

9. Gestiunea declanșatorilor

Un declanșator (*trigger*) este un bloc *PL/SQL* care se execută automat ori de câte ori are loc un anumit eveniment „declanșator”. Evenimentul poate consta din modificarea unui tabel sau a unei vizualizări, din acțiuni sistem sau chiar anumite acțiuni utilizator. Blocul *PL/SQL* poate fi asociat unui tabel, unei vizualizări, unei scheme sau unei baze de date. La fel ca și pachetele, declanșatorii nu pot fi locali unui bloc sau unui pachet, ei trebuie depuși ca obiecte independente în baza de date.

Folosirea declanșatorilor garantează faptul că atunci când o anumită operație este efectuată, automat sunt executate niște acțiuni asociate. Evident, nu trebuie introduși declanșatori care ar putea să substituie funcționalități oferite deja de sistem. De exemplu, nu are sens să fie definiți declanșatori care să implementeze regulile de integritate ce pot fi definite, mai simplu, prin constrângeri declarative.

Tipuri de declanșatori

Declanșatorii pot fi:

- ☐ la nivel de bază de date (*database triggers*);
- ☐ la nivel de aplicație (*application triggers*).

Declanșatorii bază de date se execută automat ori de câte ori are loc:

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

Declanșatorii bază de date sunt de trei tipuri:

- ☐ declanșatori *LMD* – activați de comenzi *LMD* (*INSERT*, *UPDATE* sau *DELETE*) executate asupra unui tabel al bazei de date;
- ☐ declanșatori *INSTEAD OF* – activați de comenzi *LMD* executate asupra unei vizualizări (relaționale sau obiect);
- ☐ declanșatori sistem – activați de un eveniment sistem (oprirea sau pornirea bazei), de comenzi *LDD* (*CREATE*, *ALTER*, *DROP*), de conectarea (deconectarea) unui utilizator. Ei sunt definiți la nivel de schemă sau la nivel de bază de date.

Declanșatorii asociați unui tabel (stocați în baza de date) vor acționa

indiferent de aplicația care a efectuat operația *LMD*. Dacă operația *LMD* se referă la o vizualizare, declanșatorul *INSTEAD OF* definește acțiunile care vor avea loc, iar dacă aceste acțiuni includ comenzi *LMD* referitoare la tabele, atunci declanșatorii asociați acestor tabele sunt și ei, la rândul lor, activați. Dacă declanșatorii sunt asociați unei baze de date, ei se declanșează pentru fiecare eveniment, pentru toți utilizatorii. Dacă declanșatorii sunt asociați unei scheme sau unui tabel, ei se declanșează numai dacă evenimentul declanșator implică acea schemă sau acel tabel. Un declanșator se poate referi la un singur tabel sau la o singură vizualizare.

Declanșatorii aplicație se execută implicit ori de câte ori apare un eveniment particular într-o aplicație. De exemplu, o aplicație dezvoltată cu *Developer Suite. Form Builder* utilizează frecvent acest tip de declanșatori (*form builder triggers*). Ei pot fi declanșați prin apăsarea unui buton, prin navigarea pe un câmp etc. În acest capitol se va face referință doar la declanșatorii bază de date.

Atunci când un pachet sau un subprogram este depus în dicționarul datelor, alături de codul sursă este depus și *p-codul* compilat. În mod similar se întâmplă și pentru declanșatori. Prin urmare, un declanșator poate fi apelat fără recompilare. Declanșatorii pot fi invalidați în aceeași manieră ca pachetele și subprogramele. Dacă declanșatorul este invalidat, el va fi recompilat la următoarea activare.

Crearea declanșatorilor *LMD*

Declanșatorii *LMD* sunt creați folosind comanda *CREATE TRIGGER* care are următoarea sintaxă generală:

```
CREATE [OR REPLACE] TRIGGER [schema.]nume_declanșator
  {BEFORE | AFTER}
  {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 (bloc PL/SQL sau apelul unei proceduri);
```

Numele declanșatorului trebuie să fie unic printre numele declanșatorilor din cadrul aceleiași scheme, dar poate să coincidă cu numele altor obiecte ale acesteia (de exemplu, tabele, vizualizări sau proceduri).

La crearea unui declanșator este obligatorie una dintre opțiunile *BEFORE*

sau *AFTER*, prin care se precizează momentul în care este executat corpul declanșatorului. Acesta nu poate depăși 32KB.

Până la versiunea *Oracle8i*, corpul unui declanșator trebuia să fie un bloc *PL/SQL*. În ultimele versiuni, corpul poate consta doar dintr-o singură comandă *CALL*. Procedura apelată poate fi un subprogram *PL/SQL* stocat, o rutină *C* sau o metodă *Java*. În acest caz, *CALL* nu poate conține clauza *INTO* care este specifică funcțiilor, iar pentru a referi coloanele tabelului asociat declanșatorului, acestea trebuie prefixate de atributele *:NEW* sau *:OLD*. De asemenea, în expresia parametrilor nu pot să apară variabile *bind*.

Declararea unui declanșator trebuie să cuprindă tipul comenzii *SQL* care duce la executarea declanșatorului și tabelul asociat acestuia. În ceea ce privește tipul comenzii *SQL* care va duce la executarea declanșatorului, sunt incluse următoarele tipuri de opțiuni: *DELETE*, *INSERT*, *UPDATE* sau o combinație a acestora cu operatorul logic *OR*. Cel puțin una dintre aceste opțiuni este obligatorie.

În declararea declanșatorului este specificat tabelul asupra căruia va fi executat declanșatorul. *Oracle9i* admite tablouri imbricate. Dacă declanșatorul este de tip *UPDATE*, atunci pot fi enumerate coloanele pentru care acesta se va executa.

În corpul fiecărui declanșator pot fi cunoscute valorile coloanelor atât înainte de modificarea unei linii, cât și după modificarea acesteia. Valoarea unei coloane înainte de modificare este referită prin atributul *OLD*, iar după modificare, prin atributul *NEW*. Prin intermediul clauzei opționale *REFERENCING* din sintaxa comenzii de creare a declanșatorilor, atributele *NEW* și *OLD* pot fi redenumite.

