Laborator 6 PL/SQL

Pachete

I. Definirea pachetelor

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

Spre deosebire de subprograme, pachetele **nu** pot:

- fi apelate,
- transmite parametri,
- fi încuibrite.

Un pachet are două părți, fiecare fiind stocat separat în dicționarul datelor.

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

Un pachet are următoarea formă generală:

```
CREATE PACKAGE nume_pachet {IS | AS} -- specificatia
      /* interfata utilizator, care contine: declaratii de tipuri si obiecte
      publice, specificatii de subprograme */
END [nume_pachet];
/

CREATE PACKAGE BODY nume_pachet {IS | AS} -- corpul
      /* implementarea, care contine: declaratii de obiecte si tipuri private,
      corpuri de subprograme specificate în partea de interfata */
[BEGIN]
      /* instructiuni de initializare, executate o singura data când
```

```
      pachetul este invocat prima oara de catre sesiunea utilizatorului */
END [nume_pachet];
/
```

II. Pachete predefinite

DBMS_OUTPUT permite afișarea 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 - sfârșit de linie
- NEW_LINE – depune în buffer un marcaj - sfârșit de linie
- GET_LINE – regăsește o singură linie de informație
- GET_LINES – regaseste mai multe linii de informație
- ENABLE/DISABLE – activează/dezactivează procedurile pachetului

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

- OPEN_CURSOR (deschide un nou cursor, adică se stabilește o zonă de memorie în care este procesată comanda SQL);
- PARSE (stabilește validitatea comenzii SQL, adică se verifică sintaxa instrucțiunii și se asociază cursorului deschis);
- BIND VARIABLE (leagă valoarea dată de variabila corespunzătoare din comanda SQL analizată)
- EXECUTE (execută comanda SQL și returnează numărul de linii procesate);
- FETCH_ROWS (regăsește o linie pentru un cursor specificat, iar pentru mai multe linii folosește un LOOP);
- CLOSE_CURSOR (închide cursorul specificat).

DBMS_JOB este utilizat pentru planificarea execuției programelor PL/SQL. Dintre subprogramele acestui pachet menționăm:

- SUBMIT – adaugă un nou job în coada de așteptare a job-urilor;
- REMOVE – șterge un job specificat din coada de așteptare 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.

UTL_FILE permite programului PL/SQL citirea din fișierele sistemului de operare, respectiv scrierea în aceste fișiere. El este utilizat pentru exploatarea fișierelor text. Scrierea și regăsirea informațiilor se face cu ajutorul unor proceduri asemănătoare celor din pachetul DBMS_OUTPUT. Procedura FCLOSE permite închiderea unui fiier.

Laborator 6 PL/SQL

Exerciții

I. [Pachete definite de utilizator]

1. a) Creați specificația și corpul unui pachet numit DEPT_PKG_PNU care conține:

- procedurile ADD_DEPT, UPD_DEPT și DEL_DEPT, corespunzătoare operațiilor de adăugare, actualizare (a numelui) și ștergere a unui departament din tabelul DEPT_PNU;

- funcția GET_DEPT, care determină denumirea unui departament, pe baza codului acestuia.

b) Invocați procedurile și funcția din cadrul pachetului atât prin blocuri PL/SQL cât și prin comenzi SQL.

```
SET SERVEROUTPUT ON

ALTER TABLE dept_pnu DISABLE ALL TRIGGERS;
ALTER TABLE emp_pnu DISABLE ALL TRIGGERS;

ALTER TABLE emp_pnu DROP CONSTRAINT emp_dept_pnu_fk;
ALTER TABLE emp_pnu DROP CONSTRAINT emp_pnu_pk;
ALTER TABLE dept_pnu DROP CONSTRAINT dept_pnu_pk;


--Specificatia 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) IS
    BEGIN
        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(13, 'IT');
EXECUTE dept_pkg_pnu.upd_dept(13, 'Information technology');
--sau
BEGIN
    dept_pkg_pnu.add_dept(13, 'IT');
    dept_pkg_pnu.upd_dept(13, 'Information technology');
END;
/
--Pentru invocarea functiei:
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 și 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 ieșire salariul și 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.

