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.