Laborator 8
Limbajul PL/SQL. Triggere (Declansatoare).

Trigger-ul reprezinta o categorie speciala de proceduri stocate, fiind asociat unei tabele, declanşandu-se automat (apelandu-se implicit) ca urmare a execuţiei unei comenzi SQL INSERT, DELETE, UPDATE pe tabela asociată. Spre deosebire de proedurile stocate, trigger-ele nu permit parametri de intrare.

Sintaxa generală a comenzii de creare a unui trigger este:

```
CREATE [OR REPLACE] TRIGGER nume_trigger
{BEFORE / AFTER} eveniment_declansator ON nume_tabela
[REFERENCING {NEW AS new_row_name/ OLD AS new_row_name }]
[FOR EACH ROW]
[WHEN (expresie_conditionala)]
[DECLARE
sectiune declarativa optionala]
BEGIN
      corp trigger (sectiune executabila)
[EXCEPTION
      sectiune optionala de tratare exceptii]
END;
/  --compilarea trigger-ului
```

Construcţia CREATE TRIGGER poate fi definită pentru operaţii de tip UPDATE, INSERT sau DELETE (eveniment_declanşator = comandă/comenzi DML). Astfel, în definiţia unui trigger, se pot folosi construcţii sintactice de genul:

```
... BEFORE INSERT ON nume_tabela
... AFTER INSERT OR UPDATE OR DELETE ON nume_tabela
... AFTER UPDATE OF coloana1, coloana2 ON nume_tabela
```

Ultima construcţie permite precizarea ca eveniment declanşator al trigger-ului a unei operaţii UPDATE efectuate doar pe anumite coloane ale tabelei (clauza *OF coloana1, coloana2*). O clauza [OF lista_coloane] poate fi asociată doar unui eveniment/operaţie de tip UPDATE, restricţionând aria de declanşare a trigger-ului. Evenimentul UPDATE fără o clauza OF conduce la o declanşare a trigger-ului indiferent de coloana/coloanele afectate de operaţie.

Observatii:
- Nu este posibilă definirea unui declanşator care să se activeze în urma unei operaţii SELECT (care de altfel nu este o comandă DML).
- De asemenea, nu este posibilă utilizarea în corpul unui trigger a unei clasice comenzi SQL SELECT care returnează date spre un client apelant (în schimb este posibilă o comandă SELECT...INTO, care permite încărcarea în variabile a unor date returnate prin interogarea unei tabele).

Se pot distinge două categorii de triggere -BEFORE sau AFTER- după cum trigger-ul se declanşează înainte sau după apariţia evenimentului (mai precis înainte sau după execuţia comenzii DML care a declanşat trigger-ul).

Clauza FOR EACH ROW (definită ca opţională, dar obligatorie în anumite situaţii) precizează dacă trigger-ul se declanşează o singură dată pentru un eveniment (comandă DML) sau se declanşează pentru fiecare linie afectată de eveniment. Spre exemplu, o comandă UPDATE, tratată ca eveniment declanşator al trigger-ului, poate afecta una sau mai

multe linii ale tabelei pe care este definit trigger-ul. Lipsa clauzei FOR EACH ROW conduce la declanşarea trigger-ului o singură dată, per comandă. Utilizarea clauzei conduce la o declanşare multiplă a trigger-ului, câte o dată pentru fiecare linie afectată de comanda UPDATE.

În cazul trigger-elor definite cu acţiune la nivel de linie (cu clauza FOR EACH ROW), prin intermediul variabilelor speciale NEW şi OLD pot fi referite date din liniile "vechi" (anterioare operaţiei declanşatoare), respectiv date din liniile noi (ulterioare operaţiei declanşatoare) ale tabelei, dupa cum se poate vedea:

- la o declanşare a trigger-ului în urma unei operaţii INSERT, doar referirile precedate de variabila NEW vor conţine date concrete (linia NOU adăugată), în timp ce orice referire cu variabila OLD va conţine NULL (linia neexistând înaintea operaţiei INSERT);
- la o declanşare a trigger-ului în urma unei operaţii DELETE, doar referirile precedate de variabila OLD vor conţine date concrete (linia VECHE, ştearsă), în timp ce orice referire cu variabila NEW va conţine NULL (linia ne mai existând după operaţia DELETE);
- doar la o declanşare a trigger-ului în urma unei operaţii UPDATE, atât referirile OLD cât şi NEW vor conţine date concrete (vechile valori, respectiv noile valori, chiar dacă unele dintre acestea nu se modifică practic în urma UPDATE-ului).

