

## **Laborator 6 PL/SQL**

### **Declanșatori**

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

#### **Tipuri de declanșatori:**

- la nivel de bază de date – se declanșează la un eveniment legat de date (*LMD*) sau de sistem (*logon, shutdown*);
- la nivel de aplicație – se declanșează la apariția unui eveniment legat de o aplicație particulară.
- Sintaxa comenzii de creare a unui declanșator este următoarea:

```
CREATE [OR REPLACE] TRIGGER [schema.]nume_declanșator
{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_declanșator;
```

- Informații despre declanșatori se pot obține interogând vizualizările
  - *USER\_TRIGGERS, ALL\_TRIGGERS, DBA\_TRIGGERS*
  - *USER\_TRIGGER\_COL*
- Înainte de a scrie codul *PL/SQL*, trebuie să stabilim:
  - momentul când este executat *trigger*-ul: *BEFORE, AFTER, INSTEAD OF*
  - ce fel de acțiuni îl declanșează: *INSERT, UPDATE, DELETE*
  - tipul *trigger*-ului (de câte ori se execută): la nivel de instrucțiune sau la nivel de linie (*FOR EACH ROW*).

- Dezactivarea, respectiv activarea declanșatorilor se realizează prin următoarele comenzi:

```
ALTER TABLE nume_tabel
DISABLE ALL TRIGGERS;

ALTER TABLE nume_tabel
ENABLE ALL TRIGGERS;

ALTER TRIGGER nume_trig ENABLE;

ALTER TRIGGER nume_trig DISABLE;
```

- Eliminarea unui declanșator se face prin

```
DROP TRIGGER nume_trig;
```

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

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

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

#### Varianta 1

```
CREATE OR REPLACE TRIGGER trig21_***
    BEFORE UPDATE OF salary ON emp_***
    FOR EACH ROW
BEGIN
    IF (:NEW.salary < :OLD.salary) THEN
        RAISE_APPLICATION_ERROR(-20002, 'salariul nu poate fi micșorat');
    END IF;
END;
/
UPDATE emp_***
SET     salary = salary-100;
DROP TRIGGER trig21_***;
```

#### Varianta 2

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

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

```

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

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

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

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

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

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

DROP TRIGGER trig3_***;

```

**4. a. Creați tabelul *info\_dept\_\*\*\** cu următoarele coloane:**

- id (codul departamentului) – cheie primară;
- nume\_dept (numele departamentului);
- plati (suma alocată pentru plata salariilor angajaților care lucrează în departamentul respectiv).

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

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

```

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

```

```

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

```

```

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

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

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

UPDATE emp_***
SET salary = salary + 1000
WHERE employee_id=90;

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

DELETE FROM emp_***
WHERE employee_id=90;

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

DROP TRIGGER trig4_***;

```

- 5. a.** Creați tabelul *info\_emp\_\*\*\** cu următoarele coloane:
- id (codul angajatului) – cheie primară;
  - nume (numele angajatului);
  - prenume (prenumele angajatului);
  - salariu (salariul angajatului);
  - id\_dept (codul departamentului) – cheie externă care referă tabelul *info\_dept\_\*\*\**.
- b.** Introduceți date în tabelul creat anterior corespunzătoare informațiilor existente în schemă.
- c.** Creați vizualizarea *v\_info\_\*\*\** care va conține informații complete despre angajați și departamentele acestora. Folosiți cele două tabele create anterior, *info\_emp\_\*\*\**, respectiv *info\_dept\_\*\*\**.
- d.** Se pot realiza actualizări asupra acestei vizualizări? Care este tabelul protejat prin cheie? Consultați vizualizarea *user\_updatable\_columns*.

- e. Definiți un declanșator prin care actualizările ce au loc asupra vizualizării se propagă automat în tabelele de bază (**declanșator INSTEAD OF**). Se consideră că au loc următoarele actualizări asupra vizualizării:
  - se adaugă un angajat într-un departament deja existent;
  - se elimină un angajat;
  - se modifică valoarea salariului unui angajat;
  - se modifică departamentul unui angajat (codul departamentului).
- f. Verificați dacă declanșatorul definit funcționează corect.
- g. Modificați declanșatorul definit astfel încât să permită și următoarele operații:
  - se adaugă un angajat și departamentul acestuia (departamentul este nou);
  - se adaugă doar un departament.
- h. Verificați dacă declanșatorul definit funcționează corect.
- i. Modificați prin intermediul vizualizării numele unui angajat. Ce observați?
- j. Modificați declanșatorul definit anterior astfel încât să permită propagarea în tabelele de bază a actualizărilor realizate asupra numelui și prenumelui angajatului, respectiv asupra numelui de departament.
- k. Verificați dacă declanșatorul definit funcționează corect.

