SQL - 8

ELEMENTE DE PL/SQL

STUD

MATR	NUME	AN	GRUPA	DATAN	LOC	TUTOR	PUNCTAJ	CODS
1456	GEORGE	4	1141A	12-MAR-82	BUCURESTI		2890	11
1325	VASILE	2	1122A	05-OCT-84	PITESTI	1456	390	11
1645	MARIA	3	1131B	17-JUN-83	PLOIESTI		1400	11
3145	ION	1	2112B	24-JAN-85	PLOIESTI	3251	1670	21
2146	STANCA	4	2141A	15-MAY-82	BUCURESTI		620	21
3251	ALEX	5	2153B	07-NOV-81	BRASOV		1570	21
2215	ELENA	2	2122A	29-AUG-84	BUCURESTI	2146	890	21
4311	ADRIAN	3	2431A	31-JUL-83	BUCURESTI		450	24
3514	FLOREA	5	2452B	03-FEB-81	BRASOV		3230	24
1925	OANA	2	2421A	20-DEC-84	BUCURESTI	4311	760	24
2101	MARIUS	1	2412B	02-SEP-85	PITESTI	3514	310	24
4705	VOICU	2	2421B	19-APR-84	BRASOV	4311	1290	24

SPEC si BURSA

CODS	NUME	DOMENIU
11	MATEMATICA	STIINTE EXACTE
21	GEOGRAFIE	UMANIST
24	ISTORIE	UMANIST

TIP	PMIN	PMAX	SUMA
FARA BURSA	0	399	
BURSA SOCIALA	400	899	100
BURSA DE STUDIU	900	1799	150
BURSA DE MERIT	1800	2499	200
BURSA DE EXCEPTIE	2500	9999	300

PL/SQL

- PL/SQL (Procedural Language/SQL) este o extensie procedurală a limbajului SQL pentru SGBD Oracle (extensie proprietară).
- Apariţia sa este datorată faptului că limbajul SQL este un limbaj de cereri şi nu un limbaj de programare

PROGRAM = BLOC

- Unitatea de execuţie este **blocul** care conţine:
  - cereri SQL
  - instrucţiuni PL/SQL.
- Ele sunt executate astfel:
  - Cererile SQL sunt trimise pentru a fi executate de serverul Oracle. Rezultatele acestora pot fi folosite apoi în instrucţiunile PL/SQL.
  - Instrucţiunile PL/SQL sunt executate de **Executorul de Instrucţiuni Procedurale** (Procedural Statement Executor) aflat în **Motorul PL/SQL** (PL/SQL Engine).
- Motorul PL/SQL este prezent în:
  - Serverul Oracle
  - Unelte Oracle

CE PUTEM SCRIE IN PL/SQL?

- Blocuri anonime (Anonymous blocks)
- Proceduri şi funcţii stocate (Stored procedures/functions)
- Proceduri şi funcţii folosite în aplicaţii (Application procedures/functions)
- Declanşatori ataşaţi bazei de date (Database triggers)
- Declanşatori folosiţi în aplicaţii (Application triggers)
- Pachete (Packaged procedures)

CUM ARATA UN BLOC?

DECLARE -- opțional

declarații de variabile, cursori, excepții definite de utilizator

BEGIN -- obligatoriu

cereri SQL

instrucțiuni PL/SQL

EXCEPTION -- opțional

acțiuni executate în caz de ridicare excepții

END; -- obligatoriu

DECLARE - BEGIN

- Este zona de declaraţii a blocului.
- Sunt declarate în principal variabile, cursori, excepţii utilizator necesare blocului, dar şi alte elemente (tipuri, proceduri, funcţii, etc.).
- Această parte este opţională, un bloc putând începe direct cu cuvântul cheie BEGIN (care este obligatoriu).
- Tipurile de date din SQL pot fi folosite (cu unele diferenţe) şi în PL/SQL.

BEGIN - EXCEPTION

- Corpul blocului, format din:
  - cereri SQL şi
  - instrucţiuni PL/SQL care descriu procesarea datelor în cadrul acestuia.
- Cuvântul cheie EXCEPTION poate lipsi, în care caz blocul se termină cu END.

EXCEPTION - END

- Acţiuni executate în caz de eroare (tratarea erorilor).
- Este o parte opţională a unui bloc (dar cuvântul cheie END care marchează sfârşit de bloc este obligatoriu).

EXEMPLU

```
DECLARE
  v_nume VARCHAR2(10);
BEGIN
  SELECT NUME
  INTO v_nume
  FROM SPEC
  WHERE CODS=11;
  DBMS_OUTPUT.PUT_LINE('Specializarea: '||
  v_nume);
EXCEPTION
  WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('A aparut o
  exceptie');
END;
```

REGULI DE SCRIERE

- Fiecare cerere SQL şi instrucţiune PL/SQL se încheie cu punct şi virgulă (;)
- DECLARE, BEGIN şi EXCEPTION nu se termină cu punct şi virgulă
- După END se pune punct şi virgulă
- Terminarea blocului se face cu punct (**.**)
- Rularea unui bloc (in SQL*Plus) se face cu **/** sau **r** (run) la promptul SQL>
- Ultimul bloc executat poate fi editat (ed in SQL*Plus)
- În caz de rulare cu succes, după eventualele mesaje tipărite de bloc apare:

```
PL/SQL procedure successfully completed
```

- In caz de rulare cu eroare netratată, se va afişa codul şi mesajul de eroare.