Clauza REFERENCING (optionala) permite utilizarea altor nume pentru cele două variabile speciale NEW şi OLD.

Clauza WHEN (optionala) poate impune o condiţie suplimentară pentru declanşarea trigger-ului.

***Observaţii*:**
- În cadrul corpului unui trigger variabilele NEW şi OLD trebuie precedate de caracterul două-puncte (":"). Doar în cadrul clauzei WHEN nu este necesară precedarea lor de acest caracter.
- De asemenea, variabilele NEW şi OLD nu pot fi referite în cadrul unui trigger declanşat o singură dată, la nivel de comandă (trigger fără clauza FOR EACH ROW).

Problema "*tabelei schimbatoare*" (*mutating table*) apare în cazul în care trigger-ul asociat tabelei încercă să execute o operaţie (DML) pe aceeaşi tabela, chiar alt tip de operaţie decât cel care a declanşat trigger-ul:

Spre exemplu, trigger-ul următor provoacă o astfel de eroare:

```
CREATE OR REPLACE TRIGGER tri1
AFTER INSERT ON tabela
--FOR EACH ROW
BEGIN
UPDATE tabela SET ......;
END;
```

Evitarea unor astfel de probleme specifice situaţiei în care un trigger, declanşat de o operaţie de modificare (DML) pe o tabela, efectuează la rândul lui operaţii de modificare pe aceeasi tabela, se poate face printr-o construcţie adecvată a trigger-ului.

In continuare sunt prezentate câteva dintre aspectele care trebuie avute în vedere în acest caz:
- construcţia unui trigger de tip BEFORE, cu declanşare la nivel de linie;
- utilizarea în corpul trigerr-ului (pentru o modificare de date) a unor construcţii de genul:
:NEW.cimp_de_modificat := valoare/expresie
în locul unor comenzi SQL DML, de genul UPDATE.

În acest fel, înainte de scrierea/modificarea efectivă în tabela, sunt realizate de către trigger modificările dorite care ulterior sunt salvate în tabelă (evitând problema ”*tabelei schimbătoare*”).

Observaţii:
- Declanşarea unui trigger asociat unei tabele poate duce la efectuarea unor operaţii asupra altor tabele.
- În cadrul corpului unui trigger pot fi apelate şi proceduri stocate.
- În cadrul corpului unui trigger putem avea sectiunea de tartare a exceptiilor, EXCEPTION.
- În cadrul unui trigger nu pot fi utilizate comenzile COMMIT şi ROLLBACK.
- Comanda RETURN poate fi folosită dar fără a returna nici o valoare (deci doar un simplu RETURN).

Probleme rezolvate:

