Laborator 5 PL/SQL

Trigger-i (declanșatori)

- Un trigger este un bloc PL/SQL asociat unui tabel, view, scheme sau unei baze de date.
- Trigger-ul se executa implicit ori de câte ori are loc un anumit eveniment.
- Pot fi de următoarele tipuri:
  - trigger-i la nivel de aplicație: se declanșează odată cu un anumit eveniment din aplicație;
  - trigger-i la nivel de bază de date: se declanșează atunci când apare un eveniment asupra datelor (de ex, LMD) sau un eveniment sistem (logon, shutdown) asupra unei scheme sau asupra bazei de date.
- Instrucțiunea pentru crearea trigger-ilor LMD conține următoarele informații:
  - timpul declanșării trigger-ului în raport cu evenimentul:
    - pentru tabele: BEFORE, AFTER
    - pentru view-uri nemodificabile: INSTEAD OF
  - evenimentul declanșator: INSERT, UPDATE, DELETE
  - numele tabelului
  - tipul trigger-ului – precizează de câte ori se execută corpul acestuia; trigger-ul poate fi la nivel de:
    - instrucțiune (statement): corpul triggerului se execută o singură dată pentru evenimentul declanșator. Un astfel de trigger se declanșează chiar dacă nici o linie nu este afectată.
    - linie (row): corpul triggerului se declanșează o dată pentru fiecare linie afectată de către evenimentul declanșator. Un astfel de trigger nu se execut dacă evenimentul declanșator nu afectează nici o linie.
  - clauza WHEN - precizează o condiție restrictivă
  - corpul triggerului (blocul PL/SQL)
- Sintaxa comenzii de creare a unui trigger LMD este:

```
CREATE [OR REPLACE] TRIGGER [schema.]nume_trigger
      {BEFORE | AFTER}
      [INSTEAD OF]
      {DELETE | INSERT | UPDATE [OF coloana[, coloana ...] ] }
      [OR {DELETE | INSERT | UPDATE [OF coloana[, coloana ...] ] ...} ON
      [schema.]nume_tabel
      [REFERENCING {OLD [AS] vechi NEW [AS] nou
                  | NEW [AS] nou OLD [AS] vechi } ]
```

```
      [FOR EACH ROW]
      [WHEN (condiție) ]
      corp_trigger;
```

- Informații despre trigger-i se găsesc în următoarele vizualizări ale dicționarului datelor: USER_TRIGGERS, USER_TRIGGER_COL, ALL_TRIGGERS, DBA_TRIGGERS. Modificarea unui declansator constă din redenumirea, recompilarea, activarea sau dezactivarea acestuia și se realizează prin comenzi de forma:
  `ALTER TRIGGER nume_trigger ENABLE;`
  `ALTER TRIGGER nume_trigger DISABLE;`
  `ALTER TRIGGER nume_trigger COMPILE;`
  `ALTER TRIGGER nume_trigger RENAME TO nume_nou;`
- Activarea și dezactivarea tuturor trigger-ilor asociați unui tabel se realizează prin comenzile:
  `ALTER TABLE nume_tabel DISABLE ALL TRIGGERS;`
  `ALTER TABLE nume_tabel ENABLE ALL TRIGGERS;`
- Eliminarea unui declanșator se face prin
  `DROP TRIGGER nume_trigger;`
- Sintaxa pentru crearea unui declansator sistem este următoarea:

```
CREATE [OR REPLACE] TRIGGER [schema.]nume_declansator
      {BEFORE | AFTER}
      {lista_evenimente_LDD | lista_evenimente_bază} ON {DATABASE | SCHEMA}
      [WHEN (condiție) ]
      corp_declansator;
```

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

Exerciții:

1. Să se creeze un trigger care asigură ca inserarea de angajați în tabelul EMP_PNU se poate realiza numai în zilele lucrătoare, între orele 8-18.
Obs: Trigger-ul nu are legatură directă cu datele => este

trigger la nivel de instrucțiune

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

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

DROP TRIGGER b_i_emp_pnu;
```

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

