Universitatea
Politehnica
București

Facultatea de
Automatică și
Calculatoare

Catedra de
Calculatoare

Laborator 9
Triggeri PL/SQL

Autori

Conf. Dr. Ing. Alexandru Boicea
As. Dr. Ing. Ciprian-Octavian Truică

Cuprins

- Triggeri PL/SQL
- Crearea unui trigger
- Triggeri de tip BEFORE
- Triggeri de tip AFTER
- Restricții în clauza WHEN
- Predicate condiționale
- Triggeri cu opțiunea INSTEAD OF
- Informații din dicționarul bazei de date

Triggeri PL/SQL

- Un trigger este un bloc PL/SQL stocat pe server care se execută la apariția unui eveniment care modifică starea anumitor obiecte ale bazei de date;
- Termenul corespondent în literatura de specialitate românească este *declanșator*, dar este rar folosit și de aceea în continuare se va folosi termenul în limba engleză;
- Tipuri de evenimente care pot determina execuția unui trigger sunt:
  - Comenzi INSERT, UPDATE, DELETE pe o tabelă;
  - Comenzi INSERT, UPDATE, DELETE pe un view cu opțiunea INSTEAD OF;
  - Comenzi CREATE, ALTER, DROP la nivel de schemă sau bază de date;
  - Comenzi SHUTDOWN, LOGON, LOGOFF la nivel de schemă sau bază de date.

Triggeri PL/SQL

- În general, triggerii se folosesc pentru:
  - Gestionarea restricțiilor complexe de integritate;
  - Monitorizarea tranzacțiilor;
  - Efectuarea de replicări de tabele situate în diferite noduri ale unei baze de date distribuite;
  - Păstrarea semnăturii userilor care au efectuat operații pe baza de date;
  - Prelucrarea de informații statistice în legătură cu accesul tabelelor;
  - Jurnalizarea transparentă a evenimentelor.

Triggeri PL/SQL

- Printre avantajele utilizării triggerilor, se pot menționa:
  - Declanșarea automată, la apariția evenimentului monitorizat;
  - Lansarea în execuție a unor proceduri stocate specifice;
  - Posibilitatea modificării în cascadă a mai multor obiecte corelate în baza de date;
  - Transparența față de utilizator.

Crearea unui trigger

- Sintaxa unui trigger este:

```
CREATE [OR REPLACE] TRIGGER [schema.]trigger_name
    {BEFORE | AFTER | INSTEAD OF}
    {DELETE | INSERT | UPDATE [OR {DELETE | INSERT | UPDATE }…]
        [OF column[, column …] ]}
    ON [schema.]tabel _name
    [referencing_clauses]
    [FOR EACH ROW]
    [WHEN (condition) ]
    DECLARE
        trigger_variables
    BEGIN
        trigger_body
    END
```

Crearea unui trigger

- Unde:
- **trigger_name** – numele triggerului PL/SQL
- **schema** – specifică schema pe care se definește triggerul sau în care există obiectele, în mod implicit este aleasă schema utilizatorului curent
- **tabel _name** – numele tabelul/view-ul pe care se monitorizează evenimentul
- **column** – numele coloanei (coloanelor) din tabelul/view-ul pe care se monitorizează evenimentul
- **condition** – reprezintă o condiție pentru executarea triggerului, fiind admise corelări dar nu și interogări
- **trigger_variables** – secțiunea de declarare a variabilelor locale ale triggerului
- **trigger_body** – reprezintă corpul triggerului

Crearea unui trigger

- **BEFORE | AFTER** – specifică momentul executării triggerului: înainte sau după apariția evenimentului
- **INSTEAD OF** – specifică că este permisă o operație de inserare, ștergere, modificare pe view-uri, pentru care nu este permisă operația în mod firesc
- **INSERT | UPDATE | DELETE** – specifică evenimentul pe care se declanșează triggerul
- **FOR EACH ROW** – specifică dacă execuția triggerului se face pentru fiecare linie afectată, cu respectarea condiției din **WHEN**

Crearea unui trigger

- Sintaxa **referencing_clauses:**

REFERENCING {OLD [AS] old_variable NEW [AS] new_variable |
NEW [AS] new_variable OLD [AS] old_variable}

- Se folosesc nume corelate pentru a specifica valorile noi și vechi ale rândului curent pentru care se declanșează triggerul;
- Aceste valori se pot folosi atât în clauza WHEN (**old_variable.row_field**, **new_variable.row_field**) cât și în blocul PL/SQL folosindu-se prefixate de două puncte **:** (de exemplu **:old_variable.row_field**, **:new_variable.row_field**);
- Numele implicite sunt OLD și NEW.

