

3. PL/SQL – Blocuri. Variabile. Instrucțiuni.

CUPRINS

3. PL/SQL – Blocuri. Variabile. Instrucțiuni.	1
3.1. Limbajul <i>PL/SQL</i>	2
3.2. Structura unui bloc <i>PL/SQL</i>	2
3.2.1. Separator pentru instrucțiuni.....	3
3.2.2. Comentarii.....	3
3.3. Operatori.....	4
3.4. Variabile	4
3.5. Blocuri <i>PL/SQL</i>	5
3.6. Comenzi <i>SQL</i> în <i>PL/SQL</i>	6
3.6.1. Comanda <i>SELECT ... INTO</i>	7
3.6.2. Comenzile <i>INSERT, UPDATE, DELETE</i>	8
3.7. Instrucțiuni <i>PL/SQL</i>	9
3.7.1. Instrucțiunea de atribuire	9
3.7.2. Instrucțiunea condițională <i>IF</i>	9
3.7.3. Instrucțiunea condițională <i>CASE</i>	11
3.7.4. Instrucțiunea iterativă <i>LOOP</i>	13
3.7.5. Instrucțiunea iterativă <i>WHILE</i>	13
3.7.6. Instrucțiunea iterativă <i>FOR</i>	14
3.7.7. Instrucțiunea vidă.....	15
3.7.8. Instrucțiunea de salt <i>EXIT</i>	18
3.7.9. Instrucțiunea de salt <i>CONTINUE</i>	18
3.7.10. Instrucțiunea de salt <i>GOTO</i>	19
Bibliografie.....	20

3.1. Limbajul *PL/SQL*

- Atât *PL/SQL*, cât și *server-ul Oracle* utilizează același spațiu de memorie și prin urmare nu apar supraîncărcări datorate comunicațiilor dintre acestea.
- Este un limbaj cu structură de blocuri.
- Pentru modularizarea codului *PL/SQL* se pot folosi
 - blocuri anonime
 - subprograme (proceduri și funcții)
 - funcțiile pot fi invocate direct utilizând comenzi *SQL*
 - pachete
 - *trigger-i*
 - sunt un tip special de proceduri *PL/SQL* care se execută automat la apariția unui anumit eveniment.
- Blocuri anonime versus subprograme stocate

Blocuri anonime	Subprograme stocate
Blocuri PL/SQL fără nume	Blocuri PL/SQL cu nume
Compile de fiecare dată când aplicația este executată	Compile o singură dată
Nu sunt stocate în BD	Sunt stocate în BD
Nu pot fi invocate de alte aplicații	Pot fi invocate de alte aplicații
Nu întorc valori	Funcțiile trebuie să întoarcă o valoare
Nu acceptă parametrii	Acceptă parametrii

3.2. Structura unui bloc *PL/SQL*

- Blocul este unitatea de bază a unui program *PL/SQL*.
- Mai este denumit și modul.
- Blocul *PL/SQL* conține 3 secțiuni
 - secțiunea declarativă (opțională)
 - constante și variabile
 - tipuri de date locale
 - cursoare
 - excepții definite de utilizator

- subprograme locale (vizibile doar în bloc)
- secțiunea executabilă (obligatorie)
 - instrucțiuni *SQL* pentru prelucrarea datelor
 - instrucțiuni *PL/SQL*
 - trebuie să conțină măcar o instrucțiune
- secțiunea de tratare a excepțiilor (opțională)
 - instrucțiuni efectuate atunci când apare a numită excepție/eroare
- Sintaxa generală

```
[<<nume_bloc>>]  
[DECLARE  
    instrucțiuni de declarare]  
BEGIN  
    instrucțiuni executabile SQL sau PL/SQL  
[EXCEPTION  
    tratarea erorilor/excepțiilor]  
END [nume_bloc];
```

- Dacă blocul *PL/SQL* este executat fără erori va apărea mesajul:
anonymous block completed



Într-un bloc *PL/SQL* sunt permise instrucțiuni *SQL*Plus*?

3.2.1. Separator pentru instrucțiuni

- Caracterul „;” este separator pentru instrucțiuni.