Un declanșator poate activa alt declanșator, iar acesta la rândul său poate activa alt declanșator etc. Această situație (declanșatori în cascadă) poate avea însă efecte imprevizibile. Sistemul *Oracle* permite maximum 32 declanșatori în cascadă. Numărul acestora poate fi limitat (utilizând parametrul de inițializare *OPEN_CURSORS*), deoarece pentru fiecare execuție a unui declanșator trebuie deschis un nou cursor.

Declanșatorii la nivel de baze de date pot fi de două feluri:

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

Declanșatori la nivel de instrucțiune

Declanșatorii la nivel instrucțiune sunt executați o singură dată pentru instrucțiunea declanșatoare, indiferent de numărul de linii afectate (chiar dacă

4

nici o linie nu este afectată). Un declanșator la nivel de instrucțiune este util dacă acțiunea declanșatorului nu depinde de informațiile din liniile afectate.

Exemplu:

Programul de lucru la administrația unei expoziții este de luni până vineri, în intervalul (8:00 a.m. - 10:00 p.m.). Să se construiască un declanșator la nivel de instrucțiune care împiedică orice activitate asupra unui tabel al bazei de date, în 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 (-20001, '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;
/
```

Declanșatori la nivel de linie

Declanșatorii la nivel de linie sunt creați cu opțiunea *FOR EACH ROW*. În acest caz, declanșatorul este executat pentru fiecare linie din tabelul afectat, iar dacă evenimentul declanșator nu afectează nici o linie, atunci declanșatorul nu este executat. Dacă opțiunea *FOR EACH ROW* nu este inclusă, declanșatorul este considerat implicit la nivel de instrucțiune.

Declanșatorii la nivel linie nu sunt performanți dacă se fac frecvent reactualizări pe tabele foarte mari.

Restricțiile declanșatorilor pot fi incluse prin specificarea unei expresii booleene în clauza *WHEN*. Această expresie este evaluată pentru fiecare linie afectată de către declanșator. Declanșatorul este executat pentru o linie, doar

5

dacă expresia este adevărată pentru acea linie. Clauza *WHEN* este validă doar pentru declanșatori la nivel de linie.

Exemplu:

Să se implementeze cu ajutorul unui declanșator constrângerea că valorile fotografiilor nu pot fi reduse (trei variante).

Varianta 1:

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

Varianta 2:

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

Varianta 3:

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

Accesul la vechile și noile valori ale coloanelor liniei curente, afectată de evenimentul declanșator, se face prin: *OLD.ume_coloană* (vechea valoare), respectiv prin *NEW.ume_coloană* (noua valoare). În cazul celor trei comenzi *LMD*, aceste valori devin:

<i>INSERT</i>	: <i>NEW.ume_coloană</i> → noua valoare
	(: <i>OLD.ume_coloană</i> → <i>NULL</i>);
<i>UPDATE</i>	: <i>NEW.ume_coloană</i> → noua valoare

```

: OLD.ume_coloană → vechea valoare;
DELETE (: NEW.ume_coloană → NULL )
: OLD.ume_coloană → vechea valoare.

```

Exemplu:

Se presupune că pentru fiecare sală există două câmpuri (*min_valoare* și *max_valoare*) în care se rețin limitele minime și maxime ale valorilor fotografiilor din sala respectivă. Să se implementeze cu ajutorul unui declanșator constrângerea că, dacă aceste limite s-ar modifica, orice fotografie trebuie să aibă valoarea cuprinsă între noile limite.