Crearea unui trigger

- Sintaxa pentru crearea unui trigger de sistem este următoarea:

```
CREATE [OR REPLACE] TRIGGER [schema.]trigger_name
        {BEFORE | AFTER}
        {DDL_event_list | DB_event_list}
        ON {DATABASE | SCHEMA}
        [WHEN (condition) ]
DECLARE
        trigger_variables
BEGIN
        trigger_body
END
```

Crearea unui trigger

- Unde:
- **DDL_event_list** - CREATE, DROP, ALTER
- **DB_event_list** - STARTUP, SHUTDOWN, LOGON, LOGOFF, SERVERERROR, SUSPEND

Crearea unui trigger

- Există două tipuri de triggeri:
  - Triggeri pe o comandă – sunt executați o singură dată pentru evenimentul declanșator. De exemplu dacă se execută o comandă INSERT de mai multe linii, triggerul este executat o singură dată. În acest caz, nu este limitare la numărul de linii afectate de eveniment;
  - Triggeri pe o linie – este executat ori de câte ori o linie a unei tabele este afectată de evenimentul declanșator. De exemplu, dacă se execută o comandă UPDATE care actualizează k linii, atunci triggerul este executat de k ori.

Crearea unui trigger

- Un trigger poate fi executat înainte ca un eveniment să aibă loc (opțiunea BEFORE) sau după ce evenimentul s-a consumat (opțiunea AFTER);
- În general, triggerii de tip BEFORE sunt folosiți pentru:
  - A salva valorile coloanelor înaintea executării unei comenzi UPDATE;
  - A decide dacă acțiunea triggerului trebuie sau nu executată (aceasta poate îmbunătăți performanțele serverului prin eliminarea procesării inutile).
- Triggeri de tip AFTER sunt, în general, folosiți atunci când:
  - Se dorește ca executarea triggerului să se facă după ce comanda s-a efectuat cu succes;
  - Nu au apărut erori de procesare care ar impune o comandă ROLLBACK pentru respectiva tranzacție;
  - Trebuie alterate și alte date corelate cu cele deja afectate.

Triggeri de tip BEFORE

- Triggerii de tip BEFORE se declanșează la apariția unui eveniment, dar înainte ca evenimentul să se termine;

- Ex. 1. Să se scrie un trigger de tip BEFORE care printează un mesaj ori de câte ori se face un insert în tabela emp.

```
create or replace trigger insertemp
before insert on emp
begin
  dbms_output.put_line('S-a facut o noua inserare in tabela emp');
end;
/
```

Triggeri de tip BEFORE

- Să verificăm cum lucrează făcând un insert in tabela EMP:

```
SQL> set serveroutput on;
SQL> insert into emp(empno, ename, sal) values(999, 'Preda', 1500);
Sa facut o noua inserare in tabela emp

1 row created.
```

- Se observă că triggerul s-a declanșat și operația s-a făcut cu succes;
- Dacă forțăm o eroare la inserare, rezultatul este următorul (exemplul necesită ca empno să aibă constrângerea not null):

```
SQL> alter table emp modify empno not null;

Table altered.

SQL> insert into emp(ename, sal) values('Tache', 1500);
Sa facut o noua inserare in tabela emp
insert into emp(ename, sal) values('Tache', 1500)
*
ERROR at line 1:
ORA-01400: cannot insert NULL into ("STUDENT"."EMP"."EMPNO")
```

Triggeri de tip BEFORE

- Se observă că triggerul s-a declanșat normal, nu a ținut cont că inserarea nu a fost efectuată și a generat o eroare;
- Dacă compilarea unui trigger se face cu succes, apare mesajul: **Trigger created;**
- În caz că apar erori la crearea unui trigger apare mesajul de avertizare:

  Warning: Trigger created with compilation errors.
- Pentru a vedea erorile de compilare se va folosi comanda

  show errors.