1). Considerand tabela *Evidenta_persoane*, cu doua campuri (*id* de tip integer, cheie primara, si *nume* de tip varchar(30)), scrieti un trigger care permite adaugarea de noi linii in tabela. Trigger-ul va verifica daca noua valoare ce se doreste a fi inserata in campul id este diferita de NULL si respecta conditia de unicitate. In cazul in care aceasta este NULL, se va insera in tabela, in campul de tip primary key, valorea cea mai mare din coloana respectiva incrementata cu unu si se va semnala printr-un mesaj anomalia rezolvata. In cazul in care noua valoare a id-ului nu respecta conditia de unicitate, se va abandona operatia si se va afisa un mesaj de eroare. In cazul in care noua valoare a id-ului este o valoare valida, inserarea se va efectua cu succes si se va semnala acest lucru printr-un mesaj.

```
CREATE TABLE Evidenta_persoane (id INTEGER PRIMARY KEY, nume VARCHAR(30));

SET SERVEROUTPUT ON;

CREATE OR REPLACE TRIGGER tr1
BEFORE INSERT ON Evidenta_persoane
FOR EACH ROW
DECLARE
nr integer;
maxim integer;
BEGIN

IF :NEW.id is NULL THEN
select max(id) INTO maxim from Evidenta_persoane;
:NEW.id:=maxim+1;
DBMS_OUTPUT.PUT_LINE ('Anomalie rezolvata. Inserare reusita!');
ELSE
  SELECT count(*) INTO nr from Evidenta_persoane WHERE id=:NEW.id;
  IF (nr!=0) THEN
    DBMS_OUTPUT.PUT_LINE ('Eroare: Acest id se gaseste in tabela!');
  ELSE
    DBMS_OUTPUT.PUT_LINE ('Inserare reusita!');
  END IF;
END IF;
END;
/

INSERT INTO Evidenta_persoane VALUES (1, 'Ana');
INSERT INTO Evidenta_persoane VALUES (1, 'Vali');
INSERT INTO Evidenta_persoane VALUES (NULL, 'Ana');
```

```
SELECT * FROM Evidenta_persoane;

DROP TABLE Evidenta_persoane;
//In momentul in care stergem o tabela, se sterg automat si toate trigger-
ele definite pe aceasta.
```

2). Fie tabela *Evidenta_pacienti*, avand campurile (cnp char(13) primary key, nume varchar (30), data_nasterii date). Creati un trigger care verifica in momentul introducerii uni noi linii in tabela, ca *data_nasterii* sa fie mai mica decat data curenta. Daca *data_nasterii*>data curenta, se completeaza campul cu NULL si se semnaleaza problema printr-un mesaj (linia adaugandu-se totusi).

```
CREATE TABLE Evidenta_pacienti (cnp char(13) PRIMARY KEY, nume VARCHAR(30),
data_nasterii date);

CREATE OR REPLACE TRIGGER tr2
BEFORE INSERT ON Evidenta_pacienti
FOR EACH ROW
DECLARE
BEGIN

IF :NEW.data_nasterii>sysdate THEN
   :NEW.data_nasterii:=NULL;
   DBMS_OUTPUT.PUT_LINE('Inserare cu NULL!');
ELSE
   DBMS_OUTPUT.PUT_LINE ('Inserare reusita!');
END IF;
END;
/

INSERT INTO Evidenta_pacienti VALUES ('111', 'Ana', '28-Jul-2020');
SELECT * FROM Evidenta_pacienti;

INSERT INTO Evidenta_pacienti VALUES ('222', 'Vali', '28-Jun-2016');
SELECT * FROM Evidenta_pacienti;

DROP TRIGGER tr2;
```

3). Fie tabela *Evidenta_pacienti*, avand campurile (cnp char(13) primary key, nume varchar (30), data_nasterii date). Creati un trigger care verifica in momentul introducerii uni noi linii in tabela ca *data_nasterii* sa fie mai mica decat data curenta. Daca *data_nasterii*>data curenta, se interzice adaugarea noii linii, semnalandu-se acest lucru printr-un mesaj.

Observatii:

- Pentru rezolvarea situatiei in care data nasterii e mai mare decat data curenta, se cere generarea propriei exceptii;

```
CREATE OR REPLACE TRIGGER tr3
BEFORE INSERT ON Evidenta_pacienti
FOR EACH ROW
DECLARE
er EXCEPTION;
BEGIN

IF :NEW.data_nasterii>sysdate THEN
   RAISE er;
ELSE
   DBMS_OUTPUT.PUT_LINE ('Inserare reusita!');
```

```
END IF;
EXCEPTION
WHEN er THEN
   RAISE_APPLICATION_ERROR(-20000, 'Eroare: Data_nasterii mai mare decat
   data curenta!');
END;
/
INSERT INTO Evidenta_pacienti VALUES ('333', 'Ana', '28-Jul-2018');
SELECT * FROM Evidenta_pacienti;

INSERT INTO Evidenta_pacienti VALUES ('444', 'Vali', '28-Jun-2020');
SELECT * FROM Evidenta_pacienti;

DROP TABLE Evidenta_pacienti;
```