```
CREATE OR REPLACE TRIGGER b_i_emp_pnu
      BEFORE INSERT OR UPDATE OR DELETE ON emp_pnu
BEGIN
    IF (TO_CHAR(SYSDATE, 'dy') IN ('sat', 'sun')) OR
        (TO_CHAR(SYSDATE, 'HH24:MI') NOT BETWEEN '08:00' AND '18:00') THEN
            IF DELETING THEN
                RAISE_APPLICATION_ERROR (-20501, 'Nu se pot sterge
inregistrari decat in timpul orelor de luru');
            ELSIF INSERTING THEN
                RAISE_APPLICATION_ERROR (-20500, 'Nu se pot adauga
inregistrari decat in timpul orelor de lucru');
            ELSIF UPDATING ('SALARY') THEN
                RAISE_APPLICATION_ERROR (-20502, 'Nu se poate actualiza campul
SALARY decat in timpul orelor de lucru');
            ELSE
                RAISE_APPLICATION_ERROR (-20503, 'Nu se pot actualiza
inregistrari decat in timpul orelor de lucru');
            END IF;
    END IF;
END;
/

ALTER TRIGGER b_i_emp_pnu DISABLE;
DROP TRIGGER b_i_emp_pnu;
```

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

Obs: Trigger-ul se declanșează de un număr de ori = nr de înregistrări inserate sau al căror câmp salary este modificat (deci are legtură cu datele din tabel) => este

trigger la nivel de linie

```
CREATE OR REPLACE TRIGGER b_i_u_emp_pnu
    BEFORE INSERT OR UPDATE OF salary ON emp_pnu
    FOR EACH ROW
BEGIN
    IF NOT (:NEW.job_id IN ('AD_PRES', 'AD_VP')) AND :NEW.salary > 15000
THEN
```

```
        RAISE_APPLICATION_ERROR (-20202, 'Angajatul nu poate castiga aceasta
suma');
    END IF;
END;
/
INSERT INTO emp_pnu (employee_id, last_name, first_name, email, hire_date,
job_id, salary, department_id)
VALUES (300, 'Smith', 'Robert', 'rsmith', SYSDATE, 'IT_PROG', 15001, 60);

DROP TRIGGER b_i_u_emp_pnu;
```

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

```
--Varianta 1:
CREATE OR REPLACE TRIGGER verifica_salariu_pnu
    BEFORE UPDATE OF salary ON emp_pnu
    FOR EACH ROW
    WHEN (NEW.salary < OLD.salary)
BEGIN
    RAISE_APPLICATION_ERROR (-20222, 'valoarea unui salariu nu poate fi
micsorata');
END;
/

UPDATE emp_pnu
SET salary = salary/2;

--Varianta 2:
CREATE OR REPLACE TRIGGER verifica_salariu_pnu
    BEFORE UPDATE OF salary ON emp_pnu
    FOR EACH ROW
BEGIN
    IF (:NEW.salary < :OLD.salary) THEN
        RAISE_APPLICATION_ERROR (-20222, 'valoarea unui salariu nu poate fi
micsorata');
    END IF;
END;
/

--Varianta 3:
CREATE OR REPLACE PROCEDURE p4l6_pnu
IS
BEGIN
    RAISE_APPLICATION_ERROR (-20222, 'valoarea unui salariu nu poate fi
micsorata');
END;
/

CREATE OR REPLACE TRIGGER verifica_salariu_pnu
```

```
BEFORE UPDATE OF salary ON emp_pnu
    FOR EACH ROW
    WHEN (NEW.salary < OLD.salary)
CALL p4l6_pnu;

DROP TRIGGER verifica_salariu_pnu;
```

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

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

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

UPDATE emp_pnu
SET salary = salary + 1000
WHERE job_id = 'SA_REP';

SELECT e1.salary, e2.salary, e1.commission_pct, e2.commission_pct
FROM employees e1 JOIN emp_pnu e2 ON e1.employee_id = e2.employee_id
WHERE e1.job_id = 'SA_REP' and e2.job_id = 'SA_REP';

DROP TRIGGER b_i_u_sal_emp_pnu;
```

