

Capitolul 17. Declanșatoare

Declanșatoarele (trigger-e, în original) reprezintă o categorie deosebită de proceduri stocate ale unei baze de date. Particularitatea lor esențială ține de faptul că, odată create și stocate în schema bazei, acestea sunt executate *automat* la operațiuni pentru care au fost definite. Tradițional, sunt trei tipuri de astfel de operațiuni: inserare, modificare și ștergere ale liniilor din tabela pentru care au fost definite. La această tipologie a declanșatoarelor, unele SGBD-uri au mai adăugat și alte categorii, asociate, spre exemplu, actualizărilor tabelelor virtuale, conectării unei aplicații/utilizator la baza de date, deschiderii sau închiderii bazei de date etc.

Pentru lucrarea de față problema principală a declanșatoarelor ține de faptul că acestea sunt scrise în extensiile procedurale ale SQL care, ca și funcțiile și procedurile din capitolul anterior, prezintă diferențe sensibile de la produs la produs: PL/SQL (Oracle), SQL PL (DB2), Transact-SQL (SQL Server), PL/pgSQL etc.

Cele mai importante avantaje ale declanșatoarelor sunt:

- permit instituirea unor reguli de validare cu mult mai complexe decât posibilitățile clauzei CHECK;
- pot ameliora sensibil mecanismul de securitate al bazei;
- permit instituirea (acolo unde nu este posibil prin clauza FOREIGN KEY) restricțiilor referențiale și modului de tratare a modificărilor ce pot cauza probleme de integritate referențială;
- constituie suportul calculării automate a valorilor unor atribute, ceea ce poate atrage un spor de viteză considerabil;
- posibilitatea jurnalizării explicite, în formatul dorit a tuturor operațiunilor de tip DML și DML dintr-o (sub) schemă a bazei de date;
- sincronizarea replicilor unei baze de date distribuite/replicate.

17.1. Tipologia declanșatoarelor

Așadar, aria de folosire a declanșatoarelor este foarte extinsă. Totuși, documentațiile de produs recomandă ca elanul trigger-esc să nu depășească anumite limite. Prin declanșatoare trebuie gestionate numai acele reguli imposibil de asigurat prin restricțiile la nivel de atribut (câmp), înregistrare, restricții de tip PRIMARY KEY, NOT NULL etc. Tipologia „clasică” numără nu mai puțin de 15 categorii de triggere – vezi tabelul 17.1.

Tabel 17.1. Tipologia “clasică” a declanșatoarelor

Eveniment declanșator	Declanșare pt. fiecare	Descriere
BEFORE INSERT	Comandă	Codul (programul) acestuia se lansează înaintea executării unei comenzi INSERT în tabela țintă

BEFORE INSERT	Linie	Se execută înaintea inserării fiecărei linii în tabelă
AFTER INSERT	Linie	Se execută după inserarea fiecărei linii în tabelă
AFTER INSERT	Comandă	Se lansează după execuția unei comenzi de inserare de linii în tabelă
INSTEAD OF INSERT	Linie	În loc să se insereze o linie în tabelă, se execută codul din acest declanșator
BEFORE UPDATE	Comandă	Codul acestuia se execută înaintea executării unei comenzi UPDATE pentru tabela țintă
BEFORE UPDATE	Linie	Se execută înaintea modificării fiecărei linii din tabelă
AFTER UPDATE	Linie	Se execută după modificarea fiecărei linii
AFTER UPDATE	Comandă	Se lansează după execuția comenzii UPDATE (după modificarea tuturor liniilor afectate de comandă)
INSTEAD OF UPDATE	Linie	În loc să se modifice o linie din tabela țintă, se execută codul din acest declanșator
BEFORE DELETE	Comandă	Se execută înaintea comenzii DELETE
BEFORE DELETE	Linie	Se execută înainte de ștergerea fiecărei linii
AFTER DELETE	Linie	Se execută după ștergerea fiecărei linii
AFTER DELETE	Comandă	Se lansează după execuția comenzii DELETE (după ștergerea tuturor liniilor afectate de comandă)
INSTEAD OF DELETE	Linie	Se execută în locul ștergerii unei linii din tabela țintă

Declanșatoare la nivel de linie și la nivel de comandă (statement)

La definirea triggerului, se poate stabili de câte ori este executat acesta: de fiecare dată când o linie este inserată/modificată/ștearsă, sau o dată pentru o comandă de actualizare, indiferent de câte linii sunt afectate. Să luăm, spre exemplu, comanda:

UPDATE clienti SET codcl = codcl + 100 WHERE codcl > 3

Presupunând că 135 de clienți au codul mai mare decât 3, un trigger de modificare la nivel de linie se execută de 135 de ori, în timp ce unul la nivel de comandă o singură dată. Dacă nici un client nu are codul mai mare de 3, un trigger de modificare la nivel de linie nu s-ar putea executa deloc, în timp ce declanșatorul declarat a fi la nivel de comandă ar fi lansat (evident, o singură dată).

