Laborator 1 PL/SQL

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

PL/SQL include atât instrucțiuni *SQL* pentru prelucrarea datelor și pentru gestiunea tranzacțiilor, cât și instrucțiuni proprii.

PL/SQL extinde *SQL* prin construcții specifice limbajelor procedurale (definirea variabilelor, declararea tipurilor, utilizarea structurilor de control, implementarea procedurilor și funcțiilor, introducerea tipurilor obiect și metodelor etc).

Tipurile de date scalare

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

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.
- La declararea variabilelor în *PL/SQL* pot fi utilizate atributele *%TYPE* și *%ROWTYPE*, care reprezintă tipuri de date implicite.
- Atributul *%TYPE* permite definirea unei variabile cu același tip de date ca al altei variabile sau al unei coloane dintr-un tabel.
- Atributul *%ROWTYPE* permite definirea unei variabile de tip înregistrare cu aceeași structură ca a altei variabile de tip înregistrare, a unui tabel sau cursor.

Sintaxa declarării unei variabile este următoarea:

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

- Constantele și variabilele *NOT NULL* trebuie inițializate atunci când sunt declarate, altfel apare eroare la compilare.
- Afișarea valorii variabilelor se face cu ajutorul procedurilor:
  DBMS_OUTPUT.PUT(sir_caractere);
  DBMS_OUTPUT.PUT_LINE(sir_caractere);
  ***Obs*:** Dacă se lucrează în SQL*Plus, atunci pentru activarea modului afișare se utilizează comanda SET SERVEROUTPUT ON.

Blocuri PL/SQL

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

Tipuri de blocuri:

- anonime - sunt blocuri fără nume, care nu sunt stocate în baza de date și sunt compilate de fiecare dată când sunt executate; acest tip de bloc nu permite parametrii și nu poate întoarce un rezultat;
- neanonime - sunt blocuri cu nume care sunt compilate o singură dată, sunt stocate în baza de date și pot fi apelate din alte aplicații.

Un bloc PL/SQL are structura:

```
[<<nume_bloc>>]
[DECLARE
        variabile, cursoare]
BEGIN
        instrucțiuni SQL și PL/SQL
[EXCEPTION
        tratarea erorilor]
END[nume_bloc]
```

Instrucțiuni PL/SQL

PL/SQL dispune de comenzi ce permit controlul execuției unui bloc.

Instrucțiunile limbajului pot fi: iterative (*LOOP*, *WHILE*, *FOR*), de atribuire (:=), condiționale (*IF, CASE*), de salt (*GOTO*, *EXIT*) și instrucțiunea vidă (*NULL*).

Comentarii în PL/SQL

- pe o singură linie, prefixate de simbolurile "--", care încep în orice punct al liniei și se termină la sfârșitul acesteia;
- pe mai multe linii, care sunt delimitate de simbolurile "/*" și "*/".

Caracterul ";" este separator pentru instrucțiuni.

Observație Pentru a nu se vedea codul PL/SQL la rularea unui script se setează parametrul ECHO la valoarea OFF.

1. Evaluați următoarele declarații de variabile:

a.
```
DECLARE
    v_nume, v_prenume VARCHAR2(35);  -- greșit
Corect:
DECLARE
    v_nume    VARCHAR2(35);
    v_prenume VARCHAR2(35);
```

b.
```
DECLARE
   v_nr  NUMBER(5);     --corect
```

c.
```
DECLARE
   v_nr NUMBER(5,2) = 10;     --greșit
```
Corect:
```
DECLARE
   v_nr NUMBER(5,2) := 10;
```

d.
```
DECLARE
   v_test   BOOLEAN:= SYSDATE;    --greșit
```
Corect:
```
DECLARE
   v_test   BOOLEAN:=TRUE;
```

e.
```
DECLARE
   v1  NUMBER(5) :=10;
   v2  NUMBER(5) :=15;
   v3  NUMBER(5) := v1< v2;     --greșit
```
Corect:
```
DECLARE
   v1  NUMBER(5) :=10;
   v2  NUMBER(5) :=15;
   v3  BOOLEAN := v1< v2;
```

2. Se dă următorul bloc PL/SQL:

```
<<principal>>
DECLARE
  v_client_id     NUMBER(4):= 1600;
  v_client_nume   VARCHAR2(50):= 'N1';
  v_nou_client_id NUMBER(3):= 500;
BEGIN
 <<secundar>>
 DECLARE
   v_client_id       NUMBER(4) := 0;
   v_client_nume     VARCHAR2(50) := 'N2';
   v_nou_client_id   NUMBER(3) := 300;
   v_nou_client_nume VARCHAR2(50) := 'N3';
 BEGIN
   v_client_id:= v_nou_client_id;
   principal.v_client_nume:=
             v_client_nume ||' '|| v_nou_client_nume;
   --poziția 1
 END;
 v_client_id:= (v_client_id *12)/10;
 --poziția 2
END;
/
```

Determinați:
- valoarea variabilei v_client_id la poziția 1;
- valoarea variabilei v_client_nume la poziția 1;
- valoarea variabilei v_nou_client_id la poziția 1;
- valoarea variabilei v_nou_client_nume la poziția 1;
- valoarea variabilei v_id_client la poziția 2;
- valoarea variabilei v_client_nume la poziția 2.

Soluție:

```
Poz1 v_client_id 300
Poz1 v_client_nume N2
Poz1 v_nou_client_id 300
Poz1 v_nou_client_nume N3
Poz2 v_client_id 1920
Poz2 v_client_nume N2 N3
```

3. Creați un bloc anonim care să afișeze propoziția "Invat PL/SQL" pe ecran.

Varianta 1 - Afișare folosind variabile de legătură

```
VARIABLE g_mesaj VARCHAR2(50)
BEGIN
  :g_mesaj := 'Invat PL/SQL';
END;
/
PRINT g_mesaj
```

Varianta 2 - Afișare folosind procedurile din pachetul standard DBMS_OUTPUT

```
BEGIN
  DBMS_OUTPUT.PUT_LINE('Invat PL/SQL');
END;
/
```

4. Definiți un bloc anonim în care să se afle numele departamentului cu cei mai mulți angajați. Comentați cazul în care există cel puțin două departamente cu număr maxim de angajați.

```
DECLARE
  v_dep departments.department_name%TYPE;
BEGIN
  SELECT department_name
  INTO   v_dep
  FROM   employees e, departments d
  WHERE  e.department_id=d.department_id
  GROUP BY department_name
  HAVING COUNT(*) = (SELECT MAX(COUNT(*))
                     FROM   employees
                     GROUP BY department_id);
  DBMS_OUTPUT.PUT_LINE('Departamentul '|| v_dep);
END;
/
```

5. Rezolvați problema anterioară utilizând variabile de legătură. Afișați rezultatul atât din bloc, cât și din exteriorul acestuia.

```
VARIABLE rezultat VARCHAR2(35)
BEGIN
  SELECT department_name
  INTO   :rezultat
```

```
  FROM   employees e, departments d
  WHERE  e.department_id=d.department_id
  GROUP BY department_name
  HAVING COUNT(*) = (SELECT MAX(COUNT(*))
                     FROM   employees
                     GROUP BY department_id);
  DBMS_OUTPUT.PUT_LINE('Departamentul '|| :rezultat);
END;
/
PRINT rezultat
```

6. Modificați exercițiul anterior astfel încât să obțineți și numărul de angajați din departamentul respectiv.

7. Determinați salariul anual și bonusul pe care îl primește un salariat al cărui cod este dat de la tastatură. Bonusul este determinat astfel: dacă salariul anual este cel puțin 200001, atunci bonusul este 20000; dacă salariul anual este cel puțin 100001 și cel mult 200000, atunci bonusul este 10000, iar dacă salariul anual este cel mult 100000, atunci bonusul este 5000. Afișați bonusul obținut. Comentați cazul în care nu există niciun angajat cu codul introdus.

Obs. Se folosește **instrucțiunea *IF*.**

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

```
SET VERIFY OFF
DECLARE
   v_cod           employees.employee_id%TYPE:=&p_cod;
   v_bonus         NUMBER(8);
   v_salariu_anual NUMBER(8);
BEGIN
   SELECT salary*12 INTO v_salariu_anual
   FROM   employees
   WHERE  employee_id = v_cod;
   IF v_salariu_anual>=200001
      THEN v_bonus:=20000;
   ELSIF v_salariu_anual BETWEEN 100001 AND 200000
      THEN v_bonus:=10000;
   ELSE v_bonus:=5000;
END IF;
DBMS_OUTPUT.PUT_LINE('Bonusul este ' || v_bonus);
END;
/
SET VERIFY ON
```

8. Rezolvați problema anterioară folosind instrucțiunea CASE.

```
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 altă_secvență;]
    END CASE;

    sau

    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];
```

Clauza *ELSE* este opțională.
Dacă aceasta este necesară în implementarea unei probleme, dar practic lipsește, iar *test_var* nu ia nici una dintre valorile ce apar în clauzele *WHEN*, atunci se declanșează eroarea predefinită *CASE_NOT_FOUND* (*ORA - 6592*).

```
DECLARE
   v_cod           employees.employee_id%TYPE:=&p_cod;
   v_bonus         NUMBER(8);
   v_salariu_anual NUMBER(8);
BEGIN
   SELECT salary*12 INTO v_salariu_anual
   FROM   employees
   WHERE  employee_id = v_cod;
   CASE WHEN v_salariu_anual>=200001
             THEN v_bonus:=20000;
        WHEN v_salariu_anual BETWEEN 100001 AND 200000
             THEN v_bonus:=10000;
        ELSE v_bonus:=5000;
   END CASE;
DBMS_OUTPUT.PUT_LINE('Bonusul este ' || v_bonus);
END;
/
```

9. Scrieți un bloc PL/SQL în care stocați prin variabile de substituție un cod de angajat, un cod de departament și procentul cu care se mărește salariul acestuia. Să se mute salariatul în noul departament și să i se crească salariul în mod corespunzător. Dacă modificarea s-a putut realiza (există în tabelul *emp_**** un salariat având codul respectiv) să se afișeze mesajul "Actualizare realizata", iar în caz contrar mesajul "Nu exista un angajat cu acest cod". Anulați modificările realizate.

```
DEFINE p_cod_sal= 200
DEFINE p_cod_dept = 80
DEFINE p_procent =20
DECLARE
  v_cod_sal   emp_***.employee_id%TYPE:= &p_cod_sal;
  v_cod_dept  emp_***.department_id%TYPE:= &p_cod_dept;
  v_procent   NUMBER(8):=&p_procent;
```

```
BEGIN
  UPDATE emp_***
  SET department_id = v_cod_dept,
      salary=salary + (salary* v_procent/100)
  WHERE employee_id= v_cod_sal;
  IF SQL%ROWCOUNT =0 THEN
     DBMS_OUTPUT.PUT_LINE('Nu exista un angajat cu acest cod');
  ELSE DBMS_OUTPUT.PUT_LINE('Actualizare realizata');
  END IF;
END;
/
ROLLBACK;
```

10. Creați tabelul *zile_****(id, data, nume_zi). Introduceți în tabelul *zile_**** informațiile corespunzătoare tuturor zilelor care au rămas din luna curentă.

```
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.

```
DECLARE
  contor  NUMBER(6) := 1;
  v_data  DATE;
  maxim   NUMBER(2) := LAST_DAY(SYSDATE)-SYSDATE;
BEGIN
  LOOP
    v_data := sysdate+contor;
    INSERT INTO zile_***
    VALUES (contor,v_data,to_char(v_data,'Day'));
    contor := contor + 1;
    EXIT WHEN contor > maxim;
  END LOOP;
END;
/
```

11. Rezolvați cerința anterioară folosind instrucțiunea WHILE.

```
WHILE condiție LOOP
   secvența_de_comenzi
END LOOP;
```

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

```
DECLARE
  contor  NUMBER(6) := 1;
  v_data  DATE;
  maxim   NUMBER(2) := LAST_DAY(SYSDATE)-SYSDATE;
```

```
BEGIN
  WHILE contor <= maxim LOOP
    v_data := sysdate+contor;
    INSERT INTO zile_***
    VALUES (contor,v_data,to_char(v_data,'Day'));
    contor := contor + 1;
  END LOOP;
END;
/
```

12. Rezolvați cerința anterioară folosind instrucțiunea FOR.

```
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.*
Aceasta este neidentificată în afara ciclului.
Pasul are implicit valoarea 1 și nu poate fi modificat.
Limitele domeniului pot fi variabile sau expresii, dar care pot fi convertite la întreg.

```
DECLARE
  v_data  DATE;
  maxim   NUMBER(2) := LAST_DAY(SYSDATE)-SYSDATE;
BEGIN
  FOR contor IN 1..maxim LOOP
    v_data := sysdate+contor;
    INSERT INTO zile_***
    VALUES (contor,v_data,to_char(v_data,'Day'));
  END LOOP;
END;
/
```

13. Să se declare și să se inițializeze cu 1 variabila *i* de tip *POZITIVE* și cu 10 constanta *max_loop* de tip *POZITIVE*. Să se implementeze un ciclu *LOOP* care incrementează pe *i* până când acesta ajunge la o valoare > *max_loop*, moment în care ciclul *LOOP* este părăsit și se sare la instrucțiunea i:=1.

Obs. Se utilizează **instrucțiunile GOTO/EXIT**.

Instrucțiunea *EXIT* permite ieșirea 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 aceleași reguli ca și cele definite pentru identificatori. Eticheta se plasează înaintea comenzii, fie pe aceeași linie, fie pe o linie separată. Etichetele se definesc prin intercalare între “<<” și “>>”.

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

Varianta 1

```
DECLARE
   i         POSITIVE:=1;
   max_loop CONSTANT POSITIVE:=10;
BEGIN
LOOP
   i:=i+1;
   IF i>max_loop THEN
     DBMS_OUTPUT.PUT_LINE('in loop i=' || i);
     GOTO urmator;
   END IF;
END LOOP;
<<urmator>>
i:=1;
DBMS_OUTPUT.PUT_LINE('dupa loop i=' || i);
END;
/
```

Varianta 2

```
DECLARE
  i         POSITIVE:=1;
  max_loop CONSTANT POSITIVE:=10;
BEGIN
  i:=1;
  LOOP
    i:=i+1;
    DBMS_OUTPUT.PUT_LINE('in loop i=' || i);
    EXIT WHEN i>max_loop;
  END LOOP;
  i:=1;
  DBMS_OUTPUT.PUT_LINE('dupa loop i=' || i);
END;
/
```

Exerciții

1. Se dă următorul bloc:

```
DECLARE
 numar number(3):=100;
mesaj1 varchar2(255):='text 1';
 mesaj2 varchar2(255):='text 2';
BEGIN
  DECLARE
   numar number(3):=1;
   mesaj1 varchar2(255):='text 2';
   mesaj2 varchar2(255):='text 3';
```

```
  BEGIN
   numar:=numar+1;
   mesaj2:=mesaj2||' adaugat in sub-bloc';
  END;
 numar:=numar+1;
 mesaj1:=mesaj1||' adaugat un blocul principal';
 mesaj2:=mesaj2||' adaugat in blocul principal';
END;
```

a) Valoarea variabilei *numar* în subbloc este:
b) Valoarea variabilei *mesaj1* în subbloc este:
c) Valoarea variabilei *mesaj2* în subbloc este:
d) Valoarea variabilei *numar* în bloc este:
e) Valoarea variabilei *mesaj1* în bloc este:
f) Valoarea variabilei *mesaj2* în bloc este:
Verificați răspunsul.

2. Se dă următorul enunț: Pentru fiecare zi a lunii octombrie (se vor lua în considerare și zilele din lună în care nu au fost realizate împrumuturi) obțineți numărul de împrumuturi efectuate.

a. Încercați să rezolvați problema în SQL fără a folosi structuri ajutătoare.
b. Definiți tabelul *octombrie_*** (id, data). Folosind PL/SQL populați cu date acest tabel. Rezolvați în SQL problema dată.

3. Definiți un bloc anonim în care să se determine numărul de filme (titluri) împrumutate de un membru al cărui nume este introdus de la tastatură. Tratați următoarele două situații: nu există nici un membru cu nume dat; există mai mulți membrii cu același nume.

4. Modificați problema anterioară astfel încât să afișați și următorul text:

- Categoria 1 (a împrumutat mai mult de 75% din titlurile existente)
- Categoria 2 (a împrumutat mai mult de 50% din titlurile existente)
- Categoria 3 (a împrumutat mai mult de 25% din titlurile existente)
- Categoria 4 (altfel)

5. Creați tabelul *member_*** (o copie a tabelului *member*). Adăugați în acest tabel coloana *discount*, care va reprezenta procentul de reducere aplicat pentru membrii, în funcție de categoria din care fac parte aceștia:

- 10% pentru membrii din Categoria 1
- 5% pentru membrii din Categoria 2
- 3% pentru membrii din Categoria 3
- nimic

Actualizați coloana *discount* pentru un membru al cărui cod este dat de la tastatură. Afișați un mesaj din care să reiasă dacă actualizarea s-a produs sau nu.