```
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 2100,
        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
        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 2100,
        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)
    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(emp_seq_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');
--sau
BEGIN
    emp_pkg_pnu.add_emp('Jane', 'Harris', 'jharris', p_job => 'SA_REP');
    emp_pkg_pnu.add_emp('David', 'Smith', 'dsmith', p_job => 'SA_MAN');
    COMMIT;
END;
/
```

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

```
--de adaugat in specificatia pachetului, dupa signatura celorlalte
--subprograme:
PROCEDURE add_emp(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_cod_job employees.job_id%TYPE DEFAULT 'SA_REP');

--de adaugat in corpul pachetului, dupa celelalte subprograme:
PROCEDURE add_emp(
```

```
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_cod_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_cod_job);
END add_emp;
/

--Exemplu de invocare:
EXECUTE emp_pkg_pnu.add_emp('Sam', 'Joplin', 'sa_man');
--sau
BEGIN
    emp_pkg_pnu.add_emp('Sam', 'Joplin', p_cod_job=>'sa_man');
    COMMIT;
END;
/
```

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 și va regăsi linia corespunzătoare codului respectiv;

- cealaltă funcție GET_EMP va avea un parametru p_nume_familie de tipul employees.last_name%TYPE și va regăsi linia corespunzătoare numelui respectiv;

- ambele funcții vor returna o valoare de tipul employees%ROWTYPE.

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

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

```
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 functie get_emp
--creati si procedura PRINT_EMPLOYEE
--apelati
```

5. Introduceți în pachet funcția VALID_DEPT_ID din laborator Subprograme (Laborator 4 PLSQL). Modificați prima procedură ADD_EMP astfel încât introducerea unui angajat nou să fie posibilă doar dacă departamentul este valid.

```
FUNCTION valid_dept_id
    (p_deptid IN dept_pnu.department_id%TYPE)
    RETURN boolean IS
    v_aux VARCHAR2(1);
BEGIN
    SELECT 'x'
    INTO v_aux
    FROM dept_pnu
    WHERE department_id = p_deptid;
    DBMS_OUTPUT.PUT_LINE('Exista codul de departament');
    RETURN TRUE;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Nu exista codul de departament');
        RETURN FALSE;
END valid_dept_id;

--add_emp ce foloseste functia valid_dept_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 2100,
    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)
IS
BEGIN
    IF valid_job_id(p_job) = TRUE AND valid_dept_id(p_deptid) = TRUE THEN
        INSERT INTO emp_pnu(employee_id, first_name, last_name, email,
manager_id,
                salary, job_id, commission_pct, department_id, hire_date)
        VALUES(emp_seq_pnu.nextval, p_first_name, p_last_name, p_email, p_mgr,
               p_sal, p_job, p_comm, p_deptid, p_hire_date);
        DBMS_OUTPUT.PUT_LINE('Adaugare cu succes');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Nu s-au introdus date coerente pentru tabelul
emp_pnu');
    END IF;
END add_emp;
```

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 cu coduri de departament valide. Creați această procedură.

```
--In specificatia pachetului, vom avea:
PROCEDURE init_dept;

--In corpul pachetului, se adauga înaintea specificarii subprogramelor:
TYPE boolean_tabtype IS TABLE OF BOOLEAN
    INDEX BY binary_integer;
valid_dept boolean_tabtype;
--La sfarsitul corpului pachetului, se declara procedura:
PROCEDURE init_dept IS
BEGIN
    FOR rec IN (SELECT distinct department_id FROM dept_pnu) LOOP
        valid_dept(rec.department_id) := TRUE;
    END LOOP;
END init_dept;

--La sfarsitul corpului pachetului, se creeaza un bloc de initializare,
-- care apeleaza procedura INIT_DEPT:
BEGIN
    init_dept;
END;
/
```

6. a) Modificați funcția VALID_DEPT_ID pentru a utiliza tabloul privat definit la exercițiul 5.

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