OBSERVATIE

- În exemplul de mai sus a fost folosită pentru procedura DBMS_OUTPUT.PUT_LINE.
- Aceasta face parte din pachetul DBMS_OUTPUT pus la dispoziţie de sistemul Oracle şi va fi utilizată şi în capitolele următoare. Sintaxa este:

```
DBMS_OUTPUT.PUT_LINE(expresie)
```

- Procedura evaluează valoarea expresiei, o converteşte la şir de caractere şi o tipăreşte.
- Cum implicit SQL*Plus nu afişează mesajele tipărite de server, pentru a le putea vedea opţiunea SQL*PLUS ***serveroutput*** trebuie trecută pe ON prin comanda:

```
SQL> SET SERVEROUTPUT ON
```

VARIABILE PL/SQL

- **Scalare:**
  - Aceste variabile pot conţine o singură valoare. Tipurile principale corespund celor care se pot asocia coloanelor unei tabele Oracle.
- **Compuse:**
  - Conţin mai multe valori, de exemplu o înregistrare (linie) din rezultatul unei cereri SQL
- **Referinţă:**
  - Conţin pointeri (referinţe) către alte elemente de program. Nu sunt tratate în acest curs (cu excepţia tipului REF CURSOR)
- **LOB (obiecte mari):**
  - Acestea conţin valori numite locatori (locators) care specifică locaţia unor obiecte mari (imagini de exemplu).

DECLARARE VARIABILE

Sintaxa unei declaraţii de variabile este:

```
Identificator [CONSTANT] tipdedate [NOT NULL]
  [:= | DEFAULT expresie]
```

- **Identificator -** Numele variabilei. Se aplică aceleaşi reguli ca la orice alt obiect SQL
- **[CONSTANT] -** Variabila nu-şi poate schimba valoarea. Constantele trebuiesc iniţializate la declarare.
- **Tipdedate -** Tipul variabilei (un tip scalar, compus, referinţă sau LOB).
- **[NOT NULL] -** Variabila nu poate fi nulă. Aceste variabile trebuiesc iniţializate la declarare.
- **[:= expresie] -** Iniţializare cu o expresie. Variabilele neiniţializate conţin iniţial valoarea NULL.
- **[DEFAULT expresie] -** Valoare implicită dată de o expresie (iniţializează variabila).

EXEMPLE

```
DECLARE
   v_nume          VARCHAR2(10)  := 'ION';
   v_datanasterii     DATE;
   v_cods          NUMBER(2) NOT NULL := 21;
   v_suma          CONSTANT NUMBER := 100;
```

ATRIBUIRI

- Sintaxa atribuirii este:

```
Identificator := expresie;
```

- Exemple:

```
v_nume := 'ION';
v_datanasterii := '12-APR-89';
v_data2 := to_date('14-04-79',
'DD-MM-YY');
v_numar := v_n1 + v_n2 * 1230;
```

TIPURI DE DATE

- Se pot folosi tipurile din SQL (cele pentru coloanele unei tabele)
- Pentru unele dintre acestea lungimea maxima admisibila este diferita in PL/SQL fata de SQL
- Exista in PL/SQL si tipul BOOLEAN care lipseste in SQL

ATRIBUTUL %TYPE

- Putem defini tipul unei variabile în funcţie de tipul altei variabile sau a unei coloane de tabelă folosind construcţia:

```
Variabila%TYPE
```

- sau

```
Tabela.Coloana%TYPE
```

- Exemplu:

```
DECLARE
   v_numar        NUMBER(5,2);
   v_altnumar           v_numar%TYPE;
   v_nume         stud.nume%TYPE;
```

- **Notă:** Dacă o variabilă este definită pe baza tipului unei coloane de tip NOT NULL ea nu moşteneşte acest atribut, putând conţine şi valoarea NULL.

DOMENIU DE VALABILITATE PENTRU NUME

- Un identificator definit într-un bloc este local acelui bloc şi global în toate subblocurile sale.
- În cazul în care un subbloc redefineşte un identificator, în subbloc poate fi folosit doar acesta din urmă.

EXEMPLU

```
DECLARE
   a NUMBER;
   b VARCHAR2(10);
BEGIN
   DECLARE
      a VARCHAR2(20);
      c NUMBER;
   BEGIN
    -- pot fi folositi a (VARCHAR2(20)), b si c
   END;
-- pot fi folositi a (NUMBER) si b
END;
```

OPERATORI

Operator	Semnificaţie
`**`	ridicare la putere
`*, /`	înmulţire, împărţire
`+, -, ||`	adunare, scădere, concatenare
`=, <, >, <=, >=, <>, !=, ~=, ^=, IS NULL, LIKE, BETWEEN, IN`	comparaţie
`NOT`	negaţie logică
`AND`	ŞI logic
`OR`	SAU logic

EXPRESII LOGICE

- În cazul unei expresii logice complexe evaluarea se opreşte în momentul în care valoarea expresiei este cunoscută. Exemplu:

```
DECLARE
   a NUMBER;
   b NUMBER;
BEGIN
      .   .   .
   if (a = 0) or ((b/a) < 1) then
      .   .   .
   END IF;
      .   .   .
END;
```

- În cazul în care variabila ***a*** are valoarea 0 expresia logică se evaluează la TRUE şi nu se mai testează a doua parte a ei care ar genera o excepţie de tip ZERO_DIVIDE.

VALORI NULE

- Comparaţiile care implică o valoare nulă returnează NULL.
- Negarea unei valori de NULL returnează NULL.
- O condiţie care se evaluează la NULL e tratată ca FALSE.
- În cazul expresiilor CASE (prezentate mai jos) nu se poate scrie WHEN NULL ci sintaxa este WHEN expresie IS NULL.
- Un şir de caractere de lungime 0 este tratat de PL/SQL ca o valoare de NULL.
- Operatorul de concatenare || ignoră valorile nule.
- Operatorul IN (listă de valori) ignoră valorile nule din listă
- Operatorul NOT IN (listă de valori) întoarce FALSE dacă lista conţine o valoare nulă.