Trigger de tip AFTER

- Triggerii de tip AFTER se declanșează după ce evenimentul declanșator se termină.

- Ex. 2. Să se creeze un trigger de tip AFTER care afișează un mesaj ori de câte ori se face o modificare în tabela EMP.

```
create or replace trigger updateemp
after update on emp
begin
  dbms_output.put_line('Sa facut o modificare in tabela emp');
end;
/
```

Trigger de tip AFTER

- Să verificăm cum lucrează triggerul făcând o modificare de comision în tabela EMP:

```
SQL> update emp set comm = 100 where empno = 7902;
Sa facut o modificare in tabela emp

1 row updated.
```

- Se observă că triggerul a fost declanșat la apariția evenimentului, în acest caz comanda UPDATE și s-a afișat mesajul de avertizare.
- Triggerul se declanșează chiar dacă nu este găsită nicio înregistrare care să îndeplinească condițiile din clauza WHERE.

```
SQL> update emp set comm = 100 where empno = 9999;
Sa facut o modificare in tabela emp

0 rows updated.
```

Trigger de tip AFTER

- Deoarece nu există niciun angajat cu empno=9999, nu a fost alterată nicio linie, totuși triggerul s-a declanșat, deoarece nu a apărut nicio eroare de execuție.
- Dacă însă operația se termină cu o eroare, triggerul nu se mai declanșează.

```
SQL> update emp set empno=null where empno=7902;
update emp set empno=null where empno=7902
               *
ERROR at line 1:
ORA-01407: cannot update ("STUDENT"."EMP"."EMPNO") to NULL


SQL> rollback;

Rollback complete.
```

Trigger de tip AFTER

- Am încercat să atribuim valoarea NULL pentru empno, ceea ce este interzis prin definirea tabelei, ca urmare s-a generat o eroare de sistem pe constrângerea respectivă și triggerul nu s-a declanșat;
- Putem trage concluzia că un trigger de tip BEFORE se declanșează necondiționat de rezultatul comenzii SQL (chiar dacă se generează o eroare), pe când cel de tip AFTER nu se declanșează dacă comanda SQL generează o eroare;
- S-a utilizat comanda ROLLBACK pentru ca modificările să nu rămână permanente.

Restricții în clauza WHEN

- Dacă vrem să introducem o restricție pentru declanșarea triggerului, putem să folosim clauza WHEN;
- Această clauză se poate folosi numai pentru triggerii de tip linie, deci poate fi folosită doar cu opțiunea FOR EACH ROW;
- În clauza WHEN se acceptă numai condiționări directe sau corelate, nu se acceptă cereri sau subcereri.

Restricții în clauza WHEN

- Ex. 3. Să se creeze un trigger de tip AFTER care afișează un mesaj ori de câte ori se face o modificare în tabela emp, dar doar pentru angajații care nu au funcția ‘MANAGER’.

```
create or replace trigger updateemp2
after update on emp
for each row
when (old.job <> 'MANAGER')
begin
   dbms_output.put_line('Sa facut o modificare in tabela emp');
end;
/
```

Restricții în clauza WHEN

- Să se modifice comisionul pentru angajatul cu empn = 7566

```
SQL> update emp set comm=100 where empno=7566;

1 row updated.

SQL> rollback;

Rollback complete.
```

- Se observă că triggerul nu a fost declanșat, cu toate că operația UPDATE s-a efectuat cu succes, deoarece angajatul are funcția ‘MANAGER’.

Restricții în clauza WHEN

- Ex. 4. Să se scrie un trigger de tip BEFORE care afișează un mesaj ori de câte ori se face un insert în tabela EMP, doar dacă salariul noului angajat este strict mai mare de 500 și strict mai mic de 5000.

```
create or replace trigger insertemp2
before insert on emp
for each row
when (new.sal > 500 and new.sal < 5000)
begin
  dbms_output.put_line('Sa facut o noua inserare in tabela emp');
end;
/
```

Restricții în clauza WHEN

- Să facem o inserare cu salariul în afara limitei:

```
SQL> insert into emp(empno, ename, sal) values(1111, 'Frunza', 5500);

1 row created.

SQL> rollback;

Rollback complete.
```

- Se observă că inserarea s-a făcut cu succes, dar triggerul nu s-a declanșat.

Restricții în clauza WHEN

- Ex. 5. Să se scrie un trigger de tip AFTER care se declanșează dacă salariul unui angajat este majorat.

```
create or replace trigger majorareSal
after update on emp
for each row
when (new.sal > old.sal)
begin
   dbms_output.put_line('Salariul lui '||:old.ename
     ||' a fost majorat de la '||:old.sal||' la '||:new.sal);
end;
/
```

Restricții în clauza WHEN

- Se observă că în clauza WHEN, variabilele **old** și **new**, se apelează fără caracterul ‘:’ , pe când la afișare trebuie introdus.
- Dacă se face o majorare de salariu pentru salariatul cu empno = 7566, rezultatul este:

```
SQL> update emp set sal = 3000, comm = 100 where empno=7566;
Salariul lui JONES a fost majorat de la 2975 la 3000

1 row updated.

SQL> rollback;

Rollback complete.
```

Predicate condiționate

- În cazul în care se execută mai multe comenzi DML, se pot folosi predicate condiționate în corpul triggerului.
- Predicatele condiționate sunt:
  - **INSERTING** – returnează **TRUE** dacă triggerul se declanșează pe o comandă **INSERT;**
  - **UPDATING** – retunează **TRUE** dacă triggerul se declanșează pe o comandă **UPDATE;**
  - **UPDATING (‘column_name’)** – retunează **TRUE** dacă triggerul se declanșează pe o comandă **UPDATE** care modifică coloana specificată prin **column_name;**
  - **DELETING** – returnează **TRUE** dacă triggerul se declanșează pe o comandă **DELETE.**

Predicate condiționate

- Ex. 6. Să se scrie un trigger de tip AFTER care se declanșează în momentul în care se face un insert, delete sau update pe coloanele sal și comm din tabela EMP. Să se insereze mesaje la declanșarea triggerului în tabela MESAJE.
- Pasul I – tabela mesaje

```
create table mesaje
(
  mesaj1 varchar2(40),
  mesaj2 varchar2(40)
);
```

Predicate condiționate

- Pasul II – creare trigger

```
create or replace trigger monitor
  after insert or delete or update of sal, comm
  on emp for each row
declare
  salariu emp.sal%type;
  comision emp.comm%type;
begin
  if inserting then
    insert into mesaje values('Inserare in tabela emp',
      to_char(sysdate, 'dd-mm-yyyy hh:mi:ss'));
  elsif deleting then
    insert into mesaje values('Stergere in tabela emp',
      to_char(sysdate, 'dd-mm-yyyy hh:mi:ss'));
  elsif updating('sal') then
    insert into mesaje values('Salariu modificat, old='||:old.sal
      ||' new= '||:new.sal, to_char(sysdate, 'dd-mm-yyyy hh:mi:ss'));
  elsif updating('comm') then
    insert into mesaje values('Comision modificat, old='||:old.comm||
      ' new= '||:new.comm, to_char(sysdate, 'dd-mm-yyyy hh:mi:ss'));
  end if;
end;
/
```

Predicate condiționate

- Pasul III – testare trigger

```
SQL> set line 120;
SQL> insert into emp(empno, ename, sal, comm) values(999, 'Preda', 1500, 100);

1 row created.

SQL> update emp set comm = nvl(comm,0)+100 where empno = 999;

1 row updated.

SQL> update emp set sal = nvl(sal,0)+500 where empno=999;

1 row updated.

SQL> delete from emp where empno = 999;

1 row deleted.

SQL> select * from mesaje;

MESAJ1                                   MESAJ2
---------------------------------------- ----------------------------------------
_
Inserare in tabela emp                   01-12-2013 11:49:26
Comision modificat, old=100 new= 200     01-12-2013 11:49:26
Salariu modificat, old=1500 new= 2000    01-12-2013 11:49:26
Stergere in tabela emp                   01-12-2013 11:49:26

SQL> rollback;

Rollback complete.
```

Triggere cu opțiunea INSTEAD OF

- Acest trigger se definește numai pe view-uri, nu și pe tabele;
- Unele view-uri nu pot fi modificate prin comenzi DML, dar folosind un trigger cu opțiunea INSTEAD OF acest lucru este realizabil;
- View-urile care nu pot fi modificate prin comenzile UPDATE, INSERT sau DELETE sunt cele create printr-o interogare care conține în construcție:
  - Un operator SET sau DISTINCT;
  - O funcție de agregare sau o funcție analitică;
  - Clauzele GROUP BY, ORDER BY, CONNECT BY sau START WITH;
  - O expresie tip colecție într-o clauză SELECT;
  - O subcerere într-o clauză SELECT;
  - Unele metode de JOIN.

Triggere cu opțiunea INSTEAD OF

- Orice view, aflat într-un astfel de caz, se poate face modificabil folosind un trigger cu opțiunea INSTEAD OF;
- Acest trigger trebuie să determine ce operație trebuie executată pentru modificarea tabelei pe care este creat view-ul respectiv;
- Dacă view-ul conține pseudocoloane sau expresii , acestea nu pot fi modificate direct printr-o comandă UPDATE, dar pot fi modificate prin trigger.

Triggere cu opțiunea INSTEAD OF

- Ex. 7. Să creăm un view **sefi** cu următoarea comandă:

```
create or replace view sefi as
  select d.deptno, d.dname, e.empno, e.ename, e.hiredate, e.sal, e.comm
  from dept d inner join emp e on e.deptno = d.deptno
  where e.empno in (select distinct mgr from emp);
```

- Să facem o inserare în view-ul creat

```
SQL> insert into sefi values(50, 'Mediu', 1234, 'PREDA', sysdate, 2000, 100);
insert into sefi values(50, 'Mediu', 1234, 'PREDA', sysdate, 2000, 100)
*
ERROR at line 1:
ORA-01779: cannot modify a column which maps to a non key-preserved table
```

- Comanda INSERT a generat o eroare, deoarece nu se acceptă inserarea într-un astfel de view.

Triggere cu opțiunea INSTEAD OF

- Să creăm triggerul manager, cu opțiunea INSTEAD OF, care va face o inserare și o modificare în tabela EMP și inserare în tabela DEPT.

```
create or replace trigger manager
instead of insert on sefi
  referencing new as n
  for each row
declare
  nr number;
begin
  select count(*) into nr from emp where empno = :n.empno;
  if nr = 0 then
    insert into emp(empno, ename, deptno)
      values(:n.empno, :n.ename, :n.deptno);
    update emp set sal = :n.sal, comm = :n.comm,
        hiredate = :n.hiredate, mgr = :n.empno
      where empno = :n.empno;
  end if;
  select count(*) into nr from dept where deptno = :n.deptno;
  if nr = 0 then
    insert into dept(deptno, dname) values(:n.deptno, :n.dname);
  end if;
end;
```

Triggere cu opțiunea INSTEAD OF

- Executăm din nou comanda insert:

```
SQL> insert into sefi values(50, 'Mediu', 1234, 'PREDA', sysdate, 2000, 100);

1 row created.

SQL> select ename, sal, comm from emp where empno = 1234;

ENAME             SAL       COMM
---------- ---------- ----------
PREDA             2000        100

SQL> rollback;

Rollback complete.
```

- Se observă că inserarea s-a făcut cu succes, de data aceasta, deoarece s-a declanșat triggerul manager care a executat operațiile echivalente comenzii INSERT, prin inserarea în tabelele DEPTși EMP.

Triggere cu opțiunea INSTEAD OF

- În corpul triggerului nu trebuie să fie cuprinse instrucțiuni care să afecteze starea tabelei sau view-ului pe care se monitorizează evenimentul;
- Ex. 8. Să construim un trigger **comision** care actualizează comisionul la 10% din salariu când se modifică salariul unui angajat:

```
create or replace trigger comision
after update of sal on emp
for each row
begin
  update emp set comm = 0.1*sal
    where :new.empno=:old.empno;
end;
```

Triggere cu opțiunea INSTEAD OF

- Să facem o modificare de salariu să vedem ce anume se întâmplă:

```
SQL> update emp set sal = 5000 where empno=7566;
update emp set sal = 5000 where empno=7566
       *
ERROR at line 1:
ORA-04091: table STUDENT.EMP is mutating, trigger/function may not see it
ORA-06512: at "STUDENT.COMISION", line 2
ORA-04088: error during execution of trigger 'STUDENT.COMISION'
```

- Se observă că în execuția triggerului s-a generat o eroare.

Informații din dicționarul bazei de date

- Informațiile despre triggeri se pot obține din dicționarul de date, ca pentru proceduri, funcții și pachete;
- De exemplu, dacă vrem să vedem toți triggerii creați de userul curent, data când au fost creați, data ultimei utilizări și starea lor, putem executa următoarea cerere SQL:

```
SQL> col object_name for a20;
SQL> select object_name, created, last_ddl_time, status
  2  from user_objects where object_type = 'TRIGGER';

OBJECT_NAME          CREATED   LAST_DDL_ STATUS
-------------------- --------- --------- -------
MONITOR              01-DEC-13 01-DEC-13 VALID
MANAGER              02-DEC-13 02-DEC-13 VALID
COMISION             02-DEC-13 02-DEC-13 VALID
```

Informații din dicționarul bazei de date

- Pentru a vedea structura unui trigger se va interoga view-ul user_source:

```
SQL> col text for a65;
SQL> select text from user_source where name='MANAGER' and type='TRIGGER'
  2  order by line;

TEXT
-----------------------------------------------------------------
trigger manager
instead of insert on sefi
  referencing new as n
  for each row
declare
  nr number;
begin
  select count(*) into nr from emp where empno = :n.empno;
  if nr = 0 then
    insert into emp(empno, ename, deptno)
      values(:n.empno, :n.ename, :n.deptno);
    update emp set sal = :n.sal, comm = :n.comm,
        hiredate = :n.hiredate, mgr = :n.empno
      where empno = :n.empno;
  end if;
  select count(*) into nr from dept where deptno = :n.deptno;
  if nr = 0 then
    insert into dept(deptno, dname) values(:n.deptno, :n.dname);
  end if;
end;

20 rows selected.
```

Informații din dicționarul bazei de date

- Un trigger se poate șterge din dicționar folosind comanda DDL :
  DROP TRIGGER trigger_name;

- Pentru a modifica starea unui trigger se folosească comanda DDL:
  ALTER TRIGGER trigger_name {ENABLE | DISABLE}

Exercițiu

- Ex. 9. Să se scrie un trigger care face o inserare în tabela LOG ori de câte ori se face o operație pe coloana **sal** din tabela EMP.
- Tabela LOG are următoarea structură:

```
create table log
(
  nume varchar2(20),
  salariu_vechi number,
  salariu_nou number,
  data_operare varchar2(20),
  operator varchar2(20),
  operatie varchar2(20)
)
```

- Triggerul este folosit pentru a păstra semnătura persoanei care a operat pe baza de date, operația pe care o făcut-o, pentru ce angajat a modificat salariul și la ce dată (în formatul yyyy-mm-dd hh-mi-ss).

Exercițiu

- Crearea triggerului:

```
create or replace
trigger salariu
after delete or insert or update of sal on emp
for each row
declare
   operatie varchar2(20);
   nume varchar2(20);
   sal_vec number;
   sal_nou number;
begin
```

Exercițiu

```
    if inserting then
      operatie := 'insert';
      nume := :new.ename;
      sal_vec := 0;
      sal_nou := :new.sal;
    elsif updating then
      operatie := 'update';
      nume := :old.ename;
      sal_vec := :old.sal;
      sal_nou := :new.sal;
    elsif deleting then
      operatie := 'delete';
      nume := :old.ename;
      sal_vec := :old.sal;
      sal_nou := 0;
    end if;

    insert into log values(nume, sal_vec, sal_nou,
      to_char(sysdate, 'YYYY-mm-dd hh-mi-ss'), user, operatie);
end;
```

Exercițiu

- Testarea triggerului:

```
SQL> insert into emp(empno, ename, sal, comm) values(999, 'Preda', 1500, 100);

1 row created.

SQL> update emp set sal = nvl(sal,0)+500 where empno=999;

1 row updated.

SQL> delete from emp where empno = 999;

1 row deleted.

SQL> select * from log;

NUME     SALARIU_VECHI SALARIU_NOU DATA_OPERARE          OPERATOR  OPERATIE
-------- ------------- ----------- --------------------- --------- ---------
Preda                0        1500 2013-12-02 01-18-22   STUDENT   insert
Preda             1500        2000 2013-12-02 01-18-22   STUDENT   update
Preda             2000           0 2013-12-02 01-18-22   STUDENT   delete

SQL> rollback;

Rollback complete.
```