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

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

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

UPDATE jobs_pnu
SET min_salary = 3000
WHERE min_salary = 2000;

DROP TRIGGER verifica_sal_job_pnu;
```

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

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

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

DROP TRIGGER check_sal_pnu;
```

Ce se obține și de ce? Modificați trigger-ul astfel încât să funcționeze corect.
Obs: **Tabelul** este **mutating**. ORA-04091: table [schema].EMP_PNU is mutating, trigger/function may not see it

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

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

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

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

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

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

```
/

SELECT *
FROM  emp_pnu
WHERE department_id = 90;

SELECT *
FROM dept_pnu
WHERE department_id = 90;

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

UPDATE emp_pnu
SET    salary = salary + 1000
WHERE  employee_id = 300;

DELETE FROM emp_pnu
WHERE  employee_id = 300;

DROP TRIGGER calcul_total_pnu;

ALTER TABLE dept_pnu
DROP COLUMN total_sal;
```

9. Să se creeze două tabele new_emp_pnu și new_dept_pnu pe baza tabelelor employees și departments. Să se creeze un view view_emp_pnu, care selectează codul, numele, salariul, codul departamentului, email-ul, codul job-ului, numele departamentului și codul locației pentru fiecare angajat. Să se creeze un

trigger de tip INSTEAD OF

care, în locul inserării unei linii direct în view, adaugă înregistrări corespunzătoare în tabelele new_emp_pnu și new_dept_pnu. Similar, atunci când o linie este modificată sau ștearsă prin intermediul vizualizării, liniile corespunzătoare din tabelele new_emp_pnu și new_dept_pnu sunt afectate.

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

ALTER TABLE new_dept_pnu ADD CONSTRAINT new_dept_pnu_pk PRIMARY
KEY(department_id);

CREATE TABLE new_emp_pnu AS
```

```
SELECT employee_id, last_name, salary,
       department_id, email, job_id, hire_date
FROM employees;

ALTER TABLE new_emp_pnu ADD CONSTRAINT new_emp_pnu_pk PRIMARY
KEY(employee_id);
ALTER TABLE new_emp_pnu ADD CONSTRAINT new_emp_dept_pnu_fk
    FOREIGN KEY(department_id) REFERENCES new_dept_pnu (department_id);

CREATE OR REPLACE VIEW view_emp_pnu AS
SELECT e.employee_id, e.last_name, e.salary, e.department_id,
       e.email, e.job_id, d.department_name, d.location_id, d.total_dept_sal
FROM new_emp_pnu e, new_dept_pnu d
WHERE e.department_id = d.department_id;

SELECT * FROM new_emp_pnu;
SELECT * FROM new_dept_pnu;
SELECT * FROM view_emp_pnu;

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

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

```
        UPDATE new_dept_pnu
        SET total_dept_sal = total_dept_sal - :OLD.salary
        WHERE department_id = :OLD.department_id;
        UPDATE new_dept_pnu
        SET total_dept_sal = total_dept_sal + :NEW.salary
        WHERE department_id = :NEW.department_id;
    END IF;
END;
/

-- adaugarea unui nou angajat
SELECT * FROM new_dept_pnu WHERE department_id = 10;
INSERT INTO view_emp_pnu
VALUES (400, 'N1', 3000, 10, 'n1@g.com', 'SA_REP', 'Nume dept', 1000, 0);
SELECT * FROM new_emp_pnu WHERE employee_id = 400;
SELECT * FROM new_dept_pnu WHERE department_id = 10;
-- modificarea salariului unui angajat
UPDATE view_emp_pnu
SET salary = salary + 1000
WHERE employee_id = 400;
SELECT * FROM new_emp_pnu WHERE employee_id = 400;
SELECT * FROM new_dept_pnu WHERE department_id = 10;
-- modificarea departamentului unui angajat
SELECT * FROM view_emp_pnu WHERE department_id=90;
UPDATE view_emp_pnu
SET department_id = 90
WHERE employee_id = 400;
SELECT * FROM new_emp_pnu WHERE employee_id = 400;
SELECT * FROM new_dept_pnu WHERE new_dept_pnu.department_id IN (10,90);
-- eliminarea unui angajat
DELETE FROM view_emp_pnu WHERE employee_id = 400;
SELECT * FROM new_emp_pnu WHERE employee_id = 400;
SELECT * FROM new_dept_pnu WHERE department_id = 90;
DROP TRIGGER new_emp_dept_pnu;
DROP VIEW view_emp_pnu;
DROP TABLE new_emp_pnu;
DROP TABLE new_dept_pnu;
```

10. Să se implementeze cu ajutorul unui declanșator restricția că într-un departament pot lucra maximum 45 de angajați.

Tratarea erorii MUTATING TABLE

```
SELECT *
FROM USER_TRIGGERS;

DELETE FROM emp_pnu;

ALTER TABLE emp_pnu DISABLE ALL TRIGGERS;
```

```
INSERT INTO emp_pnu
SELECT *
FROM employees;

/*
cu un singur trigger => eroare, deoarece declanșatorul consultă chiar tabelul
la care este asociat; emp_pnu e mutating table
*/
CREATE OR REPLACE TRIGGER mutating_pnu
    BEFORE INSERT OR UPDATE OF department_id ON emp_pnu
    FOR EACH ROW
DECLARE
    v_max CONSTANT NUMBER := 45;
    v_nr NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_nr
    FROM emp_pnu
    WHERE department_id = :NEW.department_id;
    IF v_nr + 1 > v_max THEN
        RAISE_APPLICATION_ERROR(-20000,'Prea multi angajati in departamentul
avand codul ' ||:NEW.department_id);
    END IF;
END mutating_pnu;
/
--merge
INSERT INTO emp_pnu VALUES
(employees_seq.NEXTVAL,'Prenume','Nume','a','t',SYSDATE,'IT_PROG',
10000,0.1,100,50);
--mutating table
INSERT INTO emp_pnu
SELECT
employees_seq.NEXTVAL,'Prenume','Nume','a','t',SYSDATE,'IT_PROG',10000,0.1,100
,50
FROM dual;
--fara PK in emp_pnu
--insert multiplu => mutating table
INSERT INTO emp_pnu
SELECT * FROM employees WHERE department_id = 50;
--mutating table
UPDATE emp_pnu
SET department_id = 50
WHERE department_id = 20;

SELECT department_id, count(*)
FROM emp_pnu
GROUP BY department_id;

ALTER TRIGGER mutating_pnu DISABLE;
-- o solutie ar fi consultarea unei copii a tabelului emp_pnu, neindicata
-- a 2-a adaugarea clauzei pragma autonomous_transaction, neindicata
-- a 3-a pachet si 2 trigger-i,
```

```
-- a 4-a trigger compus

CREATE OR REPLACE TRIGGER tranzactie_autonoma_pnu
    BEFORE INSERT OR UPDATE OF department_id ON emp_pnu
    FOR EACH ROW
DECLARE
    PRAGMA AUTONOMOUS_TRANSACTION;
    v_max CONSTANT NUMBER := 45;
    v_nr NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_nr
    FROM emp_pnu
    WHERE department_id = :NEW.department_id;
    IF v_nr + 1 > v_max THEN
        RAISE_APPLICATION_ERROR(-20000,'Prea multi angajati in ' ||
                                       'departamentul avand codul ' ||
:NEW.department_id);
    END IF;
END tranzactie_autonoma_pnu;
/
--executare operatii insert si update anterioare
--Ce se intampla cu operatiile pt. department_id = 80?
--Sunt permise inserari/actualizari chiar daca nr. de angajati depaseste 45!!!
ALTER TRIGGER tranzactie_autonoma_pnu DISABLE;

/*
Tabelul emp_pnu este mutating doar pentru un declansator la nivel de linie.
O solutie pentru aceasta problema este crearea a doi declansatori, unul la
nivel de linie si altul la nivel de instructiune.
- in declansatorul la nivel de instructiune va fi interogat tabelul emp_pnu si
va fi utilizat un tablou indexat in interiorul unui pachet pentru
inregistrarea nr de angajati pt. fiecare departament.
- in declansatorul la nivel de linie se verifica daca este respecta conditia
pentru valoarea lui :NEW.department_id, dar nu va fi interogat tabelul
emp_pnu.
*/
CREATE OR REPLACE PACKAGE pachet_pnu
AS
      TYPE tip_rec IS RECORD
            (id emp_pnu.department_id%TYPE, nr NUMBER(2));
      TYPE tip_ind IS TABLE OF tip_rec INDEX BY PLS_INTEGER;
      t tip_ind;
      contor NUMBER(2):=0;
END;
/

CREATE OR REPLACE TRIGGER comanda_pnu
      BEFORE INSERT OR UPDATE OF department_id ON emp_pnu
BEGIN
      pachet_pnu.contor:=0;
      SELECT department_id, COUNT(*)
```

```
      BULK COLLECT INTO pachet_pnu.t
      FROM emp_pnu
      GROUP BY department_id;
END comanda_pnu;
/

CREATE OR REPLACE TRIGGER linie_pnu
      BEFORE INSERT OR UPDATE OF department_id ON emp_pnu
      FOR EACH ROW
BEGIN
      FOR i IN 1..pachet_pnu.t.LAST LOOP
            IF pachet_pnu.t(i).id = :NEW.department_id AND
            pachet_pnu.t(i).nr + pachet_pnu.contor = 45 THEN
                  RAISE_APPLICATION_ERROR(-20000,'Departamentul
'||:NEW.department_id||'  depaseste numarul '||
                        ' maxim de angajati permis');
            END IF;
      END LOOP;
      pachet_pnu.contor:=pachet_pnu.contor+1;
END linie_pnu;
/
--executare operatii insert si update anterioare
BEGIN
    --DBMS_OUTPUT.PUT_LINE(pachet_pnu.t(i).nr);
    FOR i IN 1..pachet_pnu.t.LAST LOOP
        DBMS_OUTPUT.PUT_LINE(pachet_pnu.t(i).id||' '||
        pachet_pnu.t(i).nr);
    END LOOP;
    --pachet_pnu.t.DELETE;
    --pachet_pnu.contor:=0;
END;
/
CREATE OR REPLACE TRIGGER compus_pnu
    FOR INSERT OR UPDATE OF department_id ON emp_pnu
COMPOUND TRIGGER
    TYPE tip_rec IS RECORD
            (id emp_pnu.department_id%TYPE, nr NUMBER(2));
      TYPE tip_ind IS TABLE OF tip_rec INDEX BY PLS_INTEGER;
      t tip_ind;
      contor NUMBER(2):=0;
    BEFORE STATEMENT IS
    BEGIN
      contor:=0;
        SELECT department_id, COUNT(*)
        BULK COLLECT INTO t
        FROM emp_pnu
        GROUP BY department_id;
    END BEFORE STATEMENT;
    BEFORE EACH ROW IS
    BEGIN
        FOR i IN 1..t.LAST LOOP
```

```
            IF t(i).id = :NEW.department_id AND
            t(i).nr + contor = 45 THEN
                RAISE_APPLICATION_ERROR(-20000,'Departamentul
'||:NEW.department_id||'  depaseste numarul '||
                    ' maxim de angajati permis');
            END IF;
        END LOOP;
        contor:=contor+1;
    END BEFORE EACH ROW;
END compus_pnu;
/
--executare operatii insert si update anterioare
DROP TRIGGER mutating_pnu;
DROP TRIGGER tranzactie_autonoma_pnu;
DROP PACKAGE pachet_pnu;
DROP TRIGGER linie_pnu;
DROP TRIGGER comanda_pnu;
DROP TRIGGER compus_pnu;
```

11. Să se creeze un declanșator care:
a) dacă este eliminat un departament, va șterge toți angajații care lucrează în departamentul respectiv;
b) dacă se schimbă codul unui departament, va modifica această valoare pentru fiecare angajat care lucrează în departamentul respectiv.

Obs: Declanșatorul realizează

constrângerea de integritate ON DELETE CASCADE sau UPDATE,

adică ștergerea sau modificarea cheii primare a unui tabel „părinte“ se va reflecta și asupra înregistrărilor corespunzătoare din tabelul „copil“.

Obs: Se presupune că asupra tabelului emp_pnu nu există o constrângere de integritate FOREIGN KEY (department_id) REFERENCES dept_pnu(department_id) ON DELETE CASCADE.

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

```
DELETE FROM emp_pnu;
DELETE FROM dept_pnu;

INSERT INTO dept_pnu SELECT *
FROM departments;

INSERT INTO emp_pnu SELECT *
FROM employees;
```

```
CREATE OR REPLACE TRIGGER dep_cascada_pnu
    BEFORE DELETE OR UPDATE OF department_id ON dept_pnu
    FOR EACH ROW
BEGIN
    IF DELETING THEN
        DELETE FROM emp_pnu
        WHERE department_id = :OLD.department_id;
    END IF;
    IF UPDATING AND :OLD.department_id != :NEW.department_id THEN
        UPDATE emp_pnu
        SET department_id = :NEW.department_id
        WHERE department_id = :OLD.department_id;
    END IF;
END dep_cascada_pnu;
/
DELETE FROM dept_pnu
WHERE department_id = 10;

SELECT *
FROM dept_pnu
WHERE department_id = 10;

SELECT *
FROM emp_pnu
WHERE department_id = 10;

UPDATE dept_pnu
SET department_id = 12
WHERE department_id = 20;

SELECT *
FROM dept_pnu
WHERE department_id = 12;

SELECT *
FROM emp_pnu
WHERE department_id = 12;

ALTER TABLE dept_pnu ADD CONSTRAINT dept_pnu_pk PRIMARY KEY (department_id);
ALTER TABLE emp_pnu ADD CONSTRAINT emp_pnu_pk PRIMARY KEY (employee_id);
ALTER TABLE emp_pnu ADD CONSTRAINT emp_dept_pnu_fk FOREIGN KEY(department_id)
REFERENCES dept_pnu(department_id) ON DELETE CASCADE;
--executare operatii delete si update anterioare
--delete => mutating table
--update merge

ALTER TABLE emp_pnu DROP CONSTRAINT emp_dept_pnu_fk;
ALTER TABLE emp_pnu ADD CONSTRAINT emp_dept_pnu_fk FOREIGN KEY(department_id)
REFERENCES dept_pnu(department_id) ON DELETE SET NULL;
ALTER TABLE emp_pnu ADD CONSTRAINT emp_dept_pnu_fk FOREIGN KEY(department_id)
REFERENCES dept_pnu(department_id);
```

```
DROP TRIGGER dep_cascada_pnu;
/*
Daca constrangerea de cheie externa nu are optiuni de stergere specificate,
atunci comanda delete este permisa.
*/
```

12. Să se creeze un declanșator prin care să nu se permită ștergerea informațiilor din tabelul emp_pnu de către utilizatorul curent. Dezactivați, iar apoi activați trigger-ul creat. Testați, iar apoi suprimați acest trigger.

```
CREATE OR REPLACE TRIGGER p13l6_pnu
    BEFORE DELETE ON emp_pnu
BEGIN
    IF USER = UPPER('grupa4051') THEN
        RAISE_APPLICATION_ERROR(-20900,'Nu este permisa stergerea de catre '
||USER);
    END IF;
END;
/
ALTER TRIGGER p13l6_pnu DISABLE;
DELETE FROM emp_pnu WHERE employee_id = 100;
ALTER TRIGGER p13l6_pnu ENABLE;
DELETE FROM emp_pnu WHERE employee_id = 100;
DROP TRIGGER p13l6_pnu;
```

13. Să se creeze un tabel care conține următoarele câmpuri: user_id, nume_bd, eveniment_sis, nume_obj, data. Să se creeze un

trigger sistem (la nivel de schemă)

care să introducă date în acest tabel după ce utilizatorul a folosit o comandă LDD. Testați, iar apoi suprimați trigger-ul.

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

```
SELECT * FROM log_pnu;
DROP TRIGGER p14l6_pnu;
```