticheta*;

Nu este permis saltul:

- în interiorul unui bloc (subbloc);
- în interiorul unei comenzi *IF*, *CASE* sau *LOOP*;
- de la o clauză a comenzii *CASE*, la altă clauză a aceleaşi comenzi;
- de la tratarea unei excepţii, în blocul curent;
- în exteriorul unui subprogram.

13

Instrucţiunea vidă

Instrucţiunea vidă (*NULL*) este folosită pentru o mai bună lizibilitate a programului. *NULL* este instrucţiunea care nu are nici un efect, marcând faptul că nu trebuie întreprinsă nici o acţiune. Nu trebuie confundată instrucţiunea *NULL* cu valoarea *null*!

Uneori instrucţiunea *NULL* este folosită într-o comandă *IF*, indicând faptul că pentru o anumită clauză *ELSIF* nu se execută nici o acţiune.