Laborator 2 PL/SQL

Tipuri de date compuse (definite de utilizator)
Tipul de date înregistrare (RECORD)
Tipul de date colecție (tablouri indexate – INDEX-BY TABLES
tablouri imbricate – NESTED TABLES
vectori – VARRAYS)

Considerații legate de valoarea NULL

- comparațiile simple ce implică NULL sunt evaluate la NULL;
- negarea unei valori NULL (NOT NULL) este NULL;
- în comenzile condiționale, dacă o condiție este evaluată la NULL, atunci secvența de comenzi asociată nu va fi executată:

```
IF condiție THEN
  -- dacă valoarea de adevăr este TRUE
  Secvența de comenzi 1;
ELSE
  -- dacă valoarea de adevăr este FALSE sau NULL
  Secvența de comenzi 2;
END IF;
```

Nr	P	Q	NOT P	P OR Q	P AND Q
1	false	false	true	false	false
2	false	true	true	true	false
3	false	Null	true	Null	false
4	true	false	false	true	false
5	true	true	false	true	true
6	true	Null	false	true	Null
7	Null	false	Null	Null	false
8	Null	true	Null	true	Null
9	Null	Null	Null	Null	Null

1. Care este rezultatul următorului bloc PL/SQL?

```
DECLARE
  x      NUMBER(1) := 5;
  y      x%TYPE  := NULL;
BEGIN
  IF x <> y THEN
      DBMS_OUTPUT.PUT_LINE ('valoare <> null este = true');
    ELSE
      DBMS_OUTPUT.PUT_LINE ('valoare <> null este != true');
  END IF;

  x := NULL;
  IF x = y THEN
       DBMS_OUTPUT.PUT_LINE ('null = null este = true');
    ELSE
       DBMS_OUTPUT.PUT_LINE ('null = null este != true');
  END IF;
END;
/
```

Tipul de date RECORD

- definește un grup de date stocate sub formă de câmpuri, fiecare cu tipul de date și numele propriu;
- numărul de câmpuri nu este limitat;
- se pot defini valori inițiale și constrângeri NOT NULL asupra câmpurilor;
- câmpurile sunt inițializate automat cu NULL;
- tipul RECORD poate fi folosit în secțiunea declarativă a unui bloc, subprogram sau pachet;
- se pot declara sau referi tipuri RECORD imbricate;
- sintaxa generală a definirii tipului RECORD este:

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

v_nume_record   nume_tip;
```

- câmpurile unei înregistrări PL/SQL sunt accesate prin prefixare cu numele înregistrării:

```
nume_record.nume_camp
```

- utilizând tipul *RECORD*:
  - se poate insera o linie într-un tabel (*INSERT*);
  - se poate actualiza o linie într-un tabel (*UPDATE* cu sintaxa *SET ROW*);
  - se poate menține informația afectată de comenzile LMD folosind clauza *RETURNING*.

2. Definiți tipul înregistrare *emp_record* care conține câmpurile *employee_id*, *salary* și *job_id*. Apoi, definiți o variabilă de acest tip.

a. Inițializați variabila definită. Afișați variabila.

```
DECLARE
  TYPE emp_record IS RECORD
        (cod employees.employee_id%TYPE,
         salariu employees.salary%TYPE,
         job employees.job_id%TYPE);
  v_ang emp_record;
BEGIN
  v_ang.cod:=700;
  v_ang.salariu:= 9000;
  v_ang.job:='SA_MAN';
  DBMS_OUTPUT.PUT_LINE ('Angajatul cu codul '|| v_ang.cod ||
    ' si jobul ' || v_ang.job || ' are salariul ' ||  v_ang.salariu);
END;
/
```

b. Inițializați variabila cu valorile corespunzătoare angajatului având codul 101. Afișați variabila.

```
BEGIN
 /********   In loc de ...
 * SELECT employee_id, salary, job_id
 * INTO   v_ang.cod, v_ang.salariu, v_ang.job
 * FROM   employees
 * WHERE  employee_id = 101;
 ******************************/
```

```
 SELECT employee_id, salary, job_id
 INTO   v_ang
 FROM   employees
 WHERE  employee_id = 101;
 DBMS_OUTPUT.PUT_LINE ('Angajatul cu codul '|| v_ang.cod ||
    ' si jobul ' || v_ang.job || ' are salariul ' ||  v_ang.salariu);
END;
/
```

c. Ștergeți angajatul având codul 100 din tabelul *emp_*** și rețineți în variabila definită anterior informații corespunzătoare acestui angajat. Anulați modificările realizate.

```
BEGIN
 DELETE FROM emp_***
 WHERE employee_id=100
 RETURNING employee_id, salary, job_id INTO v_ang;

 DBMS_OUTPUT.PUT_LINE ('Angajatul cu codul '|| v_ang.cod ||
    ' si jobul ' || v_ang.job || ' are salariul ' ||  v_ang.salariu);
END;
/
ROLLBACK;
```

Atributul %ROWTYPE

- Este utilizat pentru a declara o variabilă de tip înregistrare cu aceeași structură ca a altei variabile de tip înregistrare, a unui tabel sau cursor.

3. Declarați două variabile cu aceeași structură ca și tabelul *emp_***. Ștergeți din tabelul *emp_*** angajații 100 și 101, menținând valorile șterse în cele două variabile definite. Folosind cele două variabile, introduceți informațiile șterse în tabelul *emp_***.

```
DECLARE
 v_ang1     employees%ROWTYPE;
 v_ang2     employees%ROWTYPE;
BEGIN
-- sterg angajat 100 si mentin in variabila linia stearsa
   DELETE FROM emp_***
   WHERE employee_id = 100
   RETURNING employee_id, first_name, last_name, email, phone_number,
             hire_date, job_id, salary, commission_pct, manager_id,
             department_id
   INTO v_ang1;

-- inserez in tabel linia stearsa
   INSERT INTO emp_***
   VALUES v_ang1;

-- sterg angajat 101
   DELETE FROM emp_***
   WHERE employee_id = 101;
```

```
-- obtin datele din tabelul employees
   SELECT *
   INTO   v_ang2
   FROM   employees
   WHERE  employee_id = 101;

-- inserez o linie oarecare in emp_***
   INSERT INTO emp_***
   VALUES(1000,'FN','LN','E',null,sysdate,'AD_VP',1000, null,100,90);

-- modific linia adaugata anterior cu valorile variabilei v_ang2
   UPDATE emp_***
   SET    ROW = v_ang2
   WHERE  employee_id = 1000;
 END;
/
```

Metode pentru colecții (tablouri indexate, tablouri imbricate, vectori)

PL/SQL oferă subprograme numite metode care operează asupra unei colecții. Acestea pot fi apelate numai din comenzi procedurale (deci, nu din *SQL*).

Metodele sunt apelate prin expresia:

nume_colecție.nume_metodă [(*parametri*)]

Metodele care se pot aplica colecțiilor *PL/SQL* sunt următoarele:

- *COUNT* întoarce numărul curent de elemente al unei colecții *PL/SQL*;
- *DELETE*(*n*) șterge elementul *n* dintr-o colecție *PL/SQL*; *DELETE*(*m*, *n*) șterge toate elementele având indecșii între *m* și *n*; *DELETE* șterge toate elementele unei colecții *PL/SQL* (**nu este validă pentru tipul *varrays***);
- *EXISTS*(*n*) întoarce *TRUE* dacă există al *n*-lea element al unei colecții *PL/SQL*; altfel, întoarce *FALSE*;
- *FIRST*, *LAST* întorc indicele primului, respectiv ultimului element din colecție;
- *NEXT*(*n*), *PRIOR*(*n*) întorc indicele elementului următor, respectiv precedent celui de rang *n* din colecție, iar dacă nu există un astfel de element întorc valoarea *null*;
- *EXTEND* adaugă elemente la sfârșitul unei colecții: *EXTEND* adaugă un element *null* la sfârșitul colecției, *EXTEND*(*n*) adaugă *n* elemente *null*, *EXTEND*(*n*, *i*) adaugă *n* copii ale elementului de rang *i* (**nu este validă pentru tipul *index-by tables***);
- *LIMIT* întoarce numărul maxim de elemente al unei colecții (cel de la declarare) pentru tipul vector și *null* pentru tablouri imbricate (**nu este validă pentru tipul *index-by tables***);
- *TRIM* șterge elementele de la sfârșitul unei colecții: *TRIM* șterge ultimul element, *TRIM*(*n*) șterge ultimele *n* elemente (**nu este validă pentru tipul *index-by tables***). Similar metodei *EXTEND*, metoda *TRIM* operează asupra dimensiunii interne a tabloului imbricat.

  - *EXISTS* este singura metodă care poate fi aplicată unei colecții atomice *null*. Orice altă metodă declanșează excepția *COLLECTION_IS_NULL*.
  - *COUNT*, *EXISTS*, *FIRST*, *LAST*, *NEXT*, *PRIOR* și *LIMIT* sunt funcții, iar restul sunt proceduri *PL/SQL*.

Observație:

- Tipul *tablou indexat* poate fi utilizat numai în declarații *PL/SQL*. Tipurile *vector* și *tablou imbricat* pot fi utilizate atât în declarații *PL/SQL*, cât și în declarații la nivelul schemei (de exemplu, pentru definirea tipului unei coloane a unui tabel).
- Tablourile indexate pot avea indice negativ, domeniul permis pentru index fiind

–2147483648..2147483647; pentru tablourile imbricate domeniul permis pentru index este 1..2147483647.

- Tablourile imbricate și vectorii trebuie inițializați și/sau extinși pentru a li se putea adăuga elemente noi.

Tablouri indexate (index-by table)

- Sunt mulțimi de perechi cheie-valoare, în care fiecare cheie este unică și utilizată pentru a putea localiza valoarea asociată.
- Tablourile indexate pot creste în dimensiune în mod dinamic neavând specificat un număr maxim de elemente.
- Un tablou indexat nu poate fi inițializat la declarare, este necesară o comandă explicită pentru a inițializa fiecare element al său.
- Sintaxa generală pentru tabloul indexat este:

```
TYPE t_tablou_indexat IS TABLE OF {
  { tip_de_date | variabila%TYPE
    | tabel.coloana%TYPE }[NOT NULL]}
    | tabel%ROWTYPE }
INDEX BY PLS_INTEGER | BINARY_INTEGER | VARCHAR2(n);

v_tablou  t_tablou_indexat;
```

4. Definiți un tablou indexat de numere. Introduceți în acest tablou primele 10 de numere naturale.
- **a.** Afișați numărul de elemente al tabloului și elementele acestuia.
- **b.** Setați la valoarea *null* elementele de pe pozițiile impare. Afișați numărul de elemente al tabloului și elementele acestuia.
- **c.** Ștergeți primul element, elementele de pe pozițiile 5, 6 și 7, respectiv ultimul element. Afișați valoarea și indicele primului, respectiv ultimului element. Afișați elementele tabloului și numărul acestora.
- **d.** Ștergeți toate elementele tabloului.

```
DECLARE
  TYPE tablou_indexat IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
  t    tablou_indexat;
BEGIN
-- punctul a
  FOR i IN 1..10 LOOP
    t(i):=i;
  END LOOP;
  DBMS_OUTPUT.PUT('Tabloul are ' || t.COUNT ||' elemente: ');
  FOR i IN t.FIRST..t.LAST LOOP
      DBMS_OUTPUT.PUT(t(i) || ' ');
  END LOOP;
  DBMS_OUTPUT.NEW_LINE;

-- punctul b
  FOR i IN 1..10 LOOP
    IF i mod 2 = 1 THEN t(i):=null;
    END IF;
  END LOOP;
  DBMS_OUTPUT.PUT('Tabloul are ' || t.COUNT ||' elemente: ');
```

```
  FOR i IN t.FIRST..t.LAST LOOP
      DBMS_OUTPUT.PUT(nvl(t(i), 0) || ' ');
  END LOOP;
  DBMS_OUTPUT.NEW_LINE;

-- punctul c
  t.DELETE(t.first);
  t.DELETE(5,7);
  t.DELETE(t.last);
  DBMS_OUTPUT.PUT_LINE('Primul element are indicele ' || t.first ||
         ' si valoarea ' || nvl(t(t.first),0));
  DBMS_OUTPUT.PUT_LINE('Ultimul element are indicele ' || t.last ||
         ' si valoarea ' || nvl(t(t.last),0));
  DBMS_OUTPUT.PUT('Tabloul are ' || t.COUNT ||' elemente: ');
  FOR i IN t.FIRST..t.LAST LOOP
     IF t.EXISTS(i) THEN
        DBMS_OUTPUT.PUT(nvl(t(i), 0)|| ' ');
     END IF;
  END LOOP;
  DBMS_OUTPUT.NEW_LINE;

-- punctul d
  t.delete;
  DBMS_OUTPUT.PUT_LINE('Tabloul are ' || t.COUNT ||' elemente.');
END;
/
```

5. Definiți un tablou indexat de înregistrări având tipul celor din tabelul *emp*_***. Ștergeți primele două linii din tabelul *emp*_***. Afișați elementele tabloului. Folosind tabelul indexat adăugați înapoi cele două linii șterse.

```
DECLARE
  TYPE tablou_indexat IS TABLE OF emp_***%ROWTYPE
                      INDEX BY BINARY_INTEGER;
  t    tablou_indexat;
BEGIN
-- stergere din tabel si salvare in tablou
   DELETE FROM emp_***
   WHERE  ROWNUM<= 2
   RETURNING employee_id, first_name, last_name, email, phone_number,
             hire_date, job_id, salary, commission_pct, manager_id,
             department_id
   BULK COLLECT INTO t;

--afisare elemente tablou
  DBMS_OUTPUT.PUT_LINE (t(1).employee_id ||' ' || t(1).last_name);
  DBMS_OUTPUT.PUT_LINE (t(2).employee_id ||' ' || t(2).last_name);

--inserare cele 2 linii in tabel
  INSERT INTO emp_*** VALUES t(1);
  INSERT INTO emp_*** VALUES t(2);
  END;
/
```

Tablouri imbricate (nested table)

Sintaxa generală pentru tabloul imbricat este:

```
[CREATE [OR REPLACE]] TYPE t_tablou_imbri IS TABLE OF
{ { tip_de_date | variabila%TYPE |
    tabel.coloana%TYPE }[NOT NULL]} | tabel%ROWTYPE };

v_tablou_imbri t_tablou_imbri;
```

- Singura diferență sintactică între tablourile indexate și cele imbricate este absența clauzei *INDEX BY*. Mai exact, dacă această clauză lipsește tipul de date declarat este tablou imbricat.
- Numărul maxim de linii al unui tablou imbricat este dat de capacitatea maximă 2 *GB.*
- Tablourile imbricate:
  - 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 ștergere;
  - metoda NEXT ne permite să ajungem la următorul element;
  - pentru a insera un element nou, tabloul trebuie extins cu metoda EXTEND.
- Un tablou imbricat este o mulțime neordonată de elemente de același tip. Valorile de acest tip:
  - pot fi stocate în baza de date;
  - pot fi prelucrate direct în instrucțiuni *SQL*;
  - au excepții predefinite proprii.
- Tablourile imbricate trebuie inițializate cu ajutorul constructorului.
  - *PL/SQL* apelează un constructor numai în mod explicit.
  - Tabelele indexate nu au constructori.
  - Constructorul primește 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, atunci când aceasta este inițializată.
  - Pentru vectori nu poate fi depășită dimensiunea maximă precizată la declarare.
  - Atunci când constructorul este apelat fără argumente se va crea o colecție fără niciun element (vidă), dar care este *not null*.

6. Rezolvați exercițiul 4 folosind tablouri imbricate.

```
DECLARE
  TYPE tablou_imbricat IS TABLE OF NUMBER;
  t    tablou_imbricat := tablou_imbricat();
BEGIN
-- punctul a
  FOR i IN 1..10 LOOP
     t.extend;
     t(i):=i;
  END LOOP;
  DBMS_OUTPUT.PUT('Tabloul are ' || t.COUNT ||' elemente: ');

  FOR i IN t.FIRST..t.LAST LOOP
      DBMS_OUTPUT.PUT(t(i) || ' ');
  END LOOP;
  DBMS_OUTPUT.NEW_LINE;
```

```
-- punctul b
  FOR i IN 1..10 LOOP
    IF i mod 2 = 1 THEN t(i):=null;
    END IF;
  END LOOP;
  DBMS_OUTPUT.PUT('Tabloul are ' || t.COUNT ||' elemente: ');
  FOR i IN t.FIRST..t.LAST LOOP
      DBMS_OUTPUT.PUT(nvl(t(i), 0) || ' ');
  END LOOP;
  DBMS_OUTPUT.NEW_LINE;

-- punctul c
  t.DELETE(t.first);
  t.DELETE(5,7);
  t.DELETE(t.last);
  DBMS_OUTPUT.PUT_LINE('Primul element are indicele ' || t.first ||
         ' si valoarea ' || nvl(t(t.first),0));
  DBMS_OUTPUT.PUT_LINE('Ultimul element are indicele ' || t.last ||
         ' si valoarea ' || nvl(t(t.last),0));
  DBMS_OUTPUT.PUT('Tabloul are ' || t.COUNT ||' elemente: ');
  FOR i IN t.FIRST..t.LAST LOOP
     IF t.EXISTS(i) THEN
        DBMS_OUTPUT.PUT(nvl(t(i), 0)|| ' ');
     END IF;
  END LOOP;
  DBMS_OUTPUT.NEW_LINE;

-- punctul d
  t.delete;
  DBMS_OUTPUT.PUT_LINE('Tabloul are ' || t.COUNT ||' elemente.');
END;
/
```

7. Declarați un tip tablou imbricat de caractere și o variabilă de acest tip. Inițializați variabila cu următoarele valori: m, i, n, i, m. Afișați conținutul tabloului, de la primul la ultimul element și invers. Ștergeți elementele 2 și 4 și apoi afișați conținutul tabloului.