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

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

```
CREATE OR REPLACE TRIGGER trig5_***
  INSTEAD OF INSERT OR DELETE OR UPDATE ON v_info_***
  FOR EACH ROW
BEGIN

IF INSERTING THEN
  -- inserarea in vizualizare determina inserarea
  -- in info_emp_*** si reactualizarea in info_dept_***
  -- se presupune ca departamentul exista
  INSERT INTO info_emp_***
  VALUES (:NEW.id, :NEW.num, :NEW.prenume, :NEW.salariu,
          :NEW.id_dept);

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

ELSIF DELETING THEN
  -- stergerea unui salariat din vizualizare determina
  -- stergerea din info_emp_*** si reactualizarea in
  -- info_dept_***
```

```

DELETE FROM info_emp ***
WHERE id_dept = :OLD.id_dept;

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

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

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

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

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

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

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

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

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

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

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

SELECT * FROM info_emp *** WHERE id=400;

SELECT * FROM info_dept *** WHERE id=10;

```

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

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

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

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

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

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

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

-- eliminarea unui angajat
DELETE FROM v_info_*** WHERE id = 400;

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

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

```
DROP TRIGGER trig5_***;
```

- 6.** Definiți un declanșator care să nu se permită ștergerea informațiilor din tabelul *emp\_\*\*\** de către utilizatorul *grupa\*\*\**.

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

- 7. a.** Creați tabelul *audit\_\*\*\** cu următoarele câmpuri:

- utilizator (numele utilizatorului);
- nume\_bd (numele bazei de date);
- eveniment (evenimentul sistem);
- nume\_obiect (numele obiectului);
- data (data producerii evenimentului).

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

*Obs.* Sintaxa pentru crearea unui declanșator sistem este următoarea

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

```
lista_evenimente_LDD - CREATE, DROP, ALTER)
lista_evenimente_bază - STARTUP, SHUTDOWN, LOGON, LOGOFF,
SERVERERROR, SUSPEND
```

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

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

```
CREATE INDEX ind_*** ON info_emp_***(nume);
DROP INDEX ind_***;
SELECT * FROM audit_***;
DROP TRIGGER trig7_***;
```

**8.** Definiți un declanșator care să nu permită modificarea:

- valorii salariului maxim astfel încât acesta să devină mai mic decât media tuturor salariilor;
- valorii salariului minim astfel încât acesta să devină mai mare decât media tuturor salariilor.

*Observație:*

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

Este necesar să se definească doi declanșatori:

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



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

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

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

SELECT AVG(salary)
FROM emp_***;

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

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

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

## EXERCIIȚII

1. Definiți un declanșator care să permită ștergerea informațiilor din tabelul *dept\_\*\*\** decât în dacă utilizatorul este SCOTT.
2. Creați un declanșator prin care să nu se permită mărirea comisionului astfel încât să depășească 50% din valoarea salariului.
3.
  - a. Introduceți în tabelul *info\_dept\_\*\*\** coloana *numar* care va reprezenta pentru fiecare departament numărul de angajați care lucrează în departamentul respectiv. Populați cu date această coloană pe baza informațiilor din schemă.
  - b. Definiți un declanșator care va actualiza automat această coloană în funcție de actualizările realizate asupra tabelului *info\_emp\_\*\*\**.
4. Definiți un declanșator cu ajutorul căruia să se implementeze restricția conform căreia într-un departament nu pot lucra mai mult de 45 persoane (se vor utiliza doar tabelele *emp\_\*\*\** și *dept\_\*\*\** fără a modifica structura acestora).
5.
  - a. Pe baza informațiilor din schemă creați și populați cu date următoarele două tabele:
    - *emp\_test\_\*\*\** (*employee\_id* – cheie primară, *last\_name*, *first\_name*, *department\_id*);
    - *dept\_test\_\*\*\** (*department\_id* – cheie primară, *department\_name*).
  - b. Definiți un declanșator care va determina ștergeri și modificări în cascadă:
    - ștergerea angajaților din tabelul *emp\_test\_\*\*\** dacă este eliminat departamentul acestora din tabelul *dept\_test\_\*\*\**;
    - modificarea codului de departament al angajaților din tabelul *emp\_test\_\*\*\** dacă departamentul respectiv este modificat în tabelul *dept\_test\_\*\*\**.

Testați și rezolvați problema în următoarele situații:

  - nu este definită constrângere de cheie externă între cele două tabele;
  - este definită constrângere de cheie externă între cele două tabele;
  - este definită constrângere de cheie externă între cele două tabele cu opțiunea ON DELETE CASCADE;
  - este definită constrângere de cheie externă între cele două tabele cu opțiunea ON DELETE SET NULL.
6.
  - a. Creați un tabel cu următoarele coloane:
    - *user\_id* (SYS.LOGIN\_USER);
    - *nume\_bd* (SYS.DATABASE\_NAME);
    - *erori* (DBMS\_UTILITY.FORMAT\_ERROR\_STACK);
    - *data*.
  - b. Definiți un declanșator sistem (la nivel de bază de date) care să introducă date în acest tabel referitoare la erorile apărute.