```

CREATE OR REPLACE TRIGGER verifica_limite
  BEFORE UPDATE OF min_valoare, max_valoare ON sala
  FOR EACH ROW
DECLARE
  v_min_val  fotografie.valoare%TYPE;
  v_max_val  fotografie.valoare%TYPE;
  e_invalid  EXCEPTION;
BEGIN
  SELECT MIN(valoare), MAX(valoare)
  INTO   v_min_val, v_max_val
  FROM   fotografie
  WHERE  cod_sala = :NEW.cod_sala;
  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
fotografii ale caror valori sunt in afara domeniului
permis');
END verifica_limite;
/

```

Ordinea de execuție a declanșatorilor

PL/SQL permite definirea a 12 tipuri de declanșatori care sunt obținuți prin combinarea proprietății de moment (timp) al declanșării (*BEFORE*, *AFTER*), cu proprietatea nivelului la care acționează (nivel linie, nivel instrucțiune) și cu tipul operației atașate declanșatorului (*INSERT*, *UPDATE*, *DELETE*).

De exemplu, *BEFORE INSERT* acționează o singură dată, înainte de executarea unei instrucțiuni *INSERT*, iar *BEFORE INSERT FOR EACH ROW* acționează înainte de inserarea fiecărei noi înregistrări.

Declanșatorii sunt activați când este executată o comandă *LMD*. La apariția unei astfel de comenzi se execută câteva acțiuni care vor fi descrise în continuare.

- 1) Se execută declanșatorii la nivel de instrucțiune *BEFORE*.
- 2) Pentru fiecare linie afectată de comanda *LMD*:
 - 2.1) se execută declanșatorii la nivel de linie *BEFORE*;
 - 2.2) se blochează și se modifică linia afectată (se execută comanda *LMD*), se verifică constrângerile de integritate (blocarea rămâne valabilă până în momentul în care tranzacția este permanentizată);
 - 2.3) se execută declanșatorii la nivel de linie *AFTER*.
- 3) Se execută declanșatorii la nivel de instrucțiune *AFTER*.

Începând cu versiunea *Oracle8i* algoritmul anterior se schimbă, în sensul că verificarea constrângerii referențiale este amânată după executarea declanșatorului la nivel linie.

Obsevații:

- ☐ În expresia clauzei *WHEN* nu pot fi incluse funcții definite de utilizator sau subcereri *SQL*.
- ☐ În clauza *ON* poate fi specificat un singur tabel sau o singură vizualizare.
- ☐ În interiorul blocului *PL/SQL*, coloanele tabelului prefixate cu *OLD* sau *NEW* sunt considerate variabile externe și deci, trebuie precedate de caracterul „:”.
- ☐ Condiția de la clauza *WHEN* poate conține coloane prefixate cu *OLD* sau *NEW*, dar în acest caz, acestea nu trebuie precedate de „:”.
- ☐ Declanșatorii bază de date pot fi definiți numai pe tabele (excepție, declanșatorul *INSTEAD OF* care este definit pe o vizualizare). Totuși, dacă o comandă *LMD* este aplicată unei vizualizări, pot fi activați declanșatorii asociați tabelelor care definesc vizualizarea.
- ☐ Corpul unui declanșator nu poate conține o interogare sau o reactualizare a unui tabel aflat în plin proces de modificare, pe timpul acțiunii declanșatorului (*mutating table*).
- ☐ Blocul *PL/SQL* care descrie acțiunea declanșatorului nu poate conține comenzi pentru gestiunea tranzacțiilor (*COMMIT*, *ROLLBACK*, *SAVEPOINT*). Controlul tranzacțiilor este permis, însă, în procedurile

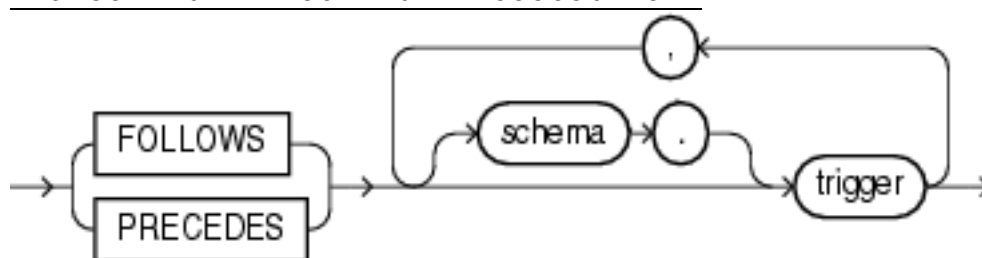
stocate. Dacă un declanșator apelează o procedură stocată care execută o comandă referitoare la controlul tranzacțiilor, atunci va apărea o eroare la execuție și tranzacția va fi anulată.

- ☐ Comenzile *LDD* nu pot să apară decât în declanșatorii sistem.
- ☐ Corpul declanșatorului poate să conțină comenzi *LMD*.
- ☐ În corpul declanșatorului pot fi referite și utilizate coloane *LOB*, dar nu pot fi modificate valorile acestora.
- ☐ În corpul declanșatorului se pot insera date în coloanele de tip *LONG* și *LONGRAW*, dar nu pot fi declarate variabile de acest tip.
- ☐ Dacă un tabel este suprimat (se șterge din dicționarul datelor), automat sunt distruși toți declanșatorii asociați tabelului.
- ☐ Nu este indicată crearea declanșatorilor recursivi.
- ☐ Este necesară limitarea dimensiunii unui declanșator. Dacă acesta solicită mai mult de 60 linii de cod, atunci este preferabil ca o parte din cod să fie inclusă într-o procedură stocată și aceasta să fie apelată din corpul declanșatorului.

Sunt două diferențe esențiale între declanșatori și procedurile stocate:

- ☐ declanșatorii se invocă implicit, iar procedurile explicit;
- ☐ instrucțiunile *LCD* (*COMMIT*, *ROLLBACK*, *SAVEPOINT*) nu sunt permise în corpul unui declanșator.

<https://docs.oracle.com/en/database/oracle/oracle-database/21/lnpls/CREATE-TRIGGER-statement.html#GUID-AF9E33F1-64D1-4382-A6A4-EC33C36F237B>



Mai multe detalii la

<https://oracle-base.com/articles/11g/trigger-enhancements-11gr1>

Exemplu schimbare ordine execuție:

```

CREATE TABLE trigger_follows_test (
  id          NUMBER,
  description VARCHAR2(50)

```


9

```
);  
CREATE OR REPLACE TRIGGER trigger_follows_test_trg_2  
BEFORE INSERT ON trigger_follows_test  
FOR EACH ROW  
BEGIN  
    DBMS_OUTPUT.put_line('TRIGGER_FOLLOWS_TEST_TRG_2 -  
Executed');  
END;  
/  
CREATE OR REPLACE TRIGGER trigger_follows_test_trg_1  
    BEFORE INSERT ON trigger_follows_test  
FOR EACH ROW  
FOLLOWS trigger_follows_test_trg_2  
BEGIN  
    DBMS_OUTPUT.put_line('TRIGGER_FOLLOWS_TEST_TRG_1 -  
Executed');  
END;  
/  
INSERT INTO trigger_follows_test VALUES (2, 'TWO');  
SELECT *  
FROM trigger_follows_test;
```

Predicate condiționale

În interiorul unui declanșator care poate fi executat pentru diferite tipuri de instrucțiuni *LMD* se pot folosi trei funcții booleene prin care se stabilește tipul operației executate. Aceste predicate condiționale (furnizate de pachetul standard *DBMS_STANDARD*) sunt *INSERTING*, *UPDATING* și *DELETING*.

Funcțiile booleene nu solicită prefixarea cu numele pachetului și determină tipul operației (*INSERT*, *DELETE*, *UPDATE*). De exemplu, predicatul *INSERTING* ia valoarea *TRUE* dacă instrucțiunea declanșatoare este *INSERT*. Similar sunt definite predicatele *UPDATING* și *DELETING*. Utilizând aceste predicate, în corpul declanșatorului se pot executa secvențe de instrucțiuni diferite, în funcție de tipul operației *LMD*.

În cazul în care corpul declanșatorului este un bloc *PL/SQL* complet (nu o comandă *CALL*), pot fi utilizate atât predicatele *INSERTING*, *UPDATING*, *DELETING*, cât și identificatorii *:OLD*, *:NEW*, *:PARENT*.

Exemplu:

Se presupune că în tabelul *sala* se păstrează (într-o coloană numită *total_val*) valoarea totală a fotografiilor expuse în sala respectivă.

10

```
UPDATE sala
SET     total_val =
        (SELECT SUM(valoare)
         FROM   fotografie
         WHERE  fotografie.cod_sala =
sala.cod_sala);
```

Reactualizarea acestui câmp poate fi implementată cu ajutorul unui declanșator în următoarea manieră:

```
CREATE OR REPLACE PROCEDURE creste
    (v_cod_sala      IN sala.cod_sala%TYPE,
     v_val           IN sala.total_val%TYPE) AS
BEGIN
    UPDATE  sala
    SET     total_val = NVL (total_val, 0) + v_val
    WHERE   cod_sala = v_cod_sala;
END creste;
/

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

Declanșatori INSTEAD OF

PL/SQL permite definirea unui nou tip de declanșator, numit *INSTEAD OF*, care oferă o modalitate de actualizare a vizualizărilor obiect și a celor relaționale.