Declanșatoare de tip înainte (BEFORE) și după (AFTER) actualizare

Un trigger de tip BEFORE este ideal pentru verificarea unui set de condiții care ar putea bloca de la bun început tranzacția respectivă (autentificări, soldul 0 pentru un cont sau pentru un material într-o magazie etc.). Dacă pentru tabelă au fost definite și restricții (NOT NULL, PRIMARY KEY, CHECK), dar și declanșatoare de tip BEFORE, declanșatoarele BEFORE se lansează *înaintea* verificării restricțiilor. Declanșatoarele de tip AFTER se execută după verificarea restricțiilor, fiind nimerite la jurnalizare (ce ține evidența strictă a modificărilor operate în tabele cu informații sensibile - spre exemplu, calculul automat al costului mediu ponderat al unui material după fiecare intrare în magazie sau calculul soldului curent al unui cont bancar după fiecare depunere sau retragere etc.). Spre deosebire de cele de tip

BEFORE , triggerele de tip AFTER blochează liniile procesate. Dacă declanșatoarele prelucreează/interoghează tabelele pe care au fost denumite¹, declanșatoarele de tip BEFORE nu „văd” modificările care atras declanșarea lor, în timp ce triggerele AFTER „văd” imaginea bazei de date după modificare.

Orice declanșator la nivel de linie sau comandă poate fi de tip BEFORE sau AFTER. Astfel, pentru orice tabelă și operație de inserare/modificare/ștergere pot definite patru tipuri de trigger care se execută în ordinea:

- înainte - la nivel de comandă (BEFORE - statement)
- înainte - la nivel de linie (BEFORE - row)
- după - la nivel de linie (AFTER - row)
- după - la nivel de comandă (AFTER - statement)

Trigger de tip în-loc-de (INSTEAD OF)

Acest gen de declanșator constituie un excelent mijloc de propagare a modificărilor dintr-o tabelă virtuală în tabelele de bază din care provine. Amânăm această discuție pentru finalul fiecărui paragraf, începând cu următorul.

Un alt element important este că, și în trigger-ele la nivel de linie (ROW) de tip BEFORE și în cele de tip AFTER, pot fi invocate și prelucrate atât valorile atributelor dinaintea operației, cât și cele de după operație. Prefixul este, după caz, :OLD/OLD sau :NEW/NEW.

17.2. Declanșatoare în PL/pgSQL

Limbajul PL/pgSQL este generos în materie de declanșatoare. Să începem cu unul simplu dedicat cheilor surogat. Atributul CodCl din tabela CLIENTI este o asemenea cheie surogat, întrucât valorile sale nu au nicio legătură cu valorile celorlalte atribute ale fiecărui client, fiind mai mult un soi de număr curent. Primii clienți au fost inserați încă din capitolul 3. Din acest moment, dorim ca, la inserare, codul fiecărui client să nu mai depindă de utilizator (valoarea specificată în comanda INSERT), ci să fie atribuită automat de sistem. Pentru aceasta, vom crea un obiect al bazei de date de care am pomenit în capitolul 2, și anume o secvență:

```
CREATE SEQUENCE seq_codcl START WITH 1011 INCREMENT BY 1  
MINVALUE 1001 MAXVALUE 8900 NO CYCLE ;
```

O secvență este un obiect dintr-o bază de date care generează o valoare unică atunci când i se cere. Chiar și atunci când câteva mii de utilizatori ar introduce simultan clienți, vrem să fim siguri că valoarea fiecăruia este unică. Or secvența

¹ Nu întotdeauna este permisă interogarea unei tabele pentru care s-au definit declanșatoare la nivel de linie (ROW), mai ales în caz declanșatoarelor de tip UPDATE. Cei dintre dvs. Au lucrat în Oracle, au avut, probabil, de a face cu problema *mutanței* tabelei.

este construită astfel ca eventualele cereri de valori unice să fie serializate. Definim acum un declanșator de tip BEFORE-INSERT-ROW – vezi listing 17.1. PL/pgSQL reclamă crearea unei funcții care să returneze TRIGGER, și apoi o comandă CREATE TRIGGER care să folosească funcția respectivă drept argument.

Listing 17.1. Funcția-declanșator PL/pgSQL executată la inserarea unei înregistrări în tabela CLIENTI

```
CREATE OR REPLACE FUNCTION trg_clienti_ins_before_row()
  RETURNS TRIGGER AS $trg_clienti_ins_before_row$
BEGIN
  NEW.CodCl = NEXTVAL('seq_codcl') ;
  RETURN NEW ;
END ;
$trg_clienti_ins_before_row$ LANGUAGE plpgsql ;
```

Extragerea următoarei valori unice disponibile din secvență se realizează prin funcția *NEXTVAL*. *NEW.CodCl* semnalizează că valoarea generată de secvență va fi atribuită atributului *CodCl* de pe linia nou-introdusă. Funcția se termină obligatoriu prin *RETURN NEW*, ceea ce înseamnă că se returnează noua linie inserată în care au fost făcute eventuale modificări. Asociem funcția declanșatorului printr-o comandă CREATE TRIGGER:

```
CREATE TRIGGER trg_clienti_ins_before_row
  BEFORE INSERT ON clienti
  FOR EACH ROW EXECUTE PROCEDURE trg_clienti_ins_before_row()
```

Reținem că modificarea valorii unuia sau multor atribute de pe linia proaspăt inserată este posibilă doar în declanșatoare BEFORE-ROW. Acum să testăm funcționalitatea declanșatorului. Introducem un client:

```
INSERT INTO clienti VALUES (999, 'Client destul de nou', 'R12345',
  'Adresa clientului nou', 700505, NULL)
```

Valoarea *CodCl* specificată în clauza VALUES este 999. Dacă însă vizualizăm noua înregistrare (ultima din figura 17.1) observăm că PostgreSQL a ținut cont de declanșator, și nu de ceea ce am specificat noi în comanda INSERT.