EXPRESII CASE – FORMA 1

```
DECLARE
   codjudet VARCHAR2(2) := 'CJ';
  numejudet VARCHAR2(20);
BEGIN
  numejudet := CASE codjudet
   WHEN 'MM' THEN 'Maramures'
   WHEN 'CT' THEN 'Constanta'
   WHEN 'CJ' THEN 'Cluj'
  ELSE 'Alt judet'    END;
END;
```

EXPRESII CASE – FORMA 2

```
DECLARE
codjud VARCHAR2(2) := 'CJ';
numejud VARCHAR2(20);
BEGIN
numejud := CASE
  WHEN codjud = 'MM' THEN 'Maramures'
  WHEN codjud ='CT' THEN 'Constanta'
  WHEN codjud ='CJ' THEN 'Cluj'
  WHEN codjud LIKE 'A%' THEN `Incepe cu A'
  WHEN codjud IS NULL THEN 'Cod judet NULL'
  ELSE 'Alt judet'
END;
END;
```

FUNCTII: SQL vs. PL/SQL

- Funcţiile SQLCODE şi SQLERRM nu pot fi folosite în cereri SQL.
- Funcţiile DEREF, REF, VALUE, DECODE, DUMP, GREATEST, LEAST şi VSIZE nu pot fi folosite în instrucţiunile procedurale ale PL/SQL (în documentaţia Oracle9 nu mai sunt amintite în această categorie GREATEST şi LEAST).
- Funcţiile statistice şi analitice (ex: AVG, SUM, COUNT, MIN, MAX, etc.) sunt specifice SQL şi nu pot fi folosite în instrucţiunile procedurale ale PL/SQL.

DECIZIE

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

EXEMPLU

```
DECLARE
   v_nr   number(2);
BEGIN
  SELECT cods
  INTO v_nr
  FROM spec
  WHERE domeniu='STIINTE EXACTE';
  dbms_output.put_line('Numarul specializarii este: ` ||v_nr);
  IF (v_nr=21) then
      dbms_output.put_line('Este 21');
  ELSIF v_nr < 21 then
      dbms_output.put_line('Mai mic decat 21');
  ELSE
      dbms_output.put_line('Mai mare decat 21');
  END IF;
EXCEPTION
  WHEN no_data_found THEN
    dbms_output.put_line('Nu exista');
  WHEN too_many_rows THEN
    dbms_output.put_line('Sunt mai multe');
  WHEN others THEN
    dbms_output.put_line('Eroare nespecificata');
END;
```

CICLURI: LOOP

```
LOOP
   Instructiuni;
         EXIT [WHEN conditie]; --
  iesire din ciclu
   Instructiuni;
END LOOP;
```

LOOP: EXEMPLU

```
DECLARE
  v_contor number(2) :=6;   v_cods number;   v_nume varchar2(10);
  v_dom varchar2(20);
BEGIN
LOOP
  BEGIN
   SELECT cods, nume, domeniu
   INTO v_cods, v_nume, v_dom
   FROM spec
   WHERE cods=v_contor;
   dbms_output.put_line('Cod '|| v_contor||  '  nume '||v_nume||
                        ' domeniu '||v_dom);
  EXCEPTION
    WHEN no_data_found THEN
             dbms_output.put_line('Nu exista codul '||
             v_contor);
  END; -- pentru subbloc
  v_contor := v_contor + 5;
  EXIT when v_contor > 40;
END LOOP;
EXCEPTION
WHEN others THEN
  dbms_output.put_line('Exceptie');
END;
```

CICLURI: FOR

```
FOR contor IN [REVERSE] val_init..val_fin LOOP
   Instructiuni PL/SQL si cereri SQL;
END LOOP;
```

FOR: EXEMPLU

```
DECLARE
  v_contor number(2);    v_cods number;
  v_nume varchar2(10);    v_dom varchar2(20);
BEGIN
FOR v_contor IN 20..25 LOOP
  BEGIN
    SELECT cods, nume, domeniu
    INTO v_cods, v_nume, v_dom
    FROM spec WHERE cods=v_contor;
    dbms_output.put_line('Cod '|| v_contor||
            ' nume '||v_nume||' domeniu '||v_dom);
  EXCEPTION
    WHEN no_data_found THEN
           dbms_output.put_line('Nu exista codul '||
  v_contor);
  END; -- pentru subbloc
END LOOP;
END;
```

CICLURI: WHILE

```
WHILE conditie LOOP
   Instructiuni PL/SQL si
    cereri SQL;
END LOOP;
```

WHILE: EXEMPLU

```
DECLARE
  v_contor number(2) :=6;   v_cods number;
  v_nume varchar2(10);   v_dom varchar2(20);
BEGIN
WHILE v_contor <=40 LOOP
  BEGIN
  SELECT cods, nume, domeniu
  INTO v_cods, v_nume, v_dom
  FROM spec
  WHERE cods=v_contor;
  dbms_output.put_line('Cod '|| v_contor||
   ' nume '||v_nume||' domeniu '||v_dom);
  EXCEPTION
  WHEN no_data_found THEN
    dbms_output.put_line('Nu exista codul '||
                         v_contor);
  END; -- pentru subbloc
  v_contor := v_contor + 5;
END LOOP;
EXCEPTION
WHEN others THEN
  dbms_output.put_line('Exceptie');
END;
```

ALEGERE: CASE

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

CASE: EXEMPLU

```
DECLARE
  v_dom varchar2(20);
BEGIN
  SELECT domeniu
  INTO v_dom
  FROM spec
  WHERE cods=21;
  CASE v_dom
    WHEN 'UMANIST' THEN
      dbms_output.put_line('UMAN');
    WHEN 'STIINTE EXACTE' THEN
      dbms_output.put_line(`REAL');
    ELSE
      dbms_output.put_line('ALTCEVA');
  END CASE;
END;
```

EXCEPTIA CASE NOT FOUND

- În cazul absenţei lui ELSE se adaugă implicit ridicarea unei excepţii predefinite:

```
ELSE RAISE CASE_NOT_FOUND;
```

- Exemplu: domeniul pentru specializarea cu cod 21 (UMANIST) nu este printre etichetele CASE şi nu există ELSE:

```
. . . .
  SELECT domeniu
  INTO v_dom
  FROM spec
  WHERE cods=21;
  CASE v_dom
    WHEN 'STIINTE EXACTE' THEN
      dbms_output.put_line('EXACTE');
  END CASE;
. . . .
```

va semnala excepţia: *ORA-06592: CASE not found while executing CASE statement*.

VARIABILE COMPUSE

- Pentru descrierea tipurilor compuse se poate folosi declaraţia de tip. Aceasta are următoarea formă:

```
TYPE denumire_tip descriere_tip;
```

- Cu ajutorul ei vom putea defini şi tipurile compuse care se pot folosi în PL/SQL:
  - Înregistrare (RECORD)
  - Tablou asociativ (TABLE .. INDEX BY)
  - Tabelă (Nested TABLE)
  - Tablou de dimensiune variabilă (VARRAY)

INREGISTRARI

- Înregistrările din PL/SQL sunt similare înregistrărilor din limbajul Pascal sau structurilor din C. Sub un acelaşi nume sunt înmagazinate mai multe valori de tipuri diferite care se numesc câmpuri.
- Definirea unei înregistrări se poate face în două moduri:
  - Folosind o descriere a fiecărei componente a înregistrării. În acest caz o variabilă de acest tip poate să nu fie asociată cu o tabelă sau cu rezultatul unei cereri SQL.
  - Folosind %ROWTYPE se poate defini o înregistrare compatibilă cu o linie a unei tabele sau cu o linie a rezultatului unei cereri SQL.

INREGISTRARI - DESCRIERE

Sintaxa:

```
TYPE denumire_tip IS RECORD
(nume_camp1 tip_camp [NOT NULL]
                     [:= | DEFAULT expresie]
[, nume_camp2 ...]);
```

Referirea unui câmp al unei variabile de tip înregistrare se face prin construcţia:

```
nume_inregistrare.nume_camp
```

EXEMPLU

```
DECLARE
v_punctaj                NUMBER(4);
TYPE t_student IS RECORD
   (matr NUMBER(4) NOT NULL := 1234,
    nume stud.nume%type,
    punctaj     v_punctaj%type,
    data DATE DEFAULT SYSDATE);
v_student t_student;
BEGIN
  v_student.matr := v_student.matr + 1;
  v_student.nume := 'MARCEL';
  v_student.punctaj := 1400;
  .   .   .
END;
```

%ROWTYPE

- Sintaxa:

```
nume_variabila   nume_tabela%ROWTYPE;
```

- Exemplu:

```
DECLARE
v_student stud%ROWTYPE;
BEGIN
  SELECT *
  INTO v_student
  FROM stud     WHERE matr=1456;
  v_student.nume := 'MARCEL';
  v_student.punctaj := 1400;
  .   .   .
END;
```

EXCEPTII

- În cazul sistemelor de gestiune a bazelor de date erorile (dar nu numai) sunt numite şi ***excepţii***.
- Un bloc PL/SQL poate conţine între EXCEPTION şi END instrucţiuni care să trateze :
  - erorile returnate de serverul Oracle sau uneltele Oracle
  - situaţii definite de utilizator (excepţii definite de utilizator).
- Excepţiile pot avea asociat un nume (identificator).
- Unele dintre aceste nume sunt predefinite (pentru o parte dintre erorile returnate de server sau unelte).

EXCEPTII – cont.

- Apariţia unei excepţii este numită şi ***ridicarea unei excepţii*** (exception raising).
- În acest caz execuţia normală a blocului este întreruptă şi se face ***tratarea excepţiei***: se sare în zona EXCEPTION pentru execuţia instrucţiunilor asociate acesteia.

EXCEPTII – CARACTERISTICI(1)

- Apariţia unei excepţii, tratată sau nu, termină execuţia blocului. Toate instrucţiunile PL/SQL sau cererile SQL care apar după cererea sau instrucţiunea care a ridicat excepţia nu mai sunt executate.
- În cazul în care o excepţie are asociat un tratament, la ridicarea ei se trece la execuţia instrucţiunilor de tratare după care se iese din blocul curent fără eroare.

EXCEPTII – CARACTERISTICI(2)

- În cazul în care apare o excepţie care nu este tratată explicit în porţiunea dintre EXCEPTION şi END, blocul se termină cu eroare, excepţia putând fi tratată de blocul înconjurător (dacă există).
- În cazul excepţiilor definite de utilizator acestea trebuiesc ridicate explicit în zona dintre BEGIN şi EXCEPTION prin instrucţiunea PL/SQL RAISE.

EXEMPLU

```
DECLARE
  v_nume varchar2(10);
BEGIN
  SELECT nume INTO v_nume FROM stud
  WHERE cods>20;
END;
```

- În cazul în care cererea SQL returnează mai mult decât o singură înregistrare, valoarile rezultate nu pot fi stocate într-o singură variabilă scalară. Numele predefinit al acestei exceptii este TOO_MANY_ROWS
- O altă eroare pentru blocul de mai sus este şi nereturnarea nici unei valori de către cererea SQL (în cazul în care nici o specializare nu are codul mai mare ca 20). Numele predefinit al acestei exceptii este NO_DATA_FOUND

EXEMPLU – cont.

```
DECLARE
  v_nume varchar2(10);
BEGIN
  SELECT nume INTO v_nume FROM stud
  WHERE cods>20;
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;
```

SINTAXA

```
DECLARE
. . . .
BEGIN
. . . .
EXCEPTION
WHEN exceptie_11 OR exceptie_12 . . .THEN
  instructiuni_1
[WHEN exceptie_21 OR exceptie_22 . . . THEN
  instructiuni_2]
.  .  .  .  .  .
[WHEN OTHERS THEN
  Instructiuni_others]
END;
```

unde:

- **exceptie_xy:** identificatorul unei excepţii
- **instructiuni_x:** instrucţiuni PL/SQL sau cereri SQL = TRATARE EXCEPTIE
- **instructiuni_others** Tratare orice alta exceptie.