4). Se considera urmatoarele trei tabele:

Tabela angajati avand campurile:
- id integer PRIMARY KEY,
- nume varchar(30) NOT NULL,
- salar_neg number(5,2) NOT NULL,
- data _angajarii DATE,
- data _concedierii DATE.

Tabela salarii avand campurile:
- id integer NOT NULL REFERENCES angajati(id),
- nr_zile integer NOT NULL,
- brut number(5,2) ,
- deducere number(5,2) ,
- impozit number(5,2) ,
- net number(5,2).

Tabela deduceri avand campurile:
- brut_min number(5,2) NOT NULL,
- brut_max number(5,2) NOT NULL,
- deducere number(5,2) NOT NULL.

Se cunosc urmatoarele:

- Coloana salar_neg din tabela angajati contine salariul brut negociat pe zi.
- Salariul brut este salariul brut negociat pe zi * numar de zile lucrate.
  brut=salar_neg*nr_zile
- Deducerile se acorda in functie de venitul brut lunar, conform urmatorului tabel:

BRUT_MINIM	BRUT_MAXIM	DEDUCERE
0	300	200
300.01	600	150
600.01	800	100
800.01	999.99	0

-Limitele transelor si respectiv cuantumurile acestor deduceri se citesc din tabela deduceri.
-Impozitul datorat statului se calculeaza cu formula:

impozit = (venit brut - deducere) * 0.16

-Venitul net este obtinut cu ajutorul formulei:

$$\text{venit net} = \begin{cases} \text{venit brut} & \text{daca impozitul} <= 0 \\ \text{venit brut - impozit} & \text{daca impozitul} > 0 \end{cases}$$

Se cere:

a). Sa se creeze cele 3 tabele si sa se populeze cu informatii tabelele angajati si deduceri.

```
CREATE TABLE angajati
(id integer PRIMARY KEY,
nume varchar(30) NOT NULL,
salar_neg number(5,2)NOT NULL,
data_angajarii DATE,
data_concedierii DATE);

CREATE TABLE salarii
(id integer NOT NULL REFERENCES angajati(id),
nr_zile integer NOT NULL,
brut number(5,2),
deducere number(5,2),
impozit number(5,2),
net number(5,2));

CREATE TABLE deduceri
(brut_min number(5,2),
brut_max number(5,2),
deducere number(5,2));

ALTER SESSION SET NLS_DATE_FORMAT='dd/mm/yyyy';
SET AUTOCOMMIT ON;

INSERT INTO angajati VALUES (1,'ion',30,'22/01/2004',NULL);
INSERT INTO angajati VALUES (2,'gheorghe',20,'02/10/2003',NULL);
INSERT INTO angajati VALUES (3,'paul',28,'01/03/2002',NULL)

INSERT INTO deduceri VALUES (0,300,200);
INSERT INTO deduceri VALUES (300.01,600,150);
INSERT INTO deduceri VALUES (600.01,800,100);
INSERT INTO deduceri VALUES (800.01,999.99,0);
```