Continuăm cu declanșatorul de inserare în tabela FACTURI. Pentru a fi siguri că orice factură nou introdusă pornește de la zero cu valorile totală, încasată și tva, declarăm un declanșator după cum urmează (listing 17.2).

Listing 17.2. Declanșatorul PL/pgSQL pentru inserări în tabela FACTURI

```
CREATE OR REPLACE FUNCTION trg_facturi_ins()
  RETURNS TRIGGER AS $trg_facturi_ins$
BEGIN
  NEW.valtotala := 0 ;
  NEW.tva := 0 ;
  NEW.valincasata := 0 ;

  RETURN NEW ;
END ;
$trg_facturi_ins$ LANGUAGE plpgsql ;
```

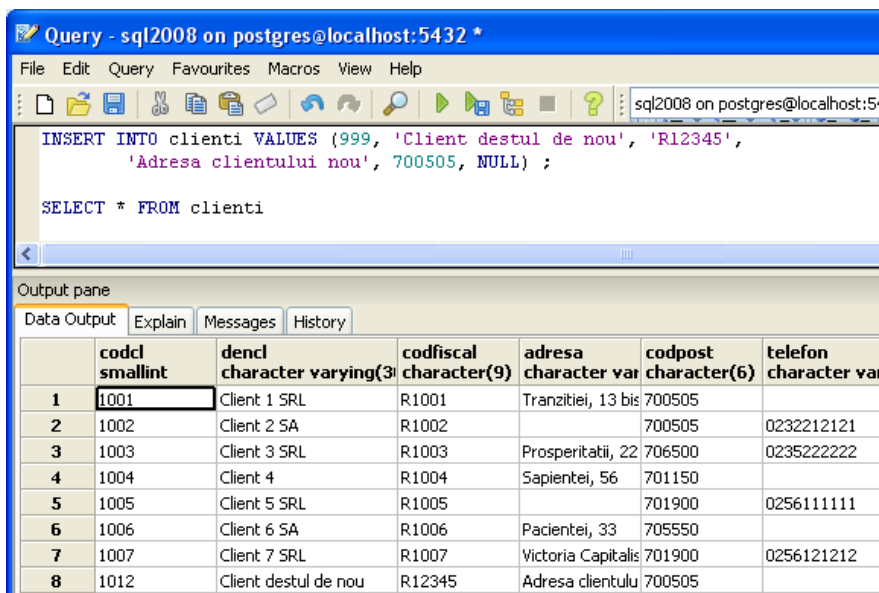


Figura 17.1. Verificarea declanșatorului de inserare în CLIENTI

**CREATE TRIGGER trg_facturi_ins BEFORE INSERT ON facturi
FOR EACH ROW EXECUTE PROCEDURE trg_facturi_ins() ;**

Atributele *ValTotala* și *TVA* ale tabelii FACTURI trebuie recalculate după fiecare inserare, (eventual) modificare și ștergere a unei linii din factura respectivă, altfel spus la INSERT-uri, DELETE-uri și unele UPDATE-uri asupra tabelii LINIFACT. Așa încât, ca model, vom redacta declanșatorul de inserare în tabela LINIIFACT – vezi listing 17.3 -, care modifică valorile atributelor:

- *Linie* și *TVALinie* din LINIIFACT,
- *TVA* și *VALTotala* din FACTURI

Listing 17.3. Declanșatorul PL/pgSQL de inserare în tabela LINIIFACT

```
DROP FUNCTION trg_linifact_ins() CASCADE ;

CREATE OR REPLACE FUNCTION trg_linifact_ins()
  RETURNS TRIGGER AS $trg_linifact_ins$
BEGIN
  NEW.tvalinie := COALESCE(NEW.cantitate,0) * COALESCE(NEW.pretunit,0) *
    f_proctva(NEW.codpr) ;
  NEW.linie := (SELECT COALESCE(MAX(Linie),0) + 1 FROM linifact
    WHERE NrFact=NEW.NrFact) ;

  UPDATE facturi SET
    tva = COALESCE(tva,0) + NEW.tvalinie,
    valtotala = COALESCE(valtotala,0) + NEW.cantitate * NEW.pretunit + NEW.tvalinie
    WHERE nrfact = NEW.nrfact ;
  RETURN NEW ;
END ;
```

```
$trg liniifact_ins$ LANGUAGE plpgsql ;
```

```
CREATE TRIGGER trg liniifact_ins BEFORE INSERT ON liniifact
FOR EACH ROW EXECUTE PROCEDURE trg liniifact_ins() ;
```