OBSERVATII

- WHEN OTHERS, dacă există, este întotdeauna ultima.
- După tratatea excepţiei apărute se iese din blocul curent.
- În consecinţă este inutilă prezenţa aceleiaşi excepţii pe mai mult de un WHEN

EXCEPTII - CATEGORII

- Excepţii Oracle predefinite.
- Excepţii Oracle care nu sunt predefinite (non-predefinite)
- Excepţii definite de utilizator

PREDEFINITE

- **ACCESS_INTO_NULL**
- **CASE_NOT_FOUND**
- **COLLECTION_IS_NULL**
- **CURSOR_ALLREADY_OPEN**
- **DUP_VAL_ON_INDEX**
- **INVALID_CURSOR**
- **INVALID_NUMBER**
- **LOGIN_DENIED**
- **NO_DATA_FOUND**
- **NOT_LOGGED_ON**

PREDEFINITE (2)

- **PROGRAM_ERROR**
- **ROWTYPE_MISMATCH**
- **SELF_IS_NULL**
- **STORAGE_ERROR**
- **SUBSCRIPT_BEYOND_COUNT**
- **SUBSCRIPT_BEYOND_LIMIT**
- **SYS_INVALID_ROWID**
- **TIMEOUT_ON_RESOURCE**
- **TOO_MANY_ROWS**
- **VALUE_ERROR**
- **ZERO_DIVIDE**

PREDEFINITE – cont.

- Excepţiile predefinite au deja un identificator asociat.
- Ele sunt ridicate automat la apariţie de către serverul Oracle sau uneltele Oracle.
- Exemplul anterior continea tratarea pentru doua erori predefinite

PREDEFINITE - SINTEZA

- NU se declară în zona DECLARE
- NU trebuiesc ridicate explicit prin RAISE
- Se tratează în zona EXCEPTION

NON-PREDEFINITE

- Reprezintă alte erori returnate de Oracle, în afara celor predefinite.
- Aceste erori se pot trata în două moduri:
  - Prin folosirea lui WHEN OTHERS
  - Prin declararea excepţiei (i se dă un nume) şi asocierea unui cod de eroare Oracle în zona DECLARE şi tratarea ei în zona EXCEPTION.

EXEMPLU

```
DECLARE
v_nume VARCHAR2(10);
v_numar NUMBER;
eroare_grupare EXCEPTION;
PRAGMA EXCEPTION_INIT(eroare_grupare, -979);
BEGIN
   SELECT NUME, COUNT(*) INTO v_nume, v_numar
   FROM STUD GROUP BY CODS;
EXCEPTION
   WHEN eroare_grupare THEN
     DBMS_OUTPUT.PUT_LINE('Eroare grupare');
   WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE('Alta exceptie');
END;
```

NON-PREDEFINITE - SINTEZA

- Se declară în zona DECLARE şi se asociază un cod de eroare
- NU trebuiesc ridicate explicit prin RAISE
- Se tratează în zona EXCEPTION

EXCEPTII UTILIZATOR

- Un program PL/SQL poate conţine excepţii definite de utilizator.
- În cazul când programul detectează o situaţie anormală el poate trece în zona de excepţii prin instrucţiunea PL/SQL

```
RAISE nume_exceptie
```

- Astfel de excepţii trebuiesc:
  - Declarate explicit
  - Ridicate explicit

EXEMPLU

```
DECLARE
prea_putini    EXCEPTION; v_nrstud     NUMBER;
BEGIN
    SELECT count(*)
    INTO v_nrstud
    FROM stud WHERE cods=21;
    IF (v_nrstud < 5) THEN
      RAISE prea_putini;
    END IF;
EXCEPTION
  WHEN prea_putini THEN
    DBMS_OUTPUT.PUT_LINE(`< 5 studenti');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Alta exceptie');
END;
```

EXCEPTII UTILIZATOR - SINTEZA

- Se declară în zona DECLARE (ca nume)
- Se ridică explicit cu RAISE
- Se tratează în zona EXCEPTION

SQLCODE SI SQLERRM

- În cazul în care dorim ca în zona EXCEPTION .. WHEN OTHERS să identificăm ce eroare a apărut putem folosi două funcţii care întorc codul numeric respectiv mesajul text al erorii respective.
- În funcţie de valoarea lor putem să tratăm diferenţiat diferite erori netratate anterior în blocul respectiv.
- Cele două funcţii sunt:
  - SQLCODE - întoarce codul numeric al excepţiei (erorii returnate)
  - SQLERRM - întoarce mesajul text asociat cu acea excepţie

CODURI RETURNATE

Valoare SQLCODE	Descriere
0	Nu a apărut nici o excepție
1	A apărut o excepție definită de utilizator
100	A apărut excepția NO_DATA_FOUND
Număr negativ	Cod eroare returnat de serverul Oracle

EXEMPLU

```
DECLARE
V_cod     NUMBER;
v_text    VARCHAR2(255);
BEGIN
.   .   .   .   .
EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
    v_cod := SQLCODE;
    v_text := SQLERRM;
    INSERT INTO ERORI VALUES(v_cod, v_text);
END;
```

DOMENIU VALABILITATE

- Ca si in cazul tuturor numelor:

```
DECLARE
    exceptia1 EXCEPTION; -
BEGIN
   DECLARE -- incepe un subbloc
   exceptia1 EXCEPTION; -- diferita de exceptia1 anterioara
    BEGIN
    ...
    IF ... THEN
            RAISE exceptia1; -- nu e tratata de bloc
            -- fiind redefinita in subbloc
    END IF;
    END; -- sfarsit subbloc
EXCEPTION
   WHEN exceptia1 THEN -- nu trateaza exceptia din subbloc
...
END;
```

RAISE_APPLICATION_ERROR

- Excepţiile de acest tip sunt similare erorilor Oracle non-predefinite dar trebuiesc ridicate explicit în zona executabilă sau în zona de excepţii folosind:

```
RAISE_APPLICATION_ERROR(cod_eroare,
  text_eroare)
```

- unde:
  - cod_eroare este un număr ales de utilizator între -20000 şi -20999
  - text_eroare este un text ales de utilizator
- Şi la aceste excepţii se poate folosi PRAGMA EXCEPTION_INIT pentru a le asigna un nume.