```
--a)
--In specificatia pachetului, vom avea:
FUNCTION valid_dept_id
    (p_deptid IN dept_pnu.department_id%TYPE)
    RETURN boolean;

--La sfarsitul corpului pachetului, se declara:
FUNCTION valid_dept_id
        (p_deptid IN dept_pnu.department_id%TYPE)
        RETURN boolean
BEGIN
    RETURN valid_dept.EXISTS(p_deptid);
EXCEPTION
    WHEN no_data_found THEN
        RETURN false;
END valid_dept_id;

--b)
BEGIN
    emp_pkg_pnu.add_emp('James', 'Bond', 'bond@g.com', p_deptid => 15);
END;
/
--Inserati un departament nou, avand codul 15.
INSERT INTO dept_pnu (department_id, department_name)
```

```
VALUES (15, 'Security');
COMMIT;

--Incercati din nou adaugarea unui angajat in departamentul având codul 15.
Comentati.

--Introduceți in pachet procedura print_tablou care afiseaza elementele
tabloului privat.
--In specificatia pachetului, vom avea:
PROCEDURE print_tablou;

--La sfarsitul corpului pachetului, se declara procedura:
PROCEDURE print_tablou IS
    BEGIN
        FOR i IN valid_dept.FIRST..valid_dept.LAST LOOP
            IF valid_dept.EXISTS(i) THEN
                DBMS_OUTPUT.PUT_LINE('valid_dept['||i||'] = TRUE');
            END IF;
        END LOOP;
    END print_tablou;

--Actualizati tabloul PL/SQL intern cu noile date ale departamentelor.
BEGIN
    emp_pkg_pnu.init_dept;
--     emp_pkg_pnu.print_tablou;
--     DBMS_OUTPUT.PUT_LINE(CASE WHEN emp_pkg_pnu.valid_dept_id(15) = TRUE
THEN 'TRUE' ELSE 'FALSE' END);
END;
/
--Inserati inregistrarea corespunzatoare angajatului din departamentul 15.

SELECT *
FROM dept_pnu
ORDER BY department_id ASC;

SELECT *
FROM emp_pnu
ORDER BY employee_id DESC;

DELETE FROM emp_pnu
WHERE department_id = 15;

DELETE FROM dept_pnu
WHERE department_id = 15;

COMMIT;
```

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

```
CREATE OR REPLACE PACKAGE p7l5_pnu AS
    CURSOR c_emp(nr NUMBER) RETURN employees%ROWTYPE;
    FUNCTION sal_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;
/
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;
/
```

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 și 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
                DBMS_OUTPUT.PUT_LINE('Combinatie valida de job si
departament');
                  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;
/
BEGIN
    verif_pkg_pnu.verifica ('SA_REP', 10);
--     verif_pkg_pnu.verifica ('IT_PROG', 60);
END;
/
```

II. [Pachete standard]

[DBMS_OUTPUT]

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

```
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 last_name, salary, department_id
    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('----------------');
    DBMS_OUTPUT.PUT_LINE(linie||' si starea '||stare);
    DBMS_OUTPUT.PUT_LINE('----------------');
END;
/
```

[DBMS_JOB]

