Declansatori

Declansatori in *PL/SQL*

Un declansator (*trigger*) este un bloc *PL/SQL* sau apelul *(CALL)* unei proceduri *PL/SQL,* care se executa automat ori de cate ori are loc un anumit eveniment „declansator“ Evenimentul poate consta din:

- modificarea unui tabel sau a unei vizualizari,
- actiuni sistem
- anumite actiuni utilizator.

Blocul *PL/SQL* poate fi asociat unui tabel, unei vizualizari, unei scheme sau unei baze de date.

La fel ca si pachetele, declansatorii nu pot fi locali unui bloc sau unui pachet, ei trebuie depusi ca obiecte independente in baza de date.

Folosirea declansatorilor garanteaza faptul ca atunci cand o anumita operatie este efectuata, automat sunt executate niste actiuni asociate. Evident, nu trebuie introdusi declansatori care ar putea sa substituie functionalitati oferite deja de sistem. De exemplu, nu are sens sa fie definiti declansatori care sa implementeze regulile de integritate ce pot fi definite, mai simplu, prin constrangeri declarative.

Tipuri de declansatori

Declansatorii pot fi:

- la nivel de baza de date *(database triggers);*
- la nivel de aplicatie *(application triggers).*

Declansatorii baza de date se executa automat ori de cate ori are loc:

- o actiune (comanda *LMD)* asupra datelor unui tabel;
- o actiune (comanda *LMD)* asupra datelor unei vizualizari;
- o comanda *LDD (CREATE, ALTER, DROP)* referitoare la anumite obiecte ale schemei sau ale bazei;
- un eveniment sistem *(SHUTDOWN, STARTUP);*
- o actiune a utilizatorului *(LOGON, LOGOFF);*
- o eroare *(SERVERERROR, SUSPEND).*

Declansatorii baza de date sunt de trei tipuri:

- declansatori *LMD* - activati de comenzi *LMD (INSERT, UPDATE* sau *DELETE)* executate asupra unui tabel al bazei de date;

- declansatori *INSTEAD OF* - activati de comenzi *LMD* executate asupra unei vizualizari (relationale sau obiect);
- declansatori sistem - activati de un eveniment sistem (oprirea sau pornirea bazei), de comenzi *LDD (CREATE, ALTER, DROP)*, de conectarea (deconectarea) unui utilizator. Ei sunt definiti la nivel de schema sau la nivel de baza de date.

Declansatorii asociati unui tabel (stocati in baza de date) vor actiona indiferent de aplicatia care a efectuat operatia *LMD.* Daca operatia *LMD* se refera la o vizualizare, declansatorul *INSTEAD OF* defineste actiunile care vor avea loc, iar daca aceste actiuni includ comenzi *LMD* referitoare la tabele, atunci declansatorii asociati acestor tabele sunt si ei, la randul lor, activati.

Daca declansatorii sunt asociati unei baze de date, ei se declanseaza pentru fiecare eveniment, pentru toti utilizatorii. Daca declansatorii sunt asociati unei scheme sau unui tabel, ei se declanseaza numai daca evenimentul declansator implica acea schema sau acel tabel. Un declansator se poate referi la un singur tabel sau la o singura vizualizare.

Declansatorii aplicatie se executa implicit ori de cate ori apare un eveniment particular intr-o aplicatie (de exemplu, o aplicatie dezvoltata cu *Developer Suite). Form Builder* utilizeaza frecvent acest tip de declansatori *(form builder triggers).* Ei pot fi declansati prin apasarea unui buton, prin navigarea pe un camp etc. In acest capitol se va face referinta doar la declansatorii baza de date.

Atunci cand un pachet sau un subprogram este depus in dictionarul datelor, alaturi de codul sursa este depus si *p-codul* compilat. In mod similar se intampla si pentru declansatori. Prin urmare, un declansator poate fi apelat fara recompilare. Declansatorii pot fi invalidati in aceeeasi maniera ca pachetele si subprogramele. Daca declansatorul este invalidat, el va fi recompilat la urmatoarea activare.

Crearea declansatorilor *LMD*

Declansatorii *LMD* sunt creati folosind comanda *CREATE TRIGGER.*

Numele declansatorului trebuie sa fie unic printre numele declansatorilor din cadrul aceleasi scheme, dar poate sa coincida cu numele altor obiecte ale acesteia (de exemplu, tabele, vizualizari sau proceduri).

La crearea unui declansator este obligatorie una dintre optiunile *BEFORE* sau *AFTER,* prin care se precizeaza momentul in care este executat corpul declansatorului.

CREATE [OR REPLACE] TRIGGER [schema.]nume_declansator
{BEFORE | AFTER}
{DELETE I INSERT | UPDATE [OFcoloana[, *coloana* **...**]] }
[OR {DELETE | INSERT | UPDATE [OFcoloana[, coloana ...]] ... }
ON *[schema.]nume_tabel*
[REFERENCING {OLD [AS] vechi NEW [AS] nou | NEW [AS] nou
OLD [AS] vechi }]
[FOR EACH ROW]
[WHEN(conditie)]
corpdeclansator (bloc PL/SQL sau apelul unei proceduri);

Declararea unui declansator trebuie sa cuprinda tipul comenzii *SQL care* duce la executarea declansatorului si tabelul asociat acestuia. In ceea ce priveste tipul comenzii *SQL* care va duce la executarea declasatorului, sunt incluse urmatoarele tipuri de optiuni: *DELETE, INSERT, UPDATE* sau o combinare a acestora cu operatorul logic *OR.* Cel putin una dintre optiuni este obligatorie.

In declararea declansatorului este specificat tabelul asupra caruia va fi executat declansatorul. Daca declansatorul este de tip *UPDATE,* atunci pot fi enumerate coloanele pentru care acesta se va executa.

In corpul fiecarui declansator pot fi cunoscute valorile coloanelor atat inainte de modificarea unei linii, cat si dupa modificarea acesteia. Valoarea unei coloane inainte de modificare este referita prin atributul *OLD,* iar dupa modificare, prin atributul *NEW.* Prin intermediul clauzei optionale *REFERENCING* din sintaxa comenzii de creare a declansatorilor, atributele *NEW* si OLD pot fi redenumite. In interiorul blocului PL/SQL, coloanele prefixate prin OLD sau *NEW* sunt considerate variabile externe, deci trebuie prefixate cu ":".

Un declansator poate activa alt declansator, iar acesta la randul sau poate activa alt declansator etc. Aceasta situatie (declansatori in cascada) poate avea insa efecte imprevizibile. Sistemul *Oracle* permite maximum 32 declansatori in cascada. Numarul acestora poate fi limitat (utilizand parametrul de initializare *OPENCURSORS),* deoarece pentru fiecare executie a unui declansator trebuie deschis un nou cursor.

Declansatorii la nivel de baze de date pot fi de doua feluri:

- la nivel de instructiune *(statement level trigger*);
- la nivel de linie *(row level trigger).*

Declansatori la nivel de instructiune

Declansatorii la nivel instructiune sunt executati o singura data pentru instructiunea declansatoare, indiferent de numarul de linii afectate (chiar daca nici o linie nu este afectata). Un declansator la nivel de instructiune este util daca actiunea declansatorului nu depinde de informatiile din liniile afectate.

Exemplu:
Programul de lucru la administratia muzeului este de luni pana vineri, in intervalul (8:00 a.m. - 10:00 p.m.). Sa se construiasca un declansator la nivel de instructiune care impiedica orice activitate asupra unui tabel al bazei de date, in afara acestui program.

```
CREATE OR REPLACE PROCEDURE verifica IS BEGIN
IF ((TO_CHAR(SYSDATE,'D') BETWEEN 2 AND 6)
AND
TO_DATE(TO_CHAR(SYSDATE,'hh24:mi'), 'hh24:mi')
NOT BETWEEN TO_DATE('08:00','hh24:mi')
                AND TO_DATE('22:00','hh24:mi'))
THEN
  RAISE_APPLICATION_ERROR (-27733, 'nu puteti reactualiza acest
                tabel deoarece sunteti in afara programului');
END IF;
END verifica;
/
CREATE OR REPLACE TRIGGER BIUD_tabel1
BEFORE INSERT OR UPDATE OR DELETE ON tabel1 BEGIN
verifica;
END;
```

Declansatori la nivel de linie

Declansatorii la nivel de linie sunt creati cu optiunea *FOR EACH ROW*. In acest caz, declansatorul este executat pentru fiecare linie din tabelul afectat, iar daca evenimentul declansator nu afecteaza nici o linie, atunci declansatorul nu este executat. Daca optiunea *FOR EACH ROW* nu este inclusa, declansatorul este considerat implicit la nivel de instructiune.
Declansatorii la nivel linie nu sunt performanti daca se fac frecvent reactualizari pe tabele foarte mari.
Restrictive declansatorilor pot fi incluse prin specificarea unei expresii booleene in clauza *WHEN*. Acesta expresie este evaluata pentru fiecare linie afectata de catre declansator. Declansatorul este executat pentru o linie, doar daca expresia este adevarata pentru acea linie. Clauza *WHEN* este valida doar pentru declansatori la nivel de linie.

Exemplu:

Sa se implementeze cu ajutorul unui declansator constrangerea ca valorile operelor de arta nu pot fi reduse (trei variante).

Varianta 1:

```
CREATE OR REPLACE TRIGGER verifica_valoare BEFORE UPDATE OF
  valoare ON opera FOR EACH ROW
WHEN (NEW.valoare < OLD.valoare)
BEGIN
RAISE_APPLICATION_ERROR (-20222, 'valoarea unei opere de
arta nu poate fi micsorata');
END;
```

Varianta 2:

```
CREATE OR REPLACE TRIGGER verifica_valoare BEFORE UPDATE OF
  valoare ON opera FOR EACH ROW BEGIN
IF (:NEW.valoare < :OLD.valoare) THEN
RAISE_APPLICATION_ERROR (-20222, 'valoarea unei opere de
arta nu poate fi micsorata');
END IF;
END;
```

Varianta 3:

```
CREATE OR REPLACE TRIGGER verifica_valoare BEFORE UPDATE OF
  valoare ON opera FOR EACH ROW
WHEN (NEW.valoare < OLD.valoare)
CALL procedura -- care va face actiunea RAISE ...
/
```

Accesul la vechile si noile valori ale coloanelor liniei curente, afectata de evenimentul declansator, se face prin: *OLD.numecoloana* (vechea valoare), respectiv prin *NEW.nume coloana* (noua valoare). In cazul celor trei comenzi *LMD*, aceste valori devin:

INSERT : NEW.nume_coloana noua valoare
(: OLD.nume_coloana *NULL*);
UPDATE : *NEW.*nume_coloana noua valoare
: OLD.nume_coloana vechea valoare;
DELETE (: NEW.nume_coloana *NULL*)
: OLD.nume_coloana vechea valoare.

Exemplu:
Se presupune ca pentru fiecare galerie exista doua campuri *(min_valoare* si *max valoare)* in care se retin limitele minime si maxime ale valorile operelor din galeria respectiva. Sa se implementeze cu ajutorul unui declansator constrangerea ca, daca aceste limite s-ar modifica, valoarea oricarei opere de arta trebuie sa ramana cuprinsa intre noile limite.

```
CREATE OR REPLACE TRIGGER verifica_limite
BEFORE UPDATE OF min_valoare, max_valoare ON galerie FOR
EACH ROW    -  -
DECLARE
v_min_val opera.valoare%TYPE; v_max_val opera.valoare%TYPE;
e~invalid EXCEPTION;
BEGIN
SELECT MIN(valoare), MAX(valoare)
INTO    v_min_val, v_max_val
FROM opera
WHERE cod_galerie = :NEW.cod_galerie;
IF (v_min_val < :NEW.min_valoare) OR (v_max_val >
:NEW.max_valoare) THEN RAISE e_invalid;
END IF;
EXCEPTION
WHEN e_invalid THEN
RAISE_APPLICATION_ERROR (-20567, 'Exista opere de arta ale
caror valori sunt in afara domeniului permis');
END verifica_limite;
/
```

Ordinea de executie a declansatorilor

PL/SQL permite definirea a mai multor tipuri de declansatori care sunt obtinuti prin combinarea proprietatii de moment (timp) al declansarii *(BEFORE, AFTER)*, cu proprietatea nivelului la care actioneaza (nivel linie, nivel intructiune) si cu tipul operatiei atasate declansatorului *(INSERT, UPDATE, DELETE)*.
De exemplu, *BEFORE INSERT* actioneaza o singura data, inaintea executarii unei instructiuni *INSERT*, iar *BEFORE INSERT FOR EACH ROW* actioneaza inainte de inserarea fiecarei noi inregistrari.
Declansatorii sunt activati cand este executata o comanda *LMD*. La aparitia unei astfel de comenzi se executa cateva actiuni care vor fi descrise in continuare.

1. Se executa declansatorii la nivel de instructiune *BEFORE*.
2. Pentru fiecare linie afectata de comanda *LMD*:

2.1. se executa declansatorii la nivel de linie *BEFORE;*

2.2. se blocheaza si se modifica linia afectata (se executa comanda *LMD),* se verifica constrangerile de integritate (blocarea ramane valabila pana in momentul in care tranzactia este permanentizata);

2.3. se executa declansatorii la nivel de linie *AFTER.*

3**.** Se executa declansatorii la nivel de instructiune *AFTER.*

Verificarea constrangerii referentiale este amanata dupa executarea declansatorului la nivel linie.

Obsevatii:

- In expresia clauzei *WHEN* nu pot fi incluse functii definite de utilizator sau subcereri *SQL.*
- In clauza *ON* poate fi specificat un singur tabel sau o singura vizualizare.
- In interiorul blocului *PL/SQL,* coloanele tabelului prefixate cu *OLD* sau *NEW* sunt considerate variabile externe si deci, trebuie precedate de caracterul „:“.
- Conditia de la clauza *WHEN* poate contine coloane prefixate cu *OLD* sau *NEW,* dar in acest caz, acestea nu trebuie precedate de „:“.
- Declansatorii baza de date pot fi definiti numai pe tabele (exceptie, declansatorul *INSTEAD OF* care este definit pe o vizualizare). Totusi, daca o comanda *LMD* este aplicata unei vizualizari, pot fi activati declansatorii asociati tabelelor care definesc vizualizarea.
- Corpul unui declansator nu poate contine o interogare sau o reactualizare a unui tabel aflat in plin proces de modificare, pe timpul actiunii declansatorului *(mutating table).*
- Blocul *PL/SQL* care descrie actiunea declansatorului nu poate contine comenzi pentru gestiunea tranzactiilor (*COMMIT*, *ROLLBACK*, *SAVEPOINT).* Controlul tranzactiilor este permis, insa, in procedurile stocate. Daca un declansator apeleaza o procedura stocata care executa o comanda referitoare la controlul tranzactiilor, atunci va aparea o eroare la executie si tranzactia va fi anulata.
- Comenzile *LDD* nu pot sa apara decat in declansatorii sistem.
- Corpul declansatorului poate sa contina comenzi *LMD.*
- In corpul declansatorului pot fi referite si utilizate coloane *LOB,* dar nu pot fi modificate valorile acestora.
- Nu este indicata crearea declansatorilor recursivi.
- In corpul declansatorului se pot insera date in coloanele de tip *LONG* si *LONGRAW,* dar nu pot fi declarate variabile de acest tip.

- Daca un tabel este suprimat (se sterge din dictionarul datelor), automat sunt distrusi toti declansatorii asociati tabelului.
- Este necesara limitarea dimensiunii unui declansator. Este preferabil ca o parte din cod sa fie inclusa intr-o procedura stocata si aceasta sa fie apelata din corpul declansatorului.
  Sunt doua diferente esentiale intre declansatori si procedurile stocate:
  - declansatorii se invoca implicit, iar procedurile explicit;
  - instructiunile *LCD* (*COMMIT*, *ROLLBACK*, *SAVEPOINT)* nu sunt permise in corpul unui declansator.

Predicate conditionale

In interiorul unui declansator care poate fi executat pentru diferite tipuri de instructiuni *LMD* se pot folosi trei functii booleene prin care se stabileste tipul operatiei executate. Aceste predicate conditionale sunt *INSERTING*, *UPDATING* si *DELETING*.

Functiile booleene nu solicita prefixarea cu numele pachetului si determina tipul operatiei *(INSERT, DELETE, UPDATE)*. De exemplu, predicatul *INSERTING* ia valoarea *TRUE* daca instructiunea declansatoare este *INSERT*. Similar sunt definite predicatele *UPDATING* si *DELETING*. Utilizand aceste predicate, in corpul declansatorului se pot executa secvente de instructiuni diferite, in functie de tipul operatiei *LMD*.

In cazul in care corpul declansatorului este un bloc *PL/SQL* complet (nu o comanda *CALL)*, pot fi utilizate atat predicatele *INSERTING*, *UPDATING*, *DELETING*, cat si identificatorii *:OLD*, *:NEW*.

Exemplu:
Se presupune ca in tabelul *galerie* se pastreaza (intr-o coloana numita *total val)* valoarea totala a operelor de arta expuse in galeria respectiva.

```
UPDATE galerie SET    total_val =
(SELECT SUM(valoare)
FROM opera
WHERE    opera.cod_galerie    =
galerie.cod_galerie);
```

Reactualizarea acestui camp poate fi implementata cu ajutorul unui declansator in urmatoarea maniera:

```
CREATE OR REPLACE PROCEDURE creste
(v_cod_galerie    IN galerie.cod_galerie%TYPE,
v_val    IN galerie.total_val%TYPE) AS
BEGIN    -  -
UPDATE galerie
SET    total val = NVL (total val, 0)    + v val
```

```
WHERE    cod_galerie = v_cod_galerie;
END creste;
/
CREATE OR REPLACE TRIGGER calcul_val
AFTER INSERT OR DELETE OR UPDATE OF valoare ON opera FOR EACH
ROW BEGIN
IF DELETING THEN
creste (:OLD.cod_galerie, -1*:OLD.valoare);
ELSIF UPDATING THEN
creste    (:NEW.cod_galerie,    :NEW.valoare    -
:OLD.valoare);
ELSE /* inserting */
creste (:NEW.cod_galerie, :NEW.valoare);
END IF;
END;
/
```

Declansatori *INSTEAD OF*

PL/SQL permite definirea unui nou tip de declansator, numit *INSTEAD OF,* care ofera o modalitate de actualizare a vizualizarilor obiect si a celor relationale.
Sintaxa acestui tip de declansator este similara celei pentru declansatori *LMD,* cu doua exceptii:

- clauza *{BEFORE | AFTER}* este lnlocuita prin *INSTEAD OF;*
- clauza *ON [schema.]nume_tabel* este lnlocuita printr-una din clauzele *ON [schema.]numeview* sau *ON NESTED TABLE (numecoloana) OF [schema.]nume_view.*

Declansatorul *INSTEAD OF* permite reactualizarea unei vizualizari prin comenzi *LMD.* O astfel de modificare nu poate fi realizata in alta maniera, din cauza regulilor stricte existente pentru reactualizarea vizualizarilor. Declansatorii de tip *INSTEAD OF* sunt necesari, deoarece vizualizarea pe care este definit declansatorul poate, de exemplu, sa se refere la *join-ul* unor tabele, si in acest caz, nu sunt actualizabile toate legaturile.

O vizualizare nu poate fi modificata prin comenzi *LMD* daca vizualizarea contine operatori pe multimi, functii grup, clauzele *GROUP BY, CONNECT BY, START WITH,* operatorul *DISTINCT* sau join-uri.

Declansatorul *INSTEAD OF* este utilizat pentru a executa operatii *LMD* direct pe tabelele de baza ale vizualizarii. De fapt, se scriu comenzi *LMD* relative la o vizualizare, iar declansatorul, in locul operatiei originale, va opera pe tabelele de baza.

De asemenea, acest tip de declansator poate fi definit asupra vizualizarilor ce au drept campuri tablouri imbricate, declansatorul furnizand o modalitate de reactualizare a elementelor tabloului imbricat.

In acest caz, el se declanseaza doar in cazul in care comenzile *LMD* opereaza asupra tabloului imbricat (numai cand elementele tabloului imbricat sunt modificate folosind clauzele *THE()* sau *TABLE())* si nu atunci cand comanda *LMD* opereaza doar asupra vizualizarii. Declansatorul permite accesarea liniei „parinte“ ce contine tabloul imbricat modificat.

Observatii:

- Spre deosebire de declansatorii *BEFORE* sau *AFTER*, declansatorii *INSTEAD OF* se executa in locul instructiunii *LMD (INSERT, UPDATE, DELETE)* specificate.
- Optiunea *UPDATE OF* nu este permisa pentru acest tip de declansator.
- Declansatorii *INSTEAD OF* se definesc pentru o vizualizare, nu pentru un tabel.
- Declansatorii *INSTEAD OF* actioneaza implicit la nivel de linie.
- Daca declansatorul este definit pentru tablouri imbricate, atributele *:OLD* si *:NEW* se refera la liniile tabloului imbricat, iar pentru a referi linia curenta din tabloul „parinte“ s-a introdus atributul *:PARENT.*

Exemplu:

Se considera *nouopera*, respectiv *nouartist*, copii ale tabelelor *opera*, respectiv *artist* si *viopar* o vizualizare definita prin compunerea naturala a celor doua tabele. Se presupune ca pentru fiecare artist exista un camp *(sum val)* ce reprezinta valoarea totala a operelor de arta expuse de acesta in muzeu.

Sa se defineasca un declansator prin care reactualizarile executate asupra vizualizarii *vi op ar* se vor transmite automat tabelelor *nou opera* si *nouartist.*

```
CREATE TABLE nou_opera AS
SELECT cod_opera, cod_artist, valoare, tip, stil FROM
opera;
CREATE TABLE nou_artist AS
SELECT cod_artist, nume, sum_val FROM artist;
CREATE VIEW vi_op_ar AS
SELECT cod_opera,o.cod_artist,valoare,tip,nume, sum
val
FROM    opera o, artist a
WHERE o.cod artist = a.cod artist
```

```
CREATE OR REPLACE TRIGGER react
INSTEAD OF INSERT OR DELETE OR UPDATE ON vi_op_ar
FOR EACH ROW
BEGIN
IF INSERTING THEN
INSERT INTO nou_opera
VALUES (:NEW.cod_opera, :NEW.cod_artist, :NEW.valoare,
:NEW.tip);
UPDATE nou_artist
SET    sum_val = sum_val + :NEW.valoare
WHERE cod_artist = :NEW.cod_artist;
ELSIF DELETING THEN
DELETE FROM nou_opera
WHERE cod_opera = :OLD.cod_opera;
UPDATE nou_artist
SET    sum_val = sum_val - :OLD.valoare
WHERE cod_artist = :OLD.cod_artist;
ELSIF UPDATING ('valoare') THEN UPDATE nou_opera SET
valoare = :NEW.valoare WHERE cod_opera = :OLD.cod_opera;
UPDATE nou_artist
SET    sum_val    = sum_val + (:NEW.valoare -
:OLD.valoare)
WHERE cod_artist = :OLD.cod_artist;
ELSIF UPDATING (’cod_artist’) THEN
        UPDATE  nou_ opera
        SET     cod_ artist   = :NEW.cod_ artist
        WHERE   cod_ opera    = :OLD.cod_opera;
        UPDATE  nou_ artist
        SET     sum_ val =   Sum_val -   :OLD.valoare
        WHERE   cod_ artist    = :OLD.cod _artist;
        UPDATE  nou_ artist
        SET     sum_ val =   sum val +   :NEW.valoare
        WHERE   cod_ artist   = :NEW.cod_ artist;
  END IF;
END;
/
```

Declansatori sistem

Declansatorii sistem sunt activati de comenzi *LDD* (*CREATE, DROP, ALTER*) si de anumite evenimente sistem *(STARTUP, SHUTDOWN, LOGON, LOGOFF, SERVERERROR, SUSPEND).* Un declansator sistem poate fi definit la nivelul bazei de date sau la nivelul schemei.

Sintaxa pentru crearea unui astfel de declansator este urmatoarea:

CREATE [OR REPLACE] TRIGGER [schema.]numedeclansator {BEFORE | AFTER}
{listaevenimenteLDD | lista evenimente baza}
ON {DATABASE | SCHEMA}
[WHEN(*conditie*)] *corpdeclansator;*

Cuvintele cheie *DATABASE* sau *SCHEMA* specifica nivelul declansatorului.

Exista restrictii asupra expresiilor din conditia clauzei *WHEN*. De exemplu, declansatorii *LOGON* si *LOGOFF* pot verifica doar identificatorul *(userid)* si numele utilizatorului *(username),* iar declansatorii *LDD* pot verifica tipul si numele obiectelor definite, identificatorul si numele utilizatorului.

Evenimentele amintite anterior pot fi asociate clauzelor *BEFORE* sau *AFTER.* De exemplu, un declansator *LOGON (AFTER)* se activeaza dupa ce un utilizator s-a conectat la baza de date, un declansator *CREATE (BEFORE* sau *AFTER)* se activeaza inainte sau dupa ce a fost creat un obiect al bazei, un declansator *SERVERERROR (AFTER)* se activeaza ori de cate ori apare o eroare (cu exceptia erorilor: *ORA-01403, ORA-01422, ORA-01423, ORA-01034, ORA-04030).*

Pentru declansatorii sistem se pot utiliza functii speciale care permit obtinerea de informatii referitoare la evenimentul declansator. Ele sunt functii *PL/SQL* stocate care trebuie prefixate de numele proprietarului *(SYS).*

Printre cele mai importante functii care furnizeaza informatii referitoare la evenimentul declansator, se remarca:

- *SYSEVENT* - returneaza evenimentul sistem care a activat declansatorul (este de tip *VARCHAR2(20)* si este aplicabila oricarui eveniment);
- *DATABASE NAME* - returneaza numele bazei de date curente (este de tip *VARCHAR2(50)* si este aplicabila oricarui eveniment);
- *LOGIN USER* - returneaza identificatorul utilizatorului care activeaza declansatorul (este de tip *VARCHAR2(30)* si este aplicabila oricarui eveniment);

Exemplu:

```
CREATE OR REPLACE TRIGGER logutiliz AFTER CREATE ON
SCHEMA BEGIN
INSERT INTO ldd_tab(user_id, object_name, creation_date)
VALUES    (USER, SYS.DICTIONARY_OBJ_NAME, SYSDATE);
END logutiliz;
```

Evenimentul *SERVERERROR* poate fi utilizat pentru a urmari erorile care apar in baza de date. Codul erorii este furnizat, prin intermediul declan satorului, de functia *SERVER ERROR*, iar mesajul asociat erorii poate fi obtinut cu procedura *DBMS_ UTILITY.FORMA T_ERROR_STA CK.*

Exemplu:

```
CREATE TABLE erori ( moment    DATE,
utilizator VARCHAR2(30), nume_baza    VARCHAR2(50),
stiva_erori VARCHAR2(2000) );
/

CREATE OR REPLACE TRIGGER logerori AFTER SERVERERROR ON
DATABASE
BEGIN
INSERT INTO erori
VALUES (SYSDATE, SYS.LOGIN_USER, SYS.DATABASE_NAME,
DBMS_UTILITY.FORMAT_ERROR_STACK);
END logerori;
/
```

Modificarea si suprimarea declansatorilor

Optiunea *OR REPLACE* din cadrul comenzii *CREATE TRIGGER* recreeaza declansatorul, daca acesta exista. Clauza permite schimbarea definitiei unui declansator existent fara suprimarea acestuia.
Similar procedurilor si pachetelor, un declansator poate fi suprimat prin:

DROP TRIGGER [schema.]nume_declansator;

Uneori actiunea de suprimare a unui declansator este prea drastica si este preferabila doar dezactivarea sa temporara. In acest caz, declansatorul va continua sa existe in dictionarul datelor.
Modificarea unui declansator poate consta din recompilarea *(COMPILE),*

redenumirea *(RENAME),* activarea *(ENABLE)* sau dezactivarea *(DISABLE)* acestuia si se realizeaza prin comanda:

ALTER TRIGGER [schema.]nume declansator {ENABLE | DISABLE | COMPILE | RENAME TO nume_nou}
{ALL TRIGGERS}

Daca un declansator este activat, atunci sistemul *Oracle* il executa ori de cate ori au loc operatiile precizate in declansator asupra tabelului asociat si cand conditia de restrictie este indeplinita. Daca declansatorul este dezactivat, atunci sistemul *Oracle* nu il va mai executa. Dupa cum s-a mai subliniat, dezactivarea unui declansator nu implica stergerea acestuia din dictionarul datelor.
Toti declansatorii asociati unui tabel pot fi activati sau dezactivati utilizand optiunea *ALL TRIGGERS (ENABLE ALL TRIGGERS,* respectiv *DISABLE ALL TRIGGERS).* Declansatorii sunt activati in mod implicit atunci cand sunt creati.
Pentru activarea *(enable)* unui declansator, server-ul *Oracle:*

- verifica integritatea constrangerilor,
- garanteaza ca declansatorii nu pot compromite constrangerile de integritate,
- garanteaza consistenta la citire a vizualizarilor,
- gestioneaza dependentele.

Activarea si dezactivarea declansatorilor asociati unui tabel se poate realiza si cu ajutorul comenzii *ALTER TABLE.*
Un declansator este compilat in mod automat la creare. Daca un *site* este neutilizabil atunci cand declansatorul trebuie compilat, sistemul *Oracle* nu poate valida comanda de accesare a bazei distante si compilarea esueaza.

Informatii despre declansatori

In DD exista vizualizari ce contin informatii despre declansatori si despre starea acestora (*USER_TRIGGERS, USER_TRIGGER_COL, ALL_TRIGGERS, DBATRIGGERS* etc.). Aceste vizualizari sunt actualizate ori de cate ori un declansator este creat sau suprimat.
Atunci cand declansatorul este creat, codul sau sursa este stocat in vizualizarea *USERTRIGGERS.* Vizualizarea *ALLTRIGGERS* contine
informatii despre toti declansatorii din baza de date. Pentru a detecta dependentele declansatorilor poate fi consultata vizualizarea *USER DEPENDENCIES,* iar *ALLDEPENDECIES* contine informatii despre dependentele tuturor obiectelor din baza de date. Erorile rezultate din compilarea declansatorilor pot fi analizate din vizualizarea *USERERRORS,* iar prin comanda *SHOW ERRORS* se vor afisa erorile corespunzatoare ultimului declansator compilat.

In operatiile de gestiune a bazei de date este necesara uneori reconstruirea instructiunilor *CREATE TRIGGER*, atunci cand codul sursa original nu mai este disponibil. Aceasta se poate realiza utilizand vizualizarea *USERTRIGGERS.*
Vizualizarea include numele declansatorului *(TRIGGERNAME),* tipul acestuia *(TRIGGERTYPE),* evenimentul declansator *(TRIGGERINGEVENT),* numele proprietarului tabelului *(TABLEOWNER),* numele tabelului pe care este definit declansatorul *(TABLENAME),* clauza *WHEN (WHENCLAUSE),* corpul declansatorului *(TRIGGERBODY),* antetul *(DESCRIPTION),* starea acestuia *(STATUS)* care poate sa fie *ENABLED* sau *DISABLED* si numele utilizate pentru a referi parametrii *OLD* si *NEW (REFERENCINGNAMES).* Daca obiectul de baza nu este un tabel sau o vizualizare, atunci *TABLENAME* este *null.*

Exemplu:

Presupunand ca nu este disponibil codul sursa pentru declansatorul *alfa*, sa se reconstruiasca instructiunea *CREATE TRIGGER* corespunzatoare acestuia.

```
SELECT 'CREATE OR REPLACE TRIGGER '    || DESCRIPTION ||
TRIGGER_BODY FROM    USER_TRIGGERS
WHERE    TRIGGER_NAME = 'ALFA';
```

Cu aceasta interogare se pot reconstrui numai declansatorii care apartin contului utilizator curent. O interogare a vizualizarilor *ALL TRIGGERS* sau *DBATRIGGERS* permite reconstruirea tuturor declansatorilor din sistem, daca se dispune de privilegii *DBA.*

Exemplu:

```
SELECT USERNAME FROM    USER_USERS;
```

Aceasta cerere furnizeaza numele "proprietarului" (creatorului) declansatorului si nu numele utilizatorului care a reactualizat tabelul.

Privilegii sistem

Sistemul furnizeaza privilegii sistem pentru gestiunea declansatorilor:

- *CREATE TRIGGER* (permite crearea declansatorilor in schema personala);
- *CREATE ANY TRIGGER* (permite crearea declansatorilor in orice schema cu exceptia celei corespunzatoare lui *SYS);*
- *ALTER ANY TRIGGER* (permite activarea, dezactivarea sau compilarea declansatorilor in orice schema cu exceptia lui *SYS);*
- *DROP ANY TRIGGER* (permite suprimarea declansatorilor la nivel de baza de date in orice schema cu exceptia celei corespunzatoate lui *SYS);*
- *ADMINISTER DATABASE TRIGGER* (permite crearea sau modificarea unui declansator sistem referitor la baza de date);
- *EXECUTE* (permite referirea, in corpul declansatorului, a procedurilor, functiilor sau pachetelor din alte scheme).

Tabele *mutating*

Asupra tabelelor si coloanelor care pot fi accesate de corpul declansatorului exista anumite restrictii. Pentru a analiza aceste restrictii este necesara definirea tabelelor in schimbare *(mutating)* si constranse *(constraining).*

Un tabel *constraining* este un tabel pe care evenimentul declansator trebuie sa-l consulte fie direct, printr-o instructiune *SQL,* fie indirect, printr-o constrangere de integritate referentiala declarata. Tabelele nu sunt considerate *constraining* in cazul declansatorilor la nivel de instructiune. Comenzile *SQL* din corpul unui declansator nu pot modifica valorile coloanelor care sunt declarate chei primare, externe sau unice (*PRIMARY KEY*, *FOREIGN KEY*, *UNIQUE KEY*) intr-un tabel *constraining*.

Un tabel *mutating* este tabelul modificat de instructiunea *UPDATE*, *DELETE* sau *INSERT*, sau un tabel care va fi actualizat prin efectele actiunii integritatii referentiale *ON DELETE CASCADE*. Chiar tabelul pe care este definit declansatorul este un tabel *mutating*, ca si orice tabel referit printr-o constrangere *FOREING KEY.*

Tabelele nu sunt considerate *mutating* pentru declansatorii la nivel de instructiune, cu exceptia celor declansati ca efect al optiunii ON *DELETE CASCADE.* Vizualizarile nu sunt considerate *mutating* in declansatorii *INSTEAD OF*.

Regula care trebuie respectata la utilizarea declansatoriilor este: **comenzile** *SQL* **din corpul unui declansator nu pot consulta sau modifica date dintr-un tabel** *mutating.*

Exceptia! Daca o comanda *INSERT* afecteaza numai o inregistrare, declansatorii la nivel de linie *(BEFORE* sau *AFTER)* pentru inregistrarea respectiva nu trateaza tabelul ca fiind *mutating.* Acesta este unicul caz in care un declansator la nivel de linie poate citi sau modifica tabelul. Comanda *INSERT INTO tabel SELECT* ... considera tabelul *mutating* chiar daca cererea returneaza o singura linie.

Exemplu:

```
CREATE OR REPLACE TRIGGER cascada
  AFTER UPDATE OF cod artist ON artist FOR EACH ROW
BEGIN
    UPDATE opera
    SET    opera.cod_artist= :NEW.cod_artist
     WHERE opera.cod_artist= :OLD.cod artist
END;

    UPDATE  artist
       SET   cod_artist = 71
     WHERE  cod artist = 23;
```

La executia acestei secvente este semnalata o eroare. Tabelul *artist* referentiaza tabelul *opera* printr-o constrangere de cheie externa. Prin urmare, tabelul *opera* este *constraining,* iar declansatorul *cascada* incearca sa schimbe date in tabelul *constraining,* ceea ce nu este permis. Exemplul va functiona corect daca nu este definita sau activata constrangerea referentiala intre cele doua tabele.

Exemplu:

Sa se implementeze cu ajutorul unui declansator restrictia ca intr-o sala pot sa fie expuse maximum 10 opere de arta.

```
     CREATE OR REPLACE TRIGGER TrLimitaopere
     BEFORE INSERT OR UPDATE OF cod_sala ON opera FOR EACH
ROW DECLARE
     v_Max_opere CONSTANT NUMBER := 10;
     v_opere_curente    NUMBER;
     BEGIN
     SELECT COUNT(*) INTO v_opere_curente FROM opera
     WHERE cod_sala = :NEW.cod_sala;
     IF v_opere_curente + 1 > v_Max_opere THEN
     RAISE_APPLICATION_ERROR(-20000,'Prea multe opere de
arta in sala avand codul '     || :NEW.cod_sala);
     END IF;
     END TrLimitaopere;
```

Cu toate ca declansatorul pare sa produca lucrul dorit, totusi dupa o reactualizare a tabelului *opera* in urmatoarea maniera:

```
     INSERT INTO opera (cod_opera, cod_sala)
     VALUES (756893, 10);
```

se obtine urmatorul mesaj de eroare:

ORA-04091: tabel opera is mutating, trigger/function may not see it
ORA-04088: error during execution of trigger

Eroarea *ORA-04091* apare deorece declansatorul *TrLimitaopere* consulta chiar tabelul *(opera)* la care este asociat declansatorul *(mutating)*.
Tabelul *opera* este *mutating* doar pentru un declansator la nivel de linie. Aceasta inseamna ca tabelul poate fi consultat in interiorul unui declansator la nivel de instructiune. Totusi, limitarea numarului operelor de arta nu poate fi facuta in interiorul unui declansator la nivel de instructiune, din moment ce este necesara valoarea *:NEW.cod_sala* in corpul declansatorului.

```
CREATE OR REPLACE PACKAGE PSalaDate AS
TYPE t_cod_sala IS TABLE OF opera.cod_sala%TYPE -   - INDEX
BY BINARY_INTEGER;-
TYPE t_cod_opera IS TABLE OF opera.cod_opera%TYPE -   - INDEX
BY BINARY_INTEGER;-
v_cod_sala    t_cod_sala;
v_cod_opera    t_cod_opera;
v_NrIntrari BINARY_INTEGER := 0;
END PSalaDate;
```

```
CREATE OR REPLACE TRIGGER TrLLimitaSala BEFORE INSERT OR
UPDATE OF cod_sala ON opera FOR EACH ROW BEGIN
PSalaDate.v_NrIntrari := PSalaDate.v_NrIntrari + 1;
PSalaDate.v_cod_sala(PSalaDate.v_NrIntrari) :=
:NEW.cod_sala;
PSalaDate.v_cod_opera(PSalaDate.v_NrIntrari) :=
:NEW.cod_opera;
END TrLLimitasala;
```

```
  CREATE OR REPLACE TRIGGER TrILimitaopere
  AFTER INSERT OR UPDATE OF cod_sala ON opera
  DECLARE
    v_Max_opere v    CONSTANT NUMBER := 10;
  opere curente v    NUMBER;
  cod operax         opera.cod_opera%TYPE;
  v_cod_salax BEGIN  opera.cod_sala%TYPE;

    FOR v_LoopIndex IN 1..PsalaDate.v_NrIntrari LOOP
      v_cod_operax := PsalaDate.v_cod_opera(v_LoopIndex);
      v_cod_salax := PsalaDate.v_cod_sala(v_LoopIndex);
      SELECT COUNT(*)    -   -   -
      INTO    v_opere_curente
      FROM    opera
      WHERE    cod_sala = v_cod_salax;
      IF v_opere_curente > v_Max_opere THEN
        RAISE_APPLICATION_ERROR(-20000, 'Prea multe opere de
        arta in sala' || v_cod_salax || 'din cauza inserarii
        operei avand codul' || v_cod_operax)
```

```
     END IF;
     END LOOP;
     /* Reseteaza contorul deoarece urmatoarea executie va
folosi date noi */
     PSalaDate.v_NrIntrari := 0;
     END TrILimitaopere;
```

O solutie pentru acesta problema este crearea a doi declansatori, unul la nivel de linie si altul la nivel de instructiune. In declansatorul la nivel de linie se inregistreaza valoarea lui *:NEW.cod_opera,* dar nu va fi interogat tabelul *opera.*

Interogarea va fi facuta in declansatorul la nivel de instructiune si va folosi valoarea inregistrata in declansatorul la nivel de linie.

O modalitate pentru a inregistra valoarea lui *:NEW.cod_opera* este utilizarea unui tablou indexat in interiorul unui pachet.

Exemplu:

Sa se creeze un declansator care:

a) daca este eliminata o sala, va sterge toate operele expuse in sala respectiva;

b) daca se schimba codul unei sali, va modifica aceasta valoare pentru fiecare opera de arta expusa in sala respectiva.

```
     CREATE OR REPLACE TRIGGER sala_cascada
     BEFORE DELETE OR UPDATE OF cod_sala ON sala FOR EACH
ROW BEGIN
     IF DELETING THEN DELETE FROM opera WHERE    cod_sala =
:OLD.cod_sala;
     END IF;
     IF UPDATING AND :OLD.cod_sala != :NEW.cod_sala THEN
UPDATE opera
     SET    cod_sala = :NEW.cod_sala
     WHERE    cod_sala = :OLD.cod_sala;
     END IF;
     END sala_cascada;
```

Declansatorul anterior realizeaza constrangerea de integritate *UPDATE* sau *ON DELETE CASCADE,* adica stergerea sau modificarea cheii primare a unui tabel „parinte“ se va reflecta si asupra inregistrarilor corespunzatoare din tabelul „copil“.

Executarea acestuia, pe tabelul *sala* (tabelul „parinte“), va duce la efectuarea a doua tipuri de operatii pe tabelul *opera* (tabelul „copil“).

La eliminarea unei sali din tabelul *sala,* se vor sterge toate operele de arta corespunzatoare acestei sali.

```
DELETE FROM sala WHERE cod_sala = 773;
```

La modificarea codului unei sali din tabelul *sala*, se va actualiza codul salii atat in tabelul *sala*, cat si in tabelul *opera*.

```
UPDATE sala
SET    cod_sala = 777
WHERE    cod~sala = 333;
```

Se presupune ca asupra tabelului *opera* exista o constrangere de integritate:

```
FOREIGN KEY (cod_sala) REFERENCES sala(cod_sala)
```

In acest caz sistemul *Oracle* va afisa un mesaj de eroare prin care se precizeaza ca tabelul *sala* este *mutating*, iar constrangerea definita mai sus nu poate fi verificata.

```
ORA-04091: table MASTER.SALA is mutating,
trigger/function may not see it
```

Pachetele pot fi folosite pentru incapsularea detaliilor logice legate de declansatori. Exemplul urmator arata un mod simplu de implementare a acestei posibilitati. Este permisa apelarea unei proceduri sau functii stocate din blocul *PL/SQL* care reprezinta corpul declansatorului.

Exemplu:

```
CREATE OR REPLACE PACKAGE pachet IS
PROCEDURE procesare_trigger(pvaloare IN NUMBER,
_    pstare    IN VARCHAR2);
END pachet;
CREATE OR REPLACE PACKAGE BODY pachet IS
PROCEDURE procesare_trigger(pvaloare IN NUMBER,
_    pstare    IN VARCHAR2) IS
BEGIN

END procesare_trigger;

END pachet;

CREATE OR REPLACE TRIGGER gama AFTER INSERT ON opera FOR
EACH ROW BEGIN
pachet.procesare_trigger(:NEW.valoare,:NEW.stare)
END;
```