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

Pachete

I. Definirea pachetelor

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

Ø Spre deosebire de subprograme, pachetele nu pot:

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

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

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

Ø Un pachet are următoarea formă generală:

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

II. Pachete predefinite

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

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

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

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

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

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

Procedura FCLOSE permite închiderea unui fişier.

Exerciții
I. [Pachete definite de utilizator]

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

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

Soluție:

Specificația pachetului:

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

Corpul pachetului:

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

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

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

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

Obs: Pentru invocarea procedurii:

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

sau

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

Pentru invocarea funcției:

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

```
FROM dual;
```

sau

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

sau

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

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

Soluție:

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

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

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

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

END emp_pkg_pnu;
/
```

Exemplu de invocare:

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

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

Soluție :

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

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

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

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

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

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

Soluție:

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

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

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

Soluție:

In specificația pachetului, vom avea:

```
PROCEDURE init_dept;
```

In corpul pachetului, se adaugă înaintea specificării subprogramelor:

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

La sfârşitul corpului pachetului, se declară procedura :

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

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

```
BEGIN
        Init_dept ;
End ;
```

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

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

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

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

Inserați un departament nou, având codul 15.

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

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

```
EXECUTE emp_pkg_pnu.init_dept
```

Inserați înregistrarea corespunzătoare angajatului din departamentul 15.

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

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

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

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

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

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

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

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

II. [Pachete standard]

[DBMS_OUTPUT]

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

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

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

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

[DBMS_JOB]

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

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

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

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

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

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

d) Stergeți job-ul din coada de aşteptare.

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

[UTL_FILE]

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

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

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

[SQL dinamic, DBMS_SQL]

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

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

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

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

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

END;

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

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

c) Listați structura tabelului contacte_pnu.

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

e) Afişați conținutul tabelului contacte_pnu.

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

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

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

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

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

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

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

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

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