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

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

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

d) Stergeți job-ul din coada de așteptare.

```
--a
VARIABLE num_job NUMBER
BEGIN
    DBMS_JOB.SUBMIT(
    job => :num_job,
    -- returneaza numarul jobului, printr-o variabila de legatura
    what => 'verif_pkg_pnu.verifica(''SA_REP'', 10);',
    -- codul care va fi executat ca job
    next_date => SYSDATE+30/86400,
    -- data primei executii (dupa 30 secunde)
    interval => 'TRUNC(SYSDATE+1)');
    -- intervalul dintre executiile job-ului
    COMMIT;
END;
/
PRINT num_job
--b
SELECT job, next_date, what
FROM user_jobs;
--c
BEGIN
    DBMS_JOB.RUN(job => x);
    --x este numarul identificat
```

```
    --pentru job-ul care va apartine
END;
/
--d
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 genereaz un raport într-un fișier al sistemului de operare, utilizând pachetul UTL_FILE. Raportul va conține lista angajatilor care au depășit media salariilor din departamentul lor. Procedura va avea doi parametri: directorul de ieșire și numele fișierului 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.FOPEN(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;
/

/*
Observaţie: in 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 D:).
Aceasta operaţie va cere oprirea bazei de date şi repornirea ei.

In  web browser introduceti URL-ul de access Enterprise Manager:
https://hostname:portnumber/em.
Autentificarea la baza de date Enterprise Manager Database trebuie facuta cu
un utilizator autorizat precum SYSTEM.
*/

--creare director la nivelul sistemului de operare
-- D:/OracleW7/Directory/curs_plsql

--setare valoare parametru de initializare Oracle
--UTL_FILE_DIR=D:/OracleW7/Directory/curs_plsql

--atrbuire privilegiu CREATE ANY DIRECTORY utilizatorului
--conectare user SYS
GRANT CREATE ANY DIRECTORY TO curs_plsql;

--conectare user curs_plsql
--definire director
CREATE DIRECTORY curs_plsql AS 'D:/OracleW7/Directory/curs_plsql';

--apel procedura

EXECUTE scriu_fisier('D:/OracleW7/Directory/curs_plsql','ex11.txt');
```

[SQL dinamic, DBMS_SQL]

12. Să se construiască o procedură care folosește SQL dinamic pentru a șterge liniile unui tabel specificat ca parametru. Subprogramul furnizează ca rezultat numărul liniilor șterse (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 ('jobs_pnu', :linii_sterse)
PRINT linii_sterse
```

Obs: Pentru a executa o instrucțiune SQL dinamic poate fi utilizata și comanda EXECUTE IMMEDIATE.

```
INSERT INTO jobs_pnu
SELECT *
FROM jobs;

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;
/

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

13. a) Creați un pachet numit TABLE_PKG_PNU care utilizează SQL nativ pentru crearea sau ștergerea unui tabel și pentru adăugarea, modificarea sau ștergerea de linii din tabel.

Specificația pachetului va conține procedurile următoare:
- PROCEDURE make (table_name VARCHAR2, col_specs VARCHAR2)
- PROCEDURE add_row (table_name VARCHAR2, col_values VARCHAR2, cols VARCHAR2 := NULL)
- PROCEDURE upd_row(table_name VARCHAR2, set_values VARCHAR2, conditions VARCHAR2 := NULL)
- PROCEDURE del_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) Afișați conținutul tabelului contacte_pnu.

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

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

h) Afișați conținutul tabelului, apoi ștergeți tabelul prin intermediul procedurii remove.

```
CREATE OR REPLACE PACKAGE table_pkg_pnu IS
    PROCEDURE execute (stmt VARCHAR2);
    PROCEDURE make_table(table_name VARCHAR2, col_specs VARCHAR2);
    PROCEDURE add_row (table_name VARCHAR2, col_values VARCHAR2, cols VARCHAR2
:= NULL);
    PROCEDURE upd_row(table_name VARCHAR2, set_values VARCHAR2, conditions
VARCHAR2 := NULL);
    PROCEDURE del_row(table_name VARCHAR2, conditions VARCHAR2 := NULL);
    PROCEDURE remove(table_name VARCHAR2);
END table_pkg_pnu;
/
CREATE OR REPLACE PACKAGE BODY table_pkg_pnu IS
    PROCEDURE execute (stmt VARCHAR2) IS
    BEGIN
        DBMS_OUTPUT.PUT_LINE(CHR(10)||'Va fi rulata comanda: '||stmt);
        EXECUTE IMMEDIATE stmt;
        DBMS_OUTPUT.PUT_LINE('Finalizare cu succes');
    END;
    PROCEDURE make_table(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(CHR(10)||stmt);
        DBMS_SQL.PARSE(csr_id, stmt, DBMS_SQL.NATIVE);
        DBMS_SQL.CLOSE_CURSOR(csr_id);
        DBMS_OUTPUT.PUT_LINE('Finalizare cu succes');
    END;
END table_pkg_pnu;
/

DECLARE
    v_table_name VARCHAR2(100):='contacte_pnu';
    v_col_specs VARCHAR2(100):='cod NUMBER(4), nume VARCHAR2(35)';
    v_col_values VARCHAR2(100):='1, ''Geoff Gallus''';
    v_cols VARCHAR2(100):='cod,nume';
    v_conditions_d VARCHAR2(100):='cod = 1';
    v_set_values VARCHAR2(100):='nume = ''Nancy Greenberg''';
    v_conditions_u VARCHAR2(100):='cod = 2';
    TYPE tab IS TABLE OF VARCHAR2(100) INDEX BY BINARY_INTEGER;
    v tab;
    comanda VARCHAR2(500);
    bloc VARCHAR2(500);
BEGIN
    table_pkg_pnu.make_table(v_table_name,v_col_specs);
    --listare structura tabel
    comanda:= 'SELECT COLUMN_NAME
                FROM USER_TAB_COLUMNS
                WHERE LOWER(TABLE_NAME) = :nume_tabel';
    DBMS_OUTPUT.PUT_LINE(CHR(10)||'Va fi rulata comanda: '||comanda);
    EXECUTE IMMEDIATE comanda BULK COLLECT INTO v USING v_table_name;
    DBMS_OUTPUT.PUT_LINE(CHR(10)||'Coloanele sunt: ');
    FOR i IN v.FIRST..v.LAST LOOP
        DBMS_OUTPUT.PUT_LINE(v(i));
    END LOOP;
    table_pkg_pnu.add_row(v_table_name,v_col_values,v_cols);
    --tiparire continut tabel
    bloc:= 'BEGIN
                    FOR j IN  (SELECT '|| v_cols ||
                                ' FROM '|| v_table_name||') LOOP

DBMS_OUTPUT.PUT_LINE(j.'||SUBSTR(v_cols,0,INSTR(v_cols,',')-1)||
                                            ');
```

```
DBMS_OUTPUT.PUT_LINE(j.'||SUBSTR(v_cols,INSTR(v_cols,',')+1,LENGTH(v_cols))||
                                                    ');
                    END LOOP;
            END;';
    table_pkg_pnu.execute(bloc);
    table_pkg_pnu.del_row(v_table_name,v_conditions_d);
    table_pkg_pnu.upd_row(v_table_name,v_conditions_u);
    table_pkg_pnu.remove(v_table_name);
END;
/
```

14. Creați o procedură stocată pentru afișarea conținutului unui tabel. Tratați cazul când tabelul nu există.

```
CREATE OR REPLACE PROCEDURE list_table_content(p_table_name VARCHAR2)
IS
    comanda VARCHAR2(500);
    TYPE tab IS TABLE OF VARCHAR2(100) INDEX BY BINARY_INTEGER;
    v tab;
    v_cols VARCHAR2(500);
BEGIN
    comanda:= 'SELECT COLUMN_NAME
                FROM USER_TAB_COLUMNS
                WHERE TABLE_NAME = UPPER(:nume_tabel)';
    EXECUTE IMMEDIATE comanda BULK COLLECT INTO v USING p_table_name;
    IF v.COUNT = 0 THEN
        RAISE_APPLICATION_ERROR(-20777,'Nu exista tabelul '||p_table_name);
    END IF;
    FOR i IN 1..v.LAST LOOP
        DBMS_OUTPUT.PUT_LINE(v(i));
        v_cols := v_cols || ',' || v(i);
    END LOOP;
    v_cols := SUBSTR(v_cols, 2);
    DBMS_OUTPUT.PUT_LINE(v_cols);
    comanda:=
    'BEGIN
        FOR j IN (SELECT '||v_cols||' FROM '||p_table_name||') LOOP
            DBMS_OUTPUT.PUT(j.'||v(1)||');
            DBMS_OUTPUT.PUT(CHR(9));
            DBMS_OUTPUT.PUT_LINE(j.'||v(2)||');
        END LOOP;
    END;';
    EXECUTE IMMEDIATE comanda;
END;
/
```