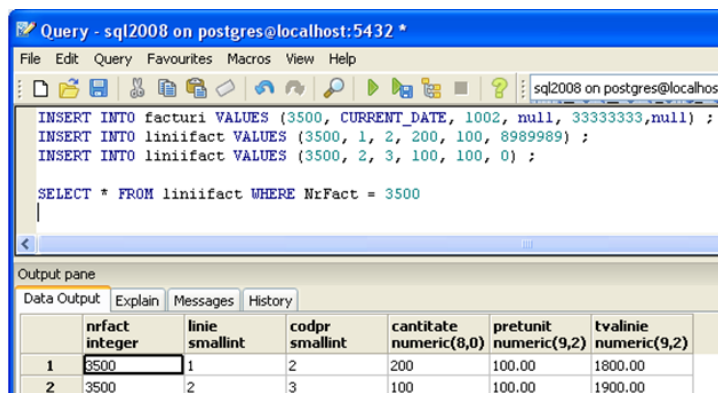
Haideți să testăm cele două declanșatoare, adăugând factura 3500 care conține două linii în care apar produsele ce au codurile 2 și 3:

```
INSERT INTO facturi VALUES (3500, CURRENT_DATE, 1002, null, 33333333,null) ;
null, 33333333,null) ;
```

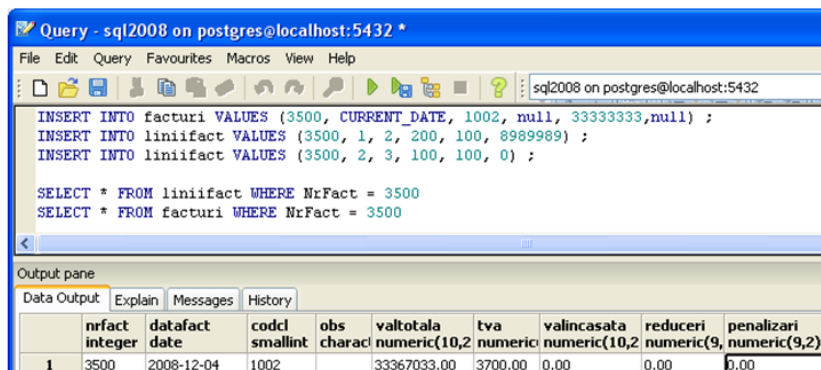
```
INSERT INTO liniifact VALUES (3500, 1, 2, 200, 100, 8989989) ;
```

```
INSERT INTO liniifact VALUES (3500, 2, 3, 100, 100, 0) ;
```

Figura 17.2 arată conținutul înregistrărilor din tabelele FACTURI și LINIIFACT care se referă la factura 3500, valorile calculate fiind, cel puțin în aparență, corecte.



	nrfact integer	linie smallint	codpr smallint	cantitate numeric(8,0)	pretunit numeric(9,2)	tvalinie numeric(9,2)
1	3500	1	2	200	100.00	1800.00
2	3500	2	3	100	100.00	1900.00



	nrfact integer	datafact date	codcl smallint	obs charac	valtotala numeric(10,2)	tva numeric	valincasata numeric(10,2)	reduceri numeric(9,2)	penalizari numeric(9,2)
1	3500	2008-12-04	1002		33367033.00	3700.00	0.00	0.00	0.00

Figura 17.2. Verificarea declanșatoarelor de inserare în tabelele FACTURI și LINIIFACT

Dacă la inserarea unei linii dintr-o factură (tabela LINIIFACT), valorile facturate și ale TVA (din FACTURI) trebuie incrementate, la ștergerea unei linii, valorile

celor două atribute trebuie scăzute. Poate apărea, însă, și o problemă de numărare a liniilor. Astfel, dacă factura are cinci linii, iar linia ștersă este a doua, în urma ștergerii după linia 1 urmează 3. Acest gol ar putea fi supărător pentru unii, așa că avem două variante. Prima este cea restrictivă: nu se permite decât ștergerea ultimei linii dintr-o factură – vezi listing 17.4.

Listing 17.4. Declanșatorul PL/pgSQL de ștergere în tabela LINIIFACT – versiunea 1.0

```
CREATE OR REPLACE FUNCTION trg_liniiifact_del()
  RETURNS TRIGGER AS $trg_liniiifact_del$
DECLARE
  v_nrlinii NUMERIC(3) ;
BEGIN
  SELECT MAX(linie) INTO v_nrlinii FROM liniifact WHERE nrfact = OLD.nrfact ;
  IF OLD.linie <> COALESCE(v_nrlinii,0) THEN
    RAISE EXCEPTION 'Nu puteti sterge decit ultima linie dintr-o factura ' ;
  END IF ;

  UPDATE facturi
    SET tva = tva - OLD.tvalinie,
        valtotala = valtotala - OLD.cantitate * OLD.pretunit
        - OLD.tvalinie
    WHERE nrfact = OLD.nrfact ;
  RETURN OLD ;
END ;
$trg_liniiifact_del$ LANGUAGE plpgsql ;
```

```
CREATE TRIGGER trg_liniiifact_del BEFORE DELETE ON liniifact
FOR EACH ROW EXECUTE PROCEDURE trg_liniiifact_del() ;
```

Dacă după crearea acestui declanșator, încercăm să ștergem prima linie a facturii 3500:

```
DELETE FROM liniifact WHERE NrFact=3500 AND linie=1
```

obținem mesajul de eroare din figura 17.3.