```
DECLARE
  TYPE tablou_imbricat IS TABLE OF CHAR(1);
  t tablou_imbricat := tablou_imbricat('m', 'i', 'n', 'i', 'm');
  i INTEGER;
BEGIN
  i := t.FIRST;
  WHILE i <= t.LAST LOOP
    DBMS_OUTPUT.PUT(t(i));
    i := t.NEXT(i);
  END LOOP;
  DBMS_OUTPUT.NEW_LINE;

  i := t.LAST;
  WHILE i >= t.FIRST LOOP
    DBMS_OUTPUT.PUT(t(i));
    i := t.PRIOR(i);
```

```
  END LOOP;
  DBMS_OUTPUT.NEW_LINE;

  t.delete(2);
  t.delete(4);

  i := t.FIRST;
  WHILE i <= t.LAST LOOP
    DBMS_OUTPUT.PUT(t(i));
    i := t.NEXT(i);
  END LOOP;
  DBMS_OUTPUT.NEW_LINE;

  i := t.LAST;
  WHILE i >= t.FIRST LOOP
    DBMS_OUTPUT.PUT(t(i));
    i := t.PRIOR(i);
  END LOOP;
  DBMS_OUTPUT.NEW_LINE;

END;
/
```

Vectori

- Sintaxa generală pentru declararea vectorilor:

```
[CREATE [OR REPLACE]] TYPE t_vector IS VARRAY(limita) OF
   { { tip_de_date | variabila%TYPE |
       Tabel.coloana%TYPE }[NOT NULL]}
     | tabel%ROWTYPE };

v_vector t_vector;
```

- Spre deosebire de tablourile imbricate, 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 și ordinea elementelor este importantă.

8. Rezolvați exercițiul 4 folosind vectori.

```
DECLARE
  TYPE vector IS VARRAY(20) OF NUMBER;
  t    vector:= vector();
BEGIN
-- punctul a
  FOR i IN 1..10 LOOP
     t.extend; t(i):=i;
  END LOOP;
  DBMS_OUTPUT.PUT('Tabloul are ' || t.COUNT ||' elemente: ');
  FOR i IN t.FIRST..t.LAST LOOP
      DBMS_OUTPUT.PUT(t(i) || ' ');
  END LOOP;
  DBMS_OUTPUT.NEW_LINE;
-- punctul b
  FOR i IN 1..10 LOOP
    IF i mod 2 = 1 THEN t(i):=null;
```

```
    END IF;
  END LOOP;
  DBMS_OUTPUT.PUT('Tabloul are ' || t.COUNT ||' elemente: ');
  FOR i IN t.FIRST..t.LAST LOOP
      DBMS_OUTPUT.PUT(nvl(t(i), 0) || ' ');
  END LOOP;
  DBMS_OUTPUT.NEW_LINE;

-- punctul c
-- metodele DELETE(n), DELETE(m,n) nu sunt valabile pentru vectori!!!
-- din vectori nu se pot sterge elemente individuale!!!

-- punctul d
  t.delete;
  DBMS_OUTPUT.PUT_LINE('Tabloul are ' || t.COUNT ||' elemente.');
END;
/
```

9. Definiți tipul *subordonati_*** (vector, dimensiune maximă 10, menține numere). Creați tabelul *manageri_*** cu următoarele câmpuri: *cod_mgr* NUMBER(10), *nume* VARCHAR2(20), *lista* subordonati_***. Introduceți 3 linii în tabel. Afișați informațiile din tabel. Ștergeți tabelul creat, apoi tipul.

```
CREATE OR REPLACE TYPE subordonati_*** AS VARRAY(10) OF NUMBER(4);
/
CREATE TABLE manageri_*** (cod_mgr NUMBER(10),
                           nume VARCHAR2(20),
                           lista subordonati_***);

DECLARE
  v_sub   subordonati_***:= subordonati_***(100,200,300);
  v_lista manageri_***.lista%TYPE;
BEGIN
  INSERT INTO manageri_***
  VALUES (1, 'Mgr 1', v_sub);

  INSERT INTO manageri_***
  VALUES (2, 'Mgr 2', null);

  INSERT INTO manageri_***
  VALUES (3, 'Mgr 3', subordonati_***(400,500));

  SELECT lista
  INTO   v_lista
  FROM   manageri_***
  WHERE  cod_mgr=1;

  FOR j IN v_lista.FIRST..v_lista.LAST loop
       DBMS_OUTPUT.PUT_LINE (v_lista(j));
  END LOOP;
END;
/
SELECT * FROM manageri_***;
```

```
DROP TABLE  manageri_***;
DROP TYPE subordonati_***;
```

10. Creați tabelul *emp_test_**** cu coloanele *employee_id* și *last_name* din tabelul *employees*. Adăugați în acest tabel un nou câmp numit *telefon* de tip tablou imbricat. Acest tablou va menține pentru fiecare salariat toate numerele de telefon la care poate fi contactat. Inserați o linie nouă în tabel. Actualizați o linie din tabel. Afișați informațiile din tabel. Ștergeți tabelul și tipul.

```
CREATE TABLE emp_test_*** AS
      SELECT employee_id, last_name FROM employees
      WHERE ROWNUM <= 2;

CREATE OR REPLACE TYPE tip_telefon_*** IS TABLE OF VARCHAR(12);
/

ALTER TABLE emp_test_***
ADD (telefon tip_telefon_***)
NESTED TABLE telefon STORE AS tabel_telefon_***;

INSERT INTO emp_test_***
VALUES (500, 'XYZ',tip_telefon_***('074XXX', '0213XXX', '037XXX'));

UPDATE emp_test_***
SET    telefon = tip_telefon_***('073XXX', '0214XXX')
WHERE  employee_id=100;

SELECT  a.employee_id, b.*
FROM    emp_test_*** a, TABLE (a.telefon) b;

DROP TABLE emp_test_***;
DROP TYPE  tip_telefon_***;
```

11. Ștergeți din tabelul *emp_**** salariații având codurile menținute într-un vector.

Obs. Comanda *FORALL* permite ca toate liniile unei colecții să fie transferate simultan printr-o singură operație. Procedeul este numit ***bulk bind***.

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

Varianta 1

```
DECLARE
  TYPE tip_cod IS VARRAY(5) OF NUMBER(3);
  coduri tip_cod := tip_cod(205,206);
BEGIN
  FOR i IN coduri.FIRST..coduri.LAST  LOOP
    DELETE FROM emp_***
    WHERE  employee_id = coduri (i);
  END LOOP;
END;
/
SELECT employee_id FROM emp_***;
ROLLBACK;
```

Varianta 2

```
DECLARE
  TYPE tip_cod IS VARRAY(20) OF NUMBER;
  coduri tip_cod := tip_cod(205,206);
BEGIN
  FORALL i IN coduri.FIRST..coduri.LAST
    DELETE FROM emp_***
    WHERE  employee_id = coduri (i);
END;
/
SELECT employee_id FROM emp_***;
ROLLBACK;
```

Exerciții

1. Mențineți într-o colecție codurile celor mai prost plătiți 5 angajați care nu câștigă comision. Folosind această colecție măriți cu 5% salariul acestor angajați. Afișați valoarea veche a salariului, respectiv valoarea nouă a salariului.

2. Definiți un tip colecție denumit tip_orase_***. Creați tabelul *excursie_*** cu următoarea structură: *cod_excursie* NUMBER(4), *denumire* VARCHAR2(20), *orase* tip_orase_*** (ce va conține lista orașelor care se vizitează într-o excursie, într-o ordine stabilită; de exemplu, primul oraș din listă va fi primul oraș vizitat), *status* (disponibilă sau anulată).

a. Inserați 5 înregistrări în tabel.

b. Actualizați coloana *orase* pentru o excursie specificată:
- adăugați un oraș nou în listă, ce va fi ultimul vizitat în excursia respectivă;
- adăugați un oraș nou în listă, ce va fi al doilea oraș vizitat în excursia respectivă;
- inversați ordinea de vizitare a două dintre orașe al căror nume este specificat;
- eliminați din listă un oraș al cărui nume este specificat.

c. Pentru o excursie al cărui cod este dat, afișați numărul de orașe vizitate, respectiv numele orașelor.

d. Pentru fiecare excursie afișați lista orașelor vizitate.

e. Anulați excursiile cu cele mai puține orașe vizitate.

3. Rezolvați problema anterioară folosind un alt tip de colecție studiat.

Laborator 3 PL/SQL
Cursoare

Serverul Oracle alocă o zonă de memorie, care se numește **cursor**, ori de câte ori este înaintată o cerere SQL.

Cursoarele pot fi de două feluri:

- **implicite** (create și gestionate de PL/SQL în mod automat)
  Cursoarele implicite sunt declarate de PL/SQL în mod implicit pentru toate comenzile LMD și comanda SELECT, inclusiv comenzile care întorc o singură linie.
- **explicite** (create și gestionate de utilizator).
  Cursoarele explicite sunt create pentru cereri care întorc mai mult de o linie.

Mulțimea de linii procesate întoarse de o cerere multiple-row se numește **set activ**.

Un cursor este o modalitate de a parcurge setul activ linie cu linie.

Etapele utilizării unui cursor:

a) **Declarare**.
Definește numele și structura cursorului, împreună cu clauza SELECT care va popula cursorul cu date. Cererea este validată, dar nu executată.
În secțiunea declarativă prin intermediul cuvântului cheie CURSOR:

```
CURSOR c_nume_cursor [(parametru tip_de_date, ..)] IS
comanda SELECT;
```

b) **Deschidere** (comanda OPEN).
Este executată cererea și se populează cursorul cu date.

```
OPEN c_nume_cursor [(parametru, …)];
```

c) **Încărcare** (comanda FETCH).
Încarcă o linie din cursor (indicată de pointerul cursorului) în variabile și mută pointerul la linia următoare.
Numărul de variabile din clauza INTO trebuie să se potrivească cu lista SELECT din definiția cursorului.

```
FETCH c_nume_cursor INTO variabila1,…;
```

d) **Verificare**
Se verifică dacă s-a ajuns la finalul setului activ folosind atributul %NOTFOUND sau %FOUND.
`c_nume_cursor%NOTFOUND = TRUE`, dacă nicio linie nu a fost procesată
`C_nume_cursor%FOUND = TRUE`, dacă cel puțin o linia a fost procesată
Dacă nu s-a ajuns la final mergi la (c).

e) **Închidere** cursor (dacă rămâne deschis cursorul consumă din resursele serverului)

```
CLOSE c_nume_cursor;
```

Șterge datele din cursor și îl închide. Acesta poate fi redeschis pentru o actualizare a datelor.

Alte atribute utile:
- *%ROWCOUNT* reprezintă numărul de linii procesate;
- *%ISOPEN* este TRUE în cazul în care cursorul este deschis.

```
DECLARE
     declarare cursor
BEGIN
     deschidere cursor (OPEN)
     WHILE rămân linii de recuperat LOOP
          recuperare linie rezultat (FETCH)
          …
     END LOOP
     închidere cursor (CLOSE)
     …
END;
```

1. Obțineți pentru fiecare departament numele acestuia și numărul de angajați, într-una din următoarele forme:

" În departamentul <nume departament> nu lucrează angajati".
" În departamentul <nume departament> lucrează un angajat".
" În departamentul <nume departament> lucrează <numar> angajati".

Rezolvați problema folosind un **cursor explicit**.

TEMĂ: Rezolvați problema în SQL.

```
DECLARE
  v_nr    number(4);
  v_nume  departments.department_name%TYPE;
  CURSOR c IS
    SELECT department_name nume, COUNT(employee_id) nr
    FROM   departments d, employees e
    WHERE  d.department_id=e.department_id(+)
    GROUP BY department_name;
BEGIN
  OPEN c;
  LOOP
      FETCH c INTO v_nume,v_nr;
      EXIT WHEN c%NOTFOUND;
      IF v_nr=0 THEN
         DBMS_OUTPUT.PUT_LINE('In departamentul '|| v_nume||
                              ' nu lucreaza angajati');
      ELSIF v_nr=1 THEN
           DBMS_OUTPUT.PUT_LINE('In departamentul '|| v_nume||
                              ' lucreaza un angajat');
      ELSE
         DBMS_OUTPUT.PUT_LINE('In departamentul '|| v_nume||
                              ' lucreaza '|| v_nr||' angajati');
     END IF;
 END LOOP;
 CLOSE c;
END;
```

2. Rezolvați exercițiul 1 menținând informațiile din cursor în colecții. Comentați. Procesați toate liniile din cursor, încărcând la fiecare pas câte 5 linii.

Temă: Rezolvați problema folosind cursorul și o singură colecție.
Rezolvați problema folosind doar colecții.

```
DECLARE
  TYPE   tab_nume IS TABLE OF departments.department_name%TYPE;
  TYPE   tab_nr IS TABLE OF NUMBER(4);
  t_nr   tab_nr;
  t_nume tab_nume;
  CURSOR c IS
    SELECT department_name nume, COUNT(employee_id) nr
    FROM   departments d, employees e
    WHERE  d.department_id=e.department_id(+)
    GROUP BY department_name;
BEGIN
  OPEN c;
  FETCH c  BULK COLLECT INTO t_nume, t_nr;
  CLOSE c;
  FOR i IN t_nume.FIRST..t_nume.LAST LOOP
      IF t_nr(i)=0 THEN
         DBMS_OUTPUT.PUT_LINE('In departamentul '|| t_nume(i)||
                              ' nu lucreaza angajati');
      ELSIF t_nr(i)=1 THEN
           DBMS_OUTPUT.PUT_LINE('In departamentul '||t_nume(i)||
                              ' lucreaza un angajat');
      ELSE
         DBMS_OUTPUT.PUT_LINE('In departamentul '|| t_nume(i)||
                              ' lucreaza '|| t_nr(i)||' angajati');
     END IF;
  END LOOP;
END;
/
```

3. Rezolvați exercițiul 1 folosind un **ciclu cursor**.

```
DECLARE
  CURSOR c IS
    SELECT department_name nume, COUNT(employee_id) nr
    FROM   departments d, employees e
    WHERE  d.department_id=e.department_id(+)
    GROUP BY department_name;
BEGIN
  FOR i in c LOOP
      IF i.nr=0 THEN
         DBMS_OUTPUT.PUT_LINE('In departamentul '|| i.nume||
                              ' nu lucreaza angajati');
      ELSIF i.nr=1 THEN
           DBMS_OUTPUT.PUT_LINE('In departamentul '|| i.nume ||
                              ' lucreaza un angajat');
      ELSE
         DBMS_OUTPUT.PUT_LINE('In departamentul '|| i.nume||
                              ' lucreaza '|| i.nr||' angajati');
     END IF;
 END LOOP;
END;
/
```

4. Rezolvați exercițiul 1 folosind un **ciclu cursor cu subcereri**.

```
BEGIN
  FOR i in (SELECT department_name nume, COUNT(employee_id) nr
            FROM   departments d, employees e
            WHERE  d.department_id=e.department_id(+)
            GROUP BY department_name) LOOP
      IF i.nr=0 THEN
         DBMS_OUTPUT.PUT_LINE('In departamentul '|| i.nume||
                              ' nu lucreaza angajati');
      ELSIF i.nr=1 THEN
           DBMS_OUTPUT.PUT_LINE('In departamentul '|| i.nume ||
                              ' lucreaza un angajat');
      ELSE
         DBMS_OUTPUT.PUT_LINE('In departamentul '|| i.nume||
                              ' lucreaza '|| i.nr||' angajati');
     END IF;
 END LOOP;
END;
/
```

5. Obțineți primii 3 manageri care au cei mai mulți subordonați. Afișați numele managerului, respectiv numărul de angajați.

a. Rezolvați problema folosind un cursor **explicit**.

b. Modificați rezolvarea anterioară astfel încât să obțineți primii 4 manageri care îndeplinesc condiția. Observați rezultatul obținut și specificați dacă la punctul a s-a obținut top 3 manageri?

TEMĂ: Rezolvați problema în SQL.

```
DECLARE
  v_cod    employees.employee_id%TYPE;
  v_nume   employees.last_name%TYPE;
  v_nr     NUMBER(4);
  CURSOR c IS
    SELECT   sef.employee_id cod, MAX(sef.last_name) nume,
             count(*) nr
    FROM     employees sef, employees ang
    WHERE    ang.manager_id = sef.employee_id
    GROUP BY sef.employee_id
    ORDER BY nr DESC;
BEGIN
  OPEN c;
    LOOP
      FETCH c INTO v_cod,v_nume,v_nr;
      EXIT WHEN c%ROWCOUNT>3 OR c%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('Managerul '|| v_cod ||
                           ' avand numele ' || v_nume ||
                           ' conduce ' || v_nr||' angajati');
    END LOOP;
  CLOSE c;
END;
/
```

6. Rezolvați exercițiul 5 folosind un **ciclu cursor**.