EXEMPLU

```
DECLARE
v_suma   NUMBER; suma_nula        EXCEPTION;
PRAGMA EXCEPTION_INIT(suma_nula, -20021);
BEGIN
SELECT suma INTO v_suma
FROM bursa WHERE pmin < 100;
IF v_suma IS NULL THEN
  RAISE_APPLICATION_ERROR(-20021,
    'Suma este nula');
END IF;
EXCEPTION
  WHEN suma_nula THEN
        dbms_output.put_line(SQLERRM);
END;
```

ALT EXEMPLU

```
DECLARE
v_suma    NUMBER;  v_coderoare NUMBER;
BEGIN
SELECT suma INTO v_suma
FROM bursa WHERE pmin < 100;
IF v_suma IS NULL THEN
  RAISE_APPLICATION_ERROR(-20021,
   'Suma este nula');
END IF;
EXCEPTION
  WHEN OTHERS THEN
   v_coderoare := SQLCODE;
   IF (v_coderoare = -20021) THEN
                dbms_output.put_line(SQLERRM);
  END IF;
END;
```

RETRATARE EXCEPTII

- În cazul în care se doreşte tratarea unei excepţii apărute într-un subbloc atât în acesta cât şi în blocul înconjurător, se procedează în modul următor:
  - Se prevede tratarea excepţiei respective în zona EXCEPTIONS a subblocului.
  - La sfârşitul secvenţei de instrucţiuni care tratează excepţia se plasează instrucţiunea RAISE fără parametri. Ea ridică excepţia curentă pentru blocul înconjurător.
  - Se prevede tratarea excepţiei respective în zona EXCEPTIONS a blocului.

EXEMPLU

```
DECLARE
exceptia1 EXCEPTION;
BEGIN
     DECLARE -- incepe un subbloc
     BEGIN
       ...
       IF ... THEN
         RAISE exceptia1;
       END IF;
     EXCEPTION -- tratare exceptii subbloc
       WHEN exceptia1 THEN
       -- instructiuni tratare in subbloc
       .  .  .  .
       RAISE;
     END; -- sfarsit subbloc
EXCEPTION
WHEN exceptia1 THEN -- retratare exceptia1 in bloc
...
END;
```

OBSERVATIE

- Exceptiile aparute:
  - In zona de declaratii
  - In zona de tratare a exceptiilor

Nu se pot trata in blocul in care au aparut ci in zona de exceptii a blocului inconjurator

CURSORI

- Pentru execuţia cererilor SQL prezente în blocurile PL/SQL şi pentru stocarea rezultatelor acestora Oracle foloseşte un spaţiu de lucru accesibil utilizatorului prin intermediul unui obiect numit ***cursor***.
- Acesta poate fi de exemplu folosit pentru a parcurge linie cu linie rezultatul şi pentru a afla câte linii are acesta.

FOLOSIRE (1)

- În zona de declaraţii se defineşte cursorul (nume, cerere SQL asociată).
- În zona executabilă (dupa BEGIN, deci inclusiv în zona de tratare a excepţiilor) se pot folosi instrucţiunile de lucru cu un cursor: OPEN, FETCH şi CLOSE.
- Pentru orice tip de cursor se pot folosi metodele asociate acestora:
  - %ISOPEN,
  - %NOTFOUND,
  - %FOUND şi
  - %ROWCOUNT

FOLOSIRE (2)

Declararea cursorului

```
DECLARE
CURSOR nume_cursor IS cerere_select;
```

Deschiderea cursorului

```
OPEN nume_cursor;
```

- se execută cererea SQL asociată cursorului.
- se aduce în zona de memorie a acestuia rezultatul cererii (o tabelă rezultat).
- se setează înregistrarea (linia) curentă la prima linie.

FOLOSIRE (3)

- **Încărcarea liniei curente**

```
FETCH nume_cursor INTO lista_variabile |
  variabila_inregistrare
```

- Se încarcă linia curentă într-o mulţime de variabile sau o singură variabilă înregistrare (având mai multe câmpuri).
- Corespondenţa este poziţională
- Se incrementează indicatorul liniei curente.
- Dacă nu s-a putut încărca o linie (mai sunt linii în rezultat) %FOUND întoarce FALSE.

FOLOSIRE (4)

- În cazul în care încărcarea se face într-o listă de variabile trebuie ca:
  - numărul de variabile din listă să fie acelaşi cu numărul de coloane din rezultatul cererii SELECT
  - tipurile variabilelor din listă să fie aceleaşi cu tipurile coloanelor din rezultatul cererii SELECT
- În cazul în care încărcarea se face într-o variabilă înregistrare trebuie ca:
  - numărul de câmpuri al variabilei să fie acelaşi cu numărul de coloane din rezultatul cererii SELECT
  - tipurile câmpurilor variabilei să fie aceleaşi cu tipurile coloanelor din rezultatul cererii SELECT
  - Se poate defini o variabilă de acelaşi tip cu liniile rezultatului folosind construcţia **nume_cursor%rowtype**.

ATRIBUTE CURSOR

- Atributele cursorului pot fi consultate după ce s-a făcut primul FETCH şi ele au următoarele valori:
- nume_cursor%ROWCOUNT: numărul de linii încărcate cu FETCH de la deschiderea cursorului.
- nume_cursor%FOUND: TRUE dacă ultimul FETCH a încărcat o nouă linie şi FALSE altfel.
- nume_cursor%NOTFOUND: TRUE dacă ultimul FETCH nu a încărcat o nouă linie şi FALSE altfel.
- nume_cursor%ISOPEN: TRUE dacă acel cursor e deschis şi FALSE altfel

EXEMPLU

```
DECLARE
CURSOR studenti IS SELECT * FROM STUD WHERE CODS = 21;
 v_stud  studenti%rowtype;
BEGIN
-- deschidere cursor
OPEN studenti;
LOOP
  -- incarcarea liniei curente
  FETCH studenti INTO v_stud;
  -- iesire cand nu mai sunt linii
  EXIT WHEN studenti%notfound;
  dbms_output.put_line(studenti%rowcount||
                       ' Student: '||v_stud.nume);
END LOOP;
-- inchidere cursor
CLOSE studenti;
END;
```

CICLURI FOR PENTRU CURSORI

- Pentru a uşura parcurgerea liniilor tabelei rezultat asociată unui cursor deschis în PL/SQL există un ciclu FOR special:

```
FOR nume_inregistrare IN nume_cursor LOOP
   instructiuni;
END LOOP
```

- Variabila înregistrare nu trebuie definită de utilizator, acest lucru făcându-se automat.
- Domeniul de valabilitate al acestei variabile este corpul ciclului.
- OPEN, FETCH şi CLOSE se execută automat
- Se execută câte un pas al ciclului pentru fiecare linie a rezultatului
- Ieşirea din ciclu se face automat la terminarea liniilor.

EXEMPLU

```
-- fara OPEN, FETCH, declarare v_stud
DECLARE
  CURSOR studenti IS
    SELECT * FROM STUD WHERE CODS = 21;
BEGIN
FOR v_stud IN studenti LOOP
  dbms_output.put_line(studenti%rowcount||
                ' Student: '||v_stud.nume);
END LOOP;  -- nu e necesar CLOSE
END;
```

CURSORI CU PARAMETRI

```
DECLARE
  CURSOR studenti (v_cods number :=21,
                   v_loc varchar2:='BUCURESTI') IS
  SELECT nume FROM stud
  WHERE cods = v_cods and loc = v_loc;
  i number;
BEGIN
FOR i IN 1..5 LOOP -- cod specializare = i*5+1
 dbms_output.put_line('Specializarea ' || (i*5+1));
 FOR v_stud in studenti((i*5+1), 'PLOIESTI') LOOP
    dbms_output.put_line(studenti%rowcount||
                         ' Student: '||v_stud.nume);
 END LOOP;  -- inchide automat cursorul
END LOOP; -- ciclul FOR i in 1..5
dbms_output.put_line('Cu parametrii impliciti');
FOR v_stud IN studenti LOOP
    dbms_output.put_line(studenti%rowcount||
                         ' Student: '||v_stud.nume);
END LOOP;  -- inchide automat cursorul
END;
```

WHERE CURRENT OF

```
DECLARE
CURSOR c_stud IS select nume, loc
             from stud FOR UPDATE; -- blocheaza liniile
                                   -- selectate
v_nume stud.nume%type; v_loc stud.loc%type;
v_nr number;
BEGIN
OPEN c_stud; v_nr := 0;
LOOP
  FETCH c_stud INTO v_nume, v_loc;
  EXIT WHEN c_stud%notfound;
  dbms_output.put_line(v_nume||' '||v_loc);
  UPDATE stud1
  SET punctaj = punctaj * 1.1
  WHERE CURRENT OF c_stud;  -- linia din tabela din care
   provine
                            -- linia curenta a cursorului
  v_nr := v_nr + 1;
END LOOP;
COMMIT;
```

PROCEDURI

- Sintaxa declarării unei proceduri este:

```
[CREATE [OR REPLACE]] - pt. proceduri stocate
PROCEDURE nume_procedura[(parametru[,
  parametru]...)]
[AUTHID {DEFINER | CURRENT_USER}] {IS | AS}
[PRAGMA AUTONOMOUS_TRANSACTION;]
[declaratii locale]
BEGIN
  instructiuni executabile sau NULL;
[EXCEPTION
  tratare erori]
END [nume_procedura];
```

PROCEDURI – cont.

- **[CREATE [OR REPLACE]]** - arată că procedura se defineşte ca obiect al bazei de date şi va fi stocată în aceasta. Altfel ea este parte a unui bloc PL/SQL.
- **[AUTHID {DEFINER | CURRENT_USER}]** - specifică dacă o procedură stocată se execută cu drepturile celui care a creat-o (valoare implicită) sau ale utilizatorului curent.
- **[PRAGMA AUTONOMOUS_TRANSACTION;]** - dacă se specifică această caracteristică, execuţia procedurii suspendă tranzacţia curentă care se reia după terminarea execuţiei acesteia.

PARAMETRI FORMALI

```
nume_parametru [IN | OUT [NOCOPY] | IN
  OUT [NOCOPY]]
   numetip [{:= | DEFAULT} expresie]
```

- **IN**, **OUT** şi **IN OUT** arată că este vorba de un parametru de intrare, de ieşire sau bidirecţional (default: IN).
- În corpul unui subprogram nu se poate asigna o valoare unui parametru transmis cu IN.
- **NOCOPY**: În mod implicit parametrii de tip IN sunt transmişi prin referinţă iar cei OUT şi IN OUT prin valoare. Se poate specifica în acest caz (OUT, IN OUT) transmiterea prin referinţă folosind NOCOPY.
- **numetip** este un nume de tip fără specificarea dimensiunii (deci VARCHAR2 de exemplu şi nu VARCHAR2(10).

EXEMPLU

```
PROCEDURE alt_tutor (cod_stud INTEGER, nou_tutor INTEGER)
IS
 tutor_actual number;
 fara_tutor EXCEPTION; este_el EXCEPTION;
BEGIN
  SELECT tutor INTO tutor_actual FROM stud
  WHERE matr = cod_stud;
  IF nou_tutor=cod_stud THEN
    RAISE este_el;
  ELSIF tutor_actual IS NULL THEN
    RAISE fara_tutor;
  ELSE
    UPDATE stud1 SET tutor = nou_tutor
    WHERE matr = cod_stud;
    dbms_output.put_line('Actualizat '||cod_stud);
  END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    dbms_output.put_line('Nu exista studentul');
  WHEN fara_tutor THEN
    dbms_output.put_line('Nu are tutor');
  WHEN este_el THEN
    dbms_output.put_line('Nu isi poate fi tutor');
 END alt_tutor;
```

PROCEDURA IN BLOC

```
DECLARE
 -- DEFINIRE PROCEDURA
 PROCEDURE alt_tutor (cod_stud INTEGER,
  nou_tutor INTEGER) IS
    . . .
 END alt_tutor;
BEGIN  -- blocul principal
 -- FOLOSIRE PROCEDURA
  alt_tutor(1325, 1645);
  alt_tutor(1456, 1645);
  alt_tutor(1645, 1645);
END;
```

FUNCTII

```
[CREATE [OR REPLACE ] ] -- FUNCTII STOCATE
FUNCTION nume_functie [ ( parametru [ ,
  parametru ]... )]
 RETURN tip_date_returnate
 [ AUTHID { DEFINER | CURRENT_USER } ]
{IS | AS}
[ PRAGMA AUTONOMOUS_TRANSACTION; ]
[declaratii locale]
BEGIN
  instructiuni executabile sau NULL;
[EXCEPTION
  tratare erori]
END [nume_functie];
```

EXEMPLU

```
FUNCTION alt_tutor (cod_stud INTEGER, nou_tutor INTEGER)
 RETURN VARCHAR2 IS
   tutor_actual number;
   fara_tutor EXCEPTION; este_el EXCEPTION;
 BEGIN
   SELECT tutor INTO tutor_actual FROM stud
   WHERE matr = cod_stud;
   IF nou_tutor=cod_stud THEN
     RAISE este_el;
   ELSIF tutor_actual IS NULL THEN
     RAISE fara_tutor;
   ELSE
     UPDATE stud1 SET tutor = nou_tutor
     WHERE matr = cod_stud;
     RETURN 'Actualizat '||cod_stud;
   END IF;
 EXCEPTION
   WHEN NO_DATA_FOUND THEN
     RETURN 'Nu exista studentul';
   WHEN fara_tutor THEN
     RETURN 'Nu are tutor';
   WHEN este_el THEN
     RETURN 'Nu isi poate fi tutor';
  END alt_tutor;
```

FUNCTIE IN BLOC

```
DECLARE
  -- DEFINIRE FUNCTIE
  FUNCTION alt_tutor (cod_stud INTEGER,
       nou_tutor INTEGER) RETURN VARCHAR2 IS
    . . . . .
  END alt_tutor;
BEGIN  -- FOLOSIRE FUNCTIE
  dbms_output.put_line(alt_tutor(1325, 1645));
  dbms_output.put_line(alt_tutor(1456, 1645));
  dbms_output.put_line(alt_tutor(1645, 1645));
END;
```

P & F STOCATE

- Pentru ca o funcţie stocată să fie apelabilă în cereri SQL trebuie să respecte următoarele restricţii:
  - Când este apelată într-o cerere SELECT funcţia nu trebuie să modifice nimic în tabelele bazei de date.
  - Când este apelată în INSERT, UPDATE sau DELETE funcţia nu poate regăsi sau modifica tabela actualizată de cererea SQL.
  - Când este apelată din SELECT, INSERT, UPDATE sau DELETE funcţia nu poate executa cereri de control pentru tranzacţii (ex.: COMMIT), cereri de control pentru sesiunea de lucru (ex.: SET ROLE) sau cereri de control sistem (ca ALTER SYSTEM). De asemenea nu poate executa cereri care se comit automat (ca de exemplu CREATE).

EXEMPLU: CREARE

```
CREATE OR REPLACE
FUNCTION SPECIALIZAREA (cod_stud INTEGER)
 RETURN VARCHAR2 IS
   V_SPEC SPEC.NUME%TYPE;
BEGIN
   SELECT SPEC.NUME
   INTO V_SPEC
   FROM STUD, SPEC
   WHERE STUD.CODS=SPEC.CODS AND
         STUD.MATR=cod_stud;
   RETURN V_SPEC;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
     RETURN 'Nu exista studentul';
   WHEN OTHERS THEN
     RETURN 'Exceptie';
END SPECIALIZAREA;
```

FOLOSIRE (1)

CERERE:

```
SELECT NUME, SPECIALIZAREA(MATR)
FROM STUD;
```

REZULTAT:

```
NUME       SPECIALIZAREA
---------- --------------------
GEORGE     MATEMATICA
VASILE     MATEMATICA
MARIA      MATEMATICA
ION        GEOGRAFIE
STANCA     GEOGRAFIE
ALEX       GEOGRAFIE
ELENA      GEOGRAFIE
ADRIAN     ISTORIE
FLOREA     ISTORIE
OANA       ISTORIE
MARIUS     ISTORIE
VOICU      ISTORIE
```

FOLOSIRE (2)

CERERE:

```
SELECT MATR, NUME, MATR+280, SPECIALIZAREA(MATR+280)
FROM STUD
```

REZULTAT:

MATR	NUME	MATR+280	SPECIALIZAREA
1456	GEORGE	1736	Nu exista studentul
1325	VASILE	1605	Nu exista studentul
1645	MARIA	1925	ISTORIE
3145	ION	3425	Nu exista studentul
2146	STANCA	2426	Nu exista studentul
3251	ALEX	3531	Nu exista studentul
2215	ELENA	2495	Nu exista studentul
4311	ADRIAN	4591	Nu exista studentul
3514	FLOREA	3794	Nu exista studentul
1925	OANA	2205	Nu exista studentul
2101	MARIUS	2381	Nu exista studentul
4705	VOICU	4985	Nu exista studentul

Starsitul capitolului

PROGRAMARE PL/SQL