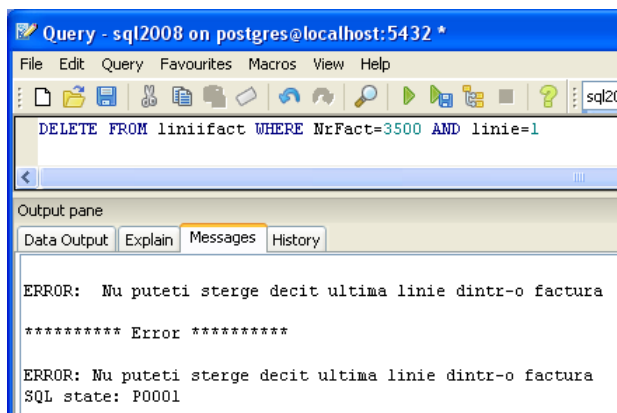


Figura 17.3. Verificarea declanșatoului de ștergere în tabela LINIIFACT (v1.0)

Ne simțim vinovați pentru rigiditatea soluției, așa că ne vom gândi la o alta, în care utilizatorul poate șterge orice linie dintr-o factură, iar liniile vor fi renumerate automat. Iată cum ar putea arăta această versiune (listing 17.5).

Listing 17.5. Declanșatorul PL/pgSQL de ștergere în tabela LINIIFACT – versiunea 2.0

-- varianta 2 - se permite stergerea oricarei linii si se renumereaza liniile acelei facturi

```
DROP FUNCTION trg liniifact_del() CASCADE ;
```

```
CREATE OR REPLACE FUNCTION trg liniifact_del()
```

```
  RETURNS TRIGGER AS $trg liniifact_del$
```

```
DECLARE
```

```
  v liniemax liniifact.linie%TYPE ;
```

```
BEGIN
```

```
  SELECT MAX(linie) INTO v liniemax FROM liniifact
```

```
  WHERE nrfact = OLD.NrFact ;
```

```
  IF v liniemax = OLD.linie THEN
```

```
    NULL ;
```

```
  ELSE
```

```
    UPDATE liniifact SET linie = linie - 1 WHERE linie > OLD.linie
```

```
    AND NrFact= OLD.NrFact ;
```

```
  END IF ;
```

```
  UPDATE facturi
```

```
  SET      tva = tva - OLD.tvalinie,
```

```
    valtotala = valtotala - OLD.cantitate * OLD.pretunit - OLD.tvalinie
```

```
  WHERE nrfact = OLD.nrfact ;
```

```
  RETURN OLD ;
```

```
END ;
```

```
$trg liniifact_del$ LANGUAGE plpgsql ;
```

```
CREATE TRIGGER trg liniifact_del AFTER DELETE ON liniifact
```

```
FOR EACH ROW
```

```
EXECUTE PROCEDURE trg liniifact_del() ;
```

Testăm validitatea declanșatorului ștergând prima linie din factură 3500², iar după cum o arată figura 17.4, ex - a doua linie a devenit prima.:

```
DELETE FROM liniifact WHERE NrFact=3500 AND linie=1
```

Până acum am rezolvat doar o parte a problemei atributelor calculate, cea mai spumoasă, și anume calcularea lor efectivă. Ne-a rămas ceva de pus la punct, fără de care corectitudinea atributelor calculate are fi îndoielnică. Să imaginăm o situație în care s-ar cuveni să restricționăm operațiunile de actualizare a tabeli FACTURI. Modificările valorilor pentru atributele *ValTotala* și *TVA* trebuie să fie permise numai dacă sunt consecință a declanșatoarelor tabeli LINIIFACT. Dacă

² Cele două linii ale facturii 3550 dinainte acestei ștergeri sunt vizibile în figura 17.2.

nu am face o asemenea verificare, orice utilizator ar putea să modifice direct (printr-o comandă UPDATE) valoarea totală a unei facturi și, astfel, s-ar crea o diferență față de informațiile din tabela LINIIFACT.