```
DECLARE
  CURSOR c IS
    SELECT   sef.employee_id cod, MAX(sef.last_name) nume,
             count(*) nr
    FROM     employees sef, employees ang
    WHERE    ang.manager_id = sef.employee_id
    GROUP BY sef.employee_id
    ORDER BY nr DESC;
BEGIN
  FOR i IN c LOOP
      EXIT WHEN c%ROWCOUNT>3 OR c%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('Managerul '|| i.cod ||
                           ' avand numele ' || i.nume ||
                           ' conduce '|| i.nr||' angajati');
  END LOOP;
END;
/
```

7. Rezolvați exercițiul 5 folosind un **ciclu cursor cu subcereri**.

```
DECLARE
  top number(1):= 0;
BEGIN
  FOR i IN (SELECT   sef.employee_id cod, MAX(sef.last_name) nume,
                     count(*) nr
            FROM     employees sef, employees ang
            WHERE    ang.manager_id = sef.employee_id
            GROUP BY sef.employee_id
            ORDER BY nr DESC)
  LOOP
      DBMS_OUTPUT.PUT_LINE('Managerul '|| i.cod ||
                           ' avand numele ' || i.nume ||
                           ' conduce '|| i.nr||' angajati');
      Top := top+1;
      EXIT WHEN top=3;
  END LOOP;
END;
/
```

8. Modificați exercițiul 1 astfel încât să obțineți doar departamentele în care lucrează cel puțin *x* angajați, unde *x* reprezintă un număr introdus de la tastatură. Rezolvați problema folosind toate cele trei tipuri de cursoare studiate.

```
DECLARE
  v_x     number(4) := &p_x;
  v_nr    number(4);
  v_nume  departments.department_name%TYPE;
```

```
  CURSOR c (paramentru NUMBER) IS
    SELECT department_name nume, COUNT(employee_id) nr
    FROM   departments d, employees e
    WHERE  d.department_id=e.department_id
    GROUP BY department_name
    HAVING COUNT(employee_id)> paramentru;
BEGIN
  OPEN c(v_x);
  LOOP
      FETCH c INTO v_nume,v_nr;
      EXIT WHEN c%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('In departamentul '|| v_nume||
                           ' lucreaza '|| v_nr||' angajati');
 END LOOP;
 CLOSE c;
END;
/
```

```
DECLARE
 v_x     number(4) := &p_x;
 CURSOR c (paramentru NUMBER) IS
    SELECT department_name nume, COUNT(employee_id) nr
    FROM   departments d, employees e
    WHERE  d.department_id=e.department_id
    GROUP BY department_name
    HAVING COUNT(employee_id)> paramentru;
BEGIN
  FOR i in c(v_x) LOOP
     DBMS_OUTPUT.PUT_LINE('In departamentul '|| i.nume||
                          ' lucreaza '|| i.nr||' angajati');
  END LOOP;
END;
/
```

```
DECLARE
 v_x     number(4) := &p_x;
 BEGIN
  FOR i in (SELECT department_name nume, COUNT(employee_id) nr
            FROM   departments d, employees e
            WHERE  d.department_id=e.department_id
            GROUP BY department_name
            HAVING COUNT(employee_id)> v_x)
  LOOP
     DBMS_OUTPUT.PUT_LINE('In departamentul '|| i.nume||
                          ' lucreaza '|| i.nr||' angajati');
END LOOP;
END;
/
```

9. Măriți cu 1000 salariile celor care au fost angajați în 2000 (din tabelul emp_***) blocând liniile înainte de actualizare (**cursor SELECT FOR UPDATE)**.

Observație: Uneori este necesară blocarea liniilor înainte ca acestea să fie șterse sau reactualizate. Blocarea se poate realiza (atunci când cursorul este deschis) cu ajutorul comenzii *SELECT* care conține clauza *FOR UPDATE*.

Comanda SELECT are următoarea extensie PL/SQL pentru blocarea explicita a înregistrărilor ce urmează a fi prelucrate (modificate sau șterse):

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

În cazul în care liniile selectate de cerere nu pot fi blocate din cauza altor blocări atunci:
- dacă se folosește NOWAIT apare imediat eroarea ORA-00054;
- dacă nu se folosește NOWAIT atunci se așteaptă până când liniile sunt deblocate;
- dacă se folosește WAIT n, atunci se așteaptă un număr determinat de secunde pentru ca liniile ce trebuie selectate pentru modificare să fie deblocate.

Pentru a modifica o anumită linie întoarsă de un astfel de cursor se folosește clauza:

```
WHERE CURRENT OF nume_cursor
```

Această clauză apare la finalul unei comenzi UPDATE și face referință la un cursor care este deschis și pentru care s-a realizat cel puțin o încărcare (FETCH).

```
SELECT last_name, hire_date, salary
FROM   emp_***
WHERE  TO_CHAR(hire_date, 'yyyy') = 2000;
```

```
DECLARE
  CURSOR c IS
    SELECT *
    FROM   emp_***
    WHERE  TO_CHAR(hire_date, 'YYYY') = 2000
    FOR UPDATE OF salary NOWAIT;
BEGIN
  FOR i IN c  LOOP
    UPDATE  emp_***
    SET     salary= salary+1000
    WHERE CURRENT OF c;
  END LOOP;
END;
/
```

```
SELECT last_name, hire_date, salary
FROM   emp_***
WHERE  TO_CHAR(hire_date, 'yyyy') = 2000;

ROLLBACK;
```

10. Pentru fiecare dintre departamentele 10, 20, 30 și 40, obțineți numele precum și lista numelor angajaților care își desfășoară activitatea în cadrul acestora. Rezolvați problema folosind:
   a. cele trei tipuri de cursoare studiate;
   b. **expresii cursor**.

Observație: În *Oracle9i* a fost introdus conceptul de expresie cursor care întoarce un cursor imbricat (*nested cursor*).

Varianta 1.1 – cursor clasic **Temă**

Varianta 1.2 – ciclu cursor **Temă**

Varianta 1.3 – ciclu cursor cu subcereri

```
BEGIN
  FOR v_dept IN (SELECT department_id, department_name
                 FROM   departments
                 WHERE department_id IN (10,20,30,40))
  LOOP
    DBMS_OUTPUT.PUT_LINE('-------------------------------------');
    DBMS_OUTPUT.PUT_LINE ('DEPARTAMENT '||v_dept.department_name);
    DBMS_OUTPUT.PUT_LINE('-------------------------------------');
    FOR v_emp IN (SELECT last_name
                  FROM    employees
                  WHERE  department_id = v_dept.department_id)
    LOOP
       DBMS_OUTPUT.PUT_LINE (v_emp.last_name);
    END LOOP;
  END LOOP;
END;
/
```

Varianta 2 – expresii cursor

```
DECLARE
  TYPE refcursor IS REF CURSOR;
  CURSOR c_dept IS
    SELECT department_name,
           CURSOR (SELECT last_name
                   FROM   employees e
                   WHERE  e.department_id = d.department_id)
    FROM   departments d
    WHERE  department_id IN (10,20,30,40);
  v_nume_dept   departments.department_name%TYPE;
  v_cursor      refcursor;
  v_nume_emp    employees.last_name%TYPE;
BEGIN
  OPEN c_dept;
  LOOP
    FETCH c_dept INTO v_nume_dept, v_cursor;
    EXIT WHEN c_dept%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('-------------------------------------');
    DBMS_OUTPUT.PUT_LINE ('DEPARTAMENT '||v_nume_dept);
    DBMS_OUTPUT.PUT_LINE('-------------------------------------');
    LOOP
      FETCH v_cursor INTO v_nume_emp;
      EXIT WHEN v_cursor%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE (v_nume_emp);
    END LOOP;
  END LOOP;
  CLOSE c_dept;
END;
/
```

11. Declarați un **cursor dinamic** care întoarce linii de tipul celor din tabelul emp_***. În funcție de o opțiune introdusă de la tastatură (una dintre valorile 1, 2 sau 3) deschideți cursorul astfel încât să regăsească:
- toate informațiile din tabelul emp_*** (pentru opțiunea 1);
- doar angajații având salariul cuprins între 10000 și 20000 (pentru opțiunea 2);
- doar salariații angajați în anul 2000 (pentru opțiunea 3).

Verificați ce se întâmplă în cazul în care introduceți o valoare diferită de 1, 2 sau 3. Modificați corespunzător.

```
DECLARE
  TYPE      emp_tip IS REF CURSOR RETURN employees%ROWTYPE;
  -- sau
  -- TYPE   emp_tip IS REF CURSOR;

  v_emp     emp_tip;
  v_optiune NUMBER := &p_optiune;
  v_ang    employees%ROWTYPE;
BEGIN
   IF v_optiune = 1 THEN
     OPEN v_emp FOR SELECT *
                    FROM employees;
   ELSIF v_optiune = 2 THEN
     OPEN v_emp FOR  SELECT *
                     FROM employees
                     WHERE salary BETWEEN 10000 AND 20000;
   ELSIF v_optiune = 3 THEN
     OPEN v_emp FOR SELECT *
                    FROM employees
                    WHERE TO_CHAR(hire_date, 'YYYY') = 2000;
   ELSE
      DBMS_OUTPUT.PUT_LINE('Optiune incorecta');
   END IF;

   LOOP
      FETCH v_emp into v_ang;
      EXIT WHEN v_emp%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE(v_ang.last_name);
   END LOOP;

   DBMS_OUTPUT.PUT_LINE('Au fost procesate '||v_emp%ROWCOUNT
                        || ' linii');
   CLOSE v_emp;
END;
/
```

12. Citiți de la tastatură o valoare *n*. Prin intermediul unui **cursor deschis cu ajutorul unui șir dinamic** obțineți angajații având salariul mai mare decât *n*. Pentru fiecare linie regăsită de cursor afișați următoarele informații:
- numele și salariul dacă angajatul nu are comision;
- numele, salariul și comisionul dacă angajatul are comision.

```
DECLARE
  TYPE  empref IS REF CURSOR;
  v_emp empref;
  v_nr  INTEGER := &n;
BEGIN
  OPEN v_emp FOR
    'SELECT employee_id, salary, commission_pct ' ||
    'FROM employees WHERE salary > :bind_var'
     USING v_nr;
 -- introduceti liniile corespunzatoare rezolvarii problemei
END;
/
```

EXERCIȚII

1. Pentru fiecare job (titlu – care va fi afișat o singură dată) obțineți lista angajaților (nume și salariu) care lucrează în prezent pe jobul respectiv. Tratați cazul în care nu există angajați care să lucreze în prezent pe un anumit job. Rezolvați problema folosind:
   a. cursoare clasice
   b. ciclu cursoare
   c. ciclu cursoare cu subcereri
   d. expresii cursor

2. Modificați exercițiul anterior astfel încât să obțineți și următoarele informații:
   - un număr de ordine pentru fiecare angajat care va fi resetat pentru fiecare job
   - pentru fiecare job
     - numărul de angajați
     - valoarea lunară a veniturilor angajaților
     - valoarea medie a veniturilor angajaților
   - indiferent job
     - numărul total de angajați
     - valoarea totală lunară a veniturilor angajaților
     - valoarea medie a veniturilor angajaților

3. Modificați exercițiul anterior astfel încât să obțineți suma totală alocată lunar pentru plata salariilor și a comisioanelor tuturor angajaților, iar pentru fiecare angajat cât la sută din această sumă câștigă lunar.

4. Modificați exercițiul anterior astfel încât să obțineți pentru fiecare job primii 5 angajați care câștigă cel mai mare salariu lunar. Specificați dacă pentru un job sunt mai puțin de 5 angajați.

5. Modificați exercițiul anterior astfel încât să obțineți pentru fiecare job top 5 angajați. Dacă există mai mulți angajați care respectă criteriul de selecție care au același salariu, atunci aceștia vor ocupa aceeași poziție în top 5.

Laborator 4 PL/SQL
Subprograme (proceduri, funcții)

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

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

Exista 2 tipuri de subprograme:
- proceduri;
- funcții (trebuie să întoarcă o valoare).

Acestea pot fi locale (declarate într-un bloc PL/SQL) sau stocate (independente sau împachetate). Procedurile și funcțiile independente sunt stocate în baza de date și de aceea ele se numesc subprograme stocate.

- 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];
```

- 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:

```
nume_parametru mod_parametru tip_de_date;
```

Mod_parametru poate fi:
- de intrare (IN) – valoare implicită; poate avea o valoare inițială;
- de intrare / ieșire (IN OUT);
- de ieșire (OUT).

- În cazul în care se modifică un obiect (vizualizare, tabel etc) de care depinde un subprogram, acesta este invalidat. Revalidarea se face ori prin recrearea subprogramului ori printr-o comanda ALTER … COMPILE:

```
ALTER PROCEDURE nume_procedura COMPILE;
ALTER FUNCTION  nume_functie   COMPILE;
```

- Ștergerea unei funcții sau proceduri se realizează prin comenzile:

```
DROP PROCEDURE nume_procedura;
DROP FUNCTION  nume_functie;
```

- Informații despre procedurile și funcțiile deținute de utilizatorul curent se pot obține interogând vizualizarea USER_OBJECTS.

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

- Codul complet al unui subprogram poate fi vizualizat folosind următoarea sintaxă:

```
SELECT    TEXT
FROM      USER_SOURCE
WHERE     NAME =UPPER('nume_subprogram');
```

Tipul unui subprogram se obține prin comanda *DESCRIBE*.

- Eroarea apărută la compilarea unui subprogram poate fi vizualizată folosind următoarea sintaxă:

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

- Erorile pot fi vizualizate și prin intermediul comenzii *SHOW ERRORS*.

1. Definiți un subprogram prin care să obțineți salariul unui angajat al cărui nume este specificat. Tratați toate excepțiile ce pot fi generate.
Apelați subprogramul pentru următorii angajați: Bell, King, Kimball.
Rezolvați problema folosind o **funcție locală**.

```
DECLARE
  v_nume employees.last_name%TYPE := Initcap('&p_nume');

  FUNCTION f1 RETURN NUMBER IS
    salariu employees.salary%type;
  BEGIN
    SELECT salary INTO salariu
    FROM   employees
    WHERE  last_name = v_nume;
    RETURN salariu;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
       DBMS_OUTPUT.PUT_LINE('Nu exista angajati cu numele dat');
    WHEN TOO_MANY_ROWS THEN
       DBMS_OUTPUT.PUT_LINE('Exista mai multi angajati '||
                            'cu numele dat');
    WHEN OTHERS THEN
       DBMS_OUTPUT.PUT_LINE('Alta eroare!');
  END f1;
```

```
BEGIN
  DBMS_OUTPUT.PUT_LINE('Salariul este '|| f1);

--EXCEPTION
--  WHEN OTHERS THEN
--    DBMS_OUTPUT.PUT_LINE('Eroarea are codul = '||SQLCODE
--            || ' si mesajul = ' || SQLERRM);
END;
/
```

2. Rezolvați exercițiul 1 folosind o **funcție stocată**.

```
CREATE OR REPLACE FUNCTION f2_***
  (v_nume employees.last_name%TYPE DEFAULT 'Bell')
RETURN NUMBER IS
    salariu employees.salary%type;
  BEGIN
    SELECT salary INTO salariu
    FROM   employees
    WHERE  last_name = v_nume;
    RETURN salariu;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
       RAISE_APPLICATION_ERROR(-20000,
                'Nu exista angajati cu numele dat');
    WHEN TOO_MANY_ROWS THEN
       RAISE_APPLICATION_ERROR(-20001,
              'Exista mai multi angajati cu numele dat');
    WHEN OTHERS THEN
       RAISE_APPLICATION_ERROR(-20002,'Alta eroare!');
END f2_***;
/
```

```
-- metode de apelare
-- 1. bloc plsql
BEGIN
  DBMS_OUTPUT.PUT_LINE('Salariul este '|| f2_***);
END;
/

BEGIN
  DBMS_OUTPUT.PUT_LINE('Salariul este '|| f2_***('King'));
END;
/
```

```
-- 2. SQL
  SELECT f2_*** FROM DUAL;
  SELECT f2_***('King') FROM DUAL;
```

```
-- 3. SQL*PLUS CU VARIABILA HOST
  VARIABLE nr NUMBER
  EXECUTE :nr := f2_***('Bell');
  PRINT nr
```

3. Rezolvați exercițiul 1 folosind o **procedură locală**.

```
-- varianta 1
DECLARE
  v_nume employees.last_name%TYPE := Initcap('&p_nume');

  PROCEDURE p3
  IS
      salariu employees.salary%TYPE;
  BEGIN
    SELECT salary INTO salariu
    FROM   employees
    WHERE  last_name = v_nume;
    DBMS_OUTPUT.PUT_LINE('Salariul este '|| salariu);

  EXCEPTION
    WHEN NO_DATA_FOUND THEN
       DBMS_OUTPUT.PUT_LINE('Nu exista angajati cu numele dat');
    WHEN TOO_MANY_ROWS THEN
       DBMS_OUTPUT.PUT_LINE('Exista mai multi angajati '||
                            'cu numele dat');
    WHEN OTHERS THEN
       DBMS_OUTPUT.PUT_LINE('Alta eroare!');
  END p3;

BEGIN
  p3;
END;
/
```

```
-- varianta 2
DECLARE
  v_nume employees.last_name%TYPE := Initcap('&p_nume');
  v_salariu employees.salary%type;

  PROCEDURE p3(salariu OUT employees.salary%type) IS
  BEGIN
    SELECT salary INTO salariu
    FROM   employees
    WHERE  last_name = v_nume;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
       RAISE_APPLICATION_ERROR(-20000,
                'Nu exista angajati cu numele dat');
    WHEN TOO_MANY_ROWS THEN
       RAISE_APPLICATION_ERROR(-20001,
              'Exista mai multi angajati cu numele dat');
```

```
    WHEN OTHERS THEN
       RAISE_APPLICATION_ERROR(-20002,'Alta eroare!');
  END p3;

BEGIN
  p3(v_salariu);
  DBMS_OUTPUT.PUT_LINE('Salariul este '|| v_salariu);
END;
/
```