b). Sa se scrie un trigger TRIG_SAL, care la orice adaugare de noi linii in tabela SALARII sau la orice actualizare a coloanei nr_zile va actualiza (recalcula) informatia din coloanele: brut, deducere, impozit si net.

```
CREATE OR REPLACE TRIGGER TRIG_SAL
BEFORE INSERT OR UPDATE of nr_zile ON salarii
FOR EACH ROW
DECLARE
salar_neg_p number;
BEGIN
-- salariu brut
SELECT salar_neg INTO salar_neg_p FROM angajati WHERE id=:NEW.id;
:NEW.brut:= :NEW.nr_zile * salar_neg_p;
-- deduceri
```

```
SELECT deducere INTO :NEW.deducere FROM deduceri WHERE brut_min<=:NEW.brut
AND brut_max>=:NEW.brut;
--impozit
:NEW.impozit:=(:NEW.brut-:NEW.deducere)*0.16;
--net
IF :NEW.impozit<=0 THEN
:NEW.net:=:NEW.brut;
ELSE
:NEW.net:=:NEW.brut-:NEW.impozit;
END IF;
END;
/

INSERT INTO salarii VALUES(1,20, NULL, NULL, NULL, NULL);
SELECT * FROM salarii;

INSERT INTO salarii VALUES(2,10, NULL, NULL, NULL, NULL);
SELECT * FROM salarii;

UPDATE salarii SET nr_zile=15 WHERE id=2;
SELECT * FROM salarii;
```

c). Sa se scrie un trigger TRIG_DED, care la orice modificare a tabelei DEDUCERI va actualiza (recalcula) informatia din coloanele: deducere, impozit si respectiv net.

```
CREATE OR REPLACE TRIGGER TRIG_DED
AFTER INSERT OR DELETE OR UPDATE ON deduceri
BEGIN
-- update deducere
UPDATE salarii SET deducere= (select deducere from deduceri WHERE
brut_min<=brut AND brut_max>=brut);
-- update impozit
UPDATE salarii SET impozit=(brut-deducere)*0.16;
-- update net
UPDATE salarii SET net=brut WHERE impozit <=0;
UPDATE salarii SET net=brut-impozit WHERE impozit >0;
END;
/

UPDATE deduceri SET deducere=300 WHERE brut_min=0;
SELECT * FROM deduceri;
SELECT * FROM salarii;
```

d). Sa se scrie un trigger TRIG_DEL, care la orice inregistrare a unei concedieri (introducerea datei concedierii in tabela ANGAJATI) va sterge toate liniile din tabela SALARII referitoare la angajatii concediati.

```
CREATE OR REPLACE TRIGGER TRIG_DEL
AFTER UPDATE of DATA_CONCEDIERII ON angajati
FOR EACH ROW
BEGIN
--verificare daca data concedierii este diferita de NULL
IF :NEW.data_concedierii is NOT NULL THEN
   DELETE FROM salarii WHERE id =:NEW.id;
END IF;
END;
/

UPDATE angajati SET data_concedierii='20/04/2020' WHERE id=1;
```

```
SELECT * FROM angajati;
SELECT * FROM salarii;

DROP TABLE DEDUCERI;
DROP TABLE SALARII;
DROP TABLE ANGAJATI;
```

Probleme propuse:

1. Fie tabela *Evidenta_angajati*, care contine trei campuri:
   - *cnp* char(13) (cod numeric personal, cu lungimea de 13 caractere, caracterele 2 pana la 7 reprezentand anul, luna si ziua nasterii);
   - *nume* varchar(20);
   - *data_nasterii* date.

Sa se scrie un trigger care in momentul adaugarii unei noi linii in tabela, extrage data din *CNP* si o completeaza automat in campul *data_nasterii*.

```
CREATE TABLE Evidenta_angajati(
cnp char(13) PRIMARY KEY,
nume varchar(20),
data_nasterii date);

--CREARE TRIGGER

INSERT INTO Evidenta_angajati VALUES('2830823350077', 'vali', NULL);
SELECT * FROM Evidenta_angajati;
```

2. Se considera urmatoarele tabele:

Tabela produse, avand campurile:
- id_produs integer PRIMARY KEY,
- nume_produs varchar(30) NOT NULL,
- pret_bucata number(5,2) NOT NULL,
- stoc integer NOT NULL.