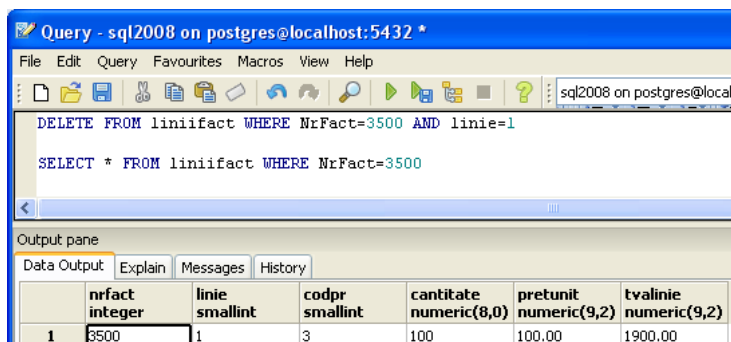


Figura 17.4. Verificarea declanșatorului de ștergere în tabela LINIIFACT (v2.0)

Ideea este una simplă: atunci când se modifică valoarea atributelor *ValTotala* și *TVA*, declanșatorul de modificare (UPDATE) din FACTURI verifică dacă modificarea provine din unul dintre declanșatoarele (INSERT, UPDATE sau DELETE) tabeli LINIIFACT. Dacă sursa modificării este un declanșator din LINIIFACT, modificarea este autorizată; în caz contrar, este interzisă.

În PostgreSQL nu există variabile publice, așa că vom folosi o tabelă pe care o vom numi, ostentativ, VAR_PUB:

```
CREATE TABLE var_pub (
    utilizator VARCHAR(40) NOT NULL,
    v_trg_liniifact BOOLEAN NOT NULL DEFAULT FALSE,
    v_trg_incasfact BOOLEAN NOT NULL DEFAULT FALSE
);
```

Tabela va avea câte o linie pentru fiecare utilizator conectat la baza de date. Celelalte două atribute, ambele de tip BOOLEAN³, vor avea valoarea logică TRUE dacă este în curs de execuție unul dintre declanșatoarele tabeli LINIIFACT, respectiv INCASFACT. Vă reamintesc (am discutat pe acest subiect și prin capitolul 6) că putem afla numele utilizatorului curent conectat la baza de date prin funcția sistem USER sau CURRENT_USER:

```
SELECT USER
```

Întrucât nu putem anticipa care sunt utilizatorii care vor lucra cu baza de date, putem gândi o procedură (știți deja că în PL/pgSQL procedurile sunt tot funcții)

³ Îmi face plăcere să vă reamintesc că, dintre cele patru servere BD, doar PostgreSQL acceptă în tabele atribute de tip BOOLEAN.

care să creeze, la apelare, o înregistrare în tabela VAR_PUB, dacă această înregistrare nu există dinainte. Iată funcția-procedură în listing 17.6.

Listing 17.6. Funcția PL/pgSQL care inserarează în VAR_PUB, dacă e nevoie, o înregistrare pentru utilizatorul curent

```
CREATE OR REPLACE FUNCTION p_init_vp () RETURNS VOID AS $$
BEGIN
    IF NOT EXISTS (SELECT * FROM var_pub WHERE utilizator = USER) THEN
        INSERT INTO var_pub VALUES (USER, FALSE, FALSE);
    END IF;
END;
$$ LANGUAGE plpgsql;
```

Pentru o apela direct, folosim:

```
SELECT p_init_vp()
```

Putem verifica dacă funcția funcționează prin:

```
SELECT * FROM var_pub
```

În continuare – vezi listing 17.7 – creăm două funcții ce furnizează valorile curente (din tabela VAR_PUB) ale atributelor *v_trg liniifact* și *v_trg incasfact* pentru utilizatorul curent.

Listing 17.7. Funcții PL/pgSQL ce extrag valorile atributelor *v_trg liniifact* și *v_trg incasfact* pentru utilizatorul curent

```
CREATE OR REPLACE FUNCTION f_trg liniifact () RETURNS BOOLEAN AS $$
DECLARE
    v BOOLEAN := FALSE;
BEGIN
    SELECT v_trg liniifact INTO v FROM var_pub WHERE utilizator = USER;
    RETURN v;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION f_trg incasfact () RETURNS BOOLEAN AS $$
DECLARE
    v BOOLEAN := FALSE;
BEGIN
    SELECT v_trg incasfact INTO v FROM var_pub WHERE utilizator = USER;
    RETURN v;
END;
$$ LANGUAGE plpgsql;
```

Tot pentru a ne ușura munca vom crea o funcție-procedură care va „seta” valoarea unuia dintre cele două câmpuri pe TRUE sau FALSE – vezi listing 17.8.

Listing 17.8. Funcție-procedură de setare a valorilor atributelor *v_trg liniifact* și *v_trg incasfact* pentru utilizatorul curent

```
CREATE OR REPLACE FUNCTION p_set (atribut_ VARCHAR, valoare_ BOOLEAN)
RETURNS VOID AS $$
BEGIN
    IF UPPER(atribut_) = 'V_TRG_LINIIFACT' THEN
```

```

UPDATE var_pub SET v_trg_liniifact = valoare_ WHERE utilizator = USER ;
ELSE
  IF UPPER(atribut_) = 'V_TRG_INCASFACT' THEN
    UPDATE var_pub SET v_trg_incasfact = valoare_ WHERE utilizator = USER ;
  END IF ;
END IF ;
END ;
$$ LANGUAGE plpgsql ;

```

Toate aceste funcții constituie accesorii pentru seria declanșatoarelor care urmează. Începem cu declanșatorul de inserare în LINIIFACT – listing 17.9. Păstrăm varianta anterioară, la care adăugăm un apel (prin comanda PERFORM) la procedura *p_init_vp*, și setarea pe TRUE a valorii atributului *v_trg_liniifact* pentru utilizatorul curent. După incrementarea valorilor atributelor *ValTotala* și *TVA*, readucem valoarea atributului *v_trg_liniifact* pentru utilizatorul curent la „starea” FALSE.