4. Rezolvați exercițiul 1 folosind o **procedură stocată**.

```
-- varianta 1
CREATE OR REPLACE PROCEDURE p4_***
      (v_nume employees.last_name%TYPE)
  IS
      salariu employees.salary%TYPE;
  BEGIN
    SELECT salary INTO salariu
    FROM   employees
    WHERE  last_name = v_nume;
    DBMS_OUTPUT.PUT_LINE('Salariul este '|| salariu);

  EXCEPTION
    WHEN NO_DATA_FOUND THEN
       RAISE_APPLICATION_ERROR(-20000,
                'Nu exista angajati cu numele dat');
    WHEN TOO_MANY_ROWS THEN
       RAISE_APPLICATION_ERROR(-20001,
              'Exista mai multi angajati cu numele dat');
    WHEN OTHERS THEN
       RAISE_APPLICATION_ERROR(-20002,'Alta eroare!');
  END p4_***;
/

-- metode apelare
-- 1. Bloc PLSQL
BEGIN
  p4_***('Bell');
END;
/

-- 2. SQL*PLUS
EXECUTE p4_***('Bell');
EXECUTE p4_***('King');
EXECUTE p4_***('Kimball');
```

```
-- varianta 2
CREATE OR REPLACE PROCEDURE
       p4_***(v_nume IN employees.last_name%TYPE,
              salariu OUT employees.salary%type) IS
  BEGIN
    SELECT salary INTO salariu
    FROM   employees
    WHERE  last_name = v_nume;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
       RAISE_APPLICATION_ERROR(-20000,
                'Nu exista angajati cu numele dat');
    WHEN TOO_MANY_ROWS THEN
       RAISE_APPLICATION_ERROR(-20001,
              'Exista mai multi angajati cu numele dat');
    WHEN OTHERS THEN
       RAISE_APPLICATION_ERROR(-20002,'Alta eroare!');
  END p4_***;
/

-- metode apelare
-- 1. Bloc PLSQL
DECLARE
   v_salariu employees.salary%type;
BEGIN
  p4_***('Bell',v_salariu);
  DBMS_OUTPUT.PUT_LINE('Salariul este '|| v_salariu);
END;
/

-- 2. SQL*PLUS
VARIABLE v_sal NUMBER
EXECUTE p4_*** ('Bell',:v_sal)
PRINT v_sal
```

5. Creați o procedură stocată care primește printr-un parametru codul unui angajat și returnează prin intermediul aceluiași parametru codul managerului corespunzător acelui angajat (**parametru de tip IN OUT**).

```
VARIABLE ang_man NUMBER
BEGIN
 :ang_man:=200;
END;
/
```

```
CREATE OR REPLACE PROCEDURE p5_***  (nr IN OUT NUMBER) IS
BEGIN
 SELECT manager_id INTO nr
 FROM employees
 WHERE employee_id=nr;
END p5_***;
/
```

```
EXECUTE p5_*** (:ang_man)
PRINT ang_man
```

6. Declarați o procedură locală care are parametrii:
- rezultat (parametru de tip OUT) de tip last_name din employees;
- comision (parametru de tip IN) de tip commission_pct din employees, inițializat cu NULL;
- cod (parametru de tip IN) de tip employee_id din employees, inițializat cu NULL.

Dacă comisionul nu este NULL atunci în rezultat se va memora numele salariatului care are comisionul respectiv. În caz contrar, în rezultat se va memora numele salariatului al cărui cod are valoarea dată în apelarea procedurii.

```
DECLARE
nume employees.last_name%TYPE;
PROCEDURE p6 (rezultat OUT employees.last_name%TYPE,
              comision IN  employees.commission_pct%TYPE:=NULL,
              cod      IN  employees.employee_id%TYPE:=NULL)
 IS
 BEGIN
 IF (comision IS NOT NULL) THEN
    SELECT last_name
    INTO rezultat
    FROM employees
    WHERE commission_pct= comision;
    DBMS_OUTPUT.PUT_LINE('numele salariatului care are
                comisionul '||comision||' este '||rezultat);
 ELSE
    SELECT last_name
    INTO rezultat
    FROM employees
    WHERE employee_id =cod;
    DBMS_OUTPUT.PUT_LINE('numele salariatului avand codul '||
                          cod ||' este '||rezultat);
 END IF;
END p6;

BEGIN
  p6(nume,0.4);
  p6(nume,cod=>200);
END;
/
```

7. Definiți două funcții locale cu același nume (**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 și 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_MAN');
  DBMS_OUTPUT.PUT_LINE('Media salariilor managerilor din'
      || ' departamentul 80 este ' || medie2);
END;
/
```

8. Calculați recursiv factorialul unui număr dat (**recursivitate**).

```
CREATE OR REPLACE FUNCTION factorial_***(n NUMBER)
 RETURN INTEGER IS
 BEGIN
  IF (n=0) THEN RETURN 1;
  ELSE RETURN n*factorial_***(n-1);
  END IF;
END factorial_***;
/
```

9. Afișați numele și salariul angajaților al căror salariu este mai mare decât media tuturor salariilor. Media salariilor va fi obținută prin apelarea unei funcții stocate.

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

Exerciții

1. Creați tabelul *info_*** cu următoarele coloane:
- utilizator (numele utilizatorului care a inițiat o comandă)
- data (data și timpul la care utilizatorul a inițiat comanda)
- comanda (comanda care a fost inițiată de utilizatorul respectiv)
- nr_linii (numărul de linii selectate/modificate de comandă)
- eroare (un mesaj pentru excepții).

2. Modificați funcția definită la exercițiul 2, respectiv procedura definită la exercițiul 4 astfel încât să determine inserarea în tabelul *info_*** a informațiile corespunzătoare fiecărui caz determinat de valoarea dată pentru parametru:
- există un singur angajat cu numele specificat;
- există mai mulți angajați cu numele specificat;
- nu există angajați cu numele specificat.

3. Definiți o funcție stocată care determină numărul de angajați care au avut cel puțin 2 joburi diferite și care în prezent lucrează într-un oraș dat ca parametru. Tratați cazul în care orașul dat ca parametru nu există, respectiv cazul în care în orașul dat nu lucrează niciun angajat. Inserați în tabelul *info_*** informațiile corespunzătoare fiecărui caz determinat de valoarea dată pentru parametru.

4. Definiți o procedură stocată care mărește cu 10% salariile tuturor angajaților conduși direct sau indirect de către un manager al cărui cod este dat ca parametru. Tratați cazul în care nu există niciun manager cu codul dat. Inserați în tabelul *info_*** informațiile corespunzătoare fiecărui caz determinat de valoarea dată pentru parametru.

5. Definiți un subprogram care obține pentru fiecare nume de departament ziua din săptămână în care au fost angajate cele mai multe persoane, lista cu numele acestora, vechimea și venitul lor lunar. Afișați mesaje corespunzătoare următoarelor cazuri:
- într-un departament nu lucrează niciun angajat;
- într-o zi din săptămână nu a fost nimeni angajat.

Observații:
- **a.** Numele departamentului și ziua apar o singură dată în rezultat.
- **b.** Rezolvați problema în două variante, după cum se ține cont sau nu de istoricul joburilor angajaților.

6. Modificați exercițiul anterior astfel încât lista cu numele angajaților să apară într-un clasament creat în funcție de vechimea acestora în departament. Specificați numărul poziției din clasament și apoi lista angajaților care ocupă acel loc. Dacă doi angajați au aceeași vechime, atunci aceștia ocupă aceeași poziție în clasament.

Laborator 5 PL/SQL
Pachete

- Pachetele sunt unităţi de program care pot cuprinde proceduri, funcţii, cursoare, tipuri de date, constante, variabile şi excepţii.
- Pachetele nu pot fi apelate, nu pot transmite parametri şi nu pot fi încuibărite.
- Un pachet are două părţi, fiecare fiind stocată separat în dicţionarul datelor:
  - **specificaţia pachetului**;

```
CREATE [OR REPLACE] PACKAGE [schema.]nume_pachet
{IS | AS}
   declaraţii;
END [nume_pachet];
```

  - **corpul pachetului**.

```
CREATE [OR REPLACE] PACKAGE BODY [schema.]nume_pachet
 {IS | AS}
[BEGIN]
 instrucţiuni;
END [nume_pachet];
```

- Recompilarea pachetului

```
ALTER PACKAGE [schema.]nume_pachet
COMPILE [ {PACKAGE | BODY} ];
```

- Eliminarea pachetului

```
DROP PACKAGE [schema.]nume_pachet
[ {PACKAGE | BODY} ];
```

I. Pachete definite de utilizator

1. Definiţi un pachet care permite prin intermediul a două funcţii calculul numărului de angajaţi şi suma ce trebuie alocată pentru plata salariilor şi a comisioanelor pentru un departament al cărui cod este dat ca parametru.

```
CREATE OR REPLACE PACKAGE pachet1_*** AS
   FUNCTION  f_numar(v_dept departments.department_id%TYPE)
        RETURN NUMBER;
   FUNCTION  f_suma(v_dept departments.department_id%TYPE)
        RETURN NUMBER;
END pachet1_***;
/
CREATE OR REPLACE PACKAGE BODY pachet1_*** AS
   FUNCTION  f_numar(v_dept  departments.department_id%TYPE)
      RETURN NUMBER IS numar NUMBER;
   BEGIN
      SELECT COUNT(*)INTO numar
      FROM   employees
      WHERE  department_id =v_dept;
   RETURN numar;
   END f_numar;
```

```
   FUNCTION  f_suma (v_dept  departments.department_id%TYPE)
      RETURN NUMBER IS
      suma NUMBER;
   BEGIN
      SELECT SUM(salary+salary*NVL(commission_pct,0))
      INTO suma
      FROM employees
      WHERE department_id =v_dept;
   RETURN suma;
   END f_suma;
END pachet1_***;
/
```

Apelare:

În SQL:

```
SELECT pachet1_***.f_numar(80)
FROM DUAL;
SELECT pachet1_***.f_suma(80)
FROM DUAL;
```

În PL/SQL:

```
BEGIN
  DBMS_OUTPUT.PUT_LINE('numarul de salariati este '||
                       pachet1_***.f_numar(80));
  DBMS_OUTPUT.PUT_LINE('suma alocata este '||
                       pachet1_***.f_suma(80));
END;
/
```

2. Creaţi un pachet ce include acţiuni pentru adăugarea unui nou departament în tabelul *dept_*** şi a unui nou angajat (ce va lucra în acest departament) în tabelul *emp_****. Procedurile pachetului vor fi apelate din SQL, respectiv din PL/SQL. Se va verifica dacă managerul departamentului există înregistrat ca salariat. De asemenea, se va verifica dacă locaţia departamentului există. Pentru inserarea codului salariatului se va utiliza o secvenţă.

```
CREATE OR REPLACE PACKAGE pachet2_*** AS
   PROCEDURE p_dept (v_codd dept_***.department_id%TYPE,
                     v_nume dept_***.department_name%TYPE,
                     v_manager dept_***.manager_id%TYPE,
                     v_loc dept_***.location_id%TYPE);
   PROCEDURE p_emp (v_first_name emp_***.first_name%TYPE,
                 v_last_name emp_***.last_name%TYPE,
                 v_email emp_***.email%TYPE,
                 v_phone_number emp_***.phone_number%TYPE:=NULL,
                 v_hire_date emp_***.hire_date%TYPE :=SYSDATE,
                 v_job_id emp_***.job_id%TYPE,
                 v_salary   emp_***.salary%TYPE :=0,
                 v_commission_pct emp_***.commission_pct%TYPE:=0,
                 v_manager_id emp_***.manager_id%TYPE,
                 v_department_id emp_***.department_id%TYPE);
```

```
  FUNCTION exista (cod_loc dept_***.location_id%TYPE,
                   manager dept_***.manager_id%TYPE)
  RETURN NUMBER;
END pachet2_***;
/

CREATE OR REPLACE PACKAGE BODY pachet2_*** AS

FUNCTION exista(cod_loc dept_***.location_id%TYPE,
                manager dept_***.manager_id%TYPE)
 RETURN NUMBER  IS
      rezultat NUMBER:=1;
      rez_cod_loc NUMBER;
      rez_manager NUMBER;
 BEGIN
    SELECT count(*) INTO   rez_cod_loc
    FROM   locations
    WHERE  location_id = cod_loc;

    SELECT count(*) INTO   rez_manager
    FROM   emp_***
    WHERE  employee_id = manager;

    IF rez_cod_loc=0 OR rez_manager=0 THEN
         rezultat:=0;
    END IF;
RETURN rezultat;
END;

PROCEDURE p_dept(v_codd dept_***.department_id%TYPE,
                 v_nume dept_***.department_name%TYPE,
                 v_manager dept_***.manager_id%TYPE,
                 v_loc dept_***. location_id%TYPE) IS
BEGIN
   IF exista(v_loc, v_manager)=0 THEN
       DBMS_OUTPUT.PUT_LINE('Nu s-au introdus date coerente pentru
            tabelul dept_***');
   ELSE
     INSERT INTO dept_***
          (department_id,department_name,manager_id,location_id)
     VALUES (v_codd, v_nume, v_manager, v_loc);
   END IF;
 END p_dept;

PROCEDURE p_emp
     (v_first_name emp_***.first_name%TYPE,
      v_last_name emp_***.last_name%TYPE,
      v_email emp_***.email%TYPE,
      v_phone_number emp_***.phone_number%TYPE:=null,
      v_hire_date emp_***.hire_date%TYPE :=SYSDATE,
      v_job_id emp_***.job_id%TYPE,
      v_salary emp_***.salary %TYPE :=0,
```

```
      v_commission_pct emp_***.commission_pct%TYPE:=0,
      v_manager_id emp_***.manager_id%TYPE,
      v_department_id  emp_***.department_id%TYPE)
AS
 BEGIN
     INSERT INTO emp_***
     VALUES (sec_***.NEXTVAL, v_first_name, v_last_name, v_email,
            v_phone_number,v_hire_date, v_job_id, v_salary,
            v_commission_pct, v_manager_id,v_department_id);
END p_emp;
END pachet2_***;
/
```

Apelare:

În SQL:

```
EXECUTE pachet2_***.p_dept(50,'Economic',200,2000);

SELECT * FROM dept_*** WHERE department_id=50;

EXECUTE pachet2_***.p_emp('f','l','e',v_job_id=>'j',
                          v_manager_id=>200,v_department_id=>50);

SELECT * FROM emp_*** WHERE job_id='j';

ROLLBACK;
```

În PL/SQL:

```
BEGIN
   pachet2_***.p_dept(50,'Economic',99,2000);
   pachet2_***.p_emp('f','l','e',v_job_id=>'j',v_manager_id=>200,
                     v_department_id=>50);
END;
/

SELECT * FROM emp_*** WHERE job_id='j';
ROLLBACK;
```

3. Definiţi un pachet cu ajutorul căruia să se obţină salariul maxim înregistrat pentru salariaţii care lucrează într-un anumit oraş şi lista salariaţilor care au salariul mai mare sau egal decât acel maxim. Pachetul va conţine un cursor şi un subprogram funcţie.

```
CREATE  OR REPLACE PACKAGE pachet3_*** AS
   CURSOR c_emp(nr NUMBER) RETURN employees%ROWTYPE;
   FUNCTION  f_max  (v_oras  locations.city%TYPE) RETURN NUMBER;
END pachet3_***;
/
```

```
CREATE OR REPLACE PACKAGE BODY pachet3_*** AS

CURSOR c_emp(nr NUMBER) RETURN employees%ROWTYPE
      IS
      SELECT *
      FROM employees
      WHERE salary >= nr;

FUNCTION  f_max (v_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(v_oras);
    RETURN  maxim;
END f_max;
END pachet3_***;
/

DECLARE
  oras    locations.city%TYPE:= 'Toronto';
  val_max NUMBER;
  lista   employees%ROWTYPE;
BEGIN
   val_max:=  pachet3_***.f_max(oras);
   FOR v_cursor IN pachet3_***.c_emp(val_max) LOOP
      DBMS_OUTPUT.PUT_LINE(v_cursor.last_name||' '||
                           v_cursor.salary);
   END LOOP;
END;
/
```

4. Definiţi un pachet care să conţină o procedură prin care se verifică dacă o combinaţie specificată dintre câmpurile *employee_id* şi *job_id* este o combinaţie care există în tabelul *employees*.

```
CREATE OR REPLACE  PACKAGE pachet4_*** IS
  PROCEDURE p_verific
      (v_cod employees.employee_id%TYPE,
       v_job   employees.job_id%TYPE);
  CURSOR c_emp RETURN employees%ROWTYPE;
END pachet4_***;
/
```

```
CREATE OR REPLACE PACKAGE BODY pachet4_*** IS

CURSOR c_emp  RETURN employees%ROWTYPE  IS
       SELECT *
       FROM   employees;
```

```
PROCEDURE p_verific(v_cod   employees.employee_id%TYPE,
                    v_job   employees.job_id%TYPE)
IS
  gasit BOOLEAN:=FALSE;
  lista employees%ROWTYPE;
BEGIN
  OPEN c_emp;
  LOOP
    FETCH c_emp INTO lista;
    EXIT WHEN c_emp%NOTFOUND;
    IF lista.employee_id=v_cod  AND lista.job_id=v_job
       THEN  gasit:=TRUE;
    END IF;
  END LOOP;
  CLOSE c_emp;
  IF gasit=TRUE THEN
     DBMS_OUTPUT.PUT_LINE('combinatia data exista');
  ELSE
     DBMS_OUTPUT.PUT_LINE('combinatia data nu exista');
  END IF;
END p_verific;
END pachet4_***;
/

EXECUTE pachet4_***.p_verific(200,'AD_ASST');
```

II. Pachete predefinite

1. **Pachetul DBMS_OUTPUT** permite afişarea de informaţii. Procedurile pachetului sunt:
   PUT – depune (scrie) în buffer informaţie;
   PUT_LINE – depune în buffer informaţia, împreună cu un marcaj de sfârşit de linie;
   NEW_LINE – depune în buffer un marcaj de sfârşit de linie;
   GET_LINE – regăseşte o singură linie de informaţie;
   GET_LINES – regăseşte mai multe linii de informaţie;
   ENABLE/DISABLE – activează/dezactivează procedurile pachetului.

Exemplul 1:

```
DECLARE
-- paramentrii de tip OUT pt procedura GET_LINE
   linie1 VARCHAR2(255);
   stare1 INTEGER;
   linie2 VARCHAR2(255);
   stare2 INTEGER;
   linie3 VARCHAR2(255);
   stare3 INTEGER;

v_emp  employees.employee_id%TYPE;
v_job  employees.job_id%TYPE;
v_dept employees.department_id%TYPE;
```

```
BEGIN
  SELECT employee_id, job_id, department_id
  INTO   v_emp,v_job,v_dept
  FROM   employees
  WHERE  last_name='Lorentz';

-- se introduce o linie in buffer fara caracter
-- de terminare linie
   DBMS_OUTPUT.PUT(' 1 '||v_emp|| ' ');

-- se incearca extragerea liniei introdusa
-- in buffer si starea acesteia
   DBMS_OUTPUT.GET_LINE(linie1,stare1);

-- se depunde informatie pe aceeasi linie in buffer
   DBMS_OUTPUT.PUT(' 2 '||v_job|| ' ');

-- se inchide linia depusa in buffer si se extrage
-- linia din buffer
   DBMS_OUTPUT.NEW_LINE;
   DBMS_OUTPUT.GET_LINE(linie2,stare2);

-- se introduc informatii pe aceeasi linie
-- si se afiseaza informatia
   DBMS_OUTPUT.PUT_LINE(' 3 ' ||v_emp|| ' '|| v_job);
   DBMS_OUTPUT.GET_LINE(linie3,stare3);

-- se afiseaza ceea ce s-a extras
   DBMS_OUTPUT.PUT_LINE('linie1 = '|| linie1||
                        '; stare1 = '||stare1);
   DBMS_OUTPUT.PUT_LINE('linie2 = '|| linie2||
                        '; stare2 = '||stare2);
   DBMS_OUTPUT.PUT_LINE('linie3 = '|| linie3||
                        '; stare3 = '||stare3);
END;
/
```

Exemplul 2:

```
DECLARE
-- parametru de tip OUT pentru NEW_LINES
-- tablou de siruri de caractere
   linii DBMS_OUTPUT.CHARARR;
-- paramentru de tip IN OUT pentru NEW_LINES
   nr_linii INTEGER;

 v_emp  employees.employee_id%TYPE;
 v_job  employees.job_id%TYPE;
 v_dept employees.department_id%TYPE;
```

```
BEGIN
  SELECT employee_id, job_id, department_id
  INTO   v_emp,v_job,v_dept
  FROM   employees
  WHERE  last_name='Lorentz';

-- se mareste dimensiunea bufferului
   DBMS_OUTPUT.ENABLE(1000000);
   DBMS_OUTPUT.PUT(' 1 '||v_emp|| ' ');
   DBMS_OUTPUT.PUT(' 2 '||v_job|| ' ');
   DBMS_OUTPUT.NEW_LINE;
   DBMS_OUTPUT.PUT_LINE(' 3 ' ||v_emp|| ' '|| v_job);
   DBMS_OUTPUT.PUT_LINE(' 4 ' ||v_emp|| ' '||
                        v_job||' ' ||v_dept);
-- se afiseaza ceea ce s-a extras
   nr_linii := 4;
   DBMS_OUTPUT.GET_LINES(linii,nr_linii);
   DBMS_OUTPUT.put_line('In buffer sunt '||
                         nr_linii ||' linii');
   FOR i IN 1..nr_linii LOOP
       DBMS_OUTPUT.put_line(linii(i));
   END LOOP;

--  nr_linii := 4;
--  DBMS_OUTPUT.GET_LINES(linii,nr_linii);
--  DBMS_OUTPUT.put_line('Acum in buffer sunt '||
                          nr_linii ||' linii');
--   FOR i IN 1..nr_linii LOOP
--       DBMS_OUTPUT.put_line(linii(i));
--   END LOOP;
--
---- DBMS_OUTPUT.disable;
---- DBMS_OUTPUT.enable;
----
---- nr_linii := 4;
---- DBMS_OUTPUT.GET_LINES(linii,nr_linii);
---- DBMS_OUTPUT.put_line('Acum in buffer sunt '||
                           nr_linii ||' linii');
END;
/
```

2. **Pachetul DBMS_JOB** este utilizat pentru planificarea execuţiei programelor PL/SQL
   SUBMIT – adaugă un nou job în coada de aşteptare a job-urilor;
   REMOVE – şterge un job din coada de aşteptare;
   RUN – execută imediat un job specificat.

Exemplu:

```
CREATE OR REPLACE PROCEDURE marire_salariu_***
     (id_angajat emp_***.employee_id%type,
      valoare    number)
IS
BEGIN
  UPDATE emp_***
  SET    salary = salary + valoare
  WHERE  employee_id = id_angajat;
END;
/
```

- INTERVAL este de tip VARCHAR2 DEFAULT 'NULL'
- se verifică trimiterea spre execuţie a procedurii (în Entreprise Manager Console → baza de date → Instance → Configuration → All Initialization parameters se setează parametrul JOB_QUEUE_PROCESSES la o valoare mai mare decât 0)

Varianta 1

```
VARIABLE nr_job NUMBER

BEGIN
 DBMS_JOB.SUBMIT(
    -- întoarce numărul jobului, printr-o variabilă de legătură
    JOB => :nr_job,

    -- codul PL/SQL care trebuie executat
    WHAT => 'marire_salariu_***(100, 1000);',

    -- data de start a execuţiei (dupa 30 secunde)
    NEXT_DATE => SYSDATE+30/86400,

    -- intervalul de timp la care se repetă execuţia
    INTERVAL => 'SYSDATE+1');

    COMMIT;
END;
/
```

```
SELECT salary FROM emp_*** WHERE employee_id = 100;
-- asteptati 30 de secunde
SELECT salary FROM emp_*** WHERE employee_id = 100;

-- numarul jobului
PRINT nr_job;

-- informatii despre joburi
SELECT JOB, NEXT_DATE, WHAT
FROM   USER_JOBS;
```

```
-- lansarea jobului la momentul dorit
SELECT salary FROM emp_*** WHERE employee_id = 100;
BEGIN
   -- presupunand ca jobul are codul 1 atunci:
   DBMS_JOB.RUN(job => 1);
END;
/
SELECT salary FROM emp_*** WHERE employee_id = 100;

-- stergerea unui job
BEGIN
   DBMS_JOB.REMOVE(job=>1);
END;
/

SELECT JOB, NEXT_DATE, WHAT
FROM   USER_JOBS;

UPDATE emp_***
SET    salary = 24000
WHERE  employee_id = 100;

COMMIT;
```

Varianta 2

```
CREATE OR REPLACE PACKAGE pachet_job_***
IS
  nr_job NUMBER;
  FUNCTION obtine_job RETURN NUMBER;
END;
/

CREATE OR REPLACE PACKAGE body pachet_job_***
IS
  FUNCTION obtine_job RETURN NUMBER IS
  BEGIN
    RETURN nr_job;
  END;
END;
/
```

```
BEGIN
 DBMS_JOB.SUBMIT(
    -- întoarce numărul jobului, printr-o variabilă de legătură
    JOB => pachet_job_***.nr_job,

    -- codul PL/SQL care trebuie executat
    WHAT => 'marire_salariu_***(100, 1000);',
```

```
    -- data de start a execuţiei (dupa 30 secunde)
    NEXT_DATE => SYSDATE+30/86400,

    -- intervalul de timp la care se repetă execuţia
    INTERVAL => 'SYSDATE+1');

    COMMIT;
END;
/

-- informatii despre joburi
SELECT JOB, NEXT_DATE, WHAT
FROM   USER_JOBS
WHERE  JOB = pachet_job_***.obtine_job;

-- lansarea jobului la momentul dorit
SELECT salary FROM emp_*** WHERE employee_id = 100;
BEGIN
   DBMS_JOB.RUN(JOB => pachet_job_***.obtine_job);
END;
/
SELECT salary FROM emp_*** WHERE employee_id = 100;

-- stergerea unui job
BEGIN
   DBMS_JOB.REMOVE(JOB=>pachet_job_***.obtine_job);
END;
/
SELECT JOB, NEXT_DATE, WHAT
FROM   USER_JOBS
WHERE  JOB = pachet_job_***.obtine_job;

UPDATE emp_***
SET    salary = 24000
WHERE  employee_id = 100;

COMMIT;
```

3. **Pachetul UTL_FILE** extinde operaţiile I/O la fişiere. Se apelează funcţia FOPEN pentru a deschide un fişier; acesta este folosit pentru operaţiile de citire sau scriere. După ce s-au încheiat operaţiile I/O se închide fişierul (FCLOSE).

   Observaţie: în Enterprise Manager Console → baza de date → Instance → Configuration → All Initialization parameters se setează parametrul UTL_FILE_DIR la o valoare care reprezintă directorul unde se face citirea/scrierea (de exemplu F:). Aceasta operaţie va cere oprirea bazei de date şi repornirea ei.

Exemplu:

Menţinem rezultatele unei comenzi SELECT într-un fişier.

```
CREATE OR REPLACE PROCEDURE scriu_fisier_***
 (director VARCHAR2,
  fisier VARCHAR2)
IS
  v_file UTL_FILE.FILE_TYPE;
  CURSOR cursor_rez IS
     SELECT department_id departament, SUM(salary) suma
     FROM employees
     GROUP BY department_id
     ORDER BY SUM(salary);
  v_rez cursor_rez%ROWTYPE;
BEGIN
 v_file:=UTL_FILE.FOPEN(director, fisier, 'w');
 UTL_FILE.PUTF(v_file, 'Suma salariilor pe departamente \n Raport
        generat pe data  ');
 UTL_FILE.PUT(v_file, SYSDATE);
 UTL_FILE.NEW_LINE(v_file);
 OPEN cursor_rez;
 LOOP
     FETCH cursor_rez INTO v_rez;
     EXIT WHEN cursor_rez%NOTFOUND;
     UTL_FILE.NEW_LINE(v_file);
     UTL_FILE.PUT(v_file, v_rez.departament);
     UTL_FILE.PUT(v_file, '         ');
     UTL_FILE.PUT(v_file, v_rez.suma);
 END LOOP;
 CLOSE cursor_rez;
 UTL_FILE.FCLOSE(v_file);
END;
/

SQL> EXECUTE scriu_fisier('F:\','test.txt');
```

EXERCIŢII

1. Definiţi un pachet care să permită gestiunea angajaţilor companiei. Pachetul va conţine:

a. o procedură care determină adăugarea unui angajat, dându-se informaţii complete despre acesta:
- codul angajatului va fi generat automat utilizându-se o secvenţă;
- informaţiile personale vor fi date ca parametrii (nume, prenume, telefon, email);
- data angajării va fi data curentă;
- salariul va fi cel mai mic salariu din departamentul respectiv, pentru jobul respectiv (se vor obţine cu ajutorul unei funcţii stocate în pachet);
- nu va avea comision;
- codul managerului se va obţine cu ajutorul unei funcţii stocate în pachet care va avea ca parametrii numele şi prenumele managerului);
- codul departamentului va fi obţinut cu ajutorul unei funcţii stocate în pachet, dându-se ca parametru numele acestuia;
- codul jobului va fi obţinut cu ajutorul unei funcţii stocate în pachet, dându-se ca parametru numele acesteia.

Observaţie: Trataţi toate excepţiile.

b. o procedură care determină mutarea în alt departament a unui angajat (se dau ca parametrii numele şi prenumele angajatului, respectiv numele departamentului, numele jobului şi numele şi prenumele managerului acestuia):
- se vor actualiza informaţiile angajatului:
  - codul de departament (se va obţine cu ajutorul funcţiei corespunzătoare definită la punctul a);
  - codul jobului (se va obţine cu ajutorul funcţiei corespunzătoare definită la punctul a);
  - codul managerului (se va obţine cu ajutorul funcţiei corespunzătoare definită la punctul a);
  - salariul va fi cel mai mic salariu din noul departament, pentru noul job dacă acesta este mai mare decât salariul curent; altfel se va păstra salariul curent;
  - comisionul va fi cel mai mic comision din acel departament, pentru acel job;
  - data angajării va fi data curentă;
- se vor înregistra informaţii corespunzătoare în istoricul joburilor.

Observaţie: Trataţi toate excepţiile.

c. o funcţie care întoarce numărul de subalterni direcţi sau indirecţi ai unui angajat al cărui nume şi prenume sunt date ca parametrii;

Observaţie: Trataţi toate excepţiile.

d. o procedură care determină promovarea unui angajat pe o treaptă imediat superioară în departamentul său; propuneţi o variantă de restructurare a arborelui care implementează ierarhia subaltern – şef din companie;

Observaţie: Trataţi toate excepţiile.

e. o procedură prin care se actualizează cu o valoare dată ca parametru salariul unui angajat al cărui nume este dat ca parametru:
- se va verifica dacă valoarea dată pentru salariu respectă limitele impuse pentru acel job;
- dacă sunt mai mulţi angajaţi care au acelaşi nume, atunci se va afişa un mesaj corespunzător şi de asemenea se va afişa lista acestora;
- dacă nu există angajaţi cu numele dat, atunci se va afişa un mesaj corespunzător;

f. un cursor care obţine lista angajaţilor care lucrează pe un job al cărui cod este dat ca parametru;

g. un cursor care obţine lista tuturor joburilor din companie;

h. o procedură care utilizează cele două cursoare definite anterior şi obţine pentru fiecare job numele acestuia şi lista angajaţilor care lucrează în prezent pe acel job; în plus, pentru fiecare angajat să se specifice dacă în trecut a mai avut sau nu jobul respectiv.

Laborator 6 PL/SQL
Declanșatori

Un declanșator este un bloc PL/SQL care se execută automat ori de câte ori are loc un anumit eveniment "declanșator" (de exemplu, inserarea unei linii într-un tabel, ștergerea unor înregistrări etc.)

Tipuri de declanșatori:

- la nivel de bază de date – pot fi declanșați de o comandă *LMD* asupra datelor unui tabel; o comandă *LMD* asupra datelor unei vizualizări; o comandă *LDD* (*CREATE*, *ALTER*, *DROP*) referitoare la anumite obiecte ale schemei sau ale bazei de date; un eveniment sistem (*SHUTDOWN*, *STARTUP*); o acțiune a utilizatorului (*LOGON*, *LOGOFF*); o eroare (*SERVERERROR, SUSPEND*).
- la nivel de aplicație – se declanșează la apariția unui eveniment într-o aplicație particulară.

- Sintaxa comenzii de creare a unui declanșator *LMD* este următoarea:

```
CREATE [OR REPLACE] TRIGGER [schema.]nume_declanșator
{BEFORE | AFTER}
{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_declanșator;
```

- În cazul declanșatorilor *LMD* este important să stabilim:
  - momentul când este executat declanșatorul: BEFORE, AFTER
  - ce fel de acțiuni îl declanșează: INSERT, UPDATE, DELETE
  - tipul declanșatorului: la nivel de instrucțiune sau la nivel de linie (*FOR EACH ROW*).

- Sintaxa comenzii de creare a unui declanșator *INSTEAD OF* este următoarea:

```
CREATE [OR REPLACE] TRIGGER [schema.]nume_trigger
--momentul când este declanșat
INSTEAD OF
--comanda/comenzile care îl declanșează
{ DELETE|INSERT|UPDATE [OF coloana[, coloana …] ] }
  [OR {DELETE|INSERT|UPDATE [OF coloana[, coloana …] ] …}
ON [schema.]nume_vizualizare
[REFERENCING {OLD [AS] vechi NEW [AS] nou
              | NEW [AS] nou OLD [AS] vechi } ]
FOR EACH ROW
[WHEN (condiție) ]
corp_trigger (bloc anonim PL/SQL sau comanda CALL);
```

- Sintaxa comenzii de creare a unui declanșator sistem este următoarea:

```
CREATE [OR REPLACE] TRIGGER [schema.]nume_trigger
{BEFORE | AFTER}
{comenzi_LDD | evenimente_sistem}
ON {DATABASE | SCHEMA}
[WHEN (condiție) ]
corp_trigger;
```

- Informații despre declanșatori se pot obține interogând vizualizările
  - *USER_TRIGGERS, ALL_TRIGGERS, DBA_TRIGGERS*
  - *USER_TRIGGER_COL*
- Dezactivarea, respectiv activarea declanșatorilor se realizează prin următoarele comenzi:

```
ALTER TABLE nume_tabel
DISABLE ALL TRIGGERS;

ALTER TABLE nume_tabel
ENABLE ALL TRIGGERS;

ALTER TRIGGER nume_trig ENABLE;

ALTER TRIGGER nume_trig DISABLE;
```

- Eliminarea unui declanșator se face prin

```
DROP TRIGGER nume_trig;
```

1. Definiți un declanșator care să permită lucrul asupra tabelului emp_*** (INSERT, UPDATE, DELETE) decât în intervalul de ore 8:00 - 20:00, de luni până sâmbătă (**declanșator la nivel de instrucțiune**).

```
CREATE OR REPLACE TRIGGER trig1_***
      BEFORE INSERT OR UPDATE OR DELETE ON emp_***
BEGIN
 IF (TO_CHAR(SYSDATE,'D') = 1)
     OR (TO_CHAR(SYSDATE,'HH24') NOT BETWEEN 8 AND 20)
 THEN
 RAISE_APPLICATION_ERROR(-20001,'tabelul nu poate fi actualizat');
 END IF;
END;
/
DROP TRIGGER trig1_***;
```

2. Definiți un declanșator prin care să nu se permită micșorarea salariilor angajaților din tabelul emp_*** (**declanșator la nivel de linie**).

Varianta 1

```
CREATE OR REPLACE TRIGGER trig21_***
  BEFORE UPDATE OF salary ON emp_***
  FOR EACH ROW
BEGIN
  IF (:NEW.salary < :OLD.salary) THEN
  RAISE_APPLICATION_ERROR(-20002,'salariul nu poate fi micsorat');
```

```
  END IF;
END;
/
UPDATE emp_***
SET    salary = salary-100;

DROP TRIGGER trig21_***;
```

Varianta 2

```
CREATE OR REPLACE TRIGGER trig22_***
  BEFORE UPDATE OF salary ON emp_***
  FOR EACH ROW
  WHEN (NEW.salary < OLD.salary)
BEGIN
  RAISE_APPLICATION_ERROR(-20002,'salariul nu poate fi micsorat');
END;
/
UPDATE emp_***
SET    salary = salary-100;

DROP TRIGGER trig22_***;
```

3. Creați un declanșator care să nu permită mărirea limitei inferioare a grilei de salarizare 1, respectiv micșorarea limitei superioare a grilei de salarizare 7 decât dacă toate salariile se găsesc în intervalul dat de aceste două valori modificate. Se va utiliza tabelul *job_grades*_***.

```
CREATE OR REPLACE TRIGGER trig3_***
  BEFORE UPDATE OF lowest_sal, highest_sal ON job_grades_***
  FOR EACH ROW
DECLARE
  v_min_sal  emp_***.salary%TYPE;
  v_max_sal  emp_***.salary%TYPE;
  exceptie EXCEPTION;
BEGIN
  SELECT MIN(salary), MAX(salary)
  INTO   v_min_sal,v_max_sal
  FROM   emp_***;

  IF (:OLD.grade_level=1) AND  (v_min_sal< :NEW.lowest_sal)
     THEN RAISE exceptie;
  END IF;

  IF (:OLD.grade_level=7) AND  (v_max_sal> :NEW.highest_sal)
     THEN RAISE exceptie;
  END IF;

EXCEPTION
  WHEN exceptie THEN
    RAISE_APPLICATION_ERROR (-20003, 'Exista salarii care se
                             gasesc in afara intervalului');
END;
/
```

```
UPDATE job_grades_***
SET    lowest_sal =3000
WHERE  grade_level=1;

UPDATE job_grades_***
SET    highest_sal =20000
WHERE  grade_level=7;

DROP TRIGGER trig3_***;
```

4. **a.** Creați tabelul *info_dept_**** cu următoarele coloane:
- id (codul departamentului) – cheie primară;
- nume_dept (numele departamentului);
- plati (suma alocată pentru plata salariilor angajaților care lucrează în departamentul respectiv).

b. Introduceți date în tabelul creat anterior corespunzătoare informațiilor existente în schemă.

c. Definiți un declanșator care va actualiza automat câmpul *plati* atunci când se introduce un nou salariat, respectiv se șterge un salariat sau se modifică salariul unui angajat.

```
CREATE OR REPLACE PROCEDURE modific_plati_***
          (v_codd  info_dept_***.id%TYPE,
           v_plati info_dept_***.plati%TYPE) AS
BEGIN
  UPDATE  info_dept_***
  SET     plati = NVL (plati, 0) + v_plati
  WHERE   id = v_codd;
END;
/
```

```
CREATE OR REPLACE TRIGGER trig4_***
  AFTER DELETE OR UPDATE  OR  INSERT OF salary ON emp_***
  FOR EACH ROW
BEGIN
  IF DELETING THEN
     -- se sterge un angajat
     modific_plati_*** (:OLD.department_id, -1*:OLD.salary);
  ELSIF UPDATING THEN
    --se modifica salariul unui angajat
    modific_plati_***(:OLD.department_id,:NEW.salary-:OLD.salary);
  ELSE
    -- se introduce un nou angajat
    modific_plati_***(:NEW.department_id, :NEW.salary);
  END IF;
END;
/
```

```
SELECT * FROM  info_dept_*** WHERE id=90;

INSERT INTO emp_*** (employee_id, last_name, email, hire_date,
                     job_id, salary, department_id)
VALUES (300, 'N1', 'n1@g.com',sysdate, 'SA_REP', 2000, 90);
```

```
SELECT * FROM  info_dept_*** WHERE id=90;

UPDATE emp_***
SET    salary = salary + 1000
WHERE  employee_id=300;

SELECT * FROM  info_dept_*** WHERE id=90;

DELETE FROM emp_***
WHERE  employee_id=300;

SELECT * FROM  info_dept_*** WHERE id=90;

DROP TRIGGER trig4_***;
```

5. **a.** Creați tabelul *info_emp_**** cu următoarele coloane:
- id (codul angajatului) – cheie primară;
- nume (numele angajatului);
- prenume (prenumele angajatului);
- salariu (salariul angajatului);
- id_dept (codul departamentului) – cheie externă care referă tabelul *info_dept_****.

b. Introduceți date în tabelul creat anterior corespunzătoare informațiilor existente în schemă.

c. Creați vizualizarea *v_info_**** care va conține informații complete despre angajați și departamentele acestora. Folosiți cele două tabele create anterior, *info_emp_****, respectiv *info_dept_****.

d. Se pot realiza actualizări asupra acestei vizualizări? Care este tabelul protejat prin cheie? Consultați vizualizarea *user_updatable_columns.*

e. Definiți un declanșator prin care actualizările ce au loc asupra vizualizării se propagă automat în tabelele de bază (**declanșator INSTEAD OF**). Se consideră că au loc următoarele actualizări asupra vizualizării:
- se adaugă un angajat într-un departament deja existent;
- se elimină un angajat;
- se modifică valoarea salariului unui angajat;
- se modifică departamentul unui angajat (codul departamentului).

f. Verificați dacă declanșatorul definit funcționează corect.

g. Modificați declanșatorul definit astfel încât să permită și următoarele operații:
- se adaugă un angajat și departamentul acestuia (departamentul este nou);
- se adaugă doar un departament.

h. Verificați dacă declanșatorul definit funcționează corect.

i. Modificați prin intermediul vizualizării numele unui angajat. Ce observați?

j. Modificați declanșatorul definit anterior astfel încât să permită propagarea în tabelele de bază a actualizărilor realizate asupra numelui și prenumelui angajatului, respectiv asupra numelui de departament.

k. Verificați dacă declanșatorul definit funcționează corect.

```
CREATE OR REPLACE VIEW v_info_*** AS
  SELECT e.id, e.nume, e.prenume, e.salariu, e.id_dept,
         d.nume_dept, d.plati
  FROM   info_emp_*** e, info_dept_*** d
  WHERE  e.id_dept = d.id;

SELECT *
FROM   user_updatable_columns
WHERE  table_name = UPPER('v_info_***');

CREATE OR REPLACE TRIGGER trig5_***
    INSTEAD OF INSERT OR DELETE OR UPDATE ON v_info_***
    FOR EACH ROW
BEGIN
IF INSERTING THEN
    -- inserarea in vizualizare determina inserarea
    -- in info_emp_*** si reactualizarea in info_dept_***
    -- se presupune ca departamentul exista
   INSERT INTO info_emp_***
   VALUES (:NEW.id, :NEW.nume, :NEW.prenume, :NEW.salariu,
           :NEW.id_dept);

   UPDATE info_dept_***
   SET    plati = plati + :NEW.salariu
   WHERE  id = :NEW.id_dept;

ELSIF DELETING THEN
   -- stergerea unui salariat din vizualizare determina
   -- stergerea din info_emp_*** si reactualizarea in
   -- info_dept_***
   DELETE FROM info_emp_***
   WHERE  id = :OLD.id;

   UPDATE info_dept_***
   SET    plati = plati - :OLD.salariu
   WHERE  id = :OLD.id_dept;

ELSIF UPDATING ('salariu') THEN
   /* modificarea unui salariu din vizualizare determina
      modificarea salariului in info_emp_*** si reactualizarea
      in info_dept_***    */

   UPDATE  info_emp_***
   SET     salariu = :NEW.salariu
   WHERE   id = :OLD.id;

   UPDATE info_dept_***
   SET    plati = plati - :OLD.salariu + :NEW.salariu
   WHERE  id = :OLD.id_dept;

ELSIF UPDATING ('id_dept') THEN
    /* modificarea unui cod de departament din vizualizare
       determina modificarea codului in info_emp_***
       si reactualizarea in info_dept_***  */
```

```
    UPDATE info_emp_***
    SET    id_dept = :NEW.id_dept
    WHERE  id = :OLD.id;

    UPDATE info_dept_***
    SET    plati = plati - :OLD.salariu
    WHERE  id = :OLD.id_dept;

    UPDATE info_dept_***
    SET    plati = plati + :NEW.salariu
    WHERE  id = :NEW.id_dept;
  END IF;
END;
/

SELECT *
FROM   user_updatable_columns
WHERE  table_name = UPPER('v_info_***');

-- adaugarea unui nou angajat
SELECT * FROM  info_dept_*** WHERE id=10;

INSERT INTO v_info_***
VALUES (400, 'N1', 'P1', 3000,10, 'Nume dept', 0);

SELECT * FROM  info_emp_*** WHERE id=400;
SELECT * FROM  info_dept_*** WHERE id=10;

-- modificarea salariului unui angajat
UPDATE v_info_***
SET    salariu=salariu + 1000
WHERE  id=400;

SELECT * FROM  info_emp_*** WHERE id=400;
SELECT * FROM  info_dept_*** WHERE id=10;

-- modificarea departamentului unui angajat
SELECT * FROM  info_dept_*** WHERE id=90;

UPDATE v_info_***
SET    id_dept=90
WHERE  id=400;

SELECT * FROM  info_emp_*** WHERE id=400;
SELECT * FROM  info_dept_*** WHERE id IN (10,90);

-- eliminarea unui angajat
DELETE FROM v_info_*** WHERE id = 400;
SELECT * FROM  info_emp_*** WHERE id=400;
SELECT * FROM  info_dept_*** WHERE id = 90;

DROP TRIGGER trig5_***;
```

6. Definiți un declanșator care să nu se permită ștergerea informațiilor din tabelul *emp_**** de către utilizatorul *grupa****.

```
CREATE OR REPLACE TRIGGER trig6_***
  BEFORE DELETE ON emp_***
 BEGIN
  IF USER= UPPER('grupa***') THEN
     RAISE_APPLICATION_ERROR(-20900,'Nu ai voie sa stergi!');
  END IF;
 END;
/
DROP TRIGGER trig6_***;
```

7. **a.** Creați tabelul *audit_**** cu următoarele câmpuri:
- utilizator (numele utilizatorului);
- nume_bd (numele bazei de date);
- eveniment (evenimentul sistem);
- nume_obiect (numele obiectului);
- data (data producerii evenimentului).

b. Definiți un declanșator care să introducă date în acest tabel după ce utilizatorul a folosit o comandă LDD (declanșator sistem - la nivel de schemă).

```
CREATE TABLE audit_***
   (utilizator     VARCHAR2(30),
    nume_bd        VARCHAR2(50),
    eveniment      VARCHAR2(20),
    nume_obiect    VARCHAR2(30),
    data           DATE);

CREATE OR REPLACE TRIGGER trig7_***
  AFTER CREATE OR DROP OR ALTER ON SCHEMA
BEGIN
  INSERT INTO audit_***
  VALUES (SYS.LOGIN_USER, SYS.DATABASE_NAME, SYS.SYSEVENT,
          SYS.DICTIONARY_OBJ_NAME, SYSDATE);
END;
/

CREATE INDEX ind_*** ON info_emp_***(nume);

DROP INDEX ind_***;

SELECT * FROM audit_***;

DROP TRIGGER trig7_***;
```

8. Definiți un declanșator care să nu permită modificarea:
- valorii salariului maxim astfel încât acesta să devină mai mic decât media tuturor salariilor;
- valorii salariului minim astfel încât acesta să devină mai mare decât media tuturor salariilor.

Observație:

În acest caz este necesară menținerea unor variabile în care să se rețină salariul minim, salariul maxim, respectiv media salariilor. Variabilele se definesc într-un pachet, iar apoi pot fi referite în declanșator prin *nume_pachet.nume_variabila*.

Este necesar să se definească doi declanșatori:

- un declanșator la nivel de comandă care să actualizeze variabilele din pachet.
- un declanșator la nivel de linie care să realizeze verificarea condițiilor.

```
CREATE OR REPLACE PACKAGE pachet_***
AS
     smin emp_***.salary%type;
     smax emp_***.salary%type;
     smed emp_***.salary%type;
END pachet_***;
/

CREATE OR REPLACE TRIGGER trig81_***
BEFORE UPDATE OF salary ON emp_***
BEGIN
  SELECT MIN(salary),AVG(salary),MAX(salary)
  INTO pachet_***.smin, pachet_***.smed, pachet_***.smax
  FROM emp_***;
END;
/

CREATE OR REPLACE TRIGGER trig82_***
BEFORE UPDATE OF salary ON emp_***
FOR EACH ROW
BEGIN
IF(:OLD.salary=pachet_***.smin)AND (:NEW.salary>pachet_***.smed)
 THEN
   RAISE_APPLICATION_ERROR(-20001,'Acest salariu depaseste
                                   valoarea medie');
ELSIF (:OLD.salary= pachet_***.smax)
       AND (:NEW.salary<  pachet_***.smed)
 THEN
   RAISE_APPLICATION_ERROR(-20001,'Acest salariu este sub
                                   valoarea medie');
END IF;
END;
/

SELECT AVG(salary)
FROM   emp_***;

UPDATE emp_***
SET    salary=10000
WHERE  salary=(SELECT MIN(salary) FROM emp_***);

UPDATE emp_***
SET    salary=1000
WHERE  salary=(SELECT MAX(salary) FROM emp_***);

DROP TRIGGER trig81_***;
DROP TRIGGER trig82_***;
```

EXERCIȚII

1. Definiți un declanșator care să permită ștergerea informațiilor din tabelul *dept_*** decât dacă utilizatorul este SCOTT.

2. Creați un declanșator prin care să nu se permită mărirea comisionului astfel încât să depășească 50% din valoarea salariului.

3. **a.** Introduceți în tabelul *info_dept_*** coloana *numar* care va reprezenta pentru fiecare departament numărul de angajați care lucrează în departamentul respectiv. Populați cu date această coloană pe baza informațiilor din schemă.

b. Definiți un declanșator care va actualiza automat această coloană în funcție de actualizările realizate asupra tabelului *info_emp_****.

4. Definiți un declanșator cu ajutorul căruia să se implementeze restricția conform căreia într-un departament nu pot lucra mai mult de 45 persoane (se vor utiliza doar tabelele *emp_*** și *dept_*** fără a modifica structura acestora).

5. **a.** Pe baza informațiilor din schemă creați și populați cu date următoarele două tabele:
- *emp_test_*** (employee_id – cheie primară, last_name, first_name, department_id);
- *dept_test_*** (department_id – cheie primară, department_name).

b. Definiți un declanșator care va determina ștergeri și modificări în cascadă:
- ștergerea angajaților din tabelul *emp_test_*** dacă este eliminat departamentul acestora din tabelul *dept_test_****;
- modificarea codului de departament al angajaților din tabelul *emp_test_*** dacă departamentul respectiv este modificat în tabelul *dept_test_****.

Testați și rezolvați problema în următoarele situații:
- nu este definită constrângere de cheie externă între cele două tabele;
- este definită constrângerea de cheie externă între cele două tabele;
- este definită constrângerea de cheie externă între cele două tabele cu opțiunea ON DELETE CASCADE;
- este definită constrângerea de cheie externă între cele două tabele cu opțiunea ON DELETE SET NULL.

Comentați fiecare caz în parte.

6. **a.** Creați un tabel cu următoarele coloane:
- user_id (SYS.LOGIN_USER);
- nume_bd (SYS.DATABASE_NAME);
- erori (DBMS_UTILITY.FORMAT_ERROR_STACK);
- data.

b. Definiți un declanșator sistem (la nivel de bază de date) care să introducă date în acest tabel referitoare la erorile apărute.

Laborator 7 PL/SQL
Tratarea excepţiilor

- PL/SQL permite utilizatorului să capteze şi să gestioneze erorile care pot apărea în timpul execuţiei unui program. În general, într-un bloc PL/SQL erorile ce apar sunt de 2 tipuri:
  - erori la compilare (detectate de motorul PL/SQL şi comunicate programatorului, care va face corectările necesare; aceste erori nu pot fi tratate în interiorul programului)
  - erori la execuţie (sunt numite excepţii; trebuie specificat în program modul de tratare a acestora, caz în care se spune ca excepţia este tratată în program; dacă aceasta nu este tratată, atunci se va propaga în mediul din care s-a invocat programul)
- O excepţie *PL/SQL* este o situaţie specială ce poate apărea în execuţia unei bloc PL/SQL.
- O excepţie poate fi gestionată:
  - în mod explicit de către utilizator (comanda RAISE);
  - în mod automat de către server, atunci când apare o eroare.
- Tratarea excepţiilor se realizează în zona EXCEPTION a unui bloc PL/SQL.

```
EXCEPTION
WHEN nume_excepţie1 [OR nume_excepţie2 …] THEN
    secvenţa_de_instrucţiuni_1;
  [WHEN nume_excepţie3 [OR nume_excepţie4 …] THEN
    secvenţa_de_instrucţiuni_2;]
  …
  [WHEN OTHERS THEN
    secvenţa_de_instrucţiuni_n;]
END;
```

- Tipuri de excepţii :
  - excepţii Oracle Server predefinite (NO_DATA_FOUND, TOO_MANY_ROWS etc);
  - excepţii Oracle Server nepredefinite (nu au un nume precum NO_DATA_FOUND, ci pot fi recunoscute doar după cod şi mesaj);
  - excepţii definite de utilizator.
- Informaţii despre erorile apărute la compilare se pot obţine 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*.

1. Remediaţi rând pe rând excepţiile din următorul exemplu.

```
SET SERVEROUT ON
DECLARE
  v NUMBER;
  CURSOR c IS
      SELECT employee_id FROM employees;
BEGIN
```

```
 -- no data found
SELECT employee_id
INTO v
FROM employees
WHERE 1=0;
-- too many rows
SELECT employee_id
INTO v
FROM employees;
  -- invalid number
SELECT employee_id
INTO v
FROM employees;
WHERE 2='s';
  -- when others
 v := 's';
  -- cursor already open
  open c;
  open c;
EXCEPTION
 WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE (' no data found: ' ||SQLCODE || ' - ' ||
SQLERRM);
 WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE (' too many rows:   ' ||SQLCODE || ' - '
|| SQLERRM);
 WHEN INVALID_NUMBER THEN
    DBMS_OUTPUT.PUT_LINE (' invalid number: ' ||SQLCODE || ' - '
|| SQLERRM);
 WHEN CURSOR_ALREADY_OPEN THEN
    DBMS_OUTPUT.PUT_LINE (' cursor already open: ' ||SQLCODE || '
- ' || SQLERRM);
 WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE (SQLCODE || ' - ' || SQLERRM);
 END;
/
SET SERVEROUT OFF
```

2. Să se creeze tabelul *error_**** care va conţine două câmpuri: *cod* de tip NUMBER şi *mesaj* de tip VARCHAR2(100). Să se creeze un bloc PL/SQL care să permită gestiunea erorii „divide by zero” în două moduri: prin definirea unei excepţii de către utilizator şi prin captarea erorii interne a sistemului. Codul şi mesajul erorii vor fi introduse în tabelul *error_****.

```
DROP TABLE error_***;

CREATE TABLE error_***
(cod    NUMBER,
 mesaj  VARCHAR2(100));
```

Varianta 1

```
DECLARE
  v_cod       NUMBER;
  v_mesaj     VARCHAR2(100);
  x           NUMBER;
  exceptie    EXCEPTION;

BEGIN
 x:=1;
 IF x=1 THEN RAISE exceptie;
 ELSE
     x:=x/(x-1);
 END IF;
EXCEPTION
 WHEN exceptie THEN
      v_cod := -20001;
      v_mesaj := 'x=1 determina o impartire la 0';
      INSERT  INTO error_***
      VALUES  (v_cod, v_mesaj);
 END;
/

SELECT *
FROM error_***;
```

Varianta 2

```
DECLARE
  v_cod        NUMBER;
  v_mesaj      VARCHAR2(100);
  x            NUMBER;
BEGIN
 x:=1;
 x:=x/(x-1);
EXCEPTION
WHEN  ZERO_DIVIDE THEN
      v_cod := SQLCODE;
      v_mesaj := SUBSTR(SQLERRM,1,100);
      -- mesajul erorii are dimensiune 512
      INSERT  INTO error_***
      VALUES  (v_cod, v_mesaj);
END;
/

SELECT *
FROM error_***;
ROLLBACK;
```

3. Să se creeze un bloc *PL/SQL* prin care să se afişeze numele departamentului care funcţionează într-o anumită locaţie. Dacă interogarea nu întoarce nicio linie, atunci să se trateze excepţia şi să se insereze în tabelul *error_**** codul erorii -20002 cu mesajul “nu exista departamente in locatia data”. Dacă interogarea întoarce o singură linie, atunci să se afişeze numele departamentului. Dacă

interogarea întoarce mai multe linii, atunci să se introducă în tabelul *error_*** codul erorii -20003 cu mesajul “exista mai multe departamente in locatia data”.

Testaţi pentru următoarele locaţii: 1400, 1700, 3000.

```
SET SERVEROUTPUT ON
SET VERIFY OFF
ACCEPT p_loc PROMPT 'Dati locatia: '

DECLARE
  v_loc      dept_***.location_id%TYPE:= &p_loc;
  v_nume     dept_***.department_name%TYPE;
BEGIN
  SELECT   department_name
  INTO     v_nume
  FROM     dept_***
  WHERE    location_id = v_loc;
  DBMS_OUTPUT.PUT_LINE('In locatia '|| v_loc ||
           ' functioneaza departamentul '||v_nume);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
     INSERT   INTO error_***
     VALUES  ( -20002, 'nu exista departamente in locatia data');
     DBMS_OUTPUT.PUT_LINE('a aparut o exceptie ');
  WHEN TOO_MANY_ROWS THEN
     INSERT   INTO error_***
     VALUES   (-20003,
                'exista mai multe departamente in locatia data');
     DBMS_OUTPUT.PUT_LINE('a aparut o exceptie ');
  WHEN OTHERS THEN
INSERT   INTO error_*** (mesaj)
VALUES   ('au aparut alte erori');
END;
/

SET VERIFY ON
SET SERVEROUTPUT OFF
```

4. Să se adauge constrângerea de cheie primară pentru câmpul *department_id* din tabelul *dept_*** şi constrângerea de cheie externă pentru câmpul *department_id* din tabelul *emp_**** care referă câmpul cu acelaşi nume din tabelul *dept_****.

Să se creeze un bloc PL/SQL care tratează excepţia apărută în cazul în care se şterge un departament în care lucrează angajaţi (**excepţie internă nepredefinită**).

```
ALTER TABLE dept_***
ADD CONSTRAINT c_pr_*** PRIMARY KEY(department_id);

ALTER TABLE emp_***
ADD CONSTRAINT c_ex_*** FOREIGN KEY (department_id)
    REFERENCES dept_***;
```

```
DELETE FROM dept_***
WHERE department_id=10;  --apare eroarea sistem -02292

SET SERVEROUTPUT ON
SET VERIFY OFF
ACCEPT p_cod PROMPT 'Dati un cod de departament '
DECLARE
  exceptie  EXCEPTION;
  PRAGMA EXCEPTION_INIT(exceptie,-02292);
  --   exceptia nu are un nume predefinit,
       cu PRAGMA EXCEPTION_INIT asociez erorii avand
  --   codul -02292 un nume
BEGIN
  DELETE FROM dept_***
  WHERE department_id = &p_cod;
EXCEPTION
  WHEN exceptie THEN
    DBMS_OUTPUT.PUT_LINE ('nu puteti sterge un departament in care
lucreaza salariati');
END;
/
SET VERIFY ON
SET SERVEROUTPUT OFF
```

5. Să se creeze un bloc *PL/SQL* prin care se afişează numărul de salariaţi care au venitul anual mai mare decât o valoare dată. Să se trateze cazul în care niciun salariat nu îndeplineşte această condiţie **(excepţii externe**).

```
SET SERVEROUTPUT ON
SET VERIFY OFF
ACCEPT p_val PROMPT 'Dati valoarea: '
DECLARE
  v_val          NUMBER := &p_val;
  v_numar     NUMBER(7);
  exceptie      EXCEPTION;
BEGIN
  SELECT  COUNT(*)
  INTO    v_numar
  FROM    emp_***
  WHERE   (salary+salary*NVL(commission_pct,0))*12>v_val;
  IF v_numar = 0 THEN
     RAISE  exceptie;
  ELSE
     DBMS_OUTPUT.PUT_LINE('NR de angajati este '||v_numar);
  END IF;

EXCEPTION
  WHEN exceptie THEN
    DBMS_OUTPUT.PUT_LINE('Nu  exista  angajati  pentru  care  sa  se
indeplineasca aceasta conditie');
```

```
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Alta eroare');
END;
/
SET VERIFY ON
SET SERVEROUTPUT OFF
```

6. Să se mărească cu 1000 salariul unui angajat al cărui cod este dat de la tastatură. Să se trateze cazul în care nu există angajatul al cărui cod este specificat. Tratarea excepţie se va face **în secţiunea executabilă**.

```
SET VERIFY OFF
ACCEPT p_cod PROMPT 'Dati codul: '
DECLARE
  v_cod          NUMBER := &p_cod;
BEGIN
UPDATE  emp_***
SET     salary=salary+1000
WHERE   employee_id=v_cod;
IF SQL%NOTFOUND THEN
  RAISE_APPLICATION_ERROR(-20999,'salariatul nu exista');
END IF;
END;
/
SET VERIFY ON
```

7. Să se afişeze numele şi salariul unui angajat al cărui cod este dat de la tastatură. Să se trateze cazul în care nu există angajatul al cărui cod este specificat. Tratarea excepţie se va face **în secţiunea de tratare a erorilor**.

```
SET SERVEROUTPUT ON
SET VERIFY OFF
ACCEPT p_cod PROMPT 'Dati codul: '
DECLARE
  v_cod    NUMBER := &p_cod;
  v_nume   emp_***.last_name%TYPE;
  v_sal    emp_***.salary%TYPE;
BEGIN
  SELECT last_name,salary
  INTO   v_nume,v_sal
  FROM   emp_***
  WHERE  employee_id=v_cod;
  DBMS_OUTPUT.PUT_LINE(v_nume||' '||v_sal);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20999, 'salariatul nu exista');
END;
/
SET VERIFY ON
SET SERVEROUTPUT OFF
```

8. Să se creeze un bloc PL/SQL care foloseşte 3 comenzi SELECT. Una dintre aceste comenzi nu va întoarce nicio linie. Să se determine care dintre cele trei comenzi SELECT determină apariţia excepţiei NO_DATA_FOUND.

```
Varianta 1 - fiecare comandă are un număr de ordine

SET SERVEROUTPUT ON
DECLARE
  v_localizare  NUMBER(1):=1;
  v_nume  emp_***.last_name%TYPE;
  v_sal    emp_***.salary%TYPE;
  v_job    emp_***.job_id%TYPE;

BEGIN
v_localizare:=1;
SELECT last_name
INTO   v_nume
FROM   emp_***
WHERE  employee_id=200;
DBMS_OUTPUT.PUT_LINE(v_nume);

v_localizare:=2;
SELECT salary
INTO   v_sal
FROM   emp_***
WHERE  employee_id=455;
DBMS_OUTPUT.PUT_LINE(v_sal);

v_localizare:=3;
SELECT job_id
INTO   v_job
FROM   emp_***
WHERE  employee_id=200;
DBMS_OUTPUT.PUT_LINE(v_job);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('comanda SELECT ' || v_localizare || ' nu
returneaza nimic');
END;
/
SET SERVEROUTPUT OFF

Varianta 2 - fiecare comandă este inclusă într-un subbloc

SET SERVEROUTPUT ON
DECLARE
  v_localizare  NUMBER(1):=1;
  v_nume  emp_***.last_name%TYPE;
  v_sal    emp_***.salary%TYPE;
  v_job    emp_***.job_id%TYPE;
BEGIN
```

```
  BEGIN
    SELECT last_name
    INTO   v_nume
    FROM   emp_***
    WHERE  employee_id=200;
     DBMS_OUTPUT.PUT_LINE(v_nume);
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('comanda SELECT1 nu returneaza nimic');
  END;

  BEGIN
    SELECT salary
    INTO   v_sal
    FROM   emp_***
    WHERE  employee_id=455;
    DBMS_OUTPUT.PUT_LINE('v_sal');
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('comanda SELECT2 nu returneaza nimic');
  END;

  BEGIN
SELECT job_id
INTO   v_job
FROM   emp_***
WHERE  employee_id=200;
DBMS_OUTPUT.PUT_LINE(v_job);
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('comanda SELECT3 nu returneaza nimic');
  END;

END;
/
SET SERVEROUTPUT OFF
```

9. Daţi un exemplu prin care să se arate că nu este permis **saltul de la secţiunea de tratare a unei excepţii, în blocul curent**.

```
DECLARE
  v_comm  NUMBER(4);
BEGIN
  SELECT ROUND(salary*NVL(commission_pct,0))
  INTO   v_comm
  FROM   emp_***
  WHERE  employee_id=455;
<<eticheta>>
  UPDATE  emp_***
  SET     salary=salary+v_comm
  WHERE employee_id=200;
```

```
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    v_comm:=5000;
    GOTO eticheta;
END;
/
```

10. Daţi un exemplu prin care să se arate că nu este permis saltul la secţiunea de tratare a unei excepţii.

```
SET SERVEROUTPUT ON
DECLARE
  v_comm_val  NUMBER(4);
  v_comm      emp_***.commission_pct%TYPE;
BEGIN
  SELECT NVL(commission_pct,0),
         ROUND(salary*NVL(commission_pct,0))
  INTO   v_comm, v_comm_val
  FROM   emp_***
  WHERE employee_id=200;
  IF v_comm=0
   THEN
     GOTO eticheta;
   ELSE
     UPDATE emp_***
     SET salary=salary+ v_comm_val
     WHERE employee_id=200;
  END IF;
<<eticheta>>
   --DBMS_OUTPUT.PUT_LINE('este ok!');
EXCEPTION
  WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE('este o exceptie!');
END;
/
SET SERVEROUTPUT OFF
```

EXERCIŢII

1. Să se creeze un bloc *PL/SQL* care afişează radicalul unei variabile introduse de la tastatură. Să se trateze cazul în care valoarea variabilei este negativă. Gestiunea erorii se va realiza prin definirea unei excepţii de către utilizator, respectiv prin captarea erorii interne a sistemului. Codul şi mesajul erorii vor fi introduse în tabelul error_***(cod, mesaj).

2. Să se creeze un bloc *PL/SQL* prin care să se afişeze numele salariatului (din tabelul *emp*_***) care câştigă un anumit salariu. Valoarea salariului se introduce de la tastatură. Se va testa programul pentru următoarele valori: 500, 3000 şi 5000.

Dacă interogarea nu întoarce nicio linie, atunci să se trateze excepţia şi să se afişeze mesajul “nu exista salariati care sa castige acest salariu ”. Dacă interogarea întoarce o singură linie, atunci să se afişeze numele salariatului. Dacă interogarea întoarce mai multe linii, atunci să se afişeze mesajul “exista mai mulţi salariati care castiga acest salariu”.

3. Să se creeze un bloc *PL/SQL* care tratează eroarea apărută în cazul în care se modifică codul unui departament în care lucrează angajaţi.

4. Să se creeze un bloc *PL/SQL* prin care se afişează numele departamentului 10 dacă numărul său de angajaţi este într-un interval dat de la tastatură. Să se trateze cazul în care departamentul nu îndeplineşte această condiţie.

5. Să se modifice numele unui departament al cărui cod este dat de la tastatură. Să se trateze cazul în care nu există acel departament. Tratarea excepţie se va face în secţiunea executabilă.

6. Să se creeze un bloc *PL/SQL* care afişează numele departamentului ce se află într-o anumită locaţie şi numele departamentului ce are un anumit cod (se vor folosi două comenzi SELECT). Să se trateze excepţia NO_DATA_FOUND şi să se afişeze care dintre comenzi a determinat eroarea. Să se rezolve problema în două moduri.