Tabela vanzari, avand campurile:
- id_produs integer NOT NULL REFERENCES produse(id_produs),
- cantitate integer,
- cost_total number (7,2).

a). Sa se creeze cele 2 tabele si sa se populeze cu urmatoarele informatii tabela produse:

ID_PRODUS	NUME_PRODUS	PRET_BUCATA	STOC
1	paine	5	1
2	lapte	8	5
3	iaurt	6	7
4	kefir	7	7

```
CREATE TABLE produse(
id_produs integer PRIMARY KEY,
nume_produs varchar(30) NOT NULL,
pret_bucata number(5,2) NOT NULL,
```

```
stoc integer NOT NULL);

CREATE TABLE vanzari(
id_produs integer NOT NULL REFERENCES produse(id_produs),
cantitate integer,
cost_total number (7,2));

INSERT INTO produse VALUES (1, 'paine', 5, 1);
INSERT INTO produse VALUES (2, 'lapte', 8, 5);
INSERT INTO produse VALUES (3, 'iaurt', 6, 7);
INSERT INTO produse VALUES (4, 'kefir', 7, 7);
```

b). Sa se scrie un trigger TRIG_VANZARI, care la orice vanzare a unui produs sa actualizeze coloana stoc din tabela produse si sa calculeze costul total aferent vanzarii produsului respectiv (cost_total=pret_bucata*cantitate). In cazul in care cantitatea ce se doreste a fi vanduta depaseste stocul existent al produsului, se va anula vanzarea, afisandu-se mesajul „Stoc insuficient!'.

Pentru testarea trigger-ului, se cere:
- vanzarea a 3 produse de tip paine avand id-ul 1;
- vanzarea a 4 produse de tip lapte avand id-ul 2.

```
--CREARE TRIGGER

INSERT INTO vanzari VALUES (1,3,NULL);
INSERT INTO vanzari VALUES (2,4,NULL);

SELECT * FROM produse;
SELECT * FROM vanzari;

DROP TABLE VANZARI;
DROP TABLE PRODUSE;
```

3). Se considera urmatoarele tabele:

Tabela useri, avand campurile:

id_user integer PRIMARY KEY,
username varchar(20) NOT NULL,
email varchar(30) NOT NULL,
pasaport_nr varchar(10),
permis_conducere_nr varchar(10).

Tabela remindere, avand campurile:

id_user integer NOT NULL REFERENCES useri(id_user),
text_reminder varchar(30),
data_reminder date,
status varchar(10).

Sa se creeze un trigger TRIG_REMINDERE care, in momentul adaugarii unui nou user in tabela useri, va verifica daca acel user are numarul pasaportului si al permisului de conducere completat. In cazul in care fie numarul permisului de conducere, fie numarul pasaportului, fie ambele sunt nule sau egale cu sirul vid, trigger-ul va insera in tabela remindere un reminder asociat noului user pentru a aminti ca datele respective trebuie

completate in sistem (se va insera un text corespunzator, o data pentru reminder, precum si status-ul „pending”).

```
CREATE TABLE useri(
id_user integer PRIMARY KEY,
username varchar(20) NOT NULL,
email varchar(30) NOT NULL,
pasaport_nr varchar(10),
permis_conducere_nr varchar(10));

CREATE TABLE remindere(
id_user integer NOT NULL REFERENCES useri(id_user),
text_reminder varchar(200),
data_reminder date,
status varchar(10));

--CREARE TRIGGER

INSERT INTO useri VALUES(1, 'adi', 'adi@yahoo.com', '224477', '333555');
SELECT * FROM useri;
SELECT * FROM remindere;

INSERT INTO useri VALUES(2, 'adina', 'adina@yahoo.com', '', '333555');

SELECT * FROM useri;
SELECT * FROM remindere;

INSERT INTO useri VALUES(3, 'maria', 'maria@yahoo.com', '224477', NULL);

SELECT * FROM useri;
SELECT * FROM remindere;

INSERT INTO useri VALUES(4, 'vali', 'vali@yahoo.com', NULL, NULL);
SELECT * FROM useri;
SELECT * FROM remindere;

DROP TABLE remindere;
DROP TABLE useri;
```