Listing 17.9. O nouă versiune a declanșatorului de inserare a unei înregistrări în LINIIFACT

```

DROP FUNCTION trg_liniifact_ins() CASCADE ;
CREATE OR REPLACE FUNCTION trg_liniifact_ins() RETURNS TRIGGER AS $trg_liniifact_ins$
BEGIN
  NEW.tvalinie := NEW.cantitate * NEW.pretunit * f_proctva(NEW.codpr) ;

  -- noutate !!!
  PERFORM p_init_vp() ;
  PERFORM p_set ('v_trg_liniifact', TRUE) ;
  UPDATE facturi SET tva = tva + NEW.tvalinie,
    valtotala = valtotala + NEW.cantitate * NEW.pretunit + NEW.tvalinie
    WHERE nrfact = NEW.nrfact ;

  -- refacem valoarea pseudo-variabilei
  PERFORM p_set ('v_trg_liniifact', FALSE) ;
  RETURN NEW ;
END ;
$trg_liniifact_ins$ LANGUAGE plpgsql ;

```

```

CREATE TRIGGER trg_liniifact_ins BEFORE INSERT ON liniifact
FOR EACH ROW EXECUTE PROCEDURE trg_liniifact_ins() ;

```

Declanșatorul de modificare a unei linii în LINIIFACT este ceva mai interesant, pentru că e prima versiune discutată (în listing 17.10).

Listing 17.10. Noua (de fapt, e prima!) versiune a declanșatorului de modificare a unei înregistrări din LINIIFACT

```

DROP FUNCTION trg_liniifact_upd_br() CASCADE ;

CREATE OR REPLACE FUNCTION trg_liniifact_upd_br()
  RETURNS TRIGGER AS $trg_liniifact_upd_br$
BEGIN
  IF NEW.nrfact = OLD.nrfact THEN          -- nu s-a schimbat nrfact !!!!
    IF NEW.codpr <> OLD.codpr OR NEW.cantitate <> OLD.cantitate OR
      NEW.pretunit <> OLD.pretunit OR NEW.tvalinie <> OLD.tvalinie THEN

```

```

NEW.tvalinie := NEW.cantitate * NEW.pretunit * f_proctva(NEW.codpr) ;

PERFORM p_init_vp() ;
PERFORM p_set ('v_trg_liniiifact', TRUE) ;

UPDATE facturi SET tva = tva - OLD.tvalinie + NEW.tvalinie,
                 valtotala = valtotala - (OLD.cantitate * OLD.pretunit + OLD.tvalinie)
                 + (NEW.cantitate * NEW.pretunit + NEW.tvalinie)
WHERE nrfact = NEW.nrfact ;

PERFORM p_set ('v_trg_liniiifact', FALSE) ;

END IF ;
ELSE -- vechile valori trebuie scazute de la "vechea" factura
-- iar noile valori adaugate "noii" facturi
PERFORM p_init_vp() ;
PERFORM p_set ('v_trg_liniiifact', TRUE) ;
UPDATE facturi SET tva = tva - OLD.tvalinie,
                 valtotala = valtotala - (OLD.cantitate * OLD.pretunit + OLD.tvalinie)
WHERE nrfact = OLD.nrfact ;

NEW.tvalinie := NEW.cantitate * NEW.pretunit * f_proctva(NEW.codpr) ;
UPDATE facturi SET tva = tva + NEW.tvalinie,
                 valtotala = valtotala + (NEW.cantitate * NEW.pretunit + NEW.tvalinie)
WHERE nrfact = NEW.nrfact ;

-- refacem valoarea pseudo-variabilei
PERFORM p_set ('v_trg_liniiifact', FALSE) ;
END IF ;
RETURN NEW ;
END ;
$strg_liniiifact_upd_br$ LANGUAGE plpgsql ;

```

CREATE TRIGGER trg_liniiifact_upd_br BEFORE UPDATE ON liniiifact

FOR EACH ROW EXECUTE PROCEDURE trg_liniiifact_upd_br() ;

Listing-ul 17.11 conține ultimul dintre cele trei declanșatoare ale tabelii LINII-FACT – cel dedicat ștergerii unei linii.

Listing 17.11. Noua versiune a declanșatorului de ștergere a unei înregistrări din LINIIFACT

```

DROP FUNCTION trg_liniiifact_del() CASCADE ;

DROP FUNCTION trg_liniiifact_del2() CASCADE ;
CREATE OR REPLACE FUNCTION trg_liniiifact_del2()
RETURNS TRIGGER AS $strg_liniiifact_del2$
BEGIN
-- noutate !!!
PERFORM p_init_vp() ;
PERFORM p_set ('v_trg_liniiifact', TRUE) ;
--
UPDATE facturi
SET tva = tva - OLD.tvalinie,
    valtotala = valtotala - OLD.cantitate * OLD.pretunit - OLD.tvalinie
WHERE nrfact = OLD.nrfact ;

PERFORM p_set ('v_trg_liniiifact', FALSE) ;
RETURN OLD ;
END ;

```

```
$trg_liniifact_del2$ LANGUAGE plpgsql ;
```

```
CREATE TRIGGER trg_liniifact_del2 BEFORE DELETE ON liniifact
FOR EACH ROW
```

```
EXECUTE PROCEDURE trg_liniifact_del2() ;
```

Cele trei declanșatoare asigură semnalizarea utilă declanșatorului de modificare în FACTURII, declanșator care trebuie să „discearnă” o modificare bună de una rea – vezi listing 17.12.

Listing 17.12. Declanșatorul de modificare a unei înregistrări din FACTURI

```
DROP FUNCTION trg_facturi_upd() CASCADE ;

CREATE OR REPLACE FUNCTION trg_facturi_upd()
RETURNS TRIGGER AS $trg_facturi_upd$
BEGIN
    IF NEW.tva <> OLD.tva OR NEW.valtotala <> OLD.valtotala THEN
        PERFORM p_init_vp() ;
        IF NOT f_trg_liniifact () THEN
            RAISE EXCEPTION
                'Nu puteti modifica interactiv nici valoarea facturii, nici TVA-ul acesteia !' ;
        END IF ;
    END IF ;
    RETURN NEW ;
END ;
$trg_facturi_upd$ LANGUAGE plpgsql ;
```

```
CREATE TRIGGER trg_facturi AFTER UPDATE ON facturi FOR EACH ROW
```

```
EXECUTE PROCEDURE trg_facturi_upd() ;
```

După atâta străduință, a sosit momentul verificărilor. Să începem printr-o tentativă frauduloasă de a modifica direct valoarea atributului ValTotală:

```
UPDATE facturi SET ValTotala = 12345678 WHERE NrFact = 1111
```

Spre meritul lor (și al nostru) declanșatoarele își fac datoria, după cum sugerează figura 17.5.

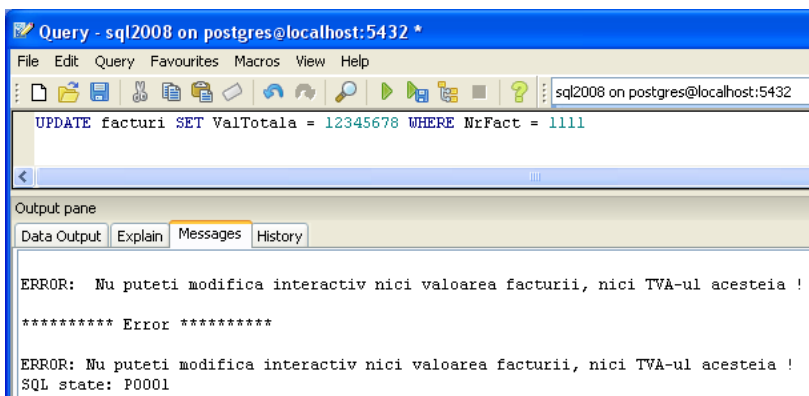


Figura 17.5. Blocarea unei modificări imorale în tabela FACTURI

În schimb, dacă ștergem liniile rămase în factura 1111 și le re-adăugăm după scenariul din capitolul trei, lucrurile decurg fără incidente:

```
DELETE FROM liniifact WHERE NrFact = 1111 ;
INSERT INTO liniifact (nrfact, linie, codpr, cantitate, pretunit)
VALUES (1111, 1, 1, 50, 1000) ;
INSERT INTO liniifact (nrfact, linie, codpr, cantitate, pretunit)
VALUES (1111, 2, 2, 75, 1050) ;
INSERT INTO liniifact (nrfact, linie, codpr, cantitate, pretunit)
VALUES (1111, 3, 5, 500, 7060) ;
```

17.3. Declanșatoare în PL/SQL

În materie de declanșatoare, limbajul PL/SQL este deopotrivă și unul dintre cele mai generoase și unul dintre cele mai năzuroase. La tipologia din tabelul 8.1 PL/SQL adaugă declanșatoarele pentru conectarea și închiderea conexiunii la/cu o bază de date, declanșatoare de pornire/oprire a bazei de date și declanșatoare asociate operațiunilor de tip DDL (CREATE/ALTER/DROP TABLE/PROCEDURE/FUNCTION/TRIGGER/...⁴).

Începem ca în paragraful precedent prin a crea o secvență – *seq_codcl*. Formatul comenzii este destul de apropiat celui din PostgreSQL:

```
CREATE SEQUENCE seq_codcl START WITH 1011 INCREMENT BY 1
MINVALUE 1001 MAXVALUE 8900 NOCYCLE NOCACHE ORDER ;
```

Cum e și normal, un declanșator se crează în PL/SQL direct prin comanda CREATE TRIGGER iar structura acestuia o respectă pe cea a oricărui bloc PL/SQL, cu câteva deosebiri. Declