Baze de date-Anul 3 (semestrul 1)
Laborator 1 PL/SQL

Tipuri de date scalare în PL/SQL. Declararea variabilelor. Instrucțiuni PL/SQL Blocuri

PL/SQL este extensia procedurală a limbajului SQL, cu trăsături specifice limbajelor de programare.

I. Tipuri de date scalare

Nu au componente interne (conțin valori atomice). Se împart în 5 clase.

- Tipurile de date ce stochează **valori numerice** cuprind
  - tipul *NUMBER* cu subtipurile *DEC*, *DECIMAL*, *DOUBLE PRECISION*, *FLOAT*, *INTEGER*, *INT*, *NUMERIC*, *REAL*, *SMALLINT*;
  - tipul *BINARY_INTEGER* cu subtipurile *NATURAL*, *NATURALN*, *POSITIVE*, *POSITIVEN*, *SIGNTYPE*; tipul *PLS_INTEGER*.
- Tipurile de date ce stochează **caractere** cuprind
  - tipul *VARCHAR2* cu subtipurile *STRING*, *VARCHAR*;
  - tipul de date *CHAR* cu subtipul *CHARACTER*;
  - tipurile *LONG*, *RAW*, *LONG RAW*, *ROWID*.
- Tipurile de date ce stochează **data calendaristică şi ora** cuprind tipurile *DATE*, *TIMESTAMP*, *TIMESTAMP WITH TIME ZONE*, *TIMESTAMP WITH LOCAL TIME ZONE*, *INTERVAL YEAR TO MONTH*, *INTERVAL DAY TO SECOND*.
- Tipurile de date **globalizare** ce stochează date *unicode* includ tipurile *NCHAR* şi *NVARCHAR2*.
- Tipul de date *BOOLEAN* stochează **valori logice** (*true*, *false* sau *null*).

Obs : Mai mute informații despre tipurile de date PL/SQL la

http://www.stanford.edu/dept/itss/docs/oracle/9i/appdev.920/a96624/03_types.htm

II. Variabile de legătură *PL/SQL*

Ø **O variabilă de legătură** (*bind variable*) este variabila care se declară într-un mediu gazdă şi este folosită pentru transferul la execuție al valorilor numerice sau de tip caracter în/din unul sau mai multe programe *PL/SQL*.

Ø Variabilele declarate în mediul gazdă sau în cel apelant pot fi referite în instrucțiuni *PL/SQL* dacă acestea nu sunt în cadrul unei proceduri, funcții sau pachet.

Ø În *SQL*Plus*, variabilele de legătură se declară folosind comanda ***VARIABLE*****,** iar pentru tipărirea acestora se utilizează comanda ***PRINT***. Ele sunt referite prin prefixare cu simbolul "**:**", pentru a putea fi deosebite de variabilele *PL/SQL* declarate.

III. Declararea variabilelor PL/SQL

- Identificatorii *PL/SQL* trebuie declarați înainte să fie referiți în blocul *PL/SQL*. Dacă în declarația unei variabile apar referiri la alte variabile, acestea trebuie să fi fost declarate anterior. Orice variabilă declarată într-un bloc este accesibilă blocurilor conținute sintactic în acesta.
- În declararea variabilelor în *PL/SQL* pot fi utilizate atributele ***%TYPE*** şi ***%ROWTYPE***, care

reprezintă tipuri de date implicite.

- Atributul *%TYPE* permite definirea unei variabile având tipul unei variabile declarate anterior sau tipul unei coloane dintr-un tabel.
- Atributul *%ROWTYPE* permite definirea unei variabile având tipul unei înregistrări dintr-un tabel.

Sintaxa declarării unei variabile este următoarea:

identificator [***CONSTANT***]{*tip_de_date* | *identificator****%TYPE*** |
*identificator****%ROWTYPE***} [***NOT NULL***]
[{:= | ***DEFAULT***} *expresie_PL/SQL*];

***Exemplu*:**

```
v_valoare          NUMBER(15) NOT NULL := 0;
v_data_achizitie DATE DEFAULT SYSDATE;
v_material         VARCHAR2(15) := 'Matase';
c_valoare          CONSTANT NUMBER := 100000;
v_stare            VARCHAR2(20) DEFAULT 'Buna';
v_clasificare      BOOLEAN  DEFAULT FALSE;
v_cod_opera        opera.cod_opera%TYPE;
v_opera            opera%ROWTYPE;
int_an_luna        INTERVAL YEAR TO MONTH :=
                   INTERVAL '3-2' YEAR TO MONTH; --interval de 3 ani si 2 luni
```

***Observații*:**

- Pentru pentru uşurința referirii se convine prefixarea numelor de variabile astfel:
- prefixarea cu litera *v* (*v_valoare*) pentru varibilele PL/SQL
- prefixarea cu litera *c* (*c_valoare*) pentru constante
- parametrii de substituție (variabilele de substituție din SQL*Plus) se prefixează cu litera *p*
- variabilele globale (bind variables) se prefixează cu *g*.
- Variabilele pot fi inițializate, iar dacă o variabilă nu este inițializată, valoarea implicită a acesteia este *NULL*. Dacă o variabilă este declarată *NOT NULL*, atunci ea va fi obligatoriu inițializată.
- Constantele trebuie inițializate când sunt declarate, altfel apare eroare la compilare.

!!! Afişarea valorii variabilelor se face cu ajutorul procedurilor:

DBMS_OUTPUT.PUT(sir_caractere);

DBMS_OUTPUT.PUT_LINE(sir_caractere);
Obs: se utilizează **SET SERVEROUTPUT ON** pentru activarea modului afişare.

IV. Instrucțiuni PL/SQL

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

Observații

- Comentariile sunt ignorate de compilatorul *PL/SQL*. Există două tipuri de comentarii:

- pe o singură linie, prefixate de simbolurile “--”, care încep în orice punct al liniei şi se termină la sfârşitul acesteia şi
- pe mai multe linii, care sunt delimitate de simbolurile “/*” şi “*/”.

- Caracterul “**;”** este separator pentru instrucțiuni.
- Operatorii din *PL/SQL*, ca şi ordinea de execuție a acestora, sunt identici cu cei din *SQL*. În *PL/SQL* este introdus un nou operator (**).
- Un identificator este vizibil în blocul în care este declarat şi în toate subblocurile, procedurile şi funcțiile încuibărite în acesta. Dacă blocul nu găseşte identificatorul declarat local, atunci îl caută în secțiunea 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 acțiunii 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 în *PL/SQL* prin prefixarea acestei variabile utilizând caracterul “**:**”.
- Cererea *SELECT* trebuie să întoarcă ca rezultat o singură linie. Dacă întoarce mai multe linii, atunci apare eroarea *TOO_MANY_ROWS*, iar dacă comanda nu găseşte date se generează eroarea *NO_DATA_FOUND*.

!!! Pentru evaluarea unei condiții logice care apare în comenzile limbajului, trebuie remarcat că orice expresie ce conține o valoare *null* este evaluată *null*. Singura excepție o constituie operatorul de concatenare.

1) Instrucțiunea de atribuire

variabila := expresie;

Obs: Nu poate fi asignată valoarea *NULL* unei variabile care a fost declarată *NOT NULL*.

2) Instrucțiunea *IF*

IF *condiție1* ***THEN***
  secvența_de_comenzi_1
[*ELSIF* *condiție2* ***THEN***
  secvența_de_comenzi_2]
…
[***ELSE***
  secvența_de_comenzi_n]
END IF;

Este permis un număr arbitrar de opțiuni *ELSIF*, dar poate fi cel mult o clauză *ELSE*. Aceasta se referă la ultimul *ELSIF*.

3) Instrucțiunea *CASE*

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

[<<*eticheta*>>]
CASE *test_var*
  WHEN *valoare_1* ***THEN*** *secvența_de_comenzi_1*;
  WHEN *valoare_2* ***THEN*** *secvența_de_comenzi_2*,
  …

WHEN *valoare_k* ***THEN*** *secvența_de_comenzi_k*;
[***ELSE*** *alta_secvența*;]
END CASE [*eticheta*];

Sau următoarea formă, în care fiecare clauză *WHEN* conține o expresie booleană.

[<<*eticheta*>>]
CASE
WHEN *condiție_1* ***THEN*** *secvența_de_comenzi_1*;
WHEN *condiție_2* ***THEN*** *secvența_de_comenzi_2*,
…
WHEN *condiție_k* ***THEN*** *secvența_de_comenzi_k*;
[***ELSE*** *alta_secvența*;]
END CASE [*eticheta*];

4) Instrucțiuni iterative

Instrucțiunile de ciclare pot fi:
- încuibărite pe multiple niveluri;
- etichetate;
- ieşirea din ciclare se poate realiza cu ajutorul comenzii *EXIT*.

a) LOOP
secvența_de_comenzi
END LOOP;

Comanda se execută cel puțin o dată. Dacă nu este utilizată comanda *EXIT*, ciclarea ar putea continua la infinit.

b) WHILE *condiție* ***LOOP***
secvența_de_comenzi
END LOOP;

Dacă condiția este evaluată ca fiind *FALSE* sau *NULL*, atunci secvența de comenzi nu este executată şi controlul trece la instrucțiunea după *END LOOP*.

Instrucțiunea repetitivă *FOR* (ciclare cu pas) permite executarea unei secvențe de instrucțiuni pentru valori ale variabilei *contor* cuprinse între două limite, *lim_inf* şi *lim_sup*. Dacă este prezentă opțiunea *REVERSE*, iterația se face (în sens invers) de la *lim_sup* la *lim_inf*.

c) FOR *contor_ciclu* ***IN*** [***REVERSE***] *lim_inf*..*lim_sup* ***LOOP***
secvența_de_comenzi
END LOOP;

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

5) Instrucțiuni de salt

Instrucțiunea *EXIT* permite ieşirea dintr-un ciclu. Controlul trece fie la prima instrucțiune situată după *END LOOP*-ul corespunzător, fie la instrucțiunea având eticheta *nume_eticheta*.

EXIT [*nume_eticheta*] [***WHEN*** *condiție*];

Numele etichetelor urmează aceleaşi reguli ca cele definite pentru identificatori. Eticheta se plasează înaintea comenzii, fie pe aceeaşi linie, fie pe o linie separată. Etichetele se definesc prin intercalare între “<<” şi “>>”.

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;
```

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 aceleaşi comenzi;
- de la tratarea unei excepții, în blocul curent;
- în exteriorul unui subprogram.

7) Instrucțiunea vidă. *NULL* este instrucțiunea care nu are nici un efect. Nu trebuie confundată instrucțiunea *NULL* cu valoarea *null*!

V. Blocuri PL/SQL

PL/SQL este un limbaj cu structura de **bloc**, adică programele sunt compuse din blocuri care pot fi complet separate sau încuibărite unul în altul.

Un program poate cuprinde unul sau mai multe blocuri. Un bloc poate fi anonim sau neanonim.

Ø **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.

Ø **Blocurile neanonime** sunt fie blocuri având un nume (etichetate), care sunt construite static sau dinamic şi sunt executate o singură dată, fie subprograme, pachete sau declanşatori..

Structura unui bloc *PL/SQL* este compusă din trei secțiuni distincte:

Blocul *PL/SQL* are următoarea structură generală:

[<<*nume_bloc*>>]
[***DECLARE***
  instrucțiuni de declarare]
BEGIN

instrucțiuni executabile (*SQL sau PL/SQL*)
[***EXCEPTION***
tratarea erorilor]
END [*nume_bloc*];

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

PL/SQL procedure successfully completed

Compatibilitate *SQL*

Din punct de vedere al compatibilității *PL/SQL versus SQL* există următoarele reguli de bază:

- *PL/SQL* furnizează toate comenzile *LMD* ale lui *SQL*, comanda *SELECT* cu clauza *INTO*, comenzile *LCD*, funcțiile, pseudo-coloanele şi operatorii *SQL*;
- *PL/SQL* nu furnizează comenzile *LDD*.
- Majoritatea funcțiilor *SQL* sunt disponibile în *PL/SQL*.

- Există funcții noi, specifice *PL/SQL*, cum sunt funcțiile *SQLCODE* şi *SQLERRM*.
- Există funcții *SQL* care nu sunt disponibile în instrucțiuni procedurale (de exemplu, ***DECODE*****, NULLIF, funcțiile grup**), dar care sunt disponibile în instrucțiunile *SQL* dintr-un bloc *PL/SQL*. *SQL* nu poate folosi funcții sau atribute specifice *PL/SQL*.

! Funcțiile grup trebuie folosite cu atenție, deoarece instrucțiunea *SELECT … INTO* nu poate conține clauza *GROUP BY*.

Exerciții

1. Care dintre urmatoarele declaratii nu sunt corecte si explicati de ce:

```
a) DECLARE
      v_id                NUMBER(4);
b) DECLARE
      v_x, v_y, v_z       VARCHAR2(10);
c) DECLARE
     v_birthdate          DATE NOT NULL;
d) DECLARE
     v_in_stock           BOOLEAN := 1;
e) DECLARE
      TYPE name_table_type IS TABLE OF VARCHAR2(20)
            INDEX BY BINARY_INTEGER;
     dept_name_table    name_table_type;
```

2. Determinati tipul de date al rezultatului in fiecare din atribuirile urmatoare:

```
a) v_days_to_go := v_due_date - SYSDATE;
b) v_sender := USER || ': '||TO_CHAR(v_dept_no);
c) v_sum := $100,000 + $250,000;
d) v_flag :=TRUE;
e) v_n1 := v_n2 > (2 * v_n3);
f) v_value :=NULL;
```

3. Se consideră următorul bloc PL/SQL:

```
<<bloc>>
DECLARE
   v_cantitate NUMBER(3) := 300;
   v_mesaj     VARCHAR2(255) := 'Produs 1';
```

```
    BEGIN
       <<subbloc>>
       DECLARE
            v_cantitate NUMBER(3) := 1;
            v_mesaj     VARCHAR2(255) := 'Produs 2';
            v_locatie    VARCHAR2(50) :=  'Europa';
       BEGIN
            v_cantitate := v_cantitate + 1;
            v_locatie   :=  v_locatie || 'de est';
        END;
    v_cantitate:= v_cantitate + 1;
    v_mesaj := v_mesaj ||' se afla in stoc';
    v_locatie :=  v_locatie || 'de est' ;
END;
/
```

Evaluați:
- valoarea variabilei v_cantitate în subbloc; (2)
- valoarea variabilei v_locatie la poziția în subbloc ; (Europe de est, având tipul VARCHAR2)
- valoarea variabilei v_cantitate în blocul principal ; (601, iar tipul este NUMBER)
- valoarea variabilei v_mesaj în blocul principal ; ('Produs 1 se afla in stoc')
- valoarea variabilei v_locație în blocul principal. (nu este corectă ; v_locatie nu este vizibilă în afara subblocului)

4. Creati un bloc anonim care sa afiseze propozitia "Invat PL/SQL" pe ecran, în două moduri.

Afişare cu ajutorul variabilelor de legătură	**Afişare cu procedurile din pachetul standard DBMS_OUTPUT**

VARIABLE g_mesaj VARCHAR2(50)
 	
BEGIN
  :g_mesaj := 'Invat PL/SQL';
END;
/
 	
PRINT g_mesaj
	SET SERVEROUTPUT ON
	
	BEGIN
	  DBMS_OUTPUT.PUT_LINE('Invat PL/SQL'° ;
	END;
	/
	SET SERVEROUTPUT OFF

5. Să se creeze un bloc anonim în care se declară o variabilă v_oras de tipul coloanei city (locations.city%TYPE). Atribuiți acestei variabile numele oraşului în care se află departamentul având codul 30. Afişați în cele două moduri descrise anterior.

```
SET SERVEROUTPUT ON
DECLARE
  v_oras locations.city%TYPE;
BEGIN
  SELECT city
  INTO v_oras
  FROM departments d, locations l
  WHERE d.location_id=l.location_id AND
department_id=30;
  DBMS_OUTPUT.PUT_LINE('Oraşul este '||
v_oras);
END;
/
SET SERVEROUTPUT OFF
```

```
VARIABLE g_oras VARCHAR2(20)
BEGIN
  SELECT city
  INTO :g_oras
  FROM departments d, locations l
  WHERE d.location_id=l.location_id AND
department_id=30;
  END;
/
PRINT g_oras
```

6. Să se creeze un bloc anonim în care să se afle media salariilor pentru angajații al căror departament este 50. Se vor folosi variabilele v_media_sal de tipul coloanei salary şi v_dept (de tip NUMBER).

```
SET SERVEROUTPUT ON
DECLARE
  v_media_sal employees.salary%TYPE;
  v_dept NUMBER:=50;
BEGIN
  SELECT AVG(salary)
  INTO   v_media_sal
  FROM employees
  WHERE department_id= v_dept;
  DBMS_OUTPUT.PUT_LINE('media salariilor este '|| v_media_sal);
END;
/
SET SERVEROUTPUT OFF
```

7. Să se specifice dacă un departament este mare, mediu sau mic după cum numărul angajaților săi este mai mare ca 30, cuprins între 10 şi 30 sau mai mic decât 10. Codul departamentului va fi cerut utilizatorului.

```
ACCEPT p_cod_dep  PROMPT ‘Introduceti codul departamentului ‘
DECLARE
  v_cod_dep  departments.department_id%TYPE := &p_cod_dep;
  v_numar        NUMBER(3) := 0;
  v_comentariu   VARCHAR2(10);
BEGIN
  SELECT  COUNT(*)
  INTO    v_numar
  FROM    employees
  WHERE   department_id = v_cod_dep;
  IF v_numar < 10 THEN
    v_comentariu := 'mic';
  ELSIF v_numar BETWEEN 10 AND 30 THEN
    v_comentariu := 'mediu';
  ELSE
    v_comentariu := 'mare';
  END IF;
  DBMS_OUTPUT.PUT_LINE('Departamentul avand codul' || v_cod_dep
       || 'este de tip' || v_comentariu);
END;
/
```

8. Stocați într-o variabilă de substituție p_cod_dep valoarea unui cod de departament. Definiți şi o variabilă p_com care reține un număr din intervalul [0, 100]. Pentru angajații din departamentul respectiv care nu au comision, să se atribuie valoarea lui p_com câmpului commission_pct. Afişați numărul de linii afectate de această actualizare. Dacă acest număr este 0, să se scrie « Nici o linie actualizata ».

```
SET SERVEROUTPUT ON
SET VERIFY OFF
DEFINE p_cod_dep= 50
DEFINE p_com =10
DECLARE
    v_cod_dep emp_pnu.department_id%TYPE:= &p_cod_dep;
    v_com        NUMBER(2);
```

```
BEGIN
    UPDATE emp_pnu
    SET        commission_pct = &p_com/100
    WHERE  department_id= v_cod_dep;
    IF SQL%ROWCOUNT = 0 THEN
       DBMS_OUTPUT.PUT_LINE('Nici o linie actualizata');
    ELSE DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT ||' linii actualizate ');
    END IF;
END;
/
SET VERIFY ON
SET SERVEROUTPUT OFF
```

Obs: (vom reveni în laboratorul despre cursoare)
Atributele cursoarelor implicite :

- SQL%ROWCOUNT – Numarul de linii afectate de cea mai recenta comanda SQL;
- SQL%FOUND – Atribut boolean ce returneaza TRUE daca ultima comanda SQL a afectat cel putin o linie;
- SQL%NOTFOUND – Atribut boolean ce returneaza TRUE daca ultima comanda SQL nu a afectat nici o linie
- SQL%ISOPEN – Atribut boolean ce returneaza TRUE daca cursorul implicit asociat ultimei comenzi a ramas deschis. Nu e niciodata true pentru ca serverul inchide automat cursorul la terminarea comenzii SQL.

9. În funcție de o valoare introdusă de utilizator, utilizând comanda *CASE* se va afişa un mesaj prin care este specificată ziua săptămânii (a cărei abreviere este chiar valoarea respectivă). Utilizați cele 2 forme ale comenzii CASE.

```
SET SERVEROUTPUT ON
DEFINE p_zi = m
DECLARE
  v_zi  CHAR(2) := UPPER('&p_zi');
  v_comentariu  VARCHAR2(20);
BEGIN
   CASE v_zi
     WHEN 'L' THEN v_comentariu := 'Luni';
     WHEN 'M' THEN v_comentariu := 'Marti';
     WHEN 'MI' THEN v_comentariu := 'Miercuri';
     WHEN 'J' THEN v_comentariu := 'JOI';
     WHEN 'V' THEN v_comentariu := 'Vineri';
     WHEN 'S' THEN v_comentariu := 'Sambata';
     WHEN 'D' THEN v_comentariu := 'Duminica';
      ELSE v_comentariu := ' eroare!';
   END CASE;
  DBMS_OUTPUT.PUT_LINE('Ziua este' || v_comentariu);
END;
/
SET SERVEROUTPUT OFF
```

```
SET SERVEROUTPUT ON
DEFINE p_zi = m
DECLARE
  v_zi  CHAR(2) := UPPER('&p_zi');
  v_comentariu  VARCHAR2(20);
BEGIN
   CASE
     WHEN v_zi = 'L' THEN v_comentariu :='Luni';
     WHEN v_zi ='M' THEN v_comentariu :='Marti';
     WHEN v_zi ='MI' THEN v_comentariu :='Miercuri';
     WHEN v_zi ='J' THEN v_comentariu :='JOI';
     WHEN v_zi ='V' THEN v_comentariu :='Vineri';
     WHEN v_zi ='S' THEN v_comentariu :='Sambata';
     WHEN v_zi ='D' THEN v_comentariu :='Duminica';
     ELSE v_comentariu :=' eroare!';
   END CASE;
  DBMS_OUTPUT.PUT_LINE('Ziua este' || v_comentariu);
END;
/

SET SERVEROUTPUT OFF
```

10. Creați structura tabelului *org_tab_pnu* constând din două coloane, *cod_tab* de tip *INTEGER* ce conține un contor al înregistrărilor şi *text_tab* de tip *VARCHAR2* ce conține un text asociat fiecărei înregistrări. Să se introducă 70 de înregistrări în acest tabel. Se cer 2 metode.

```
DECLARE
  v_contor BINARY_INTEGER := 1;
BEGIN
  LOOP
    INSERT INTO org_tab
    VALUES (v_contor, 'indice loop');
    v_contor := v_contor + 1;
    EXIT WHEN v_contor > 70;
  END LOOP;
END;
```

sau

```
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;
```

11. Scrieți un bloc PL/SQL care actualizează conținutul tabelului anterior, indicând pe coloana text_tab dacă numărul cod_tab este par sau impar.

12. În structura tabelului *emp_pnu* se va introduce un nou câmp (*stea* de tip *VARCHAR2(200)*). Să se creeze un bloc *PL/SQL* care va reactualiza acest câmp, introducând o steluță pentru fiecare 100$ din salariul unui angajat al cărui cod este specificat de către utilizator.

```
ALTER TABLE  emp_pnu
ADD stea     VARCHAR2(200);

SET VERIFY OFF
ACCEPT  p_cod_ang PROMPT ‘Dati codul unui angajat’
DECLARE
  v_cod_ang  emp_pnu.employee_id%TYPE := &p_cod_ang;
  v_salariu     emp_pnu.salary%TYPE;
  v_stea        emp_pnu.stea%TYPE:= NULL;
BEGIN
  SELECT  NVL(ROUND(salary/100),0)
  INTO       v_salariu
  FROM      emp_pnu
  WHERE   employee_id = v_cod_ang;
  FOR i IN 1..v_salariu LOOP
    v_stea := v_stea || '*'
  END LOOP;
  UPDATE   emp_pnu
  SET         stea = v_stea
  WHERE   employee_id = v_cod_ang;
  COMMIT;
END;
/
SET VERIFY ON
```

13. Să se declare şi să se iniţializeze cu 1 variabila v_i de tip POSITIVE şi cu 10 constanta c_max de tip POSITIVE. Să se implementeze un ciclu LOOP care incrementează pe v_i până când acesta ajunge la o valoare > c_max, moment în care ciclul LOOP este părăsit şi se sare la instrucţiunea v_i:=1. (GO TO/EXIT). Se cer 2 metode.

```
SET SERVEROUTPUT ON
DECLARE
v_i POSITIVE:=1;
c_max CONSTANT POSITIVE:=10;
BEGIN
LOOP
  v_i := v_i+1;
  IF v_i > c_max THEN
   DBMS_OUTPUT.PUT_LINE('in loop v_i=' ||
v_i);
   GOTO next;
  END IF;
END LOOP;
<<next>>
v_i:=1;
DBMS_OUTPUT.PUT_LINE('dupa loop v_i=' ||v_
i);
END;
/
SET SERVEROUTPUT OFF
```

```
SET SERVEROUTPUT ON
DECLARE
v_i POSITIVE:=1;
c_max CONSTANT POSITIVE:=10;
BEGIN
v_i:=1;
LOOP
  v_i := v_i+1;
  DBMS_OUTPUT.PUT_LINE('in loop v_i=' || v_i);
  EXIT WHEN v_i > c_max;
END LOOP;
v_i:=1;
DBMS_OUTPUT.PUT_LINE('dupa loop v_i=' ||
v_i);
END;
/
SET SERVEROUTPUT OFF
```

Exerciţii propuse:

1. Creaţi un bloc PL/SQL care declară 2 variabile (una de tip şir de caractere şi cealaltă numerică), iniţializate la declarare. Atribuiti valorile acestor variabile PLSQL unor variabile gazdă (VARIABLE) SQL*Plus şi tipăriti valorile variabilelor PL/SQL pe ecran. Executaţi blocul PL/SQL.
2. Creati si executati un bloc PL/SQL care cere de la tastatura 2 numere (prin variabile de substitutie SQL* Plus). Primul numar se va imparti la al doilea, si se va adauga rezultatului cel de-al doilea numar. Dacă al doilea număr este 0, rezultatul va fi pătratul primului număr (x**2). Rezultatul va fi retinut intr-o variabila PL/SQL si va fi tiparit pe ecran.
3. Să se calculeze suma salariilor pentru un job al cărui cod este introdus de utilizator. Căutarea se va face case-insensitive.
4. Creati un bloc PL/SQL care calculeaza castigul total pentru un an, salariul anual si procentul care reprezinta bonusul fiind transmise blocului PL/SQL prin variabile de substitutie SQL*Plus. Bonusul se va introduce ca numar intreg (pentru bonus de 15% se va introduce 15). Daca salariul este null, va fi setat la 0 inainte de a calcula castigul total. Executati blocul PL/SQL. Se va folosi functia NVL pentru manipularea valorilor NULL.
5. Sa se creeze un bloc PL/SQL care calculeaza si modifica valoarea comisionului pentru un angajat al carui cod este dat de la tastatura, pe baza salariului acestuia, astfel:

- daca salariul este mai mic decat 1000$, comisionul va fi 10% din salariu;
- daca salariul ete intre 1000 si 1500$, comisionul va fi 15% din salariu;
- daca salariul depaseste 1500$, comisionul va fi 20% din salariu;
- daca salariul este NULL, comisionul va fi 0.

Modificările se fac în tabelul emp_pnu.

6. Sa se creeze un bloc PL/SQL care selecteaza codul maxim de departament din tabelul

DEPARTMENTS si il stocheaza intr-o variabila SQL*Plus. Se va tipari rezultatul pe ecran.

7. Sa se creeze un bloc PL/SQL care insereaza un nou departament in tabelul DEPT_PNU. Se va folosi parametru de substitutie pentru numele departamentului. Codul este dat de valoarea variabilei calculate anterior +1. Locatia va avea valoarea null. Sa se listeze continutul tabelului DEPT_PNU.
8. Sa se creeze un bloc PL/SQL care reactualizeaza locatia pentru un departament existent (în tabelul DEPT_PNU). Se vor folosi parametri de substitutie pentru numarul departamentului si locatia acestuia. Sa se listeze codul, numele si locatia pentru departamentul reactualizat.
9. Sa se creeze un bloc PL/SQL care sterge departamentul creat la exercitiul 7. Se va folosi un parametru de substitutie pentru numarul departamentului. Se va tipari pe ecran numarul de linii afectate. Ce se intampla daca se introduce un cod de departament care nu exista?

Baze de date-Anul 3 (semestrul 1)
Laborator 2 PL/SQL

Tipuri de date compuse

- **înregistrare (*RECORD*) ;**
- **colecție (*INDEX-BY TABLE*, *NESTED TABLE*, *VARRAY*).**

I. Înregistrări (RECORD)

Ø Declararea tipului *RECORD* se face conform următoarei sintaxe:

TYPE *nume_tip* ***IS RECORD***
(*nume_câmp1* {*tip_câmp* | *variabil****ă%TYPE*** |
*nume_tabel.coloan****ă%TYPE*** | *nume_tabel****%ROWTYPE***}
[[***NOT NULL***] {:= | ***DEFAULT***} *expresie1*],
(*nume_câmp2* {*tip_câmp* | *variabil****ă%TYPE*** |
*nume_tabel.coloan****ă%TYPE*** | *nume_tabel****%ROWTYPE***}
[[***NOT NULL***] {:= | ***DEFAULT***} *expresie2*],…);

Ø *Oracle9i* introduce câteva facilități legate de acest tip de date.

- Se poate insera (*INSERT*) o linie într-un tabel utilizând tipul *RECORD*.
- Se poate actualiza (*UPDATE*) o linie într-un tabel utilizând tipul *RECORD* (cu sintaxa *SET ROW*)
- se poate regăsi şi returna sau şterge informația din clauza *RETURNING* a comenzilor *UPDATE* sau *DELETE*.
- dacă în comenzile *UPDATE* sau DELETE se modifică mai multe linii, atunci pot fi utilizate în sintaxa BULK COLLECT INTO, colecții de înregistrări.

Exerciții:
1. Să se şteargă angajatul având codul 200 din tabelul *EMP_PNU*. Să se rețină într-o variabilă de tip RECORD codul, numele, salariul şi departamentul acestui angajat (clauza *RETURNING*) . Să se afişeze înregistrarea respectivă. Rollback.

```
DECLARE
  TYPE info_ang_pnu IS RECORD (
    cod_ang  NUMBER(4),
    nume     VARCHAR2(20),
    salariu  NUMBER(8),
    cod_dep  NUMBER(4));
  v_info_ang  info_ang_pnu;
BEGIN
  DELETE FROM emp_pnu
     WHERE employee_id = 200
    RETURNING employee_id, last_name, salary, department_id
      INTO v_info_ang;
  DBMS_OUTPUT.PUT_LINE('A fost stearsa linia continand valorile ' ||
       v_info_ang.cod_ang ||' '||v_info_ang.nume||' ' ||v_info_ang.salariu ||' '
       || v_info_ang.cod_dep) ;
END;
/
ROLLBACK ;
```

2. a) Folosind tipul declarat mai sus, să se adauge o linie în tabelul *EMP_PNU* prin intermediul unei variabile de tip înregistrare inițializate. Efectuați modificările necesare asupra tipului de date, astfel încât această inserare să fie posibilă. La inițializarea unei variabile de tip record, țineți cont de constrângerile *NOT NULL* definite asupra tabelului *EMP_PNU*.

b) Modificați valoarea unei componente a variabilei definite anterior şi actualizați conținutul liniei introduse în tabel.

```
set serveroutput on
DECLARE
  TYPE info_ang_pnu IS RECORD (
    cod_ang  NUMBER(4):=500,
    nume      VARCHAR2(20):='abc',
    prenume VARCHAR2(20):='john',
    email      emp_pnu.email%TYPE:='abc@mail',
    telefon    emp_pnu.phone_number%type,
    data       emp_pnu.hire_date%TYPE:=SYSDATE,
    job        emp_pnu.job_id%TYPE:='SA_REP',
    salariu    NUMBER(8, 2):=1000,
    comision emp_pnu.commission_pct%TYPE,
    manager  emp_pnu.manager_id%TYPE,
    cod_dep  NUMBER(4):=30
    );
  v_info_ang  info_ang_pnu;
BEGIN
 --inserare; nu ar fi fost posibila maparea unei variabile de tip RECORD într-o lista
-- explicita de coloane
  INSERT INTO emp_pnu
  VALUES    v_info_ang;
  DBMS_OUTPUT.PUT_LINE('A fost introdusa linia continand valorile ' ||
       v_info_ang.cod_ang ||' '||v_info_ang.nume||' ' ||v_info_ang.salariu ||' '
       || v_info_ang.cod_dep) ;
--actualizare
  v_info_ang.nume:='smith';
   UPDATE emp_pnu
   SET ROW=v_info_ang
   WHERE employee_id = v_info_ang.cod_ang;
   DBMS_OUTPUT.PUT_LINE('A fost actualizata linia cu valorile ' ||
       v_info_ang.cod_ang ||' '||v_info_ang.nume||' ' ||v_info_ang.salariu ||' '
       || v_info_ang.cod_dep) ;
END;
/
ROLLBACK ;
```

II. Colecții

Colecțiile permit să fie prelucrate simultan mai multe variabile de acelaşi tip. Fiecare element are un indice unic, care determină poziția sa în colecție.

În *PL/SQL* există trei tipuri de colecții:

- tablouri indexate (*index-by tables*);
- tablouri imbricate (*nested tables*);
- vectori (*varrays* sau *varying arrays*).

Obs :

- Tipul *index-by table* poate fi utilizat **numai** în declarații *PL/SQL*. Tipurile *varray* şi *nested table* pot fi utilizate atât în declarații *PL/SQL*, cât şi în declarații la nivelul schemei (de exemplu, pentru definirea tipului unei coloane a unui tabel relațional).

- Singura diferență sintactică între tablourile indexate şi cele imbricate este clauza INDEX BY. Dacă această clauză lipseşte, atunci tipul este tablou imbricat.

Ø Atribute şi metode ale unei colecții: (informații complete – în curs !)

Atribut sau metodă	**Descriere**
COUNT	numărul componentelor colecției
FIRST	Indicele primului element din tablou
LAST	Indicele ultimului element din tablou
EXISTS	întoarce TRUE dacă există în tablou componenta cu indexul specificat
NEXT	returnează indicele următoarei componente
PRIOR	returnează indicele componentei anterioare
DELETE	şterge una sau mai multe componente.
EXTEND	Adaugă elemente la sfârşit
LIMIT	Numărul maxim de elemente al unei colecții (pentru vectori), null pentru tablouri imbricate
TRIM	şterge elementele de la sfârşitul unei colecții

Ultimele 3 metode nu sunt valide pentru index-by tables.

Ø ***bulk bind*** permite ca toate liniile unei colecții să fie transferate simultan printr-o singură operație.
- este realizat cu ajutorul comenzii *FORALL*, ce poate fi folosită cu orice tip de colecție:

FORALL *index* ***IN*** *lim_inf..lim_sup*
comanda_sql;

Cursorul *SQL* are un atribut compus *%BULK_ROWCOUNT* care numără liniile afectate de iterațiile comenzii *FORALL*. *%BULK_ROWCOUNT*(*i*) reprezintă numărul de linii procesate de a *i*-a execuție a comenzii *SQL.*

Ø **Regăsirea rezultatului unei interogări în colecții** (înainte de a fi trimisă motorului *PL/SQL*) se poate obține cu ajutorul clauzei *BULK COLLECT*:

*…**BULK COLLECT INTO** nume_colecție* [*,nume_colecție*]*…*

Ø Clauza poate să apară în:
- comenzile *SELECT INTO* (cursoare implicite*)*,
- comenzile *FETCH INTO* (cursoare explicite),
- clauza *RETURNING INTO* a comenzilor *INSERT*, *UPDATE*, *DELETE*.

Exerciții:

3. Analizați şi comentați exemplul următor. Afişați valorile variabilelor definite.

```
DECLARE
  TYPE  tab_index IS TABLE OF NUMBER
    INDEX BY BINARY_INTEGER;
  TYPE  tab_imbri IS TABLE OF NUMBER;
  TYPE  vector IS VARRAY(15) OF NUMBER;
  v_tab_index  tab_index;
  v_tab_imbri  tab_imbri;
  v_vector     vector;
  i INTEGER;
BEGIN
  v_tab_index(1) := 72;
  v_tab_index(2) := 23;
  v_tab_imbri := tab_imbri(5, 3, 2, 8, 7);
```

```
  v_vector := vector(1, 2);
-- afisati valorile variabilelor definite; exemplu dat pentru v_tab_imbri
i:=v_tab_imbri.FIRST;
WHILE (i <= v_tab_imbri.LAST) LOOP
  DBMS_OUTPUT.PUT_LINE('v_tab_imbri: '||v_tab_imbri(i));
  i:= v_tab_imbri.NEXT(i);
END LOOP;
END;
/
```

II.1. Tablouri indexate (index-by tables)

Ø Tabloul indexat *PL/SQL* are două componente:

- coloană ce cuprinde cheia primară pentru acces la liniile tabloului
- o coloană care include valoarea efectivă a elementelor tabloului.

Ø Declararea tipului *TABLE* se face respectând următoarea sintaxă:

TYPE** nume_tip **IS TABLE OF
{*tip_coloană* | *variabilă**%TYPE*** |
*nume_tabel.coloană**%TYPE*** [***NOT NULL***] |
*nume_tabel**%ROWTYPE***}
***INDEX BY** tip_indexare*;

***Observații*:**

- Elementele unui tablou indexat nu sunt într-o ordine particulară şi pot fi inserate cu chei arbitrare.
- Deoarece nu există constrângeri de dimensiune, dimensiunea tabloului se modifică dinamic.
- Tabloul indexat *PL/SQL* nu poate fi inițializat în declararea sa.
- Un tablou indexat neinițializat este vid (nu conține nici valori, nici chei).
- Un element al tabloului este nedefinit atâta timp cât nu are atribuită o valoare efectivă.
- Dacă se face referire la o linie care nu există, atunci se produce excepția *NO_DATA_FOUND*.

Ø Pentru inserarea unor valori din tablourile *PL/SQL* într-o coloană a unui tabel de date se utilizează instrucțiunea *INSERT* în cadrul unei secvențe repetitive *LOOP*.

Ø Pentru regăsirea unor valori dintr-o coloană a unei baze de date într-un tablou *PL/SQL* se utilizează instrucțiunea *FETCH* (cursoare) sau instrucțiunea de atribuire în cadrul unei secvențe repetitive *LOOP*.

Ø Pentru a şterge liniile unui tablou fie se asignează elementelor tabloului valoarea *null*, fie se declară un alt tablou *PL/SQL* (de acelaşi tip) care nu este inițializat şi acest tablou vid se asignează tabloului *PL/SQL* care trebuie şters. În *PL/SQL* 2.3 ştergerea liniilor unui tabel se poate face utilizând metoda *DELETE*.

Exerciții:

4. Să se definească un tablou indexat *PL/SQL* având elemente de tipul *NUMBER*. Să se introducă 20 de elemente în acest tablou. Să se afişeze, apoi să se şteargă tabloul utilizând diverse metode.

```
DECLARE
  TYPE  tablou_numar IS TABLE OF NUMBER
    INDEX BY PLS_INTEGER;
  v_tablou  tablou_numar;
  v_aux     tablou_numar; -- tablou folosit pentru stergere
BEGIN
  FOR i IN 1..20 LOOP
    v_tablou(i) := i*i;
```

```
   DBMS_OUTPUT.PUT_LINE(v_tablou(i));
  END LOOP;
  --v_tablou := NULL;
  --aceasta atribuire da eroarea PLS-00382
  FOR i IN v_tablou.FIRST..v_tablou.LAST LOOP -- metoda 1 de stergere
    v_tablou(i) := NULL;
  END LOOP;
--sau
  v_tablou := v_aux;   -- metoda 2 de stergere
--sau
  v_tablou.delete; --metoda 3 de stergere
  DBMS_OUTPUT.PUT_LINE('tabloul are ' || v_tablou.COUNT ||
                       ' elemente');
END;
/
```

5. Să se definească un tablou de înregistrări având tipul celor din tabelul *dept_pnu*. Să se inițializeze un element al tabloului şi să se introducă în tabelul *dept_pnu*. Să se şteargă elementele tabloului.

```
DECLARE
  TYPE dept_pnu_table_type IS TABLE OF dept_pnu%ROWTYPE
            INDEX BY BINARY_INTEGER;
  dept_table  dept_pnu_table_type;
  i                NUMBER;
BEGIN
  IF dept_table.COUNT <>0 THEN
    i := dept_table.LAST+1;
    ELSE i:=1;
  END IF;
  dept_table(i).department_id := 92;
  dept_table(i).department_name := 'NewDep';
  dept_table(i).location_id := 2700;
  INSERT INTO dept_pnu(department_id, department_name, location_id)
  VALUES (dept_table(i).department_id,
          dept_table(i).department_name,
          dept_table(i).location_id);
  -- sau folosind noua facilitate Oracle9i
  -- INSERT INTO dept_pnu
  -- VALUES  dept_table(i);
  dept_table.DELETE; -- sterge toate elementele
  DBMS_OUTPUT.PUT_LINE('Dupa aplicarea metodei DELETE
     sunt '||TO_CHAR(dept_table.COUNT)||' elemente');
END;
```

II.2 Vectori (*varray*)

- Vectorii (*varray*) sunt structuri asemănătoare vectorilor din limbajele *C* sau *Java*.
- Vectorii au o dimensiune maximă (constantă) stabilită la declarare. În special, se utilizează pentru modelarea relațiilor *one-to-many*, atunci când numărul maxim de elemente din partea „*many*“ este cunoscut şi ordinea elementelor este importantă.
- Fiecare element are un index, a cărui limită inferioară este 1.

Ø Tipul de date vector este declarat utilizând sintaxa:

TYPE *nume_tip* ***IS***
{***VARRAY*** | ***VARYING ARRAY***} (*lungime_maximă*)
OF *tip_elemente* [***NOT NULL***];.

Exerciții:

6. Analizați şi comentați exemplul următor.

```
DECLARE
   TYPE secventa IS VARRAY(5) OF VARCHAR2(10);
    v_sec  secventa := secventa ('alb', 'negru', 'rosu',
                        'verde');
BEGIN
  v_sec (3) := 'rosu';
  v_sec.EXTEND; -- adauga un element null
  v_sec(5) := 'albastru';
  -- extinderea la 6 elemente va genera eroarea ORA-06532
  v_sec.EXTEND;
END;
/
```

Obs : Pentru a putea reține şi utiliza tablourile imbricate şi vectorii, trebuie să declarăm în SQL tipuri de date care să îi reprezinte.

Tablourile imbricate şi vectorii pot fi utilizați drept câmpuri în tabelele bazei. Aceasta presupune că fiecare înregistrare din tabelul respectiv conține un obiect de tip colecție. Înainte de utilizare, tipul trebuie stocat în dicționarul datelor, deci trebuie declarat prin comanda:

CREATE TYPE *nume_tip* ***AS*** {***TABLE*** | ***VARRAY***} ***OF*** *tip_elemente*;

7. a) Să se declare un tip *proiect_pnu* care poate reține maxim 50 de valori de tip VARCHAR2(15).

b) Să se creeze un tabel *test_pnu* având o coloana *cod_ang* de tip NUMBER(4) şi o coloană *proiecte_alocate* de tip *proiect_pnu*. Ce relație se modelează în acest fel?

c) Să se creeze un bloc PL/SQL care declară o variabilă (un vector) de tip *proiect_pnu*, introduce valori în aceasta iar apoi valoarea vectorului respectiv este introdusă pe una din liniile tabelului test_pnu.

La promptul SQL :

```
CREATE TYPE proiect_pnu AS VARRAY(50) OF VARCHAR2(15)
/
CREATE TABLE test_pnu (cod_ang NUMBER(4),
                       proiecte_alocate proiect_pnu);
```

Blocul PL/SQL:

```
DECLARE
        v_proiect proiect_pnu := proiect_pnu(); --initializare utilizând constructorul
BEGIN
        v_proiect.extend (2);
        v_proiect(1) := 'proiect 1';
        v_proiect(2) := 'proiect 2' ;
        INSERT INTO test_pnu VALUES (1, v_proiect);
END;
/
```

8. Să se scrie un bloc care măreşte salariile angajaților din departamentul 50 cu 10%, în cazul în care salariul este mai mic decât 5000. Se va utiliza un vector corespunzător codurilor angajaților. Se cer 3 soluții.

Soluția 1 :

```
DECLARE
        TYPE t_id IS VARRAY(100) OF emp_pnu.employee_id%TYPE ;
        v_id t_id := t_id();
BEGIN
        FOR contor IN (SELECT * FROM emp_pnu) LOOP
                IF contor. Department_id =50 AND contor.salary < 5000 THEN
                        V_id.extend;
                        V_id(v_id.COUNT) := contor.employee_id;
```

```
            END IF;
        END LOOP;
        FOR contor IN 1..v_id.COUNT LOOP
            UPDATE emp_pnu
            SET salary = salary *1.1
            WHERE employee_id = v_id (contor);
        END LOOP;
END;
/
```

Soluția 2 (varianta FORALL):

```
DECLARE
        TYPE t_id IS VARRAY(100) OF emp_pnu.employee_id%TYPE ;
        v_id t_id := t_id();
BEGIN
        FOR contor IN (SELECT * FROM emp_pnu) LOOP
            IF contor. Department_id =50 AND contor.salary < 5000 THEN
                V_id.extend;
                V_id(v_id.COUNT) := contor.employee_id;
            END IF;
        END LOOP;
        FORALL contor IN 1..v_id.COUNT
            UPDATE emp_pnu
            SET salary = salary *1.1
            WHERE employee_id = v_id (contor);
END;
/
```

Obs: Prin comanda FORALL sunt trimise toate datele pe server, executându-se apoi o singură comandă SELECT, UPDATE etc.

Soluția 3 (varianta BULK COLLECT):

```
DECLARE
        TYPE t_id IS VARRAY(100) OF emp_pnu.employee_id%TYPE ;
        v_id t_id := t_id();
BEGIN
        SELECT employee_id
        BULK COLLECT INTO v_id
        FROM emp_pnu
        WHERE  department_id =50 AND salary < 5000;
        FORALL contor IN 1..v_id.COUNT
            UPDATE emp_pnu
            SET salary = salary *1.1
            WHERE employee_id = v_id (contor);
END;
/
```

II.3 Tablouri imbricate

- Tablourile imbricate (*nested table*) sunt tablouri indexate a căror dimensiune nu este stabilită.
  - folosesc drept indici numere consecutive ;
  - sunt asemenea unor tabele cu o singură coloană;
  - nu au dimensiune limitată, ele cresc dinamic;
  - inițial, un tablou imbricat este dens (are elementele pe poziții consecutive) dar pot apărea spații goale prin ştergere ;
  - metoda NEXT ne permite să ajungem la următorul element ;
  - pentru a insera un element nou, tabloul trebuie extins cu metoda EXTEND(nr_comp) ;

- Un tablou imbricat este o mulțime neordonată de elemente de acelaşi tip. Valorile de acest tip:
  - pot fi stocate în baza de date,
  - pot fi prelucrate direct în instrucțiuni *SQL*
  - au excepții predefinite proprii.

Ø Comanda de declarare a tipului de date tablou imbricat are sintaxa:

TYPE *nume_tip* ***IS TABLE OF*** *tip_ elemente* [***NOT NULL***];

Ø Pentru adaugarea de linii intr-un tablou imbricat, acesta trebuie sa fie initializat cu ajutorul **constructorului**.
  - *PL/SQL* apelează un constructor numai în mod explicit.
  - Tabelele indexate nu au constructori.
  - Constructorul primeşte ca argumente o listă de valori numerotate în ordine, de la 1 la numărul de valori date ca parametrii constructorului.
  - Dimensiunea inițială a colecției este egală cu numărul de argumente date în constructor, când aceasta este inițializată.
  - Pentru vectori nu poate fi depăşită dimensiunea maximă precizată la declarare.
  - Atunci când constructorul este fără argumente, va crea o colectie fără nici un element (vida), dar care are valoarea *not null*.

Exerciții:

9. Să se declare un tip tablou imbricat şi o variabilă de acest tip. Inițializați variabila şi afişați conținutul tabloului, de la primul la ultimul element şi invers.

```
DECLARE
  TYPE CharTab IS TABLE OF CHAR(1);
  v_Characters CharTab :=
    CharTab('M', 'a', 'd', 'a', 'm', ',', ' ',
          'I', '"', 'm', ' ', 'A', 'd', 'a', 'm');
  v_Index INTEGER;
BEGIN
  v_Index := v_Characters.FIRST;
  WHILE v_Index <= v_Characters.LAST LOOP
    DBMS_OUTPUT.PUT(v_Characters(v_Index));
    v_Index := v_Characters.NEXT(v_Index);
  END LOOP;
  DBMS_OUTPUT.NEW_LINE;
  v_Index := v_Characters.LAST;
  WHILE v_Index >= v_Characters.FIRST LOOP
    DBMS_OUTPUT.PUT(v_Characters(v_Index));
    v_Index := v_Characters.PRIOR(v_Index);
  END LOOP;
  DBMS_OUTPUT.NEW_LINE;
END;
/
```

10. Creați un tip tablou imbricat, numit NumTab. Afişați conținutul acestuia, utilizând metoda EXISTS. Atribuiți valorile tabloului unui tablou index-by. Afişați şi acest tablou, in ordine inversă.

```
DECLARE  -- cod partial, nu sunt declarate tipurile!
  v_NestedTable NumTab := NumTab(-7, 14.3, 3.14159, NULL, 0);
  v_Count BINARY_INTEGER := 1;
  v_IndexByTable IndexByNumTab;
BEGIN
   LOOP
   IF v_NestedTable.EXISTS(v_Count) THEN
```

```
      DBMS_OUTPUT.PUT_LINE(
        'v_NestedTable(' || v_Count || '): ' ||
        v_NestedTable(v_Count));
      v_IndexByTable(v_count) := v_NestedTable(v_count);
      v_Count := v_Count + 1;
    ELSE
      EXIT;
    END IF;
  END LOOP;
-- atribuire invalida
--  v_IndexByTable := v_NestedTable;
  v_Count := v_IndexByTable.COUNT;
  LOOP
    IF v_IndexByTable.EXISTS(v_Count) THEN
      DBMS_OUTPUT.PUT_LINE(
        'v_IndexByTable(' || v_Count || '): ' ||
        v_IndexByTable(v_Count));
      v_Count := v_Count - 1;
    ELSE
      EXIT;
    END IF;
  END LOOP;
END;
END;
/
```

11. Să se analizeze următorul bloc PL/SQL. Ce se obține în urma execuției acestuia ?

```
DECLARE
  TYPE  alfa IS TABLE OF VARCHAR2(50);
  -- creeaza un tablou (atomic) null
  tab1  alfa ;
  /* creeaza un tablou cu un element care este null, dar
     tabloul nu este null, el este initializat, poate
     primi elemente */
  tab2  alfa := alfa() ;
BEGIN
  IF tab1 IS NULL THEN
    DBMS_OUTPUT.PUT_LINE('tab1 este NULL');
  ELSE
    DBMS_OUTPUT.PUT_LINE('tab1 este NOT NULL');
  END IF;
  IF tab2 IS NULL THEN
    DBMS_OUTPUT.PUT_LINE('tab2 este NULL');
  ELSE
    DBMS_OUTPUT.PUT_LINE('tab2 este NOT NULL');
  END IF;
END;
/
```

12. Analizați următorul exemplu, urmărind excepțiile semnificative care apar în cazul utilizării incorecte a colecțiilor:

```
DECLARE
  TYPE  numar IS TABLE OF INTEGER;
  alfa  numar;
```

```
BEGIN
  alfa(1) := 77;
  -- declanseaza exceptia COLLECTION_IS_NULL
  alfa := numar(15, 26, 37);
  alfa(1) := ASCII('X');
  alfa(2) := 10*alfa(1);
  alfa('P') := 77;
  /* declanseaza exceptia VALUE_ERROR deoarece indicele
     nu este convertibil la intreg */
  alfa(4) := 47;
  /* declanseaza exceptia SUBSCRIPT_BEYOND_COUNT deoarece
     indicele se refera la un element neinitializat */
  alfa(null) := 7; -- declanseaza exceptia VALUE_ERROR
  alfa(0) := 7; -- exceptia SUBSCRIPT_OUTSIDE_LIMIT
  alfa.DELETE(1);
  IF alfa(1) = 1 THEN … -- exceptia NO_DATA_FOUND
  …
END;
/
```

II.4 Colecții pe mai multe niveluri

!!! De analizat exemplele din curs!

II.5 Prelucrarea colecțiilor

- *INSERT* - permite inserarea unei colecții într-o linie a unui tabel. Colecția trebuie să fie creată şi inițializată anterior.
- *UPDATE* este folosită pentru modificarea unei colecții stocate.
- *DELETE* poate şterge o linie ce conține o colecție.
- Colecțiile din baza de date pot fi regăsite în variabile *PL/SQL*, utilizând comanda *SELECT*.
- operatorul *TABLE* permite prelucrarea elementelor unui tablou imbricat care este stocat într-un tabel. Operatorul permite interogarea unei colecții în clauza *FROM* (la fel ca un tabel).
- Pentru tablouri imbricate pe mai multe niveluri, operațiile *LMD* pot fi făcute atomic sau pe elemente individuale, iar pentru vectori pe mai multe niveluri, operațiile pot fi făcute numai atomic.
- Pentru prelucrarea unei colecții locale se poate folosi şi operatorul *CAST*. *CAST* are forma sintactică:

  CAST (*nume_colecție* ***AS*** *tip_colecție*)

Exerciții:

13. a) Să se creeze un tip *LIST_ANG_PNU*, de tip vector, cu maxim 10 componente de tip NUMBER(4).

b) Să se creeze un tabel *JOB_EMP_PNU*, având coloanele: cod_job de tip NUMBER(3), titlu_job de tip VARCHAR2(25) şi info de tip *LIST_ANG_PNU*.

c) Să se creeze un bloc PL/SQL care declară şi inițializează două variabile de tip *LIST_ANG_PNU*, o variabilă de tipul coloanei info din tabelul *JOB_EMP_PNU* şi o variabilă de tipul codului job-ului. Să se insereze prin diverse metode 3 înregistrări în tabelul *JOB_EMP_PNU*.

```
CREATE OR REPLACE TYPE list_ang_pnu AS VARRAY(10) OF
                        NUMBER(4)
/
```

```
CREATE TABLE job_emp_pnu (
  cod_job   NUMBER(3),
  titlu_job  VARCHAR2(25),
  info         list_ang_pnu);

DECLARE
  v_list    list_ang_pnu := list_ang_pnu (123, 124, 125);
  v_info_list  list_ang_pnu := list_ang_pnu (700);
  v_info     job_emp_pnu.info%TYPE;
  v_cod      job_emp_pnu.cod_job%TYPE := 7;
  i  INTEGER;
BEGIN
  INSERT INTO job_emp_pnu
  VALUES (5, 'Analist', list_ang_pnu (456, 457));
  INSERT INTO job_emp_pnu
  VALUES (7, 'Programator', v_list);
  INSERT INTO job_emp_pnu
  VALUES (10, 'Inginer', v_info_list);
  SELECT info
  INTO   v_info
  FROM   job_emp_pnu
  WHERE  cod_job = v_cod;
    --afisare v_info
  DBMS_OUTPUT.PUT_LINE('v_info:');
  i := v_info.FIRST;
  while (i <= v_info.last) loop
    DBMS_OUTPUT.PUT_LINE(v_info(i));
    i := v_info.next(i);
  end loop;
END;
/
ROLLBACK;
```

14. Creați un tip de date tablou imbricat *DateTab_pnu* cu elemente de tip *DATE*. Creati un tabel *FAMOUS_DATES_PNU* având o coloană de acest tip. Declarați o variabilă de tip *DateTab_pnu* şi adăugați-i 5 date calendaristice. Ştergeți al doilea element şi apoi introduceți tabloul în tabelul *FAMOUS_DATES_PNU*. Selectați-l din tabel. Afişați la fiecare pas.
Obs: După crearea tabelului (prin comanda *CREATE TABLE*), pentru fiecare câmp de tip tablou imbricat din tabel este necesară clauza de stocare:

NESTED TABLE *nume_câmp* ***STORE AS*** *nume_tabel*;

În SQL*Plus:

```
DROP TABLE famous_dates_pnu;
DROP TYPE DateTab_pnu;
CREATE OR REPLACE TYPE DateTab_pnu AS
  TABLE OF DATE;
/
CREATE TABLE famous_dates_pnu (
  key       VARCHAR2(100) PRIMARY KEY,
  date_list  DateTab_pnu)
  NESTED TABLE date_list STORE AS dates_tab;
```

Blocul PL/SQL:

```
DECLARE
  v_Dates DateTab_pnu := DateTab_pnu(TO_DATE('04-JUL-1776', 'DD-MON-YYYY'),
                        TO_DATE('12-APR-1861', 'DD-MON-YYYY'),
```

```
                    TO_DATE('05-JUN-1968', 'DD-MON-YYYY'),
                    TO_DATE('26-JAN-1986', 'DD-MON-YYYY'),
                    TO_DATE('01-JAN-2001', 'DD-MON-YYYY'));

  -- Procedura locala  pentru afisarea unui  DateTab.
  PROCEDURE Print(p_Dates IN DateTab_pnu) IS
    v_Index BINARY_INTEGER := p_Dates.FIRST;
  BEGIN
    WHILE v_Index <= p_Dates.LAST LOOP
      DBMS_OUTPUT.PUT('  ' || v_Index || ': ');
      DBMS_OUTPUT.PUT_LINE(TO_CHAR(p_Dates(v_Index),
                           'DD-MON-YYYY'));
      v_Index := p_Dates.NEXT(v_Index);
    END LOOP;
  END Print;

BEGIN
  DBMS_OUTPUT.PUT_LINE('Valoarea initiala a tabloului');
  Print(v_Dates);
  INSERT INTO famous_dates_pnu (key, date_list)
    VALUES ('Date importante', v_Dates);
  v_Dates.DELETE(2); -- tabloul va avea numai 4 elemente
  SELECT date_list
    INTO v_Dates
    FROM famous_dates
    WHERE key = 'Date importante';
  DBMS_OUTPUT.PUT_LINE('Tabloul dupa  INSERT si SELECT:');
  Print(v_Dates);
END;
/

ROLLBACK;
```

15. Să se adauge o coloană *info* de tip tablou imbricat în tabelul *DEPT_PNU*. Acest tablou are ***două componente*** în care pentru fiecare departament sunt depuse codul unui angajat şi job-ul acestuia. Să se insereze o linie în tabelul imbricat. Să se listeze codurile departamentelor şi colecția angajaților corespunzători.

```
CREATE OR REPLACE TYPE rec_info_pnu IS OBJECT (cod_ang NUMBER(4), job
VARCHAR2(20));
/
CREATE OR REPLACE TYPE dept_info_pnu IS TABLE OF rec_info_pnu;
/
ALTER TABLE dept_pnu
 ADD (info dept_info_pnu)
 NESTED TABLE info STORE AS info_tab;

SET DESCRIBE DEPTH ALL LINENUM ON INDENT ON

DESC dept_pnu

UPDATE  dept_pnu
SET         info=dept_info_pnu()
WHERE department_id=90;

INSERT INTO TABLE (SELECT info
                FROM   dept_pnu
                WHERE  department_id = 90)
VALUES  (100, 'MyEmp');
```

```
SELECT  d.department_id, t.*
FROM    dept_pnu d, TABLE (d.info) t;
```

16. Să se creeze un tabel temporar *TEMP_TABLE_PNU* cu datele persistente la nivel de sesiune, având o coloană de tip numeric şi alta de tip şir de caractere. Prin intermediul unui tablou indexat, să se adauge 500 de linii în acest tabel. Se cer două variante. Comentați diferențele între variantele propuse.

```
CREATE GLOBAL TEMPORARY TABLE temp_table_pnu (
  num_col NUMBER(6),
  char_col VARCHAR2(20))
ON COMMIT PRESERVE ROWS;
```

```
DECLARE
  TYPE t_Numbers IS TABLE OF temp_table_pnu.num_col%TYPE
    INDEX BY BINARY_INTEGER;
  TYPE t_Chars IS TABLE OF temp_table_pnu.char_col%TYPE
    INDEX BY BINARY_INTEGER;
  v_Numbers t_Numbers;
  v_Chars t_Chars;
BEGIN
  FOR v_Count IN 1..500 LOOP
    v_Numbers(v_Count) := v_Count;
    v_Chars(v_Count) := 'Row number ' || v_Count;
  END LOOP;
  FOR v_Count IN 1..500 LOOP
    INSERT INTO temp_table_pnu VALUES
      (v_Numbers(v_Count), v_Chars(v_Count));
  END LOOP;
END;
/
```

```
DECLARE
  TYPE t_Numbers IS TABLE OF temp_table_pnu.num_col%TYPE
    INDEX BY BINARY_INTEGER;
  TYPE t_Chars IS TABLE OF temp_table_pnu.char_col%TYPE
    INDEX BY BINARY_INTEGER;
  v_Numbers t_Numbers;
  v_Chars t_Chars;
BEGIN
  FOR v_Count IN 1..500 LOOP
    v_Numbers(v_Count) := v_Count;
    v_Chars(v_Count) := 'Row number ' || v_Count;
  END LOOP;
  FORALL v_Count IN 1..500
    INSERT INTO temp_table_pnu VALUES
      (v_Numbers(v_Count), v_Chars(v_Count));
END;
/
```

17. Să se insereze o linie nouă în tabelul *EMP_PNU*, obținându-se *rowid*-ul acesteia. Să se afişeze valoarea obținută. Măriți cu 30% salariul angajatului cu *rowid*-ul respectiv şi obțineți numele şi prenumele acestuia. Ştergeți apoi linia corespunzătoare acelui *rowid* şi afişați informațiile corespunzătoare.

```
DECLARE
```

```
  v_NewRowid ROWID;
  v_FirstName employees.first_name%TYPE;
  v_LastName employees.last_name%TYPE;
  v_ID employees.employee_id%TYPE;
BEGIN
  -- Insereaza o linie noua in tabelul emp_pnu si obtine rowid-ul acesteia
  INSERT INTO emp_pnu
    (employee_id, first_name, last_name, email, job_id, salary, hire_date)
  VALUES
    (emp_sequence.NEXTVAL, 'Xavier', 'Juan','jxav', 'MK_MAN', 2000, SYSDATE)
  RETURNING rowid INTO v_NewRowid;

  DBMS_OUTPUT.PUT_LINE(' rowid-ul noii linii este ' || v_NewRowid);
  UPDATE emp_pnu
    SET salary = salary * 1.3
    WHERE rowid = v_NewRowid
    RETURNING first_name, last_name INTO v_FirstName, v_LastName;
  DBMS_OUTPUT.PUT_LINE('Name: ' || v_FirstName || ' ' || v_LastName);
  DELETE FROM emp_pnu
    WHERE rowid = v_NewRowid
    RETURNING employee_id INTO v_ID;
  DBMS_OUTPUT.PUT_LINE('ID-ul noii linii a fost ' || v_ID);
END;
/
```

18. Să se declare un tip tablou indexat de numere *T_NUMBERS* şi un tip tablou indexat de elemente de tip *T_NUMBERS*, numit *T_MULTINUMBERS*. Să se declare un *TIP T_MULTIVARRAY* de elemente *T_NUMBERS* şi un tip tablou imbricat de elemente *T_NUMBERS*, numit *T_MULTINESTED*. Declarați o variabilă de fiecare din aceste tipuri, inițializați-o şi apoi afişați.

```
DECLARE –acesta este doar codul partial!
  TYPE t_Numbers IS TABLE OF NUMBER
    INDEX BY BINARY_INTEGER;
  TYPE t_MultiNumbers IS TABLE OF t_Numbers
    INDEX BY BINARY_INTEGER;

  TYPE t_MultiVarray IS VARRAY(10) OF t_Numbers;
   TYPE t_MultiNested IS TABLE OF t_Numbers;
  v_MultiNumbers t_MultiNumbers;
BEGIN
  v_MultiNumbers(1)(1) := 12345;
END;
/
```

19. Definiți un tip tablou imbricat *EmpTab* cu elemente de tipul liniilor tabelului *EMP_PNU*. Definiți o variabilă de tip *EmpTab* şi apoi inserați linia corespunzătoare în tabelul *EMP_PNU*.

```
DECLARE –cod partial!
  TYPE EmpTab IS TABLE OF emp_pnu%ROWTYPE;
  v_EmpList EmpTab;
BEGIN
  v_EmpList(1) :=
    EmpTab( … );
…
END;
/
```

20. Declarați un tip *EmpTab* de tip tablou indexat de tablouri imbricate de linii din tabelul *EMPLOYEES*. Declarați o variabilă de acest tip. Inserați într-unul din elementele tabloului informațiile corespunzătoare angajatului având codul 200. Atribuiți valori pentru câmpurile *last_name* şi *first_name* ale altui element. Afişați.

```
DECLARE
  TYPE EmpTab IS TABLE OF employees%ROWTYPE
    INDEX BY BINARY_INTEGER;
  /* Fiecare element al lui v_Emp este o înregistrare */
  v_Emp EmpTab;
BEGIN
  SELECT *
    INTO v_Emp(200)
    FROM employees
    WHERE employee_id = 200;
  v_Emp(1).first_name := 'Larry';
  v_Emp(1).last_name := 'Lemon';
--afisare …
END;
/
```

Baze de date-Anul 3 (semestrul 1)
Laborator 3 PL/SQL

Cursoare

Un cursor este o modalitate de a parcurge (linie cu linie) multimea de linii procesate returnate de o cerere 'multiple-row'. Această mulțime se numeşte *active set*.

Ø Cursoarele pot fi:
- implicite – care sunt declarate de PL/SQL in mod implicit pentru toate comenzile LMD si comanda SELECT, inclusiv comenzile care returneaza o singura linie.
- explicite – pentru cereri care returneaza mai mult de o linie, sunt definite cursoare explicite, denumite de programator si manipulate prin intermediul unor comenzi specifice.

Ø **Etapele** utilizarii unui cursor:
a) ***Declarare*** (în sectiunea declarativa a blocului PL/SQL):
CURSOR *c_nume_cursor* [*(parametru tip_de_Date, ..)*] *IS*
Comanda SELECT;

b) ***Deschidere*** (comanda OPEN), operatie ce identifică mulțimea de linii (*active set*):
OPEN *c_nume_cursor* [*(parametru, …)*]*;*

c) ***Incarcare*** (comanda FETCH). Numarul de variabile din clauza INTO trebuie sa se potriveasca cu lista SELECT returnata de cursor.
FETCH *c_nume_cursor* ***INTO*** *variabila, …;*

d) ***Verificare*** dacă nu am ajuns cumva la finalul mulțimii de linii folosind atributele:
C_nume_cursor%NOTFOUND – valoare booleana
C_nume_cursor%FOUND – valoare booleana

Daca nu s-a ajuns la final mergi la c).

e) ***Inchidere*** cursor (operatiune foarte importanta avand in vedere ca daca nu e inchis cursorul ramane deschis si consuma din resursele serverului, MAX_OPEN_CURSORS)
CLOSE *c_nume_cursor;*

Ø **Atributele** cursoarelor

Atribut	**Tip**	**Descriere**
%ISOPEN	Boolean	TRUE atunci când cursorul este deschis
%NOTFOUND	Boolean	TRUE dacă cea mai recentă operație FETCH nu a regăsit o linie
%FOUND	Boolean	TRUE dacă cea mai recentă operație FETCH a întors o linie
%ROWCOUNT	Number	Întoarce numărul de linii returnate până la momentul respectiv.

Ø **Clauza FOR UPDATE**
Comanda SELECT are urmatoarea extensie PL/SQL pentru blocarea explicita inregistrarilor ce urmeaza a fi prelucrate (modificate sau sterse):

SELECT …
FROM …
WHERE …
…
ORDER BY …
FOR UPDATE [OF lista_coloane] [NOWAIT | WAIT n];

- Daca liniile selectate de cerere nu pot fi blocate din cauza altor blocari atunci

- daca se foloseste NOWAIT este ridicata imediat eroarea ORA-00054
- daca nu se foloseste NOWAIT atunci se asteapta pana cand liniile sunt deblocate.
- daca se foloseste WAIT n atunci se asteapta un numar determinat de secunde inainte de a da eroare ca liniile ce trebuie selectate pentru modificare sunt blocate.

- Nu este recomandata anularea (ROLLBACK) sau permanentizarea schimbarilor inainte de a inchide cursorul ce foloseste FOR UPDATE pentru ca aceasta ar elibera blocarile realizate de acesta.
- Pentru a modifica o anumită linie returnata de un cursor se poate folosi clauza:

WHERE CURRENT OF nume_cursor

Aceasta clauza apare la finalul unei comenzi UPDATE si face referinta la un cursor care este deschis si s-a facut cel putin o incarcare din el (FETCH).

Exerciții: [Cursoare implicite]

1. Să se actualizeze liniile tabelului emp_pnu, mărind cu 10% valoarea comisionului pentru salariații având salariul mai mic decât o valoare introdusă de utilizator. Să se afişeze dacă au fost actualizate linii sau nu (SQL%FOUND), iar în caz afirmativ să se afişeze numărul de linii afectate (SQL%ROWCOUNT). Ce fel de cursor folosim?

2. Să se creeze un tabel DEP_EMP_PNU având câmpurile cod_dep şi cod_ang. Să se introducă într-o variabilă de tip tablou imbricat codurile departamentelor (în care există angajați), iar apoi, prin intermediul unei comenzi FORALL să se insereze aceste coduri şi codurile angajaților corespunzători în tabelul DEP_EMP_PNU. Pentru fiecare departamentm să se afişeze câți angajați au fost introduşi.

```
SET SERVEROUTPUT ON
DECLARE
  TYPE t_dep IS TABLE OF NUMBER;
  V_dep t_dep;
BEGIN
  SELECT department_id BULK COLLECT INTO v_dep FROM emp_pnu;
  FORALL j IN 1..v_dep.COUNT
    INSERT INTO dep_emp_pnu
      SELECT department_id,employee_id
      FROM   emp_pnu
      WHERE  department_id_id = v_dep(j);
  FOR j IN 1..v_dep.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE ('Pentru departamentul avand codul ' ||
      V_dep(j) || ' au fost inserate ' ||
      SQL%BULK_ROWCOUNT(j)
      || inregistrari (angajati)');
  END LOOP;
  DBMS_OUTPUT.PUT_LINE ('Numarul total de inregistrari
     inserate este '||SQL%ROWCOUNT);
END;
/
SET SERVEROUTPUT OFF
```

Exerciții: [Introducere cursoare explicite]

3. Să se obțină câte o linie de forma ‘ <nume> are salariul anual <salariu annual> pentru fiecare angajat din departamentul 50. Se cer 4 soluții (WHILE, LOOP, FOR specific cursoarelor cu varianta de scriere a cursorului în interiorul său).

Soluția 1:

```
DECLARE
        CURSOR c_emp IS
                SELECT last_name, salary*12 sal_an
                FROM emp_pnu
```

```
                WHERE department_id = 50;
        V_emp c_emp%ROWTYPE;
BEGIN
        OPEN c_emp;
        FETCH c_emp INTO v_emp;
        WHILE (c_emp%FOUND) LOOP
                DBMS_OUTPUT.PUT_LINE (‘ Nume:’ || v_emp.last_name ||
                ‘ are salariul anual : ‘ || v_emp.sal_an);
                FETCH c_emp INTO v_emp;
        END LOOP;
        CLOSE c_emp;
END;
```

Soluția 2:

```
DECLARE
        CURSOR c_emp IS
                SELECT last_name, salary*12 sal_an
                FROM emp_pnu
                WHERE department_id = 50;
        V_emp c_emp%TYPE;
BEGIN
        OPEN c_emp;
        LOOP
                FETCH c_emp INTO v_emp;
                EXIT WHEN c_emp%NOTFOUND;
                DBMS_OUTPUT.PUT_LINE (‘ Nume:’ || v_emp.last_name ||
                  ‘ are salariul anual : ‘ || v_emp.sal_an);
        END LOOP;
        CLOSE c_emp;
END;
```

Soluția 3: // nu mai este nevoie explicit de OPEN, FETCH, CLOSE !!!

```
DECLARE
        CURSOR c_emp IS
                SELECT last_name, salary*12 sal_an
                FROM emp_pnu
                WHERE department_id = 50;
        BEGIN
                FOR v_emp IN c_emp LOOP
                        DBMS_OUTPUT.PUT_LINE (‘ Nume:’ || v_emp.last_name ||
                  ‘ are salariul anual : ‘ || v_emp.sal_an);
                END LOOP;
END;
```

Soluția 4:

```
BEGIN
  FOR v_rec IN  (SELECT last_name, salary*12 sal_an
              FROM   employees
              WHERE department_id = 50) LOOP
    DBMS_OUTPUT.PUT_LINE (‘ Nume:’ || v_rec.last_name ||
                ‘ are salariul anual : ‘ || v_rec.sal_an);
  END LOOP;
END;
/
```

4. Sa se afiseze salariatii care au salariul mai mic de 7000$, in urmatoarea forma:

   -Salariatul <nume> castiga <salariu>-.

5. Creați un bloc PL/SQL care determină cele mai mari n salarii, urmând paşii descrişi în continuare:
   a) creați un tabel top_salarii_pnu, având o coloană *salary*.
   b) Numărul n (al celor mai bine plătiți salariați) se va introduce de către utilizator (se va folosi comanda SQL*Plus ACCEPT şi o varianilă de substituție p_num).
   c) În secțiunea declarativă a blocului PL/SQL se vor declara 2 variabile: v_num de tip NUMBER (corespunzătoare lui p_num) şi v_sal de tipul coloanei salary. Se va declara un cursor emp_cursor pentru regăsirea salariilor în ordine descrescătoare (*se presupune că nu avem valori duplicate*).
   d) Se vor introduce cele mai mari mai bine plătiți n angajați în tabelul top_salarii_pnu;
   e) Afişați conținurtul tabelului top_salarii_pnu.
   f) Testați cazuri speciale, de genul n = 0 sau n mai mare decât numărul de angajați. Se vor elimina înregistrările din tabelul top_salarii_pnu după fiecare test.

Soluție:

```
ACCEPT p_num PROMPT ' … '
DECLARE
        V_num   NUMBER(3) := &p_num;
        V_sal     emp_pnu.salary%TYPE;
        CURSOR emp_cursor IS
                SELECT DISTINCT salary
                FROM    emp_pnu
                ORDER BY salary DESC;
BEGIN
        OPEN emp_cursor; -- folositi si alte variante de lucru cu cursorul !!!!
        FETCH emp_cursor INTO v_sal;
        WHILE emp_cursor%ROWCOUNT <= v_num AND emp_cursor%FOUND LOOP
                INSERT INTO top_salarii_pnu (salary)
                VALUES (v_sal);
                FETCH emp_cursor INTO v_sal;
        END LOOP;
        CLOSE emp_cursor;
END;
/
SELECT * FROM top_salarii_pnu;
```

Exerciții: [Cursoare cu parametru]

6. Să se declare un cursor cu un parametru de tipul codului angajatului, care regăseşte numele şi salariul angajaților având codul transmis ca parametru sau ale numele si salariile tuturor angajatilor daca valoarea parametrului este null. Să se declare o variabilă v_nume de tipul unei linii a cursorului. Să se declare două tablouri de nume (v_tab_nume), respectiv salarii (v_tab_sal). Să se parcurgă liniile cursorului în două moduri: regăsindu-le în v_nume sau în cele două variabile de tip tablou.

```
DECLARE
  CURSOR c_nume (p_id employees.employee_id%TYPE) IS
     SELECT last_name, salary
     FROM   employees
     WHERE p_id IS NULL  // conditia este adevarata pentru toate liniile tabelului atunci cand
                         // parametrul este null
    OR    employee_id = p_id;
  V_nume c_nume%ROWTYPE;
  -- sau
  /* TYPE t_nume IS RECORD
       (last_name employees.employee_id%TYPE,
        salary    employees.salary%TYPE
     v_nume t_nume;
  */
```

```
  TYPE t_tab_nume IS TABLE OF employees.last_name%TYPE;
  TYPE t_tab_sal  IS TABLE OF employees.salary%TYPE;
  V_tab_nume t_tab_nume;
  V_tab_Sal  t_tab_sal;
BEGIN
  IF c_nume%ISOPEN THEN
    CLOSE c_nume;
  END IF;
  OPEN c_nume (104);
  FETCH c_nume INTO v_nume;
  WHILE c_nume%FOUND LOOP
     DBMS_OUTPUT.PUT_LINE (‘ Nume:’ || v_nume.last_name ||
                    ‘ salariu : ‘ || v_nume.salary);
     FETCH c_nume INTO v_nume;
  END LOOP;
  CLOSE c_nume;
  -- eroare INVALID CURSOR
  --  FETCH c_nume INTO v_nume;
  DBMS_OUTPUT.PUT_LINE (‘ SI o varianta mai eficienta’);
  OPEN c_nume (null);
  FETCH c_nume BULK COLLECT INTO v_tab_nume, v_tab_sal;
  CLOSE c_nume;
  FOR i IN v_tab_nume.FIRST..v_tab_nume.LAST LOOP
     DBMS_OUTPUT.PUT_LINE (i || ‘ Nume:’ || v_tab_nume(i) ||
                  ‘ salariu : ‘ || v_tab_sal(i));
  END LOOP;
END;
/
```

7. Să se rezolve exercițiul 6 utilizând comanda LOOP.

```
DECLARE
  CURSOR c_nume (p_id employees.employee_id%TYPE) IS
     SELECT last_name, salary
     FROM   employees
     WHERE p_id IS NULL
    OR    employee_id = p_id;
  V_nume c_nume%ROWTYPE;
  -- sau
  /* TYPE t_nume IS RECORD
        (last_name employees.employee_id%TYPE,
         salary    employees.salary%TYPE
     v_nume t_nume;
  */
BEGIN
  OPEN c_nume (104);
  LOOP
    FETCH c_nume INTO v_nume;
    EXIT WHEN c_nume%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE (‘ Nume:’ || v_nume.last_name ||
                    ‘ salariu : ‘ || v_nume.salary);
  END LOOP;
  CLOSE c_nume;
END;
/
```

8. Să se rezolve exercițiul 6 folosind comanda FOR specifică lucrului cu cursoare.
Obs: La cursoare, comanda FOR realizează deschidere , incarcare si inchidere automata

```
DECLARE
  CURSOR c_nume (p_id employees.employee_id%TYPE) IS
     SELECT last_name, salary*12 sal_an
     FROM   employees
     WHERE p_id IS NULL
     OR    employee_id = p_id;
BEGIN
  FOR v_rec IN  c_nume (104) LOOP
    DBMS_OUTPUT.PUT_LINE (‘ Nume:’ || v_rec.last_name ||
                        ‘ salariu : ‘ || v_rec.sal_an);
  END LOOP;
END;
/
```

9. Utilizând un cursor parametrizat să se obțină codurile angajaților din fiecare departament şi pentru fiecare job. Rezultatele să fie inserate în tabelul *mesaje*, sub forma câte unui şir de caractere obținut prin concatenarea valorilor celor 3 coloane.

```
DECLARE
  v_cod_dep  departments.department_id%TYPE;
  v_cod_job  departments.job_id%TYPE;
  v_mesaj    VARCHAR2(75);
  CURSOR dep_job IS
    SELECT  department_id, job_id
    FROM    emp_pnu;
  CURSOR emp_cursor (v_id_dep NUMBER,v_id_job VARCHAR2) IS
    SELECT  employee_id || department_id || job_id
    FROM    emp_pnu
    WHERE   department_id = v_id_dep
    AND     job_id = v_id_job;
BEGIN
  OPEN dep_job;
  LOOP
    FETCH dep_job INTO v_cod_dep,v_cod_job;
    EXIT WHEN dep_job%NOTFOUND;
    IF emp_cursor%ISOPEN THEN
      CLOSE emp_cursor;
    END IF;
    OPEN emp_cursor (v_cod_dep, v_cod_job);
    LOOP
      FETCH emp_cursor INTO v_mesaj;
      EXIT WHEN emp_cursor%NOTFOUND;
      INSERT INTO mesaje (rezultat)
      VALUES (v_mesaj);
    END LOOP;
    CLOSE emp_cursor;
  END LOOP;
  CLOSE dep_job;
  COMMIT;
END;
/
```

Exerciții: [FOR UPDATE, WHERE CURRENT OF]

10. Să se dubleze valoarea salariilor angajaților înainte de 1 ianuarie 1995, care nu câştigă comision.

```
DECLARE
  CURSOR before95 IS
    SELECT *
    FROM   emp_pnu
```

```
   WHERE  commission_pct IS NULL
   AND    hire_date <= TO_DATE('01-JAN-1995','DD-MON-YYYY')
   FOR UPDATE OF salary NOWAIT;
BEGIN
  FOR x IN before95 LOOP
    UPDATE  emp_pnu
    SET     salary = salary*2
    WHERE CURRENT OF before95;
  END LOOP;
  COMMIT;  -- se permanentizeaza actiunea si se elibereaza blocarea
END;
/
```

11. Să se declare un cursor cu un parametru de tipul coloanei location_id, care determină departamentele din locația respectivă şi blochează liniile pe perioada prelucrării acestora. Să se deschidă cursorul folosind o variabilă de substituție pentru furnizarea parametrului. Să se actualizeze tabelul dep_pnu, dând valoarea 100 locației corespunzătoare liniei curente a cursorului.

```
DECLARE
 CURSOR c_test (p_loc departments.location_id%TYPE) IS
    SELECT 'x'
    FROM   dep_pnu
    WHERE location_id = &p_loc
     FOR UPDATE OF department_name NOWAIT;
  V_test c_test%ROWTYPE;
  -- sau
  -- v_test VARCHAR2(1);
BEGIN
  OPEN c_test (&loc);
  LOOP
    FETCH c_test INTO v_test;
    EXIT WHEN c_test%NOTFOUND;
    UPDATE dep_pnu
    SET  location_id = 100,
         -- aceasta coloana nu e blocata asa ca nu avem garantia
         -- daca va merge UPDATE-ul fara probleme sau se va bloca
         Department_name = 'x' -- acesta e selectat pentru update (blocare coloana)
    WHERE CURRENT OF c_test;
  END LOOP;
  CLOSE c_test;
   --ROLLBACK;
END;
/
```

Exerciții [Cursoare dinamice]

12. Să se declare un cursor dinamic care întoarce linii de tipul celor din tabelul emp_pnu. Să se citească o opțiune de la utilizator, care va putea lua valorile 1, 2 sau 3. Pentru opțiunea 1 deschideți cursorul astfel încât să regăsească toate informațiile din tabelul EMP_pnu, pentru opțiunea 2, cursorul va regăsi doar angajații având salariul cuprins între 10000 şi 20000, iar pentru opțiunea 3 se vor regăsi salariații angajați în anul 1990.

```
ACCEPT p_optiune PROMPT ‘Introduceti optiunea (1,2 sau 3) ‘
DECLARE
  TYPE  emp_tip IS REF CURSOR RETURN emp_pnu%ROWTYPE;
  V_emp emp_tip;
  V_optiune NUMBER := &p_optiune;
```

```
BEGIN
  IF v_optiune = 1 THEN
   OPEN v_emp FOR SELECT * FROM emp_pnu;
   --!!! Introduceți cod pentru afişare
  ELSIF v_optiune = 2 THEN
   OPEN v_emp FOR SELECT * FROM emp_pnu
               WHERE salary BETWEEN 10000 AND 20000;
   --!!! Introduceți cod pentru afişare
  ELSIF v_optiune = 3 THEN
   OPEN emp_pnu FOR SELECT * FROM emp_pnu
                WHERE TO_CHAR(hire_date, 'YYYY') = 1990;
   --!!! Introduceți cod pentru afişare
  ELSE
   DBMS_OUTPUT.PUT_LINE('Optiune incorecta');
  END IF;
END;
/
```

13. Să se citească o valoare n de la tastatura. Prin intermediul unui cursor deschis cu ajutorul unui şir dinamic să se regăsească angajații având salariul mai mare decât n. Pentru fiecare linie regăsită de cursor, dacă angajatul are comision, să se afişeze numele său şi salariul.

```
ACCEPT p_nr PROMPT 'Introduceți limita inferioara a salariului'

DECLARE
  TYPE  empref IS REF CURSOR;
  V_emp empref;
  v_nr  INTEGER := 100000;
BEGIN
  OPEN v_emp FOR
    'SELECT employee_id, salary FROM emp_pnu WHERE salary> :bind_var'
     USING v_nr;
 -- introduceti liniile corespunzatoare rezolvării problemei
END;
```

Exerciții: [Expresii cursor]

14. Să se listeze numele regiunilor şi pentru fiecare regiune să se afişeze numele țărilor. Se cer 2 metode de rezolvare (secvențial şi cu expresii cursor).

Varianta 1:

```
BEGIN
  FOR c_regiune IN (SELECT region_id, region_name
                FROM   regions)
  LOOP
    DBMS_OUTPUT.PUT_LINE (c_regiune.region_name);
    FOR c_tara IN (SELECT country_id, country_name
                FROM   countries
                WHERE  region_id =  c_regiune.region_id)
    LOOP
      DBMS_OUTPUT.PUT_LINE (c_tara.country_name);
    END LOOP;
  END LOOP;
END;
```

Varianta 2:

```
DECLARE
  CURSOR c_regiune IS
```

```
   SELECT region_name,
         CURSOR (SELECT country_name
              FROM   countries c
              WHERE  c.region_id = r.region_id)
   FROM   regions r;
  v_regiune   regions.region_name%TYPE;
  v_tara      SYS.REFCURSOR;
  TYPE tara_nume IS TABLE OF countries.country_name%TYPE
                INDEX BY BINARY_INTEGER;
  v_nume_tara  tara_nume;
BEGIN
  OPEN c_regiune;
  LOOP
   FETCH c_regiune INTO v_regiune, v_tara;
   EXIT WHEN c_regiune%NOTFOUND;
   DBMS_OUTPUT.PUT_LINE (v_regiune);
   FETCH v_tara BULK COLLECT INTO v_nume_tara;
   FOR ind IN v_nume_tara.FIRST..v_nume_tara.LAST
    LOOP
     DBMS_OUTPUT.PUT_LINE (v_nume_tara (ind));
    END LOOP;
  END LOOP;
  CLOSE c_regiune;
END;
/
```

Baze de date-Anul 3 (semestrul 1)
Laborator 4 PL/SQL

Subprograme PL/SQL (funcții şi proceduri)

Un subprogram este un bloc PL/SQL cu nume (spre deosebire de blocurile anonime) care poate primi parametri şi poate fi invocat dintr-un anumit mediu (de exemplu, SQL*Plus, Oracle Forms, Oracle Reports etc.)

Subprogramele sunt bazate pe structura de bloc PL/SQL. Similar, ele conțin o parte declarativă facultativă, o parte executabilă obligatorie şi o parte de tratare de excepții facultativă.

- Exista 2 tipuri de subprograme:
  - proceduri;
  - funcții (trebuie să conțină cel puțin o comandă RETURN);
- Subprogramele pot fi :
  - locale (în cadrul altui bloc PL/SQL sau subprogram)
  - stocate (create cu comanda CREATE) - odată create, procedurile şi funcțiile sunt stocate în baza de date de aceea ele se numesc subprograme stocate.

Ø Sintaxa simplificată pentru crearea unei proceduri este următoarea:

[*CREATE* [*OR REPLACE*]] *PROCEDURE nume_procedură* [(*lista_parametri*)]
{*IS* | *AS*}
[*declarații locale*]
BEGIN
partea executabilă
[*EXCEPTION*
partea de tratare a excepțiilor]
END [*nume_procedură*];

Ø Sintaxa simplificată pentru crearea unei funcții este următoarea:

[*CREATE* [*OR REPLACE*]] *FUNCTION nume_funcție*
[(*lista_parametri*)]
RETURN tip_de_date
{*IS* | *AS*}
[*declarații locale*]
BEGIN
partea executabilă
[*EXCEPTION*
partea de tratare a excepțiilor]
END [*nume_funcție*];

- Lista de parametri conține specificații de parametri separate prin virgulă de forma :

  nume_parametru mod_parametru tip_parametru;

  - *mod_parametru* specifică dacă parametrul este:
    - de intrare (IN) – singurul care poate avea o valoare inițială
    - de intrare / ieşire (IN OUT)
    - de ieşire (OUT)
  - *mod_parametru* are valoarea implicită IN.

- O functie îndeplineste urmatoarele conditii:-
  - Accepta numai parametrii de tip IN
  - Accepta numai tipuri de date SQL, nu si tipuri specifice PL/SQL
  - Returneaza valori de tipuri de date SQL.
  - Nu modifica tabelul care este blocat pentru comanda respectiva (*mutating tables*)

- Poate fi folosita in lista de expresii a comenzii SELECT, clauza WHERE si HAVING, CONNECT BY, START WITH, ORDER BY, GROUP BY, clauza VALUES a comenzii INSERT, clauza SET a comenzii UPDATE.

• În cazul în care se modifică un obiect (vizualizare, tabel etc) de care depinde un subprogram, acesta este invalidat. Revalidarea se face fie prin recrearea subprogramului fie prin comanda:

ALTER PROCEDURE nume_proc COMPILE;
ALTER FUNCTION nume_functie COMPILE;

• Ştergerea unei funcții sau proceduri se realizează prin comenzile:
DROP PROCEDURE nume_proc;
DROP FUNCTION nume_functie;

Ø Informații despre procedurile şi funcțiile deținute de utilizatorul curent se pot obține interogând vizualizarea *USER_OBJECTS* din dicționarul datelor.

```
SELECT   OBJECT_NAME, OBJECT_TYPE, STATUS
FROM     USER_OBJECTS
WHERE    OBJECT_TYPE IN ('PROCEDURE','FUNCTION');
```

Obs: STATUS – starea subprogramului (validă sau invalidă).

- Codul complet al unui subprogram poate fi vizualizat folosind următoarea sintaxă:
```
SELECT   TEXT
FROM     USER_SOURCE
WHERE    NAME = 'nume_subprogram'
ORDER BY LINE;
```
- Eroarea apărută la compilarea unui subprogram poate fi vizualizată folosind următoarea sintaxă:
```
SELECT   LINE, POSITION, TEXT
FROM     USER_ERRORS
WHERE    NAME = 'nume';
```
- Erorile pot fi vizualizate şi prin intermediul comenzii *SHOW ERRORS*.
- Descrierea specificației unui subprogram se face prin comanda *DESCRIBE*.

Ø Când este apelată o procedură *PL/SQL*, sistemul *Oracle* furnizează două metode pentru definirea parametrilor actuali:

- specificarea explicită prin nume;
- specificarea prin poziție.

Exemplu: subprog(a tip_a, b_tip_b, c tip_c, d tip_d)
- specificare prin poziție:
subprog(var_a,var_b,var_c,var_d);
- specificare prin nume
subprog(b=>var_b,c=>var_c,d=>var_d,a=>var_a);
- specificare prin nume şi poziție
subprog(var_a,var_b,d=>var_d,c=>var_c);

Exerciții:

I. [Proceduri locale]

1. Să se declare o procedură locală într-un bloc PL/SQL anonim prin care să se introducă în tabelul DEP_pnu o nouă înregistrare precizând, prin intermediul parametrilor, valori pentru toate câmpurile. Invocați procedura în cadrul blocului. Interogați tabelul DEP_pnu şi apoi anulați modificările (ROLLBACK).

DECLARE

```
 PROCEDURE add_dept
          (p_cod         dep_pnu.department_id %TYPE,
           p_nume        dep_pnu.department_name %TYPE,
           p_manager  dep_pnu.manager_id %TYPE,
           p_location   dep_pnu.location_id%TYPE)
  IS
 BEGIN
  INSERT  INTO dep_pnu
  VALUES  (p_cod, p_nume, p_manager, p_location);
 END;

BEGIN
 Add_dept(45, 'DB Administration' , 100, 2700);
END;
/
SELECT  *
FROM    dep_pnu;
ROLLBACK;
```

2. Să se declare o procedură locală care are parametrii următori:
- p_rezultat (parametru de tip IN OUT) de tipul coloanei last_name din tabelul employees;
- p_comision (parametru de tip OUT) de tipul coloanei commission_pct din employees, inițializat cu NULL;
- p_cod (parametru de tip IN) de tipul coloanei employee_id din employees, inițializat cu NULL.

Dacă p_comision nu este NULL atunci în p_rezultat se va memora numele salariatului care are salariul maxim printre salariații având comisionul respectiv. În caz contrar, în p_rezultat se va memora numele salariatului al cărui cod are valoarea dată la apelarea procedurii.

```
SET SERVEROUTPUT ON
DECLARE
        v_nume employees.last_name%TYPE;
PROCEDURE p2l4_pnu (p_rezultat     IN OUT  employees.last_name% TYPE,
                            p_comision   IN     employees.commission_pct %TYPE:=NULL,
                            p_cod          IN     employees.employee_id %TYPE:=NULL)
 IS
 BEGIN
        IF (p_comision IS NOT NULL) THEN
                SELECT last_name
                INTO       p_rezultat
                FROM     employees
                WHERE  commission_pct= p_comision
                AND salary = (SELECT MAX(salary)
                                FROM     employees
                                WHERE  commission_pct = p_comision);
                DBMS_OUTPUT.PUT_LINE('Numele salariatului care are comisionul '||p_comision||
                        ' este '||p_rezultat);
        ELSE
                SELECT last_name
                INTO       p_rezultat
                FROM     employees
                WHERE  employee_id = p_cod;
                DBMS_OUTPUT.PUT_LINE('numele salariatului avand codul '||p_cod||
                        ' este '||p_nume);
        END IF;
 END;

BEGIN -- partea executabilă a blocului
```

```
        p2l4_pnu (nume,0.4);
        p2l4_pnu (nume,cod=>205);
END;
/
SET SERVEROUTPUT OFF
```

II. [Proceduri stocate]

3. Să se creeze o procedură stocată fără parametri care afişează un mesaj “Programare PL/SQL”, ziua de astăzi în formatul DD-MONTH-YYYY şi ora curentă, precum şi ziua de ieri în formatul DD-MON-YYYY.

```
CREATE PROCEDURE first_pnu IS
        azi DATE := SYSDATE;
        ieri azi%TYPE;
BEGIN
        DBMS_OUTPUT.PUT_LINE('Programare PL/SQL') ;
        DBMS_OUTPUT.PUT_LINE(TO_CHAR(azi, 'dd-month-yyyy hh24:mi:ss'));
        ieri := azi -1;
        DBMS_OUTPUT.PUT_LINE(TO_CHAR(ieri, 'dd-mon-yyyy'));
END;
/
```

La promptul SQL apelam procedura astfel: EXECUTE first_pnu;

4. Să se şteargă procedura precedentă şi să se re-creeze, astfel încât să accepte un parametru IN de tip VARCHAR2, numit p_nume. Mesajul afişat de procedură va avea forma « <p_nume> invata PL/SQL». Invocați procedura cu numele utilizatorlui curent furnizat ca parametru.

```
DROP PROCEDURE first_pnu ;

CREATE PROCEDURE first_pnu(p_nume VARCHAR2) IS
….
```

Pentru apel: EXECUTE first_pnu(USER);

5. a) Creați o copie JOBS_pnu a tabelului JOBS. Implementați constrângerea de cheie primară asupra lui JOBS_pnu.

```
CREATE TABLE jobs_pnu AS SELECT * FROM jobs ;
ALTER TABLE jobs_pnu ADD CONSTRAINT pk_jobs_pnu PRIMARY KEY(job_id);
```

b) Creați o procedură ADD_JOB_pnu care inserează un nou job în tabelul JOBS_pnu. Procedura va avea 2 parametri IN p_id şi p_title corespunzători codului şi denumirii noului job.

```
CREATE OR REPLACE PROCEDURE ADD_JOB_pnu
        (p_job_id IN jobs.job_id%TYPE, p_job_title IN jobs.job_title%TYPE)
IS
BEGIN
        INSERT INTO jobs_pnu (job_id, job_title)
        VALUES (p_job_id, p_job_title);
        COMMIT;
END add_job_pnu;
```

c) Testați procedura, invocând-o astfel:

```
EXECUTE ADD_JOB_pnu('IT_DBA', 'Database Administrator);
  SELECT * FROM JOBS_pnu;
EXECUTE ADD_JOB_pnu('ST_MAN', 'Stock Manager');
   SELECT * FROM JOBS_pnu;
```

6. a) Creați o procedură stocată numită UPD_JOB_pnu pentru modificarea unui job existent în tabelul JOBS_pnu. Procedura va avea ca parametri codul job-ului şi noua sa denumire (parametri IN). Se va trata cazul în care nu are loc nici o actualizare.

```
CREATE OR REPLACE PROCEDURE UPD_JOB_pnu
```

```
        (p_job_id IN jobs.job_id%TYPE, p_job_title IN jobs.job_title%TYPE)
IS
BEGIN
        UPDATE jobs_pnu
        SET job_title =  p_job_title
        WHERE job_id = p_job_id;
        IF SQL%NOTFOUND THEN
                RAISE_APPLICATION_ERROR(-20202, 'Nici o actualizare);
                -- sau doar cu afisare mesaj
                -- DBMS_OUTPUT.PUT_LINE('Nici o actualizare');
        END IF;
END upd_job_pnu;
```

b) Testați procedura, invocând-o astfel:

```
EXECUTE UPD_JOB_pnu('IT_DBA', 'Data Administrator');
  SELECT * FROM job_pnu
  WHERE UPPER(job_id) = 'IT_DBA'
EXECUTE UPD_JOB('IT_WEB', 'Web master');
```

Obs : A doua invocare va conduce la apariția excepției. Analizați ce s-ar fi întâmplat dacă nu prevedeam această excepție, punând între comentarii liniile aferente din procedură şi recreând-o cu CREATE OR REPLACE PROCEDURE…

7. a) Creați o procedură stocată numită DEL_JOB_pnu care şterge un job din tabelul JOBS_pnu. Procedura va avea ca parametru (IN) codul job-ului. Includeți o excepție corespunzătoare situației în care nici un job nu este şters.

b) Testați procedura, invocând-o astfel:

```
DEL_JOB_pnu('IT_DBA');
DEL_JOB_pnu('IT_WEB');
```

8. a) Să se creeze o procedură stocată care calculează salariul mediu al angajaților, returnându-l prin intermediul unui parametru de tip OUT.

b) Să se apeleze procedura regăsind valoarea medie a salariilor într-o variabilă gazdă. Afişați valoarea variabilei.

```
CREATE OR REPLACE PROCEDURE p8l4_pnu (p_salAvg OUT employees.salary%TYPE)
AS
BEGIN
    SELECT AVG(salary)
    INTO p_salAvg
    FROM employees;
END;
/
```

La *prompt*-ul SQL (sau într-in fişier *script*):

```
VARIABLE g_medie NUMBER
EXECUTE p8l4_pnu (:g_medie)
PRINT g_medie
```

9. a) Să se creeze o procedură stocată care primeşte printr-un parametru salariul unui angajat şi returnează prin intermediul aceluiaşi parametru salariul actualizat astfel: dacă salariul este mai mic decât 3000, valoarea lui creşte cu 20%, dacă este cuprins între 3000 şi 7000 valoarea lui creşte cu 15%, dacă este mai mare decât 7000 va fi mărit cu 10%, iar dacă este null va lua valoarea 1000.

b) Să se declare o variabilă gazdă g_sal (VARIABLE). Să se scrie un bloc anonim PL/SQL prin care se va atribui variabilei g_sal valoarea unei variabile de substituție citite de la tastatură (ACCEPT). Să se apeleze procedura pentru această valoare şi să se afişeze valoarea returnată.

In fişierul p9l4a.sql:

```
CREATE OR REPLACE PROCEDURE p9l4_pnu  (p_sal IN OUT NUMBER)
```

```
IS
BEGIN
        CASE
                WHEN p_sal < 3000 THEN p_sal := p_sal*1.2;
        .................
                ELSE p_sal := 1000;
        END CASE;
END;
/
```

In fişierul p9l4b.sql:

```
VARIABLE g_sal NUMBER
ACCEPT p_sal PROMPT 'Introduceti salariul '
BEGIN
 :g_sal:=&p_sal;
END;
/
PRINT g_sal
EXECUTE p9l4_pnu (:g_sal)
PRINT g_sal
```

III. [Funcții locale]

10. Să se creeze o procedură stocată care pentru un anumit cod de departament (dat ca parametru) calculează prin intermediul unor funcții locale numărul de salariați care lucrează în el, suma salariilor şi numărul managerilor salariaților care lucrează în departamentul respectiv.

```
CREATE OR REPLACE PROCEDURE p11l4_pnu
                (p_dept  employees.department_id%TYPE) AS
  FUNCTION nrSal (v_dept  employees.department_id %TYPE)
     RETURN NUMBER IS
       v_numar  NUMBER(3);
     BEGIN
       SELECT COUNT(*)
       INTO   v_numar
       FROM   employees
       WHERE  department_id = v_dept;
       RETURN v_numar;
  END nrSal;

  FUNCTION sumaSal(v_dept  employees.department_id %TYPE)
     RETURN NUMBER IS
        v_suma  employees.salary%TYPE;
     BEGIN
       SELECT SUM(salary)
       INTO      v_suma
       FROM    employees
       WHERE  department_id = v_dept;
        RETURN v_suma;
  END sumaSal;

  FUNCTION nrMgr(v_dept  employees.department_id %TYPE)
     RETURN NUMBER IS
       v_numar  NUMBER(3);
     BEGIN
       SELECT COUNT(DISTINCT manager_id)
```

```
        INTO   v_numar
        FROM   employees
        WHERE  department_id = v_dept;
        RETURN v_numar;
  END nrMgr;

BEGIN – partea executabila a procedurii p11l4
  DBMS_OUTPUT.PUT_LINE('Numarul salariatilor care lucreaza in departamentul '||p_dept||  ' este '|| nrSal(p_dept));
  DBMS_OUTPUT.PUT_LINE('Suma salariilor angajatilor din departamentul '|| p_dept || ' este '|| sumaSal(p_dept));
  DBMS_OUTPUT.PUT_LINE('Numarul de manageri din departamentul '|| p_dept || ' este '|| nrMgr(p_dept));
END;
/
EXECUTE p11l4_pnu(50);
```

11. Să se creeze două funcții (locale) supraîncărcate (overload) care să calculeze media salariilor astfel:

- prima funcție va avea ca argument codul departamentului, adică funcția calculează media salariilor din departamentul specificat;
- a doua funcție va avea două argumente, unul reprezentând codul departamentului, iar celălalt reprezentând job-ul, adică funcția va calcula media salariilor dintr-un anumit departament şi care aparțin unui job specificat.

```
DECLARE
        medie1 NUMBER(10,2);
        medie2 NUMBER(10,2);
        FUNCTION medie (v_dept employees.department_id%TYPE)
                        RETURN NUMBER IS
                rezultat NUMBER(10,2);
        BEGIN
                SELECT AVG(salary)
                INTO   rezultat
                FROM   employees
                WHERE  department_id = v_dept;
                RETURN rezultat;
        END;
        FUNCTION medie (v_dept employees.department_id%TYPE,
                                v_job employees.job_id %TYPE)
                        RETURN NUMBER IS
                rezultat NUMBER(10,2);
        BEGIN
                SELECT AVG(salary)
                INTO   rezultat
                FROM   employees
                WHERE  department_id = v_dept AND job_id = v_job;
                RETURN rezultat;
        END;
BEGIN
        medie1:=medie(80);
        DBMS_OUTPUT.PUT_LINE('Media salariilor din departamentul  80  este ' || medie1);
        medie2 := medie(80, 'SA_REP');
        DBMS_OUTPUT.PUT_LINE('Media salariilor reprezentantilor de vanzari din
                                        departamentul 80 este ' || medie2);
        END;
```

IV. [Funcții stocate]

12. Să se creeze o funcție stocată care determină numărul de salariați din employees angajați după 1995, într-un departament dat ca parametru. Să se apeleze această funcție prin diferite modalități:
- printr-o variabilă de legătură;
- folosind comanda CALL;
- printr-o comandă SELECT;
- într-un bloc PL/SQL.

```
CREATE OR REPLACE FUNCTION  p14l4_pnu (p_dept  employees.department_id%TYPE)
  RETURN NUMBER
   IS
  rezultat NUMBER;
BEGIN
  SELECT  COUNT(*)
  INTO    rezultat
  FROM    employees
  WHERE   department_id=p_dept
  AND     TO_CHAR(hire_date,'yyyy')>1995;
  RETURN rezultat;
END p14l4_pnu;
/
```

```
1)  VARIABLE nr NUMBER
      EXECUTE :nr := p14l4_pnu (80);
      PRINT nr
```

```
2)   VARIABLE nr NUMBER
      CALL p14l4_pnu (50) INTO :nr;
      PRINT nr
```

```
3)  SELECT p14l4_pnu (80)
      FROM   dual;
```

```
4)   SET SERVEROUTPUT ON
      DEFINE p_dep=50 – sau ACCEPT p_dep PROMPT ‘Instroduceti codul departamentului ’
      DECLARE
      nr  NUMBER;
      v_dep  employees.department_id%TYPE := &p_dep;
     BEGIN
       nr := p14l4_pnu (v_dep);
       IF nr<>0 THEN
         DBMS_OUTPUT.PUT_LINE('numarul salariatilor angajati după 1995 in
                                departamentul '||v_dep || ' este '||nr);
       ELSE
         DBMS_OUTPUT.PUT_LINE('departamentul cu numarul '|| v_dep || ' nu are angajati');
       END IF;
     END;
     /
SET SERVEROUTPUT OFF
```

13. a) Creați o funcție numită VALID_DEPTID_pnu pentru validarea unui cod de departament specificat ca parametru. Funcția va întoarce o valoare booleana (TRUE dacă departamentul există).

b) - Creați o procedură numită ADD_EMP_pnu care adaugă un angajat în tabelul EMP_pnu. Linia respectivă va fi adăugată în tabel doar dacă departamentul specificat este valid, altfel utilizatorul va primi un mesaj adecvat.

- Procedura va avea următorii parametrii, cu valorile DEFAULT specificate între paranteze : first_name, last_name, email, job_id (SA_REP), manager_id (145), salary (1000), commission_pct (0), department_id (30).

- Pentru codul angajatului se va utiliza o secvență EMP_SEQ_pnu, iar data angajării se consideră a fi TRUNC(SYSDATE).

c) Testați procedura, adăugând un angajat pentru care se specifică numele, prenumele, codul departamentului = 15, iar restul parametrilor se lasă DEFAULT.

Adăugați un angajat pentru care se specifică numele, prenumele, codul departamentului =80, iar restul parametrilor rămân la valorile DEFAULT.

Adăugați un angajat precizând valori pentru toți parametrii procedurii.

```
CREATE OR REPLACE FUNCTION valid_deptid_pnu
        (p_deptid IN departments.department_id%TYPE)
        RETURN boolean
IS
        v_aux VARCHAR2(1);
BEGIN
        SELECT 'x'
        INTO     v_aux
        FROM   departments
        WHERE department_id = p_deptid;
        RETURN (TRUE);
        EXCEPTION
                WHEN NO_DATA_FOUND THEN
                        RETURN(FALSE);
END valid_deptid_pnu;

CREATE OR REPLACE PROCEDURE add_emp_pnu
        (p_lname     employees.last_name%TYPE,
         p_fname     employees.first_name%TYPE,
         p_email     employees.email%TYPE,
         p_ job      employees.job_id%TYPE              DEFAULT 'SA_REP',
         p_mgr       employees.manager_id%TYPE      DEFAULT 145,
         p_sal       employees.salary%TYPE              DEFAULT 1000,
         p_comm      employees.commission_pct%TYPE DEFAULT 0,
         p_deptid    employees.department_id%TYPE   DEFAULT 30)
IS
BEGIN
        IF valid_deptid_pnu(p_deptid) THEN
                INSERT INTO emp_pnu (employee_id, last_name, …)
                VALUES (emp_seq_pnu.NEXTVAL, p_lname, …);
        ELSE
                RAISE _APPLICATION_ERROR(-20204, 'Cod invalid de departament.');
        END IF;
END add_emp_pnu;

EXECUTE add_emp_pnu(p_lname => 'Harris', p_fname=>'Jane', p_email =>'JHarris',
        p_dept_id=>15);
EXECUTE add_emp_pnu(p_lname => 'Harris', p_fname=>'Joe', p_email =>'JoHarris',
        p_dept_id=>80);
```

14. Să se calculeze recursiv numărul de permutări ale unei mulțimi cu n elemente, unde n va fi transmis ca parametru.

```
CREATE OR REPLACE FUNCTION permutari_pnu(p_n NUMBER)
        RETURN INTEGER IS
```

```
BEGIN
        IF (n=0) THEN
                RETURN 1;
        ELSE
                RETURN p_n*permutari_pnu (n-1);
        END IF;
END permutari_pnu;
/
VARIABLE g_n NUMBER
EXECUTE :g_n := permutari_pnu (5);
PRINT g_n
```

15. Să se afişeze numele, job-ul şi salariul angajaților al căror salariu este mai mare decât media salariilor din tabelul employees.

```
CREATE OR REPLACE FUNCTION medie_pnu
  RETURN NUMBER IS
medie NUMBER;
BEGIN
  SELECT AVG(salary)
  INTO      medie
  FROM   employees;
  RETURN medie;
END;
/
SELECT last_name, job_id, salary
FROM   employees
WHERE  salary >= medie_pnu;
```

Baze de date-Anul 3 (semestrul 1)
Laborator 5 PL/SQL

Triggeri (declanşatori)

Ø Un trigger
- este un bloc PL/SQL asociat unui tabel, view, scheme sau unei baze de date.
- trigger-ul se executa implicit ori de câte ori are loc un anumit eveniment
- pot fi de următoarele tipuri:
  - trigger-i la nivel de aplicație: se declanşează odată un un anumit eveniment din aplicație;
  - trigger-i la nivel de bază de date: se declanşează atunci când un eveniment asupra datelor (de ex, LMD) sau un eveniment sistem (logon, shutdown) apare asupra unei scheme sau asupra bazei de date.

Ø Instrucțiunea pentru crearea trigger-ilor LMD conține următoarele informații:
- o timpul declanşării trigger-ului în raport cu evenimentul:
  - pentru tabele: BEFORE, AFTER
  - pentru view-uri nemodificabile: INSTEAD OF
- o evenimentul declanşator: INSERT, UPDATE, DELETE
- o numele tabelului
- o tipul trigger-ului – precizează de câte ori se execută corpul acestuia; trigger-ul poate fi la nivel de:
  - instrucțiune (statement): corpul triggerului se execută o singură dată pentru evenimentul declanşator. Un astfel de trigger se declanşează chiar dacă nici o linie nu este afectată.
  - linie (row): corpul triggerului se declanşează o dată pentru fiecare linie afectată de către evenimentul declanşator. Un astfel de trigger nu se execută dacă evenimentul declanşator nu afectează nici o linie.
- o clauza WHEN – precizează o condiție restrictivă
- o corpul trigger-ului (blocul PL/SQL)

Ø Sintaxa comenzii de creare a unui trigger LMD este:

CREATE [*OR REPLACE*] *TRIGGER* [*schema.*]*nume_trigger*
{*BEFORE* | *AFTER*}
[*INSTEAD OF*]
{*DELETE* | *INSERT* | *UPDATE* [*OF coloana*[, *coloana …*]] }
[*OR* {*DELETE* | *INSERT* | *UPDATE* [*OF coloana*[, *coloana …*]] …}
ON [*schema.*]*nume_tabel*
[*REFERENCING* {*OLD* [*AS*] *vechi NEW* [*AS*] *nou*
| *NEW* [*AS*] *nou OLD* [*AS*] *vechi* }]
[*FOR EACH ROW*]
[*WHEN* (*condiție*)]
corp_trigger;

Ø Informații despre triggeri se găsesc în următoarele vizualizări ale dicționarului datelor: *USER_TRIGGERS*, *USER_TRIGGER_COL*, *ALL_TRIGGERS*, *DBA_TRIGGERS.*

Ø *Modificarea unui declanşator constă din redenumirea, recompilarea, activarea sau dezactivarea acestuia şi se realizează prin comenzi de forma:*

ALTER TRIGGER nume_trigger ENABLE;
ALTER TRIGGER nume_trigger DISABLE;
ALTER TRIGGER nume_trigger COMPILE;
ALTER TRIGGER nume_trigger RENAME TO nume_nou;

Activarea şi dezactivarea tuturor triggerilor asociați unui tabel se realizează prin comenzile:

ALTER TABLE nume_tabel
DISABLE ALL TRIGGERS;
ALTER TABLE nume_tabel
ENABLE ALL TRIGGERS;

Eliminarea unui declanşator se face prin
DROP TRIGGER nume_trigger;

Ø Sintaxa pentru crearea unui declanşator sistem este următoarea

CREATE [*OR REPLACE*] *TRIGGER* [*schema.*]*nume_declanşator*
{*BEFORE* | *AFTER*}
{*lista_evenimente_LDD* | *lista_evenimente_bază*}
ON {*DATABASE* | *SCHEMA*}
[*WHEN* (*condiție*)]
corp_declanşator;

unde: *lista_evenimente_LDD - CREATE*, *DROP*, *ALTER*)
lista_evenimente_bază - STARTUP, *SHUTDOWN*, *LOGON*, *LOGOFF*, *SERVERERROR*, *SUSPEND*)

Exerciții

1. Să se creeze un trigger care asigură ca inserarea de angajați în tabelul EMP_PNU se poate realiza numai în zilele lucrătoare, între orele 8-18.

Obs:Trigger-ul nu are legătură directă cu datele => este un ***trigger la nivel de instrucțiune.***

```
CREATE OR REPLACE TRIGGER b_i_emp_pnu
    BEFORE INSERT ON emp_pnu
    BEGIN
        IF (TO_CHAR(SYSDATE, 'dy') IN ('sat', 'sun')) OR
            (TO_CHAR(SYSDATE, 'HH24:MI')
                NOT BETWEEN '08:00' AND '18:00')
        THEN
            RAISE_APPLICATION_ERROR (-20500, 'Nu se pot introduce
                inregistrari decat in timpul orelor de lucru');
        END IF;
END;
/
```

Testați trigger-ul:

```
INSERT INTO emp_pnu (employee_id, last_name, first_name, email, hire_date,
        job_id, salary, department_id)
VALUES (300, 'Smith', 'Robert', 'rsmith', SYSDATE, 'IT_PROG', 4500, 60);
```

2. Modificați trigger-ul anterior, astfel încât să fie generate erori cu mesaje diferite pentru inserare, actualizare, actualizarea salariului, ştergere.

```
CREATE OR REPLACE TRIGGER b_i_emp_pnu
    BEFORE INSERT OR UPDATE OR DELETE ON emp_pnu
    BEGIN
    IF (TO_CHAR(SYSDATE, 'dy') IN ('sat', 'sun')) OR
        (TO_CHAR(SYSDATE, 'HH24:MI') NOT BETWEEN '08:00' AND '18:00')
    THEN
        IF DELETING THEN
```

```
                RAISE_APPLICATION_ERROR (-20501, 'Nu se pot
                        sterge inregistrari decat in timpul orelor de luru');
            ELSIF INSERTING THEN
                RAISE_APPLICATION_ERROR (-20500, 'Nu se pot
                        adauga inregistrari decat in timpul orelor de lucru');
            ELSIF UPDATING ('SALARY') THEN
                RAISE_APPLICATION_ERROR (-20502, 'Nu se poate
                        actualiza campul SALARY decat in timpul orelor de
                        lucru');
            ELSE
                RAISE_APPLICATION_ERROR (-20503, 'Nu se pot
                        actualiza inregistrari decat in timpul orelor de
                        lucru');
            END IF;
        END IF;
END;
/
```

3. Să se creeze un trigger care să permită ca numai salariații având codul job-ului AD_PRES sau AD_VP să poată câştiga mai mult de 15000.

Obs: Trigger-ul se declanşează de un număr de ori = nr de înregistrări inserate sau al căror câmp salary este modificat (deci are legătură cu datele din tabel) => este un ***trigger la nivel de linie***.

```
CREATE OR REPLACE TRIIGER b_i_u_emp_pnu
    BEFORE INSERT OR UPDATE OF salary ON emp_pnu
    FOR EACH ROW
    BEGIN
        IF NOT (:NEW>job_id IN ('AD_PRES', 'AD_VP'))
                AND :NEW.salary > 15000
        THEN
                RAISE_APPLICTION_ERROR (-20202, 'Angajatul nu poate
    castiga aceasta suma');
        END IF;
END;
/
```

4. Să se implementeze cu ajutorul unui declanşator constrângerea că valorile salariilor nu pot fi reduse (trei variante). După testare, suprimați trigger-ii creați.

Varianta 1:

```
CREATE OR REPLACE TRIGGER verifica_salariu_pnu
  BEFORE UPDATE OF salary ON emp_pnu
  FOR EACH ROW
  WHEN (NEW.salary < OLD.salary)
BEGIN
  RAISE_APPLICATION_ERROR (-20222, 'valoarea unui salariu nu poate fi micşorată');
END;
/
Update emp_pnu
Set salary = salary/2;
Drop trigger verifica_salariu_pnu;
```

Varianta 2:

```
CREATE OR REPLACE TRIGGER verifica_salariu_pnu
  BEFORE UPDATE OF salary ON emp_pnu
  FOR EACH ROW
BEGIN
  IF (:NEW.salary < :OLD.salary) THEN
    RAISE_APPLICATION_ERROR (-20222, 'valoarea unui salariu nu poate fi micsorata');
  END IF;
END;
/
```

Varianta 3:

```
CREATE OR REPLACE PROCEDURE p4l6_pnu IS
   BEGIN
         RAISE_APPLICATION_ERROR (-20222, 'valoarea unui salariu nu poate fi
                                   micsorata');
END;
/
CREATE OR REPLACE TRIGGER verifica_salariu_pnu
  BEFORE UPDATE OF salary ON emp_pnu
  FOR EACH ROW
  WHEN (NEW.salary < OLD.salary)
  CALL p4l6_pnu;
```

5. Să se creeze un trigger care calculează comisionul unui angajat 'SA_REP' atunci când este adăugată o linie tabelului emp_pnu sau când este modificat salariul.

Obs: Dacă se doreşte atribuirea de valori coloanelor utilizând NEW, trebuie creați triggeri BEFORE ROW. Dacă se încearcă scrierea unui trigger AFTER ROW, atunci se va obține o eroare la compilare.

```
CREATE OR REPLACE TRIGGER b_i_u__sal_emp_pnu
      BEFORE INSERT OR UPDATE OF salary ON emp_pnu
      FOR EACH ROW
      WHEN (NEW.job_id = 'SA_REP')
BEGIN
      IF INSERTING
          THEN :NEW.commission_pct := 0;
      ELSIF :OLD.commission_pct IS NULL
          THEN :NEW.commission_pct := 0;
      ELSE :NEW.commission_pct := :OLD.commission_pct * (:NEW.salary/:OLD.salary);
      END IF;
END;
/
```

6. Să se implementeze cu ajutorul unui declanşator constrângerea că, dacă salariul minim şi cel maxim al unui job s-ar modifica, orice angajat având job-ul respectiv trebuie să aibă salariul între noile limite .

```
CREATE OR REPLACE TRIGGER verifica_sal_job_pnu
         BEFORE UPDATE OF min_salary, max_salary ON jobs_pnu
         FOR EACH ROW
DECLARE
         v_min_sal  emp_pnu.salary%TYPE;
         v_max_sal  emp_pnu.salary%TYPE;
```

```
        e_invalid  EXCEPTION;
BEGIN
        SELECT MIN(salary), MAX(salary)
          INTO   v_min_sal, v_max_sal
          FROM   emp_pnu
          WHERE  job_id = :NEW.job_id;
        IF (v_min_sal < :NEW.min_salary) OR
           (v_max_sal > :NEW.max_salary) THEN
                RAISE e_invalid;
        END IF;
EXCEPTION
        WHEN e_invalid THEN
                RAISE_APPLICATION_ERROR (-20567, 'Exista angajati avand salariul in afara
                        domeniului permis pentru job-ul corespunzator');
END verifica_sal_job_pnu;
/
```

7. Să se creeze un trigger check_sal_pnu care garantează ca, ori de câte ori un angajat nou este introdus în tabelul EMPLOYEES sau atunci când este modificat salariul sau codul job-ului unui angajat, salariul se încadrează între minimul şi maximul salariior corespunzătoare job-ului respectiv. Se vor exclude angajatii AD_PRES.

```
CREATE OR REPLACE TRIGGER check_sal_pnu
        BEFORE INSERT OR UPDATE OF salary, job_id
        ON emp_pnu
        FOR EACH ROW
        WHEN (NEW.job_id <> 'AD_PRES')
DECLARE
        v_min employees.salary %TYPE;
        v_max employees.salary %TYPE;
BEGIN
        SELECT MIN(salary), MAX(salary)
           INTO v_min, v_max
          FROM emp_pnu         -- FROM copie_emp_pnu
          WHERE job_id = :NEW.job_id;
        IF :NEW.salary < v_min OR
           :NEW.salary > v_max THEN
                RAISE_APPLICATION_ERROR (-20505, 'In afara domeniului');
        END IF;
END;
/
```

Testați trigger-ul anterior:

```
UPDATE emp_pnu
SET salary = 3500
WHERE last_name= 'Stiles';
```

Ce se obține şi de ce? Modificați trigger-ul astfel încât să funcționeze corect.

Obs: Tabelul este mutating. Pentru ca trigger-ul să funcționeze, utilizați o copie a tabelului emp_pnu în instrucțiunea SELECT din corpul trigger-ului. (aceasta este doar una dintre solutii, se vor vedea ulterior si altele).

8. a) Se presupune că în tabelul *dept_pnu* se păstrează (într-o coloană numită *total_sal*) valoarea totală a salariilor angajaților în departamentul respectiv. Introduceți această coloană în tabel şi actualizați conținutul.

```
ALTER TABLE dept_pnu
ADD (total_sal NUMBER(11, 2));

UPDATE dept_pnu
SET    total_sal =
         (SELECT  SUM(salary)
          FROM    emp_pnu
          WHERE   emp_pnu.department_id = dept_pnu.department_id);
```

b) Creați un trigger care permite reactualizarea automată a acestui câmp.

```
CREATE OR REPLACE PROCEDURE creste_total_pnu
        (v_cod_dep  IN dept_pnu.department_id%TYPE,
         v_sal      IN dept_pnu.total_sal%TYPE) AS
BEGIN
  UPDATE dept_pnu
  SET     total_sal = NVL (total_sal, 0) + v_sal
  WHERE   department_id = v_cod_dep;
END creste_total_pnu;
/
CREATE OR REPLACE TRIGGER calcul_total_pnu
  AFTER INSERT OR DELETE OR UPDATE OF salary ON emp_pnu
  FOR EACH ROW
BEGIN
  IF DELETING THEN
    creste_total_pnu (:OLD.department_id, -1*:OLD.salary);
  ELSIF UPDATING THEN
    creste_total_pnu (:NEW.department_id, :NEW.salary - :OLD.salary);
  ELSE /* inserting */
    Creste_total_pnu (:NEW.department_id, :NEW.salary);
  END IF;
END;
/
```

9. Să se creeze două tabele noi new_emp_pnu şi new_dept_pnu pe baza tabelelor employees şi departments. Să se creeze un view view_emp_pnu, care selectează codul, numele, salariul, codul departamentului, email-ul, codul job-ului, numele departamentului şi codul locației pentru fiecare angajat.
Să se creeze un ***trigger de tip INSTEAD OF*** care, în locul inserării unei linii direct în view, adaugă înregistrări corespunzătoare în tabelele new_emp_pnu şi new_dept_pnu. Similar, atunci când o linie este modificată sau ştearsă prin intermediul vizualizării, liniile corespunzătoare din tabelele new_emp_pnu şi new_dept_pnu sunt afectate.

```
CREATE TABLE new_emp_pnu AS
        SELECT employee_id, last_name, salary, department_id, email,
                        job_id, hire_date
        FROM employees;

CREATE TABLE new_dept_pnu AS
        SELECT d.department_id, d.department_name, d.location_id,
                SUM(e.salary) total_dept_sal
        FROM employees e, departments d
        WHERE e.department_id = d.department_id
```

```
        GROUP BY d.department_id, d.department_name, d.location_id;

    CREATE VIEW view_emp_pnu AS
        SELECT e.employee_id, e.last_name, e.salary, e.department_id, e.email,
            e.job_id, d.department_name, d.location_id
        FROM employees e, departments d
        WHERE e.department_id = d.department_id;

CREATE OR REPLACE TRIGGER new_emp_dept_pnu
INSTEAD OF INSERT OR UPDATE OR DELETE ON view_emp_pnu
FOR EACH ROW
BEGIN
    IF INSERTING THEN
        INSERT INTO new_emp_pnu
        VALUES(:NEW.employee_id, :NEW.last_name, :NEW.salary,
            :NEW.department_id, :NEW.email, :NEW.job_id, SYSDATE);
        UPDATE new_dept_pnu
        SET total_dept_sal = total_dept_sal + :NEW.salary
        WHERE department_id = :NEW.department_id;
    ELSIF DELETING THEN
        DELETE FROM new_emp_pnu
        WHERE employee_id = :OLD.employee_id;
        UPDATE new_dept_pnu
        SET total_dept_sal = total_dept_sal - :OLD.salary
        WHERE department_id = :OLD.department_id;
    ELSIF UPDATING ('salary') THEN
        UPDATE new_emp_pnu
        SET salary = :NEW.salary
        WHERE employee_id = :NEW.employee_id;
        UPDATE new_dept_pnu
        SET total_dept_sal = total_dept_sal + (:NEW.salary - :OLD.salary)
        WHERE department_id = :OLD.department_id;
    ELSIF UPDATING ('department_id') THEN
        UPDATE new_emp_pnu
        SET department_id = :NEW.department_id
        WHERE employee_id = :OLD.employee_id;
        UPDATE new_dept_pnu
        SET total_dept_sal = total_dept_sal  - :OLD.salary
        WHERE department_id = :OLD.department_id;
        UPDATE new_dept_pnu
        SET total_dept_sal = total_dept_sal + :NEW.salary
        WHERE department_id = :NEW.department_id;
    END IF;
    END;
    /
```

10. Să se implementeze cu ajutorul unui declanşator restricția că într-un departament pot lucra maximum 50 de angajați.

```
CREATE OR REPLACE TRIGGER TrLimitaDep_pnu
  BEFORE INSERT ON emp_pnu
  FOR EACH ROW
```

```
DECLARE
  v_Max_emp CONSTANT NUMBER := 50;
  v_emp_curent      NUMBER;
BEGIN
  SELECT  COUNT(*) INTO v_emp_curent
  FROM    emp_pnu
  WHERE   department_id = :NEW.department_id;
  IF v_emp_curent + 1 > v_Max_emp THEN
    RAISE_APPLICATION_ERROR(-20000,'Prea multi angajati in departamentul
                              avand codul ' || :NEW.department_id);
  END IF;
END TrLimitaDep_pnu;
/
```

Testați trigger-ul.

```
INSERT INTO emp_pnu (employee_id, …, department_id)
VALUES (emp_seq_pnu.nextval, …, 30);  --se va executa de mai multe ori astfel încât să existe mai mult de 50 de angajați în departamentul 30.
```

Obs: Declanşatorul *TrLimitaDep_pnu* consultă chiar tabelul (*emp_pnu*) la care este asociat declanşatorul (*mutating*).

Tabelul *emp_pnu* este *mutating* doar pentru un declanşator la nivel de linie.

O solutie pentru acestă problemă este crearea a doi declanşatori, unul la nivel de linie şi altul la nivel de instrucțiune :

- în declanşatorul la nivel de linie se înregistrează valoarea lui *:NEW.department_id*, dar nu va fi interogat tabelul *emp_pnu*.
- interogarea va fi făcută în declanşatorul la nivel de instrucțiune şi va folosi valoarea înregistrată în declanşatorul la nivel de linie.

O modalitate pentru a înregistra valoarea lui :*NEW.department_id* este ***utilizarea unui tablou indexat în interiorul unui pachet***.

```
CREATE OR REPLACE PACKAGE PdepDate_pnu AS
  TYPE t_cod IS TABLE OF dept_pnu.department_id%TYPE
             INDEX BY BINARY_INTEGER;
  v_cod_dep   t_cod;
  v_NrIntrari  BINARY_INTEGER := 0;
END PdepDate_pnu;
/
CREATE OR REPLACE TRIGGER TrLLimitaDep_pnu
  BEFORE INSERT ON dept_pnu
  FOR EACH ROW
BEGIN
  PDepDate_pnu.v_NrIntrari := PDepDate_pnu.v_NrIntrari + 1;
  PdepDate_pnu.v_cod_dep (PdepDate_pnu.v_NrIntrari) :=
                              :NEW.department_id;
END TrLLimitaDep_pnu;
/
CREATE OR REPLACE TRIGGER TrILimitaDep_pnu
  BEFORE INSERT ON emp_pnu
DECLARE
  v_Max_emp      CONSTANT NUMBER := 50;
  v_emp_curent  NUMBER;
  v_cod_dep       dep_pnu.department_id%TYPE;
```

```
BEGIN
  /* Parcurge fiecare departament inserat sau actualizat si
     verifica daca se incadreaza in limita stabilita */
  FOR v_LoopIndex IN 1..PdepDate_pnu.v_NrIntrari LOOP
    v_cod_dep := PdepDate_pnu.v_cod_dep(v_LoopIndex);
    SELECT  COUNT(*)
    INTO    v_emp_curent
    FROM    emp_pnu
    WHERE   department_id = v_cod_dep;
    IF v_emp_curent > v_Max_emp THEN
      RAISE_APPLICATION_ERROR(-20000, 'Prea multi angajati
              in departamentul avand codul: ' || v_cod_sala);
    END IF;
  END LOOP;
  /* Reseteaza contorul deoarece urmatoarea executie
     va folosi date noi */
  PdepDate_pnu.v_NrIntrari := 0;
END TrILimitaopere;
/
```

Obs: Această soluție funcționează pentru departamentele nou introduse.
Pentru testare:
- introduceți un nou departament
-introduceti mai mult de 50 de angajați în departamentul inserat anterior (eventual, cu o comandă de tip INSERT INTO ... (SELECT ...)).

11. Să se creeze un declanşator care:

a) dacă este eliminat un departament, va şterge toți angajații care lucrează în departamentul respectiv;

b) dacă se schimbă codul unui departament, va modifica această valoare pentru fiecare angajat care lucrează în departamentul respectiv.

```
CREATE OR REPLACE TRIGGER dep_cascada_pnu
  BEFORE DELETE OR UPDATE OF department_id ON dept_pnu
  FOR EACH ROW
BEGIN
  IF DELETING THEN
    DELETE FROM emp_pnu
    WHERE  department_id = :OLD.department_id;
  END IF;
  IF UPDATING AND :OLD. department_id != :NEW. department_id THEN
    UPDATE emp_pnu
    SET    department_id = :NEW. department_id
    WHERE  department_id = :OLD. department_id;
  END IF;
END dep_cascada_pnu;
```

Obs: Declanşatorul anterior realizează constrângerea de integritate *UPDATE* sau *ON DELETE CASCADE*, adică ştergerea sau modificarea cheii primare a unui tabel „părinte“ se va reflecta şi asupra înregistrărilor corespunzătoare din tabelul „copil“.
Testați trigger-ul:

```
DELETE FROM  dept_pnu
WHERE  department_id = 10;
```

```
UPDATE  dept_pnu
SET     department_id = 12
WHERE   department_id = 10;
```
Obs : Se presupune că asupra tabelului *emp_pnu* există o constrângere de integritate:
FOREIGN KEY (department_id) REFERENCES dept_pnu(department_id)

În acest caz sistemul *Oracle* va afişa un mesaj de eroare prin care se precizează că tabelul *dept_pnu* este *mutating*, iar constrângerea definită mai sus nu poate fi verificată.
ORA-04091: *table MASTER.DEPT_PNU is mutating*, *trigger/function may not see it*

12. Să se creeze un ***declanşator la nivelul bazei de date*** prin care să nu se permită ştergerea informațiilor din tabelul emp_pnu de către utilizatorul curent. Dezactivați, iar apoi activați trigger-ul creat. Testați, iar apoi suprimați acest trigger.

```
CREATE OR REPLACE TRIGGER p13l6_pnu
  BEFORE DELETE ON emp_pnu
 BEGIN
  IF USER= UPPER('g231') THEN
    RAISE_APPLICATION_ERROR(-20900,'Nu este permisa stergerea de catre ' ||USER);
  END IF;
 END;
/

ALTER TRIGGER p13l6_pnu DISABLE;

DELETE FROM emp_pnu WHERE employee_id = 100;

ALTER TRIGGER p13l6_pnu DISABLE;

DELETE FROM emp_pnu WHERE employee_id = 100;

DROP TRIGGER p13l6_pnu;
```

13. Să se creeze un tabel care conține următoarele câmpuri: user_id, nume_bd, eveniment_sis, nume_obj, data. Să se creeze un **trigger sistem** (la nivel de schemă) care să introducă date în acest tabel după ce utilizatorul a folosit o comandă LDD. Testați, iar apoi suprimați trigger-ul.

```
CREATE TABLE log_pnu
        (user_id        VARCHAR2(30),
          nume_bd     VARCHAR2(50),
          eveniment_sis  VARCHAR2(20),
          nume_obj          VARCHAR2(30),
          data              DATE);
CREATE OR REPLACE TRIGGER p14l6_pnu
  AFTER CREATE OR DROP OR ALTER ON SCHEMA
BEGIN
  INSERT INTO info_pnu
  VALUES (SYS.LOGIN_USER, SYS.DATABASE_NAME, SYS.SYSEVENT,
                SYS.DICTIONARY_OBJ_NAME, SYSDATE);
END;
/
CREATE VIEW v_test_pnu AS SELECT *  FROM jobs;

DROP      VIEW v_test_pnu;

SELECT * FROM log_pnu;

DROP TRIGGER p14l6_pnu;
```

Baze de date-Anul 3 (semestrul 1)
Laborator 6 PL/SQL

Pachete

I. Definirea pachetelor

Ø Pachetul (*package*) permite încapsularea într-o unitate logică în baza de date a procedurilor, funcțiilor, cursoarelor, tipurilor, constantelor, variabilelor şi excepțiilor.

Ø Spre deosebire de subprograme, pachetele nu pot:

- fi apelate,
- transmite parametri,
- fi încuibărite.

Ø Un pachet are două parți, fiecare fiind stocată separat în dicționarul datelor.

- Specificarea pachetului (*package specification*) – partea „vizibilă”, adică interfața cu aplicații sau cu alte unități program. Se declară tipuri, constante, variabile, excepții, cursoare şi subprograme folositoare utilizatorului.
- Corpul pachetului (*package body*) – partea „acunsă”, mascată de restul aplicației, adică realizarea specificației. Corpul defineşte cursoare şi subprograme, implementând specificația. Obiectele conținute în corpul pachetului sunt fie private, fie publice.

Ø Un pachet are următoarea formă generală:

```
CREATE PACKAGE nume_pachet {IS | AS} -- specificația
        /* interfața utilizator, care conține: declarații de tipuri şi obiecte
           publice, specificații de subprograme */
END [nume_pachet];
/
CREATE PACKAGE BODY nume_pachet {IS | AS} -- corpul
        /* implementarea, care conține: declarații de obiecte şi tipuri private,
           corpuri de subprograme specificate în partea de interfață */
[BEGIN]
        /* instrucțiuni de inițializare, executate o singură dată când
           pachetul este invocat prima oară de către sesiunea utilizatorului */
END [nume_pachet];
/
```

II. Pachete predefinite

Ø *DBMS_OUTPUT* permite afişarea de informații. *DBMS_OUTPUT* lucrează cu un *buffer* (conținut în *SGA*) în care poate fi scrisă sau regăsită informație. Procedurile pachetului sunt:

PUT – depune (scrie) în buffer informație
PUT_LINE – depune în buffer informația, împreună cu un marcaj - sfârşit de linie
NEW_LINE – depune în buffer un marcaj - sfârşit de linie
GET_LINE – regăseşte o singură linie de informație;
GET_LINES – regăseşte mai multe linii de informație;
ENABLE/DISABLE – activează/dezactivează procedurile pachetului.

Ø *DBMS_SQL* permite folosirea dinamică a comenzilor *SQL* în proceduri stocate sau în blocuri anonime şi analiza gramaticală a comenzilor *LDD*.

- *OPEN_CURSOR* (deschide un nou cursor, adică se stabileşte o zonă de memorie în care este procesată comanda *SQL*);
- *PARSE* (stabileşte validitatea comenzii *SQL*, adică se verifică sintaxa instrucțiunii şi se asociază cursorului deschis);
- *BIND VARIABLE* (leaga valoarea data de variabila corespunzatoare din comanda *SQL* analizata)
- *EXECUTE* (execută comanda *SQL* şi returnează numărul de linii procesate);
- *FETCH_ROWS* (regăseşte o linie pentru un cursor specificat, iar pentru mai multe linii foloseşte un *LOOP*);
- *CLOSE_CURSOR* (închide cursorul specificat).

Ø *DBMS_JOB* este utilizat pentru planificarea execuției programelor PL/SQL. Dintre subprogramele acestui pachet menționăm:
- *SUBMIT* – adaugă un nou *job* în coada de aşteptare a *job*-urilor;
- *REMOVE* – şterge un *job* specificat din coada de aşteptare a *job*-urilor;
- *RUN* – execută imediat un *job* specificat;
- *NEXT_DATE* – modifică momentul următoarei execuții a unui *job*;
- *INTERVAL* – modifică intervalul între diferite execuții ale unui *job*.

Ø *UTL_FILE* permite programului *PL/SQL* citirea din fişierele sistemului de operare, respectiv scrierea în aceste fişiere. El este utilizat pentru exploatarea fişierelor text. Scrierea şi regăsirea informațiilor se face cu ajutorul unor proceduri asemănătoare celor din pachetul DBMS_OUTPUT.

Procedura FCLOSE permite închiderea unui fişier.

Exerciții
I. [Pachete definite de utilizator]

1. a) Creați specificația şi corpul unui pachet numit DEPT_PKG_PNU care conține:
- procedurile ADD_DEPT, UPD_DEPT şi DEL_DEPT, corespunzătoare operațiilor de adăugare, actualizare (a numelui) şi ştergere a unui departament din tabelul DEPT_PNU;
- funcția GET_DEPT, care determină denumirea unui departament, pe baza codului acestuia.

Obs: Salvați specificația şi corpul pachetului în fişiere separate (p1l5_s.sql şi p1l5_b.sql pentru specificație, respectiv corp). Includeți câte o instrucțiune SHOW ERRORS la sfârşitul fiecărui fişier.
b) Invocați procedurile şi funcția din cadrul pachetului atât prin blocuri PL/SQL cât şi prin comenzi SQL.

Soluție:

Specificația pachetului:

```
CREATE OR REPLACE PACKAGE dept_pkg_pnu IS
        PROCEDURE add_dept (p_deptid departments.department_id%TYPE,
                            p_deptname departments.department_name%TYPE);
        PROCEDURE del_dept (p_deptid departments.department_id%TYPE);
        FUNCTION get_dept (p_deptid departments.department_id%TYPE)
                        RETURN departments.Department_name%TYPE;
        PROCEDURE upd_dept (p_deptid departments.department_id%TYPE,
                            p_deptname departments.department_name%TYPE);
END dept_pkg_pnu;
/
SHOW ERRORS
```

Corpul pachetului:

```
CREATE OR REPLACE PACKAGE BODY dept_pkg_pnu IS
        PROCEDURE add_dept (p_deptid departments.department_id%TYPE,
                                p_deptname departments.department_name%TYPE)  IS
        BEGIN
                INSERT INTO dept_pnu(department_id, department_name)
                VALUES (p_deptid, p_deptname);
                COMMIT;
        END add_dept;

        PROCEDURE del_dept (p_deptid departments.department_id%TYPE) IS
        BEGIN
                DELETE FROM dept_pnu
                WHERE department_id = p_deptid;
                IF SQL%NOTFOUND THEN
                        RAISE_APPLICATION_ERROR(-20203, 'Nici un departament sters');
                END IF;
        END del_dept;

        FUNCTION get_dept (p_deptid departments.department_id%TYPE) RETURN
departments.Department_name%TYPE IS
                v_nume departments.department_name%TYPE;
        BEGIN
                SELECT department_name
                INTO v_nume
                FROM dept_pnu
                WHERE department_id = p_deptid;
                RETURN v_nume;
        END get_dept;

        PROCEDURE upd_dept (p_deptid departments.department_id%TYPE,
                                p_deptname departments.department_name%TYPE)
                UPDATE dept_pnu
                SET department_name = p_deptname
                WHERE department_id = p_deptid;
                IF SQL%NOTFOUND THEN
                        RAISE_APPLICATION_ERROR(-20204, 'Nici un departament actualizat');
                END IF;
        END upd_dept;
END dept_pkg_pnu;
/
```

Obs: Pentru invocarea procedurii:

```
EXECUTE dept_pkg_pnu.add_dept(12, 'IT');
EXECUTE dept_pkg_pnu.upd_dept(12, 'Information technology');
```

sau

```
BEGIN
        dept_pkg_pnu.add_dept(12, 'IT');
        upd_dept(12, 'Information technology');
END;
/
```

Pentru invocarea funcției:

```
SELECT dept_pkg_pnu.get_dept(20)
```

```
FROM dual;
```

sau

```
EXECUTE DBMS_OUTPUT.PUT_LINE('Departamentul cautat este: '||dept_pkg_pnu.get_dept(20));
```

sau

```
BEGIN
        DBMS_OUTPUT.PUT_LINE('Departamentul cautat este: '||dept_pkg_pnu.get_dept(20));
END;
```

2. Creați specificația şi corpul unui pachet numit EMP_PKG_PNU care conține:
- procedura publică ADD_EMP - adaugă o înregistrare în tabelul EMP_PNU; utilizează o secvență pentru generarea cheilor primare; vor fi prevăzute valori implicite pentru parametrii nespecificați;
- procedura publică GET_EMP - pe baza unui cod de angajat transmis ca parametru, întoarce în doi parametri de ieşire salariul şi job-ul corespunzător;
- funcția privată VALID_JOB_ID - rezultatul acestei funcții indică dacă job-ul unui angajat corespunde unei valori existente în tabelul JOBS. Funcția va fi utilizată în cadrul procedurii ADD_EMP, făcând posibilă doar introducerea de înregistrări având coduri de job valide.
Tratați eventualele excepții.

Soluție:

```
CREATE OR REPLACE PACKAGE emp_pkg_pnu IS
        Procedure add_emp (
                p_first_name employees.first_name%TYPE,
                p_last_name employees.last_name%TYPE,
                p_email employees.email%TYPE,
                p_mgr employees.manager_id%TYPE DEFAULT 145,
                p_sal employees.salary%TYPE DEFAULT 1000,
                p_job employees.job_id%TYPE DEFAULT 'SA_REP',
                p_comm employees.commission_pct%TYPE DEFAULT 0,
                p_deptid employees.department_id%TYPE DEFAULT 30,
                p_hire_date employees.hire_date%TYPE DEFAULT SYSDATE);
        procedure get_emp (p_empid IN employees;employee_id%TYPE,
                                p_sal OUT employees.salary%TYPE,
                                p_job OUT employees.job_id%TYPE);
END emp_pkg_pnu;
/
CREATE OR REPLACE PACKAGE BODY emp_pkg_pnu IS
        FUNCTION valid_job_id (p_job IN jobs.job_id%TYPE) RETURN BOOLEAN IS
                x PLS_INTEGER;
        BEGIN
                SELECT 1
                INTO x
                FROM jobs_id
                WHERE LOWER(job_id) = LOWER(p_job);
                RETURN TRUE;
        EXCEPTION
                WHEN NO_DATA_FOUND THEN
                RETURN FALSE;
        END valid_job_id;

        PROCEDURE add_emp (
                p_first_name employees.first_name%TYPE,  --implicit de tip IN
                p_last_name employees.last_name%TYPE,
```

```
            p_email employees.email%TYPE,
            p_mgr employees.manager_id%TYPE DEFAULT 145,
            p_sal employees.salary%TYPE DEFAULT 1000,
            p_job employees.job_id%TYPE DEFAULT 'SA_REP',
            p_comm employees.commission_pct%TYPE DEFAULT 0,
            p_deptid employees.department_id%TYPE DEFAULT 30) IS
        BEGIN
            INSERT INTO emp_pnu(employee_id, first_name, last_name, email, manager_id,
                    salary, job_id, commission_pct, department_id, hire_date)
            VALUES(seq_emp_pnu.nextval, p_first_name, p_last_name, p_email,
                        p_mgr, p_sal, p_ job, p_comm, p_deptid, p_hire_date);
        END add_emp;

        PROCEDURE get_emp (p_empid IN employees;employee_id%TYPE,
                           p_sal OUT employees.salary%TYPE,
                           p_job OUT employees.job_id%TYPE) IS
        BEGIN
            SELECT salary, job_id
            INTO p_sal, p_job
            FROM emp_pnu
            WHERE employee_id = p_empid;
        END get_emp;

END emp_pkg_pnu;
/
```

Exemplu de invocare:

```
EXECUTE emp_pkg_pnu.add_emp('Jane', 'Harris', 'jharris', p_job => 'SA_REP');
EXECUTE emp_pkg_pnu.add_emp('David', 'Smith', 'dsmith', p_job => 'SA_MAN');
```

3. Modificați pachetul EMP_PKG_PNU anterior supraîncărcând procedura ADD_EMP. Noua procedură va avea 3 parametri, corespunzători numelui, prenumelui şi codului job-ului. Procedura va formata câmpul email astfel încât acesta să fie scris cu majuscule, prin concatenarea primei litere a prenumelui şi a primelor 7 litere ale numelui. Va fi apelată vechea procedură ADD_EMP pentru inserarea efectivă a unei înregistrări.

Soluție :

```
PROCEDURE add_emp(p_first_name employees.first_name%TYPE,
                  p_last_name employees.last_name%TYPE,
                  p_job employees.job_id%TYPE DEFAULT 'SA_REP') IS
    v_email employees.email%TYPE;
BEGIN
    v_email := UPPER(SUBSTR(p_first_name, 1, 1)||SUBSTR(p_last_name, 1, 7));
    add_emp(p_first_name, p_last_name, v_email, p_job=>p_job);
END add_emp;
```

Exemplu de invocare: `EXECUTE emp_pkg_pnu.add_emp('Sam', 'Joplin', 'sa_man');`

4. a) Creați două funcții supraîncărcate GET_EMP în pachetul EMP_PKG_PNU:

- o funcție GET_EMP va avea un parametru p_emp_id de tipul employees.employee_id%TYPE şi va regăsi linia corespunzătoare codului respectiv;
- cealaltă funcție GET_EMP va avea un parametru p_nume_familie de tipul employees.last_name%TYPE şi va regăsi linia corespunzătoare numelui respectiv;
- ambele funcții vor returna o valoare de tipul employees%ROWTYPE.

b) In pachet se va mai adăuga procedura PRINT_EMPLOYEE având un parametru de tipul EMPLOYEES%ROWTYPE, care afişează codul departamentului, codul angajatului, prenumele, numele, codul job-ului şi salariul, utilizând DBMS_OUTPUT.

c) Utilizați un bloc anonim pentru apelarea funcțiilor şi a procedurii anterioare.

Soluție:

```
FUNCTION get_emp(p_emp_id employees.employee_id%TYPE)
    RETURN employees%ROWTYPE IS
        v_emp employees%rowtype;
BEGIN
        SELECT * INTO v_emp
        FROM emp_pnu
        WHERE employee_id = p_emp_id;
        RETURN v_emp;
END;
--analog cea de-a doua funcție get_emp
--creați şi procedura PRINT_EMPLOYEE
```

5. Introduceți în pachet funcția valid_deptid din laboratorul precedent. Modificați prima procedură add_emp astfel încât introducerea unui angajat nou să fie posibilă doar dacă departamentul este valid.

Presupunând că firma nu actualizează frecvent datele despre departamente, pachetul EMP_PKG_PNU poate fi îmbunătățit prin adăugarea procedurii publice INIT_DEPT care populează un tablou privat PL/SQL de coduri de departament valide. Creați această procedură.

Soluție:

In specificația pachetului, vom avea:

```
PROCEDURE init_dept;
```

In corpul pachetului, se adaugă înaintea specificării subprogramelor:

```
TYPE boolean_tabtype IS TABLE OF BOOLEAN
        INDEX BY binary_integer;
valid_dept boolean_tabtype;
```

La sfârşitul corpului pachetului, se declară procedura :

```
PROCEDURE init_dept IS
BEGIN
        FOR rec IN (SELECT distinct department_id FROM dept_pnu)
        LOOP
                Valid_dep(rec.department_id) := TRUE;
        END LOOP;
END;
```

La sfârşitul corpului pachetului, se crează un bloc de inițializare care apelează procedura INIT_DEPT :

```
BEGIN
        Init_dept ;
End ;
```

6. a) Modificați funcția VALID_DEPTID pentru a utiliza acest tablou PL/SQL.

```
FUNCTION valid_deptid (p_deptid …)
    RETURN ….
BEGIN
        RETURN valid_dept.exists(p_deptid);
EXCEPTION
        WHEN no_data_found THEN
        RETURN false;
END valid_deptid;
```

b) Testați procedura ADD_EMP adăugâng un salariat în departamentul 15. Ce se întâmplă ?

```
EXECUTE emp_pkg_pnu.add_emp('James', 'Bond', p_deptid => 15)
```

Inserați un departament nou, având codul 15.

```
INSERT INTO dept_pnu (department_id, department_name)
VALUES (15, 'Security');
COMMIT;
```

Incercați din nou adăugarea unui angajat în departamentul având codul 15. Comentați.
Actualizați tabloul PL/SQL intern cu noile date ale departamentelor.

```
EXECUTE emp_pkg_pnu.init_dept
```

Inserați înregistrarea corespunzătoare angajatului din departamentul 15.

7. Să se creeze un pachet cu ajutorul căruia, utilizând un cursor şi un subprogram funcție, să se obțină salariul maxim înregistrat pentru salariații care lucrează într-un anumit oraş şi lista salariaților care au salariul mai mare sau egal decât maximul salariilor din oraşul respectiv.

```
CREATE  OR REPLACE PACKAGE p7l5_pnu AS
  CURSOR c_emp(nr NUMBER) RETURN employees%ROWTYPE;
  FUNCTION  f_max  (p_oras  locations.city%TYPE) RETURN NUMBER;
END p7l5_pnu;
/
CREATE OR REPLACE PACKAGE BODY p7l5_pnu AS
  CURSOR c_emp(nr NUMBER) RETURN employees%ROWTYPE  IS
    SELECT * FROM employees WHERE salary >= nr;
  FUNCTION  sal_max (p_oras  locations.city%TYPE) RETURN NUMBER  IS
    maxim  NUMBER;
   BEGIN
       SELECT  MAX(salary)
       INTO    maxim
       FROM    employees e, departments d,locations l
       WHERE   e.department_id=d.department_id AND d.location_id=l.location_id
             AND UPPER(city)=UPPER(p_oras);
       RETURN  maxim;
   END sal_max;
END p7l5_pnu;
/

SET SERVEROUTPUT ON
DECLARE
 v_oras locations.city%TYPE:= 'Oxford';
 v_max NUMBER;
 v_emp employees%ROWTYPE;
BEGIN
  v_max:=  p7l5_pnu.sal_max(v_oras);
  OPEN p7l5_pnu.c_emp(v_max);
  LOOP
    FETCH p7l5_pnu.c_emp INTO v_emp;
    EXIT WHEN p7l5_pnu.c_emp%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_emp.last_name||' '||v_emp.salary);
  END LOOP;
  CLOSE p7l5_pnu.c_emp;
END;
/
SET SERVEROUTPUT OFF
```

8. Să se creeze un pachet *verif_pkg_pnu* ce include o procedură prin care se verifică dacă o combinație specificată de valori ale atributelor *job_id* şi *department_id* este o combinație care există în tabelul *EMPLOYEES*.

```
CREATE PACKAGE verif_pkg_pnu IS
  PROCEDURE verifica
     (p_jobid   IN employees.job_id%TYPE,
      p_deptid IN employees.department_id%TYPE);
END verif_pkg_pnu;
/
CREATE OR REPLACE PACKAGE BODY verif_pkg_pnu IS
        i NUMBER := 0;
        CURSOR emp_crs IS
                SELECT distinct job_id, department_id
                FROM     employees;
        TYPE emp_table_tip IS TABLE OF emp_crs%ROWTYPE
                INDEX BY BINARY INTEGER;
        job_dep emp_table_tip;

  PROCEDURE verifica
     (p_jobid   IN employees.job_id%TYPE,
      p_deptid IN employees.department_id%TYPE)  IS
  BEGIN
        FOR k IN job_dep.FIRST..job_dep.LAST LOOP
            IF p_jobid = job_dep(k).job_id
               AND p_deptid = job_dep(k).department_id THEN
                RETURN;
            END IF;
        END LOOP;
        RAISE_APPLICATION_ERROR (-20777,'nu este o combinatie valida
                                                de job si departament');
  END verifica;

BEGIN
        FOR v_emp IN emp_crs LOOP
                job_dep(i) := v_emp;
                i := i+1;
        END LOOP;
END verif_pkg_pnu;
/

EXECUTE verif_pkg_pnu.verifica ('SA_REP', 10);
```

II. [Pachete standard]

[DBMS_OUTPUT]

9. Să se scrie un bloc anonim care reține în 3 variabile PL/SQL numele, salariul şi departamentul angajatului având codul 145. Să se afişeze aceste informații (implicit, se va introduce o linie în buffer-ul specific DBMS_OUTPUT). Să se regăsească această linie şi starea corespunzătoare (0, dacă există linii în buffer şi 1, altfel). Să se afişeze linia şi starea.

```
SET SERVEROUTPUT ON
DECLARE
        linie varchar2(255);
        stare number;
        v_nume  employees.last_name%TYPE;
```

```
        v_sal employees.salary%TYPE;
        v_dept employees.department_id%TYPE;

BEGIN
        SELECT …
        INTO v_nume,v_sal,v_dept
        FROM employees
        WHERE employee_id=145;
        DBMS_OUTPUT.PUT_LINE(v_nume||' '||v_sal||' '||v_dept);
        DBMS_OUTPUT.GET_LINE(linie,stare);
        DBMS_OUTPUT.NEW_LINE;
        DBMS_OUTPUT.PUT_LINE(linie||'    '||stare);
END;
/
SET SERVEROUTPUT OFF
```

[DBMS_JOB]

10. a) Să se utilizeze pachetul *DBMS_JOB* pentru a plasa pentru execuție în coada de aşteptare a *job*-urilor, procedura *verifica* din pachetul *verif_pkg_pnu.* Prima execuție va avea loc peste 5 minute.

```
VARIABLE num_job NUMBER
BEGIN
  DBMS_JOB.SUBMIT(
    job => :num_job, ---- returnează numărul jobului, printr-o variabilă de legătură
    what => 'verif_pkg_pnu.verifica('SA_MAN', 20);' –codul care va fi executat ca job
    next_date => SYSDATE+1/288, -- data primei execuții
    interval => 'TRUNC(SYSDATE+1)'); -- intervalul dintre execuțiile job-ului
  COMMIT;
END;
/
PRINT num_job
```

b) Aflați informații despre job-urile curente în vizualizarea USER_JOBS.

```
SELECT job, next_date,what
FROM user_jobs;
```

c) Identificați în coada de aşteptare job-ul pe care l-ați lansat şi executați-l.

```
BEGIN
        DBMS_JOB.RUN(job => x); --x este numărul identificat
                                --pentru job-ul care vă aparține
END;
/
```

d) Stergeți job-ul din coada de aşteptare.

```
EXECUTE DBMS_JOB.REMOVE(job=>x);
SELECT job, next_date,what
FROM user_jobs;
```

[UTL_FILE]

11. Creați o procedură numită EMP_REPORT_PNU care generează un raport într-un fişier al sistemului de operare, utilizând pachetul UTL_FILE. Raportul va conține lista angajaților care au depăşit media salariilor din departamentul lor. Procedura va avea doi parametri: directorul de ieşire şi numele fişierului text în care va fi scris raportul. Tratați excepțiile care pot apărea la utilizarea pachetului UTL_FILE.

```
CREATE OR REPLACE PROCEDURE emp_report_pnu (
```

```
        p_dir IN VARCHAR2, p_filename IN VARCHAR2) IS
        v_file UTL_FILE.FILE_TYPE;
        CURSOR avg_csr IS
                SELECT last_name, department_id, salary
                FROM employees e
                WHERE salary > (SELECT AVG(salary)
                                FROM employees
                                GROUP BY e.department_id)
                ORDER BY department_id;
BEGIN
        v_file := UTL_FILE(p_dir, p_filename, 'w');
        UTL_FILE.PUT_LINE(v_file, 'Angajati care castiga mai mult decat salariul mediu:');
        UTL_FILE.PUT_LINE(v_file,'Raport generat la date de '|| SYSDATE);
        UTL_FILE.NEW_LINE(v_file);
        FOR emp IN avg_csr
        LOOP
                UTL_FILE.PUT_LINE(v_file,
                        RPAD(emp.last_name, 30) || ' ' ||
                        LPAD(NVL(TO_CHAR(emp.department_id, '9999'), '-'), 5)|| ' ' ||
                        LPAD(TO_CHAR(emp.salary, '$99,999.00'), 12));
        END LOOP;
        UTL_FILE.NEW_LINE(v_file);
        UTL_FILE.PUT_LINE(v_file, '***Sfârsitul raportului ***');
        UTL_FILE.FCLOSE(v_file);
END emp_report_pnu;
/
```

[SQL dinamic, DBMS_SQL]

12. Să se construiască o procedură care foloseşte *SQL* dinamic pentru a şterge liniile unui tabel specificat ca parametru. Subprogramul furnizează ca rezultat numărul liniilor şterse (*nr_lin*).

```
CREATE OR REPLACE PROCEDURE sterge_linii
  (num_tab IN VARCHAR2, nr_lin OUT NUMBER)
AS
  nume_cursor INTEGER;
BEGIN
  nume_cursor := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE (nume_cursor, 'DELETE FROM ' ||
              num_tab, DBMS_SQL.V7);
  nr_lin := DBMS_SQL.EXECUTE (nume_cursor);
  DBMS_SQL.CLOSE_CURSOR (nume_cursor);
END;

VARIABLE linii_sterse NUMBER
        EXECUTE sterge_linii ('opera', :linii_sterse)
PRINT linii_sterse
```

Obs: Pentru a executa o instrucțiune *SQL* dinamic poate fi utilizată şi comanda *EXECUTE IMMEDIATE*.

```
CREATE OR REPLACE PROCEDURE sterge_linii
  (num_tab IN VARCHAR2, nr_lin OUT NUMBER)
IS
BEGIN
  EXECUTE IMMEDIATE 'DELETE FROM ' || num_tab;
  nr_lin := SQL%ROWCOUNT;
```

END;

13. a) Creați un pachet numit TABLE_PKG_PNU care utilizează SQL nativ pentru crearea sau ştergerea unui tabel şi pentru adăugarea, modificarea sau ştergerea de linii din tabel.
Specificația pachetului va conține procedurile următoare:
- PROCEDURE make (table_name VARCHAR2, col_specs VARCHAR2)
- PROCEDURE add_row (table_name VARCHAR2, values VARCHAR2, cols VARCHAR2 := NULL)
- PROCEDURE upd_row(table_name VARCHAR2, set_values VARCHAR2, conditions VARCHAR2 := NULL)
- PROCEDURE upd_row(table_name VARCHAR2, conditions VARCHAR2 := NULL)
- PROCEDURE remove(table_name VARCHAR2)

b) Executați procedura MAKE pentru a crea un tabel, astfel:
make('contacte_pnu', 'cod NUMBER(4), nume VARCHAR2(35)');

c) Listați structura tabelului contacte_pnu.

d) Adăugați înregistrări prin intermediul procedurii ADD_ROW.
Exemplu: add_row('contacte_pnu', '1, ''Geoff Gallus''', 'cod, nume');

e) Afişați conținutul tabelului contacte_pnu.

f) Executați procedura DEL_ROW pentru ştergerea contactului având codul 1.

g) Executați procedura UPD_ROW.
Exemplu: upd_row('contacte_pnu', 'nume = ''Nancy Greenberg''', 'id=2');

h) Afişați conținutul tabelului, apoi ştergeți tabelul prin intermediul procedurii remove.
Corpul pachetului va fi:

```
Create or replace package body table_pkg_pnu IS
        Procedure execute (stmt VARCHAR2) IS
        BEGIN
                Dbms_output.put_line(stmt);
                Execute immediate stmt;
        END;

        PROCEDURE make(table_name VARCHAR2, col_specs VARCHAR2) IS
                Stmt VARCHAR2(200) := 'CREATE TABLE '||table_name||' ('||col_specs ||');
        BEGIN
                EXECUTE(stmt);
        END;

        PROCEDURE add_row(table_name VARCHAR2, col_values VARCHAR2,
                                        cols VARCHAR2 := null) IS
                stmt VARCHAR2(200) := 'INSERT INTO '||table_name;
        BEGIN
                IF cols IS NOT NULL THEN
                        Stmt := stmt ||' (' ||cols||')';
                END IF;
                stmt := stmt || ' VALUES(' || col_values || ')';
                execute(stmt);
        END;

        PROCEDURE upd_row(table_name VARCHAR2, set_values VARCHAR2,
                  conditions VARCHAR2 := NULL) IS
```

```
            stmt VARCHAR2(200) := 'UPDATE '||table_name || ' SET ' ||set_values;
        BEGIN
            IF conditions IS NOT NULL THEN
                stmt := stmt || ' WHERE ' || conditions ;
            END IF;
            execute(stmt);
        END;
        PROCEDURE del_row(table_name VARCHAR2, conditions VARCHAR2 := NULL) IS
            stmt VARCHAR2(200) := 'DELETE FROM '||table_name;
        BEGIN
            IF conditions IS NOT NULL THEN
                stmt := stmt || ' WHERE ' || conditions ;
            END IF;
            execute(stmt);
        END;

        PROCEDURE remove(table_name VARCHAR2) IS
            csr_id INTEGER;
            stmt VARCHAR2(100) := 'DROP TABLE '||table_name;
        BEGIN
            csr_id := DBMS_SQL.OPEN_CURSOR;
            DBMS_OUTPUT.PUT_LINE(stmt);
            DBMS_SQL.PARSE(csr_id, stmt, DBMS_SQL.NATIVE);
            DBMS_SQL.CLOSE_CURSOR(csr_id);
        END;
END table_pkg_pnu;
/
```

Baze de date-Anul 3 (semestrul 1)
Laborator 7 PL/SQL

Tratarea erorilor

Ø Tratarea erorilor se realizează în secțiunea EXCEPTION a blocului PL/SQL:

```
EXCEPTION
  WHEN nume_excepție1 [OR nume_excepție2 …] THEN
    secvența_de_instrucțiuni_1;
  [WHEN nume_excepție3 [OR nume_excepție4 …] THEN
    secvența_de_instrucțiuni_2;]
  …
  [WHEN OTHERS THEN
    secvența_de_instrucțiuni_n;]
END;
```

Ø Cu ajutorul funcțiilor *SQLCODE* şi *SQLERRM* se pot obține codul şi mesajul asociate excepției declanşate.

- Codul erorii este:
  - un număr negativ, în cazul unei erori sistem;
  - numărul +100, în cazul excepției *NO_DATA_FOUND*;
  - numărul 0, în cazul unei execuții normale (fără excepții);
  - numărul 1, în cazul unei excepții definite de utilizator.

Ø Excepțiile pot fi :

- Interne - se produc atunci când un bloc *PL/SQL* nu respectă o regulă *Oracle* sau depăşeşte o limită a sistemului de exploatare.
  - Predefinite - nu trebuie declarate în secțiunea declarativă şi sunt tratate implicit de către *server*-ul *Oracle*. Ele sunt referite prin nume (*CURSOR_ALREADY_OPEN*, *DUP_VAL_ON_INDEX*, *NO_DATA_FOUND, TOO_MANY_ROWS, ZERO_DIVIDE)*
  - Excepțiile interne nepredefinite sunt declarate în secțiunea declarativă şi sunt tratate implicit de către *server*-ul *Oracle*. Ele pot fi gestionate prin clauza *OTHERS*, în secțiunea *EXCEPTION* sau prin [vezi mai jos].
- Externe - definite în partea declarativă a blocului, deci posibilitatea de referire la ele este asigurată. În mod implicit, toate excepțiile externe au asociat acelaşi cod (+1) şi acelaşi mesaj (*USER DEFINED EXCEPTION*)

  Declararea şi prelucrarea excepțiilor externe respectă următoarea sintaxă:

```
DECLARE
      nume_excepție  EXCEPTION; -- declarare excepție
BEGIN
      …
      RAISE  nume_excepție;  --declanşare excepție
      -- codul care urmează nu mai este executat
      …
EXCEPTION
      WHEN nume_excepție THEN
      -- definire mod de tratare a erorii
      …
END;
```

Ø Altă metodă pentru tratarea unei *erori interne nepredefinite* (diferită de folosirea clauzei *OTHERS* drept detector universal de excepții) este utilizarea directivei de compilare (pseudo-instrucțiune)

PRAGMA EXCEPTION_INIT. Această directivă permite asocierea numelui unei excepții cu un cod de eroare intern.

În acest caz, tratarea erorii se face în următoarea manieră:

1) se declară numele excepției în partea declarativă sub forma:

nume_excepție ***EXCEPTION****;*

2) se asociază numele excepției cu un cod eroare standard *Oracle*, utilizând comanda:

PRAGMA EXCEPTION_INIT (*nume_excepție*, *cod_eroare*)*;*

3) se referă excepția în secțiunea de gestiune a erorilor (excepția este tratată automat, fără a fi necesară comanda *RAISE*).

Ø Activarea unei excepții externe poate fi făcută şi cu ajutorul procedurii *RAISE_APPLICATION_ERROR*, furnizată de pachetul *DBMS_STANDARD*.

RAISE_APPLICATION_ERROR poate fi folosită pentru a returna un mesaj de eroare unității care o apelează, mesaj mai descriptiv decât identificatorul erorii. Unitatea apelantă poate fi *SQL*Plus*, un subprogram *PL/SQL* sau o aplicație *client*.

Procedura are următorul antet:

RAISE_APPLICATION_ERROR (*numar_eroare* ***IN NUMBER***,
mesaj_eroare ***IN VARCHAR2***, [{***TRUE*** | ***FALSE***}]);

Atributul *numar_eroare* este un număr cuprins între –20000 şi –20999, specificat de utilizator pentru excepția respectivă, iar *mesaj_eroare* este un text asociat erorii, care poate avea maximum 2048 octeți.

Ø Informații despre erorile apărute la compilare se pot obține consultând vizualizarea USER_ERRORS.

SELECT LINE, POSITION, TEXT
FROM USER_ERRORS
WHERE NAME = UPPER('nume');

LINE specifică numărul liniei în care apare eroarea, dar acesta nu corespunde liniei efective din fişierul text (se referă la codul sursă depus în *USER_SOURCE*). Dacă nu sunt erori, apare mesajul *NO ROWS SELECTED*.

Exerciții:

1. Creați tabelul *erori_pnu* având două coloane: *cod_eroare* de tip NUMBER şi *mesaj_eroare* de tip VARCHAR2(100). Să se scrie un bloc *PL/SQL* care să determine şi să afişeze salariatul angajat cel mai recent într-un departament al cărui cod este introdus de către utilizator. Pentru orice eroare apărută, vor fi inserate codul şi mesajul erorii în tabelul erori_pnu.

Varianta 1 (captarea erorii interne a sistemului)

```
ACCEPT p_cod PROMPT 'Introduceti un cod de departament '
DECLARE
  eroare_cod    NUMBER;
  eroare_mesaj  VARCHAR2(100);
  v_dep departments.department_id%TYPE := &p_cod;
  v_emp emp_pnu%ROWTYPE;
BEGIN
  SELECT  * INTO v_emp
  FROM emp_pnu
  WHERE department_id = v_dep
  AND  hire_date = (SELECT MAX(hire_date ) FROM emp_pnu
                          WHERE department_id = v_dep);
  DBMS_OUTPUT.PUT_LINE(v_emp.last_name || ' '||v_emp.salary);
```

```
EXCEPTION
  WHEN OTHERS THEN
    eroare_cod := SQLCODE;
    eroare_mesaj := SUBSTR(SQLERRM,1,100);
    INSERT  INTO erori_pnu
    VALUES  (eroare_cod, eroare_mesaj);
END;
```

Varianta 2 (definirea unei excepții de către utilizator)

```
DECLARE
  eroare_cod       NUMBER;
  eroare_mesaj   VARCHAR2(100);
  v_dep departments.department_id%TYPE := &p_cod;
  v_emp emp_pnu%ROWTYPE;
  v_num NUMBER;
  exceptie   EXCEPTION;
BEGIN
  SELECT  COUNT(*) INTO v_num -- (1)
  FROM emp_pnu
  WHERE department_id = v_dep
  AND  hire_date = (SELECT MAX(hire_date ) FROM emp_pnu
                          WHERE department_id = v_dep);
   IF v_num != 1 THEN  --(2)
        RAISE exceptie;
    END IF;
--daca nu s-a declansat exceptia, sigur cererea de mai jos va returna o singura linie
  SELECT  * INTO v_emp (3)
  FROM emp_pnu
  WHERE department_id = v_dep
  AND  hire_date = (SELECT MAX(hire_date ) FROM emp_pnu
                          WHERE department_id = v_dep);

/***** Daca nu aveam instructiunile (1) si (2), s-ar fi ridicat exceptia cu acest IF? ****/
 /* IF SQL%ROWCOUNT >!=1 THEN  -- fara (1) si (2) nu se ajunge la aceasta instructiune dacă
                                        --(3) declanşează NO_DATA_FOUND sau TOO_MANY_ROWS
         RAISE exceptie;                -- ar merge utilizarea lui SQL%ROWCOUNT
                                        --pentru INSERT, UPDATE, DELETE
                                        -- (nu pentru SELECT .. INTO …)
  END IF;
*****************************************/

  DBMS_OUTPUT.PUT_LINE(v_emp.last_name || ' '||v_emp.salary);
 EXCEPTION
 WHEN exceptie THEN
             eroare_cod := -20200;
             eroare_mesaj := 'prea multe sau prea putine linii';
             INSERT  INTO erori_pnu
             VALUES  (eroare_cod, eroare_mesaj);
 END;
/
```

2. [Excepții interne predefinite]

Creați tabelul *mesaje_pnu* având o singură coloană, numită *rezultate*, de tip varchar2(50). Să se scrie un bloc *PL/SQL* prin care să se afişeze numele departamentelor dintr-o anumită locație care au

angajați.

a) Dacă rezultatul interogării returnează mai mult decât o linie, atunci să se trateze excepția şi să se insereze în tabelul *mesaje_pnu* textul „mai multe departamente“.
b) Dacă rezultatul interogării nu returnează nici o linie, atunci să se trateze excepția şi să se insereze în tabelul *mesaje_pnu* textul „nici un departament“.
c) Dacă rezultatul interogării este o singură linie, atunci să se insereze în tabelul *mesaje_pnu* numele departamentului şi managerul acestuia.
d) Să se trateze orice altă eroare, inserând în tabelul *mesaje_pnu* textul „alte erori au apărut“.

```
SET VERIFY OFF
ACCEPT p_locatie PROMPT 'Introduceti locatia:'
DECLARE
  v_nume_dep  dep_pnu.department_name%TYPE;
  v_manager     dep_pnu.manager_id%TYPE;
  v_locatie        dep_pnu.location_id%TYPE:='&p_locatie';
BEGIN
  SELECT   department_name, manager_id
  INTO        v_nume_dep, v_manager_id
  FROM       dep_pnu
  WHERE    location = v_locatie;
  INSERT   INTO mesaje_pnu (rezultate)
  VALUES   (v_nume_dep||'-'||v_manager);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
         INSERT   INTO mesaje (rezultate)
         VALUES   ('nici un departament');
  WHEN TOO_MANY_ROWS THEN
         INSERT   INTO mesaje (rezultate)
         VALUES   ('mai multe departamente');
  WHEN OTHERS THEN
         INSERT   INTO mesaje (rezultate)
         VALUES   ('alte erori au aparut');
END;
/
SET VERIFY ON
```

3. [Excepții interne nepredefinite]

Dacă există angajați ai unui anumit departament, să se tipărească un mesaj prin care utilizatorul este anunțat că departamentul respectiv nu poate fi şters din baza de date (încălcarea constrângerii de integritate având codul eroare *Oracle* -2292). Dacă nu există, implementați această constrângere înainte de execuția blocului.

```
ALTER TABLE dep_pnu
ADD CONSTRAINT pk_dep_pnu PRIMARY KEY(department_id);

ALTER TABLE emp_pnu
ADD CONSTRAINT fk_emp_dep_pnu FOREIGN KEY (department_id) REFERENCES dep_pnu;
```

Ce se întâmplă dacă se încearcă ştergerea unui departament în care lucrează angajați?

```
DELETE FROM dep_pnu
WHERE department_id=30;
--apare eroarea sistem ORA-02292


SET VERIFY OFF
DEFINE p_nume = Sales
```

```
DECLARE
  ang_exista  EXCEPTION;
  PRAGMA EXCEPTION_INIT(ang_exista,-2292);
BEGIN
  DELETE FROM dep_pnu WHERE department_name = '&p_nume';
  COMMIT;
EXCEPTION
  WHEN ang_exista THEN
    DBMS_OUTPUT.PUT_LINE ('nu puteti sterge departamentul cu
        numele ' || '&p_nume' || ' deoarece exista angajati care lucreaza in cadrul
        acestuia');
END;
/
SET VERIFY ON
```

4. [Excepții externe]

Să se scrie un bloc *PL/SQL* care afişează numărul departamentelor în care lucrează angajați al căror salariu este mai mare sau mai mic cu 1000 decât o valoare specificată. Să se tipărească un mesaj adecvat, dacă nu există nici un departament care îndeplineşte această condiție.

```
VARIABLE g_mesaj VARCHAR2(100)
SET VERIFY OFF
ACCEPT p_val PROMPT 'introduceti valoarea:'
DECLARE
  v_val       emp_pnu.salary%TYPE := &p_val;
  v_inf       emp_pnu.salary%TYPE := v_val - 1000;
  v_sup       emp_pnu.salary %TYPE := v_val + 1000;
  v_numar     NUMBER(7);
  e_nimeni    EXCEPTION;
  e_mai_mult  EXCEPTION;
BEGIN
  SELECT  COUNT(DISTINCT employee_id)
  INTO    v_numar
  FROM    emp_pnu
  WHERE   salary BETWEEN v_inf AND v_sup;
  IF v_numar = 0 THEN
     RAISE  e_nimeni;
  ELSIF v_numar > 0 THEN
     RAISE e_mai_mult;
  END IF;
EXCEPTION
  WHEN e_nimeni THEN
    :g_mesaj:='nu exista nici un departament in care sunt angajati
        cu salariul cuprins intre '||v_inf ||' si '||v_sup;
  WHEN e_mai_mult THEN
    :g_mesaj:='exista '||v_numar||' departamente cu angajati avand salariul cuprins intre
        '||v_inf||' si '||v_sup;
  WHEN OTHERS THEN
    :g_mesaj:='au aparut alte erori';
END;
/

SET VERIFY ON
PRINT g_mesaj
```

[Procedura RAISE_APPLICATION_ERROR]

5. a) Să se şteargă salariații asignați unui cod de departament inexistent în tabelul *departments*. Dacă nu există nici un angajat care a îndeplinit această condiție, să se lanseze o excepție cu mesajul „nici un angajat nu lucreaza in departament inexistent”.

b) Să se şteargă angajații al căror comision reprezintă mai mult decât jumătatea diferenței de salariu dintre şeful angajatului respectiv şi angajat. Dacă nu există nici un angajat care a îndeplinit această condiție, să se lanseze o excepție cu mesajul „nici un angajat cu comisionul specificat”.

Invocați procedura RAISE_APPLICATION_ERROR în secțiuni diferite ale blocului PL/SQL.

```
DECLARE
  e_comision  EXCEPTION;
  PRAGMA EXCEPTION_INIT (e_comision, -20777);
BEGIN
 DELETE FROM emp_pnu e          -- punctul b)
  WHERE salary * commission_pct > ((SELECT salary
                                      FROM     emp_pnu
                                    WHERE  employee_id = e.manager_id) – salary)/2;
  IF SQL%NOTFOUND THEN
         RAISE e_comision;
    END IF;

  DELETE FROM emp_pnu          -- punctul a)
  WHERE department_id NOT IN ( .... ); -- completati subcererea corespunzătoare
                                       --ce problema poate aparea la utilizarea lui
                                          -- NOT IN ?
  IF SQL%NOTFOUND THEN
    RAISE_APPLICATION_ERROR(-20777, 'nici o stergere'); --apel in sectiunea executabila
  END IF;

EXCEPTION
   WHEN e_comision THEN
      RAISE_APPLICATION_ERROR(-20777, ' nici un angajat cu codul specificat '); -- apel in
                                 --sectiunea EXCEPTION
END;
/
```

6. Să se implementeze un declanşator care nu permite introducerea de salariați în tabelul emp_pnu având salariul mai mic decât 1000.

Să se scrie un program care detectează şi tratează eroarea „ridicată” de trigger.

```
CREATE OR REPLACE TRIGGER sal_mini_pnu
BEFORE INSERT ON emp_pnu
FOR EACH ROW
BEGIN
  IF :NEW.salary < 1000 THEN
    RAISE_APPLICATION_ERROR
      (-20005,’angajatii trebuie sa aiba salariul mai mare de 1000’);
  END IF;
END;
```

Obs: *RAISE_APPLICATION_ERROR* facilitează comunicația dintre *client* şi *server*, transmițând aplicației *client* erori specifice aplicației de pe *server* (de obicei, un declanşator). Prin urmare, procedura este doar un mecanism folosit pentru comunicația *server* → *client* a unei erori definite de utilizator, care permite ca procesul *client* să trateze excepția.

Pe stația *client* poate fi scris un program care detectează şi tratează eroarea.

```
DECLARE
```

```
   --declarare excepție
     nu_accepta  EXCEPTION;
    --asociază nume, codului eroare folosit in trigger
      PRAGMA EXCEPTION_INIT(nu_accepta,-20005);
   -- variabila PL/SQL in care retinem codul maxim + 1
     v_cod emp_pnu.employee_id%TYPE;
BEGIN
  SELECT MAX(employee_id) + 1 INTO v_cod
   FROM emp_pnu;
  -- incercare inserare
  INSERT   INTO emp_pnu (employee_id, last_name, email, hire_date,  salary)
  VALUES (v_cod, 'someone', 'smth@smth.com', SYSDATE, 900);
EXCEPTION
  /* tratare exceptie */
  WHEN nu_accepta THEN
    DBMS_OUTPUT.PUT_LINE('trigger-ul lanseaza o eroare cu mesajul' || SQLERRM );
   /* SQLERRM va returna mesaj din RAISE_APPLICATION_ERROR */
END;
```

[Cazuri speciale în tratarea excepțiilor]

7. Să se scrie un program (bloc) PL/SQL care să determine codul şi salariul angajatului având salariul minim în departamentul cerut de utilizator şi apoi să se introducă informațiile găsite în tabelul mesaje_pnu.

Obs: Dacă se declanşează o excepție într-un bloc simplu, atunci se face saltul la partea de tratare (*handler*) a acesteia, iar după ce este terminată tratarea erorii se iese din bloc (instrucțiunea *END*).

Comentați corectitudinea următoarei soluții propuse:

```
ACCEPT p_dep PROMPT 'Introduceti un cod de departament '
DECLARE
  v_cod emp_pnu.employee_id%TYPE;
  v_sal emp_pnu.salary%TYPE;
  v_dep emp_pnu.department_id%TYPE := &p_dep;
BEGIN
  DELETE from mesaje_pnu;
    v_cod := -1;
    v_sal := 0;
  SELECT employee_id, salary INTO v_cod, v_sal
   FROM    emp_pnu e
   WHERE salary = (SELECT MIN(salary)
                          FROM     emp_pnu
                          WHERE  department_id = e.department_id);
-poate declansa exceptia NO_DATA_FOUND sau TOO_MANY_ROWS
--se ajunge la comanda urmatoare?
   INSERT INTO mesaje_pnu (v_cod || v_sal);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20500, 'Nici o linie gasita') ;
  WHEN TOO_MANY_ROWS THEN
        RAISE_APPLICATION_ERROR(-20501, 'Prea multi angajati') ;
END;
```

Deficiența anterioară (iesirea din bloc inaintea se poate rezolva incluzând într-un subbloc comanda *SELECT* care a declanşat excepția.

```
BEGIN
        DELETE from mesaje_pnu;
        v_cod := -1;
```

```
        v_sal := 0;
        <<subbloc>>
        BEGIN
                SELECT employee_id, salary INTO v_cod, v_sal
                FROM    emp_pnu e
                WHERE salary = (SELECT MIN(salary)
                                FROM     emp_pnu
                                WHERE  department_id = e.department_id);
        EXCEPTION
                WHEN NO_DATA_FOUND THEN
                        RAISE_APPLICATION_ERROR(-20500, 'Nici o linie gasita') ;
                WHEN TOO_MANY_ROWS THEN
                        RAISE_APPLICATION_ERROR(-20501, 'Prea multi angajati') ;
                  /* dupa ce se trateaza exceptiile, controlul este
                        transferat blocului de nivel superior, de fapt
                    comenzii INSERT */
        END <<subbloc>>;
        INSERT INTO mesaje_pnu (v_cod || v_sal);
END;
```

8. Să se creeze un bloc PL/SQL care determină:
- numele, salariul si vechimea angajatului având salariul maxim în departamentul în care salariul mediu este minim;
- codul şi data angajării celui mai bine platit angajat din Oxford.
- numele şi salariul angajatului având cea mai mica vechime.

Dacă vreuna dintre comenzi lansează excepția TOO_MANY_ROWS, să se introducă în tabelul *mesaje_pnu* informații despre comanda care a lansat aceasta exceptie.

Solutia 1: Introducerea unui contor care să identifice instrucțiunea *SQL*.

```
DECLARE
  v_sel_cont  NUMBER(2):=1;
   v_nume employees.last_name%TYPE;
   v_sal employees.salary%TYPE;
   v_vechi NUMBER(2);
   v_cod employees.employee_id%TYPE;
   v_data employees.hire_date%TYPE;
BEGIN
  SELECT last_name, salary, ROUND((sysdate-hire_date)/365)
   INTO      v_nume, v_sal, v_vechi
   FROM    employees e
   WHERE department_id IN (SELECT department_id
                                FROM employees
                                GROUP BY department_id
                                HAVING AVG(salary) = (SELECT MIN(AVG(salary))
                                                      FROM employees
                                                      GROUP BY department_id))
    AND salary = (SELECT MAX(salary)
                        FROM employees
                        WHERE e.department_id = department_id);
  v_sel_cont:=2;
    SELECT employee_id, hire_date
    INTO v_cod, v_data
    FROM employees e, departments d, locations l
    WHERE e.department_id = d.department_id AND d.department_id = l.location_id
```

```
   AND (city, salary)  IN (SELECT city, MAX(salary)
                           FROM employees e, departments d, locations l
                           WHERE e.department_id = d.department_id
                            AND d.department_id = l.location_id
                            AND INITCAP(city) = 'Oxford');
  v_sel_cont:=3;
  SELECT last_name, salary
   INTO v_nume, v_sal
   FROM employees
   WHERE hire_date  = (SELECT MAX(hire_date)
                           FROM employees);
EXCEPTION
  WHEN TOO_MANY_ROWS THEN
    INSERT INTO mesaje_pnu(rezultat)
    VALUES ('comanda SELECT ' || TO_CHAR(v_sel_cont) ||
              ' gaseste prea multe date');
END;
```

Solutia 2: Introducerea fiecărei instrucțiuni *SQL* într-un subbloc.

```
BEGIN
  BEGIN
    SELECT …
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      INSERT INTO mesaje_pnu(rezultat)
      VALUES('SELECT 1 gaseste prea multe date');
  END;
  BEGIN
    SELECT …
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      INSERT INTO mesaje_pnu(rezultat)
      VALUES('SELECT 2 gaseste prea multe date');
  END;
  …
END;
```

Ce deosebire este înte cele două variante? (care este mai avantajoasă?)

[Propagarea excepțiilor]

9. Să se rețină în tabelul mesaje_pnu raportul dintre salariu şi comision pentru angajatul având cea mai mare vechime.

Variante:

```
DECLARE
  v_var  NUMBER(10,3);
BEGIN
  SELECT salary/comission_pct
  INTO   v_var
  FROM   emp_pnu
  WHERE  hire_date  =  (SELECT
MIN(hire_date) FROM emp_pnu);
  <<eticheta>>
  INSERT  INTO  mesaje_pnu  VALUES
(v_var);
```

```
DECLARE
  v_var  NUMBER(10,3);
BEGIN
  IF nvl(commission_pct, 0)=0 THEN
       GOTO eticheta; --este posibil?
  END IF;
  SELECT salary/comission_pct
  INTO   v_var
  FROM   emp_pnu
  WHERE  hire_date  =  (SELECT
MIN(hire_date) FROM emp_pnu);
```

EXCEPTION WHEN ZERO_DIVIDE THEN v_var:=0; GOTO <<eticheta>>; --este posibil? END;	INSERT INTO mesaje_pnu VALUES (v_var); EXCEPTION <<eticheta>> WHEN ZERO_DIVIDE THEN v_var:=0; END;

Obs: Instrucțiunea *GOTO* nu permite:

- saltul la secțiunea de tratare a unei excepții;
- saltul de la secțiunea de tratare a unei excepții, în blocul curent.

Comanda *GOTO* permite totuşi saltul de la secțiunea de tratare a unei excepții la un bloc care include blocul curent.

Cum se poate remedia situația de mai sus? Schimbați blocul propus (utilizând un subbloc) astfel încât să funcționeze corespunzător.

10. [Excepție sesizată în secțiunea executabilă a unui subbloc, dar netratată în subbloc]
Să se declare un bloc în care se va crea un subbloc ce lansează o excepție e2. Subblocul nu va conține handler pentru e2, în schimb un astfel de handler se va afla în bloc. Ce se întâmplă la execuția blocului?

Obs: Excepția este sesizată în subbloc, dar nu este tratată în acesta şi atunci se propagă spre blocul exterior. Regula poate fi aplicată de mai multe ori.

```
DECLARE
  e1  EXCEPTION;
  e2  EXCEPTION;
BEGIN
  <<subbloc>>
  BEGIN
    RAISE e2; --exceptia e2 sesizata in subbloc
  EXCEPTION
    WHEN e1 THEN
          RAISE_APPLICATION_ERROR(-20200, ‘Mesaj pentru e1 in subbloc’);
     --exceptia e2 nu este tratata in subbloc
  END <<subbloc>>;
EXCEPTION
  WHEN e2 THEN
          RAISE_APPLICATION_ERROR(-20202, ’Mesaj pentru e2 in bloc’);
  /* exceptia e2 s-a propagat spre blocul exterior unde a
  fost tratata, apoi controlul trece in exteriorul blocului */
END;
```

11. [Excepție sesizată în secțiunea declarativă]
Să se realizeze un program care calculează numărul departamentelor în care lucrează angajați, exemplificând erorile care pot apărea în secțiunea declarativă.

```
BEGIN
  DECLARE
    nr_dep   NUMBER(3) := 'XYZ'; --eroare
  BEGIN
    SELECT  COUNT (DISTINCT department_id)
    INTO    nr_dep
    FROM    emp_pnu;
  EXCEPTION
    WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE('Eroare bloc intern:' || SQLERRM);
```

```
  END;
EXCEPTION
  WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE('Eroare bloc extern:' || SQLERRM );
END;
```

Obs: Dacă în secțiunea declarativă este generată o excepție, atunci aceasta se propagă către blocul exterior, unde are loc tratarea acesteia.

12. [Excepție sesizată în secțiunea *EXCEPTION*]

Să se declare un bloc PL/SQL, care conține un subbloc ce lansează o excepție e1 care la rândul ei lansează o altă excepție e2. Excepția e2 va avea câte un handler atât în bloc, cât şi în subbloc. Unde va fi tratată e2 în urma execuției blocului?

Obs: Dacă excepția este sesizată în secțiunea *EXCEPTION*, ea se propagă imediat spre blocul exterior.

```
DECLARE
   e1  EXCEPTION;
   e2  EXCEPTION;
BEGIN
  <<subbloc>>
  BEGIN
    RAISE e1; --sesizare exceptie e1
  EXCEPTION
    WHEN e1 THEN
      RAISE e2; --sesizare exceptie e2
    WHEN e2 THEN
       RAISE_APPLICATION_ERROR(-20200, ‘Handler in subbloc’);
     /* exceptia este propagata spre blocul exterior
      cu toate ca exista aici un handler pentru ea */
  END <<subbloc>>;
EXCEPTION
  WHEN e2 THEN
        RAISE_APPLICATION_ERROR(-20202, ‘Handler in bloc’);
  --exceptia e2 este tratata in blocul exterior
END;
```

13. Compilați una din procedurile de la laboratoarele precedente şi afişați erorile de compilare.

```
SELECT LINE, POSITION, TEXT
FROM   USER_ERRORS
WHERE  UPPER(NAME) = UPPER('nume_procedura');
```