Sintaxa acestui tip de declanșator este similară celei pentru declanșatori

LMD, cu două excepții:

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

Declanșatorul *INSTEAD OF* permite reactualizarea unei vizualizări prin comenzi *LMD*. O astfel de modificare nu poate fi realizată în altă manieră, din cauza regulilor stricte existente pentru reactualizarea vizualizărilor. Declanșatorii de tip *INSTEAD OF* sunt necesari, deoarece vizualizarea pe care este definit declanșatorul poate, de exemplu, să se refere la *join*-ul unor tabele, și în acest caz, nu sunt actualizabile toate legăturile.

O vizualizare nu poate fi modificată prin comenzi *LMD* dacă vizualizarea conține operatori pe mulțimi, funcții grup, clauzele *GROUP BY*, *CONNECT BY*, *START WITH*, operatorul *DISTINCT* sau *join*-uri. Declanșatorul *INSTEAD OF* este utilizat pentru a executa operații *LMD* direct pe tabelele de bază ale vizualizării. De fapt, se scriu comenzi *LMD* relative la o vizualizare, iar declanșatorul, în locul operației originale, va opera pe tabelele de bază.

De asemenea, acest tip de declanșator poate fi definit asupra vizualizărilor ce au drept câmpuri tablouri imbricate, declanșatorul furnizând o modalitate de reactualizare a elementelor tabloului imbricat. În acest caz, el se declanșează doar în cazul în care comenzile *LMD* operează asupra tabloului imbricat (numai când elementele tabloului imbricat sunt modificate folosind clauzele *THE()* sau *TABLE()*) și nu atunci când comanda *LMD* operează doar asupra vizualizării. Declanșatorul permite accesarea liniei „părinte“ ce conține tabloul imbricat modificat.

Observații:

- ☐ Spre deosebire de declanșatorii *BEFORE* sau *AFTER*, declanșatorii *INSTEAD OF* se execută în locul instrucțiunii *LMD* (*INSERT*, *UPDATE*, *DELETE*) specificate.
- ☐ Opțiunea *UPDATE OF* nu este permisă pentru acest tip de declanșator.
- ☐ Declanșatorii *INSTEAD OF* se definesc pentru o vizualizare, nu pentru un tabel.
- ☐ Declanșatorii *INSTEAD OF* acționează implicit la nivel de linie.
- ☐ Dacă declanșatorul este definit pentru tablouri imbricate, atributele *:OLD* și *:NEW* se referă la liniile tabloului imbricat, iar pentru a referi linia curentă din tabloul „părinte“ s-a introdus atributul *:PARENT*.

Exemplu:

Se consideră *nou_fotografie*, respectiv *nou_artist*, copii ale tabelelor *fotografie*, respectiv *artist* și *vi_foto_art* o vizualizare definită prin compunerea naturală a celor două tabele. Se presupune că pentru fiecare artist există un câmp (*sum_val*) ce reprezintă valoarea totală a fotografiilor expuse de acesta în expoziții.

Să se definească un declanșator prin care reactualizările executate asupra vizualizării *vi_foto_art* se vor transmite automat tabelelor *nou_fotografie* și *nou_artist*.

```
CREATE TABLE nou_fotografie AS
  SELECT cod_fotografie, cod_artist, valoare, gen
  FROM   fotografie;

CREATE TABLE nou_artist AS
  SELECT cod_artist, nume, sum_val
  FROM   artist;

CREATE VIEW vi_foto_art AS
  SELECT cod_fotografie, o.cod_artist, valoare, gen,
  nume, sum_val
  FROM   nou_fotografie f, nou_artist a
  WHERE  f.cod_artist = a.cod_artist

CREATE OR REPLACE TRIGGER react
  INSTEAD OF INSERT OR DELETE OR UPDATE ON vi_foto_art
  FOR EACH ROW
  BEGIN
    IF INSERTING THEN
      INSERT INTO nou_fotografie
      VALUES (:NEW.cod_fotografie, :NEW.cod_artist,
:NEW.valoare, :NEW.gen);
      UPDATE nou_artist
      SET    sum_val = sum_val + :NEW.valoare
      WHERE  cod_artist = :NEW.cod_artist;
    ELSIF DELETING THEN
      DELETE FROM nou_fotografie
      WHERE  cod_fotografie = :OLD.cod_fotografie;
      UPDATE nou_artist
      SET    sum_val = sum_val - :OLD.valoare
      WHERE  cod_artist = :OLD.cod_artist;
```

13

```
ELSIF UPDATING ('valoare') THEN
    UPDATE nou_fotografie
    SET      valoare = :NEW.valoare
    WHERE    cod_fotografie = :OLD.cod_fotografie;
    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_fotografie
    SET      cod_artist = :NEW.cod_artist
    WHERE    cod_fotografie = :OLD.cod_fotografie;
    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;
/
```

Declanșatori sistem

Declanșatorii sistem sunt activați de comenzi *LDD* (*CREATE*, *DROP*, *ALTER*) și de anumite evenimente sistem (*STARTUP*, *SHUTDOWN*, *LOGON*, *LOGOFF*, *SERVERERROR*, *SUSPEND*). Un declanșator sistem poate fi definit la nivelul bazei de date sau la nivelul schemei.

Sintaxa pentru crearea unui astfel de declanșator este următoarea:

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

Cuvintele cheie *DATABASE* sau *SCHEMA* specifică nivelul declanșatorului.

Există restricții asupra expresiilor din condiția clauzei *WHEN*. De exemplu, declanșatorii *LOGON* și *LOGOFF* pot verifica doar identificatorul (*userid*) și numele utilizatorului (*username*), iar declanșatorii *LDD* pot verifica tipul și

numele obiectelor definite, identificatorul și numele utilizatorului.

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

Declanșatorii *LDD* se activează numai dacă obiectul creat este de tip *table*, *cluster*, *function*, *index*, *package*, *role*, *sequence*, *synonym*, *tablespace*, *trigger*, *type*, *view* sau *user*.

Pentru declanșatorii sistem se pot utiliza funcții speciale care permit obținerea de informații referitoare la evenimentul declanșator. Ele sunt funcții *PL/SQL* stocate care trebuie prefixate de numele proprietarului (*SYS*).

Printre cele mai importante funcții care furnizează informații referitoare la evenimentul declanșator, se remarcă:

- *SYSEVENT* – returnează evenimentul sistem care a activat declanșatorul (este de tip *VARCHAR2(20)* și este aplicabilă oricărui eveniment);
- *DATABASE_NAME* – returnează numele bazei de date curente (este de tip *VARCHAR2(50)* și este aplicabilă oricărui eveniment);
- *SERVER_ERROR* – returnează codul erorii a cărei poziție în stiva erorilor este dată de argumentul de tip *NUMBER* al funcției (este de tip *NUMBER* și este aplicabilă evenimentului *SERVERERROR*);
- *LOGIN_USER* – returnează identificatorul utilizatorului care activează declanșatorul (este de tip *VARCHAR2(30)* și este aplicabilă oricărui eveniment);
- *DICTIONARY_OBJ_NAME* – returnează numele obiectului la care face referință comanda *LDD* ce a activat declanșatorul (este de tip *VARCHAR2(30)* și este aplicabilă evenimentelor *CREATE*, *ALTER*, *DROP*).

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 urmări erorile care apar în baza de date. Codul erorii este furnizat, prin intermediul declanșatorului, de funcția *SERVER_ERROR*, iar mesajul asociat erorii poate fi obținut cu procedura *DBMS_UTILITY.FORMAT_ERROR_STACK*.

Exemplu:

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

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

Modificarea și suprimarea declanșatorilor

Opțiunea *OR REPLACE* din cadrul comenzii *CREATE TRIGGER* recrează declanșatorul, dacă acesta există. Clauza permite schimbarea definiției unui declanșator existent fără suprimarea acestuia.

Similar procedurilor și pachetelor, un declanșator poate fi suprimat prin:

DROP TRIGGER [*schema.*]*nume_declanșator*;

Uneori acțiunea de suprimare a unui declanșator este prea drastică și este preferabilă doar dezactivarea sa temporară. În acest caz, declanșatorul va continua să existe în dicționarul datelor. Modificarea unui declanșator poate consta din recompilarea (*COMPILE*), redenumirea (*RENAME*), activarea (*ENABLE*) sau dezactivarea (*DISABLE*) acestuia și se realizează prin comanda:

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

Dacă un declanșator este activat, atunci sistemul *Oracle* îl execută ori de

câte ori au loc operațiile precizate în declanșator asupra tabelului asociat și când condiția de restricție este îndeplinită. Dacă declanșatorul este dezactivat, atunci sistemul *Oracle* nu îl va mai executa. După cum s-a mai subliniat, dezactivarea unui declanșator nu implică ștergerea acestuia din dicționarul datelor.

Toți declanșatorii asociați unui tabel pot fi activați sau dezactivați utilizând opțiunea *ALL TRIGGERS* (*ENABLE ALL TRIGGERS*, respectiv *DISABLE ALL TRIGGERS*). Declanșatorii sunt activați în mod implicit atunci când sunt creați.

Activarea și dezactivarea declanșatorilor asociați unui tabel se poate realiza și cu ajutorul comenzii *ALTER TABLE*.

Un declanșator este compilat în mod automat la creare. Dacă un *site* este neutilizabil atunci când declanșatorul trebuie compilat, sistemul *Oracle* nu poate valida comanda de accesare a bazei distante și compilarea eșuează.

Informații despre declanșatori

În dicționarul datelor există vizualizări ce conțin informații despre declanșatori și despre starea acestora (*USER_TRIGGERS*, *USER_TRIGGER_COL*, *ALL_TRIGGERS*, *DBA_TRIGGERS* etc.). Aceste vizualizări sunt actualizate ori de câte ori un declanșator este creat sau șters.

Atunci când declanșatorul este creat, codul său sursă este stocat în vizualizarea *USER_TRIGGERS*. Vizualizarea *ALL_TRIGGERS* conține informații despre toți declanșatorii din baza de date. Pentru a detecta dependențele declanșatorilor poate fi consultată vizualizarea *USER_DEPENDENCIES*, iar *ALL_DEPENDENCIES* conține informații despre dependențele tuturor obiectelor din baza de date. Erorile rezultate din compilarea declanșatorilor pot fi analizate din vizualizarea *USER_ERRORS*, iar prin comanda *SHOW ERRORS* se vor afișa erorile corespunzătoare ultimului declanșator compilat.

În operațiile de gestiune a bazei de date este necesară uneori reconstruirea instrucțiunilor *CREATE TRIGGER*, atunci când codul sursă original nu mai este disponibil. Aceasta se poate realiza utilizând vizualizarea *USER_TRIGGERS*. Vizualizarea include numele declanșatorului (*TRIGGER_NAME*), tipul acestuia (*TRIGGER_TYPE*), evenimentul declanșator (*TRIGGERING_EVENT*), numele proprietarului tabelului (*TABLE_OWNER*), numele tabelului pe care este definit declanșatorul (*TABLE_NAME*), clauza *WHEN* (*WHEN_CLAUSE*), corpul declanșatorului (*TRIGGER_BODY*), antetul (*DESCRIPTION*), starea acestuia (*STATUS*) care poate să fie *ENABLED* sau *DISABLED* și numele utilizate pentru a referi parametrii *OLD* și *NEW* (*REFERENCING_NAMES*). Dacă obiectul de bază nu este un tabel sau o vizualizare, atunci *TABLE_NAME* este *null*.

Exemplu:

Presupunând că nu este disponibil codul sursă pentru declanșatorul *alfa*, să se reconstruiască instrucțiunea *CREATE TRIGGER* corespunzătoare acestuia.

```
CREATE OR REPLACE PROCEDURE create_trigger
(v_nume VARCHAR2)
IS
    sir1 VARCHAR2(4000);
    sir2 LONG;
BEGIN
    SELECT DESCRIPTION, TRIGGER_BODY
    INTO sir1,sir2
    FROM USER_TRIGGERS
    WHERE TRIGGER_NAME=UPPER(v_nume);
    DBMS_OUTPUT.PUT('CREATE OR REPLACE TRIGGER
'||sir1);
    DBMS_OUTPUT.PUT_LINE(sir2);
END;
/
```

```
EXECUTE create_trigger('ALFA')
```

Utilizând cursorul se pot reconstrui numai declanșatorii care aparțin contului utilizator curent. O interogare a vizualizărilor *ALL_TRIGGERS* sau *DBA_TRIGGERS* permite reconstruirea tuturor declanșatorilor din sistem, dacă se dispune de privilegii *DBA*.

Privilegii sistem

Sistemul furnizează următoarele privilegii sistem pentru gestiunea declanșatorilor:

- ☐ *CREATE TRIGGER* (permite crearea declanșatorilor în schema personală);
- ☐ *CREATE ANY TRIGGER* (permite crearea declanșatorilor în orice schemă cu excepția celei corespunzătoare lui *SYS*);
- ☐ *ALTER ANY TRIGGER* (permite activarea, dezactivarea sau compilarea declanșatorilor în orice schemă cu excepția lui *SYS*);
- ☐ *DROP ANY TRIGGER* (permite suprimarea declanșatorilor la nivel de bază de date în orice schemă cu excepția celei corespunzătoare lui *SYS*);
- ☐ *ADMINISTER DATABASE TRIGGER* (permite crearea sau modificarea unui declanșator sistem referitor la baza de date);
- ☐ *EXECUTE* (permite referirea, în corpul declanșatorului, a procedurilor, funcțiilor sau pachetelor din alte scheme).

Tabele mutating

Asupra tabelelor și coloanelor care pot fi accesate de corpul declanșatorului există anumite restricții. Pentru a analiza aceste restricții este necesară definirea tabelelor în schimbare (*mutating*) și constrânse (*constraining*).

Un tabel *constraining* este un tabel pe care evenimentul declanșator trebuie să-l consulte fie direct, printr-o instrucțiune *SQL*, fie indirect, printr-o constrângere de integritate referențială declarată. Tabelele nu sunt considerate *constraining* în cazul declanșatorilor la nivel de instrucțiune.

Un tabel *mutating* este tabelul modificat de instrucțiunea *UPDATE*, *DELETE* sau *INSERT*, sau un tabel care va fi actualizat prin efectele acțiunii integrității referențiale *ON DELETE CASCADE*. Chiar tabelul pe care este definit declanșatorul este un tabel *mutating*, ca și orice tabel referit printr-o constrângere *FOREIGN KEY*. Tabelele nu sunt considerate *mutating* pentru declanșatorii la nivel de instrucțiune, iar vizualizările nu sunt considerate *mutating* în declanșatorii *INSTEAD OF*.

Principalele reguli care trebuie respectate la utilizarea declanșatoriilor sunt:

- comenzile *SQL* din corpul unui declanșator nu pot consulta sau modifica valorile coloanelor care sunt declarate chei primare, externe sau unice (*PRIMARY KEY*, *FOREIGN KEY*, *UNIQUE KEY*) într-un tabel *constraining*;
- comenzile *SQL* din corpul unui declanșator nu pot consulta sau modifica date dintr-un tabel *mutating*.

Dacă o comandă *INSERT* afectează numai o înregistrare, declanșatorii la nivel de linie (*BEFORE* sau *AFTER*) pentru înregistrarea respectivă nu tratează tabelul ca fiind *mutating*. Acesta este unicul caz în care un declanșator la nivel de linie poate citi sau modifica tabelul. Comanda *INSERT INTO tabel SELECT ...* consideră tabelul *mutating* chiar dacă cererea returnează o singură linie.

Exemplu:

```
/*Trigger-ul realizeaza actualizari in cascada:
actualizarea cheii primare din tabelul parinte
determina actualizarea cheii externe din tabelul
copil*/
```

19

```
DROP TABLE fotografie_prof;
DROP TABLE artist_prof;

CREATE TABLE artist_prof AS
SELECT * FROM artist;

CREATE TABLE fotografie_prof AS
SELECT * FROM fotografie;

ALTER TABLE artist_prof ADD CONSTRAINT art_pk
PRIMARY KEY (cod_artist);

ALTER TABLE fotografie_prof ADD CONSTRAINT foto_pk
PRIMARY KEY (cod_fotografie);

ALTER TABLE fotografie_prof ADD CONSTRAINT
art_foto_fk
FOREIGN KEY(cod_artist) REFERENCES
artist_prof(cod_artist);

/*ALTER TABLE fotografie_prof DROP CONSTRAINT
art_foto_fk;

ALTER TABLE fotografie_prof ADD CONSTRAINT
art_foto_fk FOREIGN KEY(cod_artist) REFERENCES
artist_prof(cod_artist) ON DELETE CASCADE;

ALTER TABLE fotografie_prof ADD CONSTRAINT
art_foto_fk FOREIGN KEY(cod_artist) REFERENCES
artist_prof(cod_artist) ON DELETE SET NULL;*/

CREATE OR REPLACE TRIGGER modifica_copil
AFTER UPDATE OF cod_artist ON artist_prof
FOR EACH ROW
BEGIN
UPDATE fotografie_prof
SET cod_artist=:NEW.cod_artist
WHERE cod_artist=:OLD.cod_artist;
END;
/

/*constrangerea de cheie externa este definita (cu
optiuni la stergere sau nu);
exista fotografii create de artistul 1;
actualizarea urmatoare este permisa.*/

UPDATE artist_prof
SET cod_artist=71
WHERE cod_artist=1;
```

20

```
SELECT *
FROM artist_prof;

SELECT *
FROM fotografie_prof;

ROLLBACK;

ALTER TRIGGER modifica_copil DISABLE;

/* Trigger-ul realizeaza actualizari in cascada:
actualizarea cheii externe din tabelul copil
determina actualizarea primare din tabelul parinte*/

CREATE OR REPLACE TRIGGER modifica_parinte
  BEFORE UPDATE OF cod_artist ON fotografie_prof
  FOR EACH ROW
BEGIN
  UPDATE artist_prof
  SET cod_artist=:NEW.cod_artist
  WHERE cod_artist=:OLD.cod_artist;
END;

/

--actualizarea urmatoare este permisa

UPDATE fotografie_prof
SET cod_artist=71
WHERE cod_artist=1;

SELECT *
FROM artist_prof;

SELECT *
FROM fotografie_prof;

ROLLBACK;

ALTER TRIGGER modifica_copil ENABLE;

/*daca ambii trigger-i definiti anterior ar fi activi
simultan, atunci urmatoarele comenzi nu ar fi
permise;

eroarea aparuta: "table is mutating, trigger/function
may not see it".*/

UPDATE artist_prof
SET cod_artist=71
WHERE cod_artist=1;
```

21

```
UPDATE fotografie_prof
SET cod_artist=71
WHERE cod_artist=1;
ROLLBACK;

ALTER TRIGGER modifica_parinte DISABLE;
ALTER TRIGGER modifica_copil DISABLE;

CREATE OR REPLACE TRIGGER trig_mutating_table
  BEFORE DELETE ON artist_prof
  FOR EACH ROW
DECLARE
  v_nume VARCHAR2(50);
BEGIN
  SELECT nume INTO v_nume
  FROM artist_prof
  WHERE cod_artist=:OLD.cod_artist;
END;
/

/*trigger-ul consulta tabelul de care este asociat;
comanda urmatoare nu este permisa;
eroarea aparuta "table is mutating, trigger/function
may not see it".*/

DELETE FROM artist_prof WHERE cod_artist=1;

/*comanda urmatoare nu este permisa
(artistul 2001 nu exista in tabel)*/
DELETE FROM artist_prof WHERE cod_artist=2001;
ALTER TRIGGER trig_mutating_table DISABLE;

/*Exemplu

Trigger-ul realizeaza stergeri in cascada:
stergerea unei inregistrari din tabelul parinte
determina stergerea inregistrarilor copil asociate*/

CREATE OR REPLACE TRIGGER sterge_copil
  BEFORE DELETE ON artist_prof
  FOR EACH ROW
BEGIN
  DELETE FROM fotografie_prof
  WHERE cod_artist=:OLD.cod_artist;
END;
/
```

22

```
/*cazul 1 - constrangerea de cheie externa nu are
optiuni de stergere specificate;
urmatoarea comanda este permisa.*/
DELETE FROM artist_prof WHERE cod_artist=1;
/*cazul 2 - constrangerea de cheie externa are
optiuni de stergere (CASCADE/SET NULL);
urmatoarea comanda nu este permisa;
eroarea aparuta "table is mutating, trigger/function
may not see it".*/
DELETE FROM artist_prof WHERE cod_artist=1;
ROLLBACK;
```

Exemplu:

Să se implementeze cu ajutorul unui declanșator restricția că într-o sală pot să fie expuse maximum 3 fotografii.

```
CREATE OR REPLACE TRIGGER trig_mutating
  BEFORE INSERT OR UPDATE OF cod_sala ON fotografie
  FOR EACH ROW
DECLARE
  v_max CONSTANT NUMBER := 3;
  v_nr NUMBER;
BEGIN
  SELECT COUNT(*) INTO v_nr
  FROM fotografie
  WHERE cod_sala = :NEW.cod_sala;
  IF v_nr + 1 > v_max THEN
    RAISE_APPLICATION_ERROR(-20000, 'Prea multe
fotografii in sala avand codul ' || :NEW.cod_sala);
  END IF;
END trig_mutating;
```

Cu toate că declanșatorul pare să producă lucrul dorit, totuși după o reactualizare a tabelului *fotografie* în următoarea manieră:

```
INSERT INTO fotografie
SELECT 756, 'dsds', 'ceva', 10, TO_DATE('1-10-
2022', 'DD-MM-YYYY'), 1, 1, SYSDATE
FROM dual;
```

se obține următorul mesaj de eroare:

```
ORA-04091: tabel fotografie is mutating,
trigger/function may not see it
```

23

ORA-04088: error during execution of trigger TrLimitaFotografii

Eroarea *ORA-04091* apare deoarece declanșatorul *trig_mutating* consultă chiar tabelul (*fotografie*) la care este asociat declanșatorul (*mutating*).

```
INSERT INTO fotografie
SELECT 757, 'dsds2', 'ceva2', 12, TO_DATE('1-10-2022', 'DD-MM-YYYY'), 1, 1, SYSDATE
FROM dual;
```

```
INSERT INTO fotografie
SELECT 758, 'dsds3', 'ceva3', 13, TO_DATE('1-10-2022', 'DD-MM-YYYY'), 1, 1, SYSDATE
FROM dual;
```

```
UPDATE fotografie
SET cod_sala=1
WHERE cod_sala=3;
```

```
UPDATE fotografie
SET cod_sala=2
WHERE cod_sala=1;
```

```
BEGIN
UPDATE fotografie
SET cod_sala = CASE
                    WHEN cod_fotografie < 3 THEN 2
                    ELSE 1
                END;
```

```
END;
/
```

```
ROLLBACK;
```

```
DELETE FROM fotografie WHERE cod_fotografie IN
(756, 757);
```

```
COMMIT;
```

Tabelul *fotografie* este *mutating* doar pentru un declanșator la nivel de linie. Aceasta înseamnă că tabelul poate fi consultat în interiorul unui declanșator la nivel de instrucțiune. Totuși, limitarea numărului fotografiilor nu poate fi făcută în interiorul unui declanșator la nivel de instrucțiune, din moment ce este necesară valoarea *:NEW.cod_sala* în corpul declanșatorului.

Există mai multe soluții pentru această problemă.

24

Prima: consultarea unei copii a tabelului fotografie în declanșatorul la nivel de linie, neindicată,

a 2-a: adăugarea clauzei pragma autonomous_transaction, neindicată,

a 3-a: crearea unui pachet și a doi declanșatori, unul la nivel de linie și altul la nivel de instrucțiune,

a 4-a: crearea unui declanșator compus (compound trigger).

Prin definirea declanșatorului la nivel de linie ca o tranzacție autonomă, am scăpat de eroarea tabelului mutating, dar rezultatul nu este corect. Cele mai recente actualizări nu se reflectă în setul nostru de rezultate, spre deosebire de declanșatorul la nivel de comandă. Așa că trebuie să fim foarte atenți când folosim această abordare.

Soluția 2:

```
CREATE OR REPLACE TRIGGER trig_tranzactie_autonoma
  BEFORE INSERT OR UPDATE OF cod_sala ON fotografie
  FOR EACH ROW
DECLARE
pragma autonomous_transaction;
  v_max CONSTANT NUMBER := 3;
  v_nr NUMBER;
BEGIN
  SELECT COUNT(*) INTO v_nr
  FROM   fotografie
  WHERE  cod_sala = :NEW.cod_sala;
  IF v_nr + 1 > v_max THEN
    RAISE_APPLICATION_ERROR(-20000, 'Prea multe
fotografii in sala având codul ' || :NEW.cod_sala);
  END IF;
END trig_tranzactie_autonoma;
```

Soluția 3:

În declanșatorul la nivel de linie se înregistrează valoarea lui :NEW.cod_sala, dar nu va fi interogată tabelul *fotografie*.

Interogarea va fi făcută în declanșatorul la nivel de instrucțiune și va folosi valoarea înregistrată în declanșatorul la nivel de linie.

O modalitate pentru a înregistra valoarea lui :NEW.cod_sala este utilizarea unui tablou indexat în interiorul unui pachet.

```
CREATE OR REPLACE PACKAGE pachet AS
  TYPE tip_rec IS RECORD (
    id fotografie.cod_sala%TYPE,
```


25

```
        nr NUMBER(2));
TYPE t_ind IS TABLE OF tip_rec
        INDEX BY BINARY_INTEGER;
t t_ind;
contor NUMBER(2) := 0;
END pachet;

CREATE OR REPLACE TRIGGER trig_comanda
BEFORE INSERT OR UPDATE OF cod_sala ON fotografie
BEGIN
    pachet.contor:=0;
    SELECT cod_sala, COUNT(*)
    BULK COLLECT INTO pachet.t
    FROM fotografie
    GROUP BY cod_sala;
END trig_comanda;

CREATE OR REPLACE TRIGGER trig_linie
BEFORE INSERT OR UPDATE OF cod_sala ON fotografie
FOR EACH ROW
BEGIN
    FOR i IN 1..pachet.t.LAST LOOP
        IF pachet.t(i).id = :NEW.cod_sala AND
            (pachet.t(i).nr + pachet.contor = 3 OR
pachet.t(i).nr > 3) THEN
            RAISE_APPLICATION_ERROR(-20000, 'Sala
'||:NEW.cod_sala||' depaseste numarul maxim de
fotografii permis');
        END IF;
    END LOOP;
    pachet.contor:=pachet.contor+1;
END trig_linie;
```

Soluția 4:

```
CREATE OR REPLACE TRIGGER trig_compus
FOR INSERT OR UPDATE OF cod_sala
ON fotografie
COMPOUND TRIGGER
    TYPE tip_rec IS RECORD (
        id fotografie.cod_sala%TYPE,
        nr NUMBER(2));
    TYPE t_ind IS TABLE OF tip_rec
```

```

                                INDEX BY BINARY_INTEGER;
        t t_ind;
        contor NUMBER(2) := 0;
    BEFORE STATEMENT IS
    BEGIN
        contor:=0;
        SELECT cod_sala, COUNT(*)
        BULK COLLECT INTO t
        FROM fotografie
        GROUP BY cod_sala;
    END BEFORE STATEMENT;
    BEFORE EACH ROW IS
    BEGIN
        FOR i IN 1..t.LAST LOOP
            IF t(i).id = :NEW.cod_sala AND
                (t(i).nr + contor = 3 OR t(i).nr > 3)
THEN
                                RAISE_APPLICATION_ERROR(-20000, 'Sala
'||:NEW.cod_sala||' depaseste numarul maxim de
fotografii permis');
                                END IF;
                            END LOOP;
                            contor:=contor+1;
                        END BEFORE EACH ROW;
    END trig_compus;
/

```

Exemplu:

Să se creeze un declanșator care:

a) dacă este eliminată o sală, va șterge toate fotografiile expuse în sala respectivă;

b) dacă se schimbă codul unei săli, va modifica această valoare pentru fiecare fotografie expusă în sala respectivă.

```

CREATE OR REPLACE TRIGGER sala_cascada
    BEFORE DELETE OR UPDATE OF cod_sala ON sala
    FOR EACH ROW
    BEGIN
        IF DELETING THEN
            DELETE FROM fotografie
            WHERE cod_sala = :OLD.cod_sala;

```

27

```
END IF;
IF UPDATING AND :OLD.cod_sala != :NEW.cod_sala THEN
    UPDATE fotografie
    SET     cod_sala = :NEW.cod_sala
    WHERE  cod_sala = :OLD.cod_sala;
END IF;
END sala_cascada;
```

Declanșatorul anterior realizează constrângerea de integritate *UPDATE* sau *ON DELETE CASCADE*, adică ștergerea sau modificarea cheii primare a unui tabel „părinte” se va reflecta și asupra înregistrărilor corespunzătoare din tabelul „copil”.

Executarea acestuia, pe tabelul *sala* (tabelul „părinte”), va duce la efectuarea a două tipuri de operații pe tabelul *fotografie* (tabelul „copil”).

La eliminarea unei săli din tabelul *sala*, se vor șterge toate fotografiile corespunzătoare acestei săli.

```
DELETE FROM sala
WHERE  cod_sala = 3;
```

La modificarea codului unei săli din tabelul *sala*, se va actualiza codul sălii atât în tabelul *sala*, cât și în tabelul *fotografie*.

```
UPDATE sala
SET     cod_sala = 7
WHERE  cod_sala = 3;
```

Se presupune că asupra tabelului *fotografie* există o constrângere de integritate:

```
FOREIGN KEY (cod_sala) REFERENCES sala(cod_sala) ON
DELETE CASCADE/SET NULL
```

În acest caz sistemul *Oracle* va afișa un mesaj de eroare prin care se precizează că tabelul *sala* este *mutating*, iar constrângerea definită mai sus nu poate fi verificată.

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

Pachetele pot fi folosite pentru încapsularea detaliilor logice legate de declanșatori. Exemplul următor arată un mod simplu de implementare a acestei posibilități. Este permisă apelarea unei proceduri sau funcții stocate din blocul *PL/SQL* care reprezintă corpul declanșatorului.

28

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 fotografie
    FOR EACH ROW
BEGIN
    pachet.procesare_trigger(:NEW.valoare,:NEW.stare)
END;
```