3.2.2. Comentarii

- Comentariile sunt ignorate de compilatorul *PL/SQL*:
 - pe o singură linie
 - sunt prefixate de simbolurile „--”
 - încep în orice punct al liniei și se termină la sfârșitul acesteia
 - pe mai multe linii
 - sunt delimitate de simbolurile „/*” și „*/”

3.3. Operatori

- Operatorii din *PL/SQL* sunt identici cu cei din *SQL*.
- În *PL/SQL* este introdus operatorul „**“ pentru ridicare la putere.

Exemplul 3.1

```
SELECT POWER(3, 2)
FROM DUAL;
```

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(POWER(3, 2));
END;
```

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(3**2);
END;
```



Care secvență poate fi executată local? În ce condiții?

3.4. Variabile

- Stochează datele în același format binar intern ca și baza de date, astfel nefiind necesare conversii suplimentare.
- Pot fi declarate doar în zona declarativă a unui bloc, unde pot fi și inițializate.

Exemplul 3.2

```
DECLARE
    nume_utilizator    VARCHAR2(30);
    data_creatie_cont  DATE DEFAULT SYSDATE;
    numar_credite      NUMBER(4) NOT NULL := 1000;
    limita_inferioara_credite CONSTANT NUMBER := 100;
    limita_superioara_credite NUMBER := 5000;
    este_valid         BOOLEAN := TRUE;
```

- Fiecare variabilă se declară individual.

<pre>DECLARE -- declaratie -- incorecta i, j INTEGER; BEGIN NULL; END;</pre>		<pre>DECLARE /*declaratie corecta*/ i INTEGER; j INTEGER; BEGIN NULL; END;</pre>
--	--	--

- Li se pot atribui valori noi și pot fi utilizate în zona executabilă a blocului.
- Pot fi transmise ca parametrii subprogramelor *PL/SQL*.
- Pot fi declarate pentru a menține rezultatul obținut de un subprogram *PL/SQL*.
- Sunt vizibile în blocul în care sunt declarate și în toate subblocurile declarate în acesta.
- Dacă o variabilă nu este declarată local în bloc, atunci este căutată în secțiunea declarativă a blocurilor care includ blocul respectiv.

Exemplul 3.3

```
DECLARE
    v_principal VARCHAR2(50);
BEGIN
    v_principal := 'variabila din blocul principal';

    DECLARE
        v_secundar VARCHAR2(50) :=
            'variabila din blocul secundar';
    BEGIN
        DBMS_OUTPUT.PUT_LINE('<<Bloc Secundar>>');
        DBMS_OUTPUT.PUT_LINE('Folosesc '||v_principal);
        DBMS_OUTPUT.PUT_LINE('Folosesc '||v_secundar);
        v_secundar := 'Modific '||v_secundar;
        v_principal := 'Modific '||v_principal;
        DBMS_OUTPUT.PUT_LINE(v_secundar);
        DBMS_OUTPUT.PUT_LINE(v_principal);
    END;

    DBMS_OUTPUT.PUT_LINE('<<Bloc Principal>>');
    DBMS_OUTPUT.PUT_LINE(v_secundar);
    DBMS_OUTPUT.PUT_LINE(v_principal);
END;
```



Care comandă va genera o eroare?

3.5. Blocuri *PL/SQL*

Exemplul 3.4

- Bloc fără secțiune declarativă, fără secțiune de tratare a excepțiilor.

```
BEGIN
    DBMS_OUTPUT.PUT_LINE('SGBD');
END;
```

- Bloc cu secțiune declarativă, fără secțiune de tratare a excepțiilor.

```
DECLARE
  v_data DATE := SYSDATE;
BEGIN
  DBMS_OUTPUT.PUT_LINE(v_data);
END;
```

- Bloc cu secțiune declarativă, cu secțiune de tratare a excepțiilor.

```
DECLARE
  x NUMBER := &p_x;
  y NUMBER := &p_y;
BEGIN
  DBMS_OUTPUT.PUT_LINE(x/y);
EXCEPTION
  WHEN ZERO_DIVIDE THEN
    DBMS_OUTPUT.PUT_LINE('Nu poti sa imparti la 0!');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Alta eroare!');
END;
```

3.6. Comenzi SQL în PL/SQL

- Comenzi SQL care pot fi utilizate direct în PL/SQL
 - LMD (SELECT, INSERT, UPDATE, DELETE, MERGE)
 - Comanda SELECT poate fi utilizată doar cu clauza INTO
 - LCD (COMMIT, SAVEPOINT, ROLLBACK)
- Comenzi SQL care nu pot fi utilizate direct în PL/SQL
 - LDD (CREATE, ALTER, DROP)
 - LCD (GRANT, REVOKE)
 - Aceste comenzi nu pot fi folosite direct în PL/SQL deoarece sunt construite și executate la *runtime* (sunt dinamice). De aceea pot fi utilizate în PL/SQL doar cu *SQL Dinamic*.
 - *SQL Static* cuprinde comenzi care sunt stabilite la momentul în care programul este compilat. Acestea pot fi utilizate direct în PL/SQL.

3.6.1. Comanda *SELECT ... INTO*

Exemplul 3.5

```
DECLARE
    v_clasificare clasific_clienti.clasificare%TYPE;
    v_categorie   clasific_clienti.id_categorie%TYPE;
    v_client      clasific_clienti.id_client%TYPE
                := &p_client;
BEGIN
    SELECT clasificare, id_categorie
    INTO    v_clasificare, v_categorie
    FROM    clasific_clienti
    WHERE   id_client = v_client;
    DBMS_OUTPUT.PUT_LINE(v_categorie || ' '
                        || v_clasificare);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Nicio linie!');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('Mai multe linii!');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Alta eroare!');
END;
```



Comanda *SELECT ... INTO* trebuie să obțină exact o singură înregistrare?

Comanda *SELECT* poate fi utilizată în *PL/SQL* fără clauza *INTO*?

Exemplul 3.6

```
VARIABLE h_clasificare VARCHAR2
VARIABLE h_categorie   NUMBER

BEGIN
    SELECT clasificare, id_categorie
    INTO    :h_clasificare, :h_categorie
    FROM    clasific_clienti
    WHERE   id_client = 82;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Clientul nu exista!');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Alta eroare!');
END;
/

PRINT h_clasificare
PRINT h_categorie
```



Comanda *SELECT* poate fi utilizată într-o procedură *Microsoft T-SQL* fără clauza *INTO*?



În *Microsoft T-SQL* comanda *SELECT* cu clauza *INTO* funcționează la fel ca și în *PL/SQL*?



Comanda *SELECT* poate fi utilizată într-o procedură *MySQL* fără clauza *INTO*?



Ce opțiuni permite comanda *SELECT ... INTO* în *MySQL*?



În cadrul acestui curs vor fi considerate corecte și punctate doar soluțiile implementate în *PL/SQL*.

3.6.2. Comenzile *INSERT*, *UPDATE*, *DELETE*

Exemplul 3.7

```
BEGIN
  DELETE FROM clasific_clienti WHERE id_client=209;
  INSERT INTO clasific_clienti VALUES (209,2,1,null);
  UPDATE clasific_clienti
  SET    clasificare = 'D'
  WHERE  id_client = 209;
  COMMIT;
END;
```

Exemplul 3.8 - vezi curs



Un bloc *PL/SQL* poate conține mai multe comenzi *COMMIT*, *SAVEPOINT* sau *ROLLBACK*?

3.7. Instrucțiuni *PL/SQL*

- Instrucțiunea de atribuire (*:=*)
- Instrucțiuni condiționale (*IF*, *CASE*)
- Instrucțiuni iterative (*LOOP*, *WHILE*, *FOR*)
- Instrucțiuni de salt (*GOTO*, *EXIT*, *CONTINUE*)
- Instrucțiunea vidă (*NULL*)

3.7.1. Instrucțiunea de atribuire

variabila ***:=*** expresie;

- Variabilele care sunt declarate *NOT NULL* trebuie inițializate la declarare.
 - Codul din partea stângă a exemplului de mai jos va genera eroarea
PLS-00218: a variable declared NOT NULL must have an initialization assignment

Exemplul 3.9

```
DECLARE
  x NUMBER(2) NOT NULL;
BEGIN
  x:=2;
  DBMS_OUTPUT.PUT_LINE(x);
END;
```

```
DECLARE
  x NUMBER(2) NOT NULL :=2;
BEGIN
  DBMS_OUTPUT.PUT_LINE(x);
END;
```

3.7.2. Instrucțiunea condițională *IF*

```
IF expresie_booleană
  THEN comandă [comandă]...
[ELSIF expresie_booleană
  THEN comandă [comandă]...]
[ELSIF expresie_booleană
  THEN comandă [comandă]...]...
[ELSE comandă [comandă]...]
END IF;
```

- Comenzile din instrucțiune sunt executate dacă expresia booleană corespunzătoare are valoare *TRUE*. În caz contrar (expresia booleană are valoarea *FALSE* sau *NULL*), secvența nu este executată.
- Instrucțiunea *IF* poate conține mai multe clauze *ELSIF*, dar o singură clauză *ELSE*. Aceasta se referă la ultima clauză *ELSIF*.

Exemplul 3.10

```
DECLARE
  v_nr NATURAL;
  v_clasificare CHAR(1) := UPPER('&p_clasificare');
BEGIN
  SELECT COUNT(*) INTO v_nr
  FROM   clasific_clienti
  WHERE  clasificare = v_clasificare;

  IF v_nr=0
  THEN
    DBMS_OUTPUT.PUT_LINE('Nu exista clienti de ' ||
      'tipul ' || v_clasificare);
  ELSE
    DBMS_OUTPUT.PUT_LINE('Exista ' || v_nr ||
      ' clienti de tipul ' || v_clasificare);
  END IF;
END;
```

Exemplul 3.11

```
DECLARE
  v_nr NATURAL;
  v_clasificare CHAR(1) := UPPER('&p_clasificare');
BEGIN
  SELECT COUNT(*) INTO v_nr
  FROM   clasific_clienti
  WHERE  clasificare = v_clasificare
  AND    id_categorie = 1;

  IF v_nr=0
  THEN
    DBMS_OUTPUT.PUT_LINE('Nu exista clienti de ' ||
      'tipul ' || v_clasificare);
  ELSE
    IF v_nr = 1
    THEN
      DBMS_OUTPUT.PUT_LINE('Exista 1 client ' ||
        'de tipul ' || v_clasificare);
    ELSE
      DBMS_OUTPUT.PUT_LINE('Exista ' || v_nr ||
        ' clienti de tipul ' || v_clasificare);
    END IF;
  END IF;
END;
```

Exemplul 3.12 - vezi curs

3.7.3. Instrucțiunea condițională *CASE*

```
[<<eticheta>>]
CASE selector
  WHEN valoare_1_selector THEN secvență_comenzi_1;
  WHEN valoare_2_selector THEN secvență_comenzi_2;
  ...
  WHEN valoare_n_selector THEN secvență_comenzi_n;
  [ELSE secvență_comenzi;]
END CASE [eticheta];
```

- *Selectorul* este o expresie a cărei valoare este evaluată o singură dată și este utilizată pentru a selecta una dintre alternativele specificate prin clauzele *WHEN*.
 - Poate avea orice tip *PL/SQL*, cu excepția tipurilor *BLOB*, *BFILE* și tipuri definite de utilizator.
- Dacă valoarea selectorului este egală cu *valoare_k_selector*, atunci sunt executate comenzile cuprinse în *secvență_comenzi_k* și comanda *CASE* se încheie.
 - *Valoare_k_selector* poate avea orice tip *PL/SQL*, cu excepția tipurilor *BLOB*, *BFILE* și tipuri definite de utilizator.
- Secvența de comenzi din clauza *ELSE* este executată doar dacă selectorul nu are niciuna dintre valorile cuprinse în clauzele *WHEN*.
 - Clauza *ELSE* este opțională.
 - Dacă această clauză lipsește și selectorul nu are niciuna dintre valorile specificate în clauzele *WHEN*, atunci apare eroarea *CASE_NOT_FOUND*.

Exemplul 3.13

```
DECLARE
  v_nr NATURAL;
  v_clasificare CHAR(1) := UPPER('&p_clasificare');
BEGIN
  SELECT COUNT(*) INTO v_nr
  FROM   clasific_clienti
  WHERE  clasificare = v_clasificare
  AND    id_categorie = 1;

  CASE v_nr
    WHEN 0 THEN
      DBMS_OUTPUT.PUT_LINE('Nu exista clienti de ' ||
                           'tipul ' || v_clasificare);
    WHEN 1 THEN
      DBMS_OUTPUT.PUT_LINE('Exista 1 client ' ||
                           'de tipul ' || v_clasificare);
```

```
ELSE
    DBMS_OUTPUT.PUT_LINE('Exista ' || v_nr ||
        ' clienti de tipul ' || v_clasificare);
END CASE;
END;
```

- Comanda *CASE* permite o formă alternativă:

```
[<<eticheta>>]
CASE
    WHEN expresie_booleană_1 THEN secvență_comenzi_1;
    WHEN expresie_booleană_2 THEN secvență_comenzi_2;
    ...
    WHEN expresie_booleană_n THEN secvență_comenzi_n;
    [ELSE secvență_comenzi;]
END CASE [eticheta];
```

- Selectorul lipsește.
- Fiecare clauză *WHEN* conține o expresie booleană.
- Dacă expresie booleană *expresie_booleană_k* are valoarea *TRUE*, atunci sunt executate comenzile cuprinse în *secvență_comenzi_k* și comanda *CASE* se încheie.
- Secvența de comenzi din clauza *ELSE* este executată doar dacă nicio expresie booleană din clauzele *WHEN* nu are valoare *TRUE*.
- Și în acest caz clauza *ELSE* este opțională. Dacă această clauză lipsește și nicio expresie booleană din clauzele *WHEN* nu are valoare *TRUE*, atunci pare eroarea *CASE_NOT_FOUND*.

Exemplul 3.14 - **vezi curs**



Nu confundați comanda *CASE* din *PL/SQL* cu expresia *CASE* din *SQL*.

Exemplul 3.15 - **vezi curs**

Expresia *CASE* are sintaxa similară comenzii *CASE*, dar:

- clauzele *WHEN* nu se termină cu caracterul „;”;
- în clauzele *WHEN* nu se realizează atribuiri;
- clauza *END* nu include cuvântul cheie *CASE*.

Exemplul 3.16 - **vezi curs**

3.7.4. Instrucțiunea iterativă *LOOP*

```
LOOP
    secvență_de_comenzi;
END LOOP;
```

- Este denumită ciclare simplă.
- Comenzile incluse între cuvintele cheie *LOOP* și *END LOOP* sunt executate cel puțin o dată.
- Pentru a nu cicla la infinit trebuie utilizată comanda *EXIT*.

Exemplul 3.17

```
DECLARE
    cod_ascii NUMBER := ASCII('A');
BEGIN
    LOOP
        DBMS_OUTPUT.PUT(CHR(cod_ascii) || ' ');
        cod_ascii := cod_ascii + 1;
        -- EXIT;
        EXIT WHEN cod_ascii > ASCII('E');
        /* IF cod_ascii > ASCII('E') THEN EXIT;
           END IF; */
    END LOOP;

    DBMS_OUTPUT.NEW_LINE;
    DBMS_OUTPUT.PUT_LINE('Iesire cand am ajuns la ' ||
                          CHR(cod_ascii));

END;
```

- Pentru a transfera controlul iterației următoare se utilizează comanda *CONTINUE*.

Exemplul 3.18 - **vezi curs**

3.7.5. Instrucțiunea iterativă *WHILE*

```
WHILE condiție LOOP
    secvență_de_comenzi;
END LOOP;
```

- Este denumită ciclare condiționată.
- Comenzile incluse între cuvintele cheie *LOOP* și *END LOOP* sunt executate atâta timp cât condiția are valoarea *TRUE*.
- Condiția este evaluată la începutul fiecărei iterații.

Exemplul 3.19

```
DECLARE
  i NATURAL := 1;
BEGIN
  WHILE i<=10 LOOP
    DBMS_OUTPUT.PUT(i**2|| ' ');
    i := i + 1;
  END LOOP;

  DBMS_OUTPUT.NEW_LINE;
  DBMS_OUTPUT.PUT_LINE('Iesire cand i = '|| i );
END;
/

DECLARE
  i NATURAL := 1;
BEGIN
  WHILE i<=10 LOOP
    DBMS_OUTPUT.PUT(i**2|| ' ');
    i := i + 1;
    CONTINUE WHEN i<=5;
    DBMS_OUTPUT.NEW_LINE;
  END LOOP;

  DBMS_OUTPUT.PUT_LINE('Iesire cand i = '|| i );
END;
/
```

3.7.6. Instrucțiunea iterativă *FOR*

```
FOR contor IN [REVERSE] lim_inf..lim_sup LOOP
  secvență_de_comenzi;
END LOOP;
```

- Este denumită ciclare cu pas și este utilizată dacă numărul de iterații este cunoscut.
- Comenzile incluse între cuvintele cheie *LOOP* și *END LOOP* sunt executate pentru toate valorile întregi din intervalul [*lim_inf*, *lim_sup*].
- Dacă este utilizată opțiunea *REVERSE*, iterația se realizează în sens invers (de la *lim_sup* la *lim_inf*).
- Variabila *contor* nu trebuie declarată.
 - Este neidentificată în afara ciclului.
 - Implicit este de tip *BINARY_INTEGER*.
- Pasul are valoarea 1 (nu poate fi modificat).
- Limitele domeniului pot fi variabile sau expresii de tip întreg sau care pot fi convertite la întreg.

Exemplul 3.20

```
BEGIN
  FOR i IN 1..10 LOOP
    DBMS_OUTPUT.PUT(i**2 || ' ');
  END LOOP;
  DBMS_OUTPUT.NEW_LINE;
END;
/

BEGIN
  FOR i IN REVERSE 1..10 LOOP
    DBMS_OUTPUT.PUT(i**2 || ' ');
  END LOOP;
  DBMS_OUTPUT.NEW_LINE;
END;
/

BEGIN
  FOR i IN REVERSE 1..10 LOOP
    DBMS_OUTPUT.PUT(i**2 || ' ');
    CONTINUE WHEN i <= 5;
    DBMS_OUTPUT.NEW_LINE;
  END LOOP;
  DBMS_OUTPUT.NEW_LINE;
END;
/
```

3.7.7. Instrucțiunea vidă

```
NULL;
```

- Nu există nicio corespondență între valoarea *NULL* și instrucțiunea *NULL*.
- Nu realizează nicio operație.
- Plasează controlul următoarei comenzi.
- În *PL/SQL* anumite structuri trebuie să conțină cel puțin o comandă executabilă (de exemplu, instrucțiunea *IF* sau zona de gestiune a excepțiilor).

- Un bloc care nu are nicio acțiune.

Exemplul 3.21

```
DECLARE
  x NUMBER(2) NOT NULL :=2;
BEGIN
  NULL;
END;
```

- Captarea unei excepții pentru care nu se realizează nicio acțiune.

Exemplul 3.22

```
DECLARE
  v_clasificare clasific_clienti.clasificare%TYPE;
  v_categorie clasific_clienti.id_categorie%TYPE;
  v_client clasific_clienti.id_categorie%TYPE := 978;
BEGIN
  SELECT clasificare, id_categorie
  INTO v_clasificare, v_categorie
  FROM clasific_clienti
  WHERE id_client = v_client;
  DBMS_OUTPUT.PUT_LINE(v_categorie || ' '
                        || v_clasificare);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    -- DBMS_OUTPUT.PUT_LINE('Nicio linie!');
    NULL;

  WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE('Mai multe linii!');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Alta eroare!');
END;
```


- Salt la o etichetă după care nu urmează nicio instrucțiune executabilă (de exemplu urmează *END* sau *END IF*). Următoarele 3 exemple ilustrează opțiuni posibile.

Exemplul 3.23

```
DECLARE
  i INT(1);
BEGIN
  FOR i in 1..5 loop
    IF i=3 THEN
      GOTO eticheta;
    ELSE
      DBMS_OUTPUT.PUT_LINE('i='||i);
    END IF;
  END LOOP;

  <<eticheta>>
  --instrucțiunea NULL nu este necesara
  DBMS_OUTPUT.PUT_LINE('STOP cand i='||i);
END;
```

```
DECLARE
  j INT(1);
BEGIN
  FOR i in 1..5 loop
    j:=i;
    IF i=3 THEN
      GOTO eticheta;
    ELSE
      DBMS_OUTPUT.PUT_LINE('i='||i);
    END IF;
  END LOOP;

  <<eticheta>>
  --instrucțiunea NULL nu este necesara
  DBMS_OUTPUT.PUT_LINE('STOP cand i='||j);
END;
```

```
BEGIN
    FOR i in 1..5 loop
        IF i=3 THEN
            DBMS_OUTPUT.PUT_LINE('STOP cand i='||i);
            GOTO eticheta;
        ELSE
            DBMS_OUTPUT.PUT_LINE('i='||i);
        END IF;
    END LOOP;
    <<eticheta>>
    --instrucțiunea NULL este necesara
    NULL;
END;
```

- Este des utilizată în instrucțiunile condiționale pentru a sugera că într-un anumit caz nu se întâmplă nimic.

```
BEGIN
    FOR i in 1..5 loop
        IF i=3 THEN
            NULL;
        ELSE
            DBMS_OUTPUT.PUT_LINE('i='||i);
        END IF;
    END LOOP;
END;
```

3.7.8. Instrucțiunea de salt *EXIT*

```
EXIT [etichetă] [WHEN condiție];
```

- Permite ieșirea dintr-un ciclu. (Exemplele 3.17 și 3.18).
- Controlul trece fie la prima instrucțiune situată după clauza *END LOOP* corespunzătoare, fie după instrucțiunea *LOOP* având eticheta specificată.

3.7.9. Instrucțiunea de salt *CONTINUE*

```
CONTINUE [WHEN condiție];
```

- Permite transferarea controlului iterației următoare. (Exemplele 3.18, 3.19 și 3.20).

3.7.10. Instrucțiunea de salt *GOTO*

```
GOTO nume_eticheta;
```

- Permite saltul necondiționat la o instrucțiune executabilă sau la începutul unui bloc care are eticheta specificată în comandă. (Exemplul 3.23).
- 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 aceleiași comenzi;
 - de la tratarea unei excepții, în blocul curent;
 - în exteriorul unui subprogram.

Bibliografie

1. Connolly T.M., Begg C.E., *Database Systems: A Practical Approach to Design, Implementation and Management*, 5th edition, Pearson Education, 2005
2. Dollinger R., Andron L., *Baze de date și gestiunea tranzacțiilor*, Editura Albastră, Cluj-Napoca, 2004
3. Oracle and/or its affiliates, *Oracle Database Concepts*, 1993, 2017
4. Oracle and/or its affiliates, *Oracle Database Performance Tuning Guide*, 2013, 2017
5. Oracle and/or its affiliates, *Oracle Database SQL Language Reference*, 1996, 2017
6. Oracle and/or its affiliates, *Oracle Database PL/SQL Language Reference*, 1996, 2017
7. Oracle and/or its affiliates, *Oracle Database Administrator's Guide*, 2001, 2010
8. Oracle University, *Oracle Database 11g: PL/SQL Fundamentals, Student Guide*, 2009
9. Popescu I., Alecu A., Velcescu L., Florea (Mihai) G., *Programare avansată în Oracle9i*, Ed. Tehnică, 2004
10. Microsoft Online Documentation
<http://msdn.microsoft.com>
11. MySQL Online Documentation
<http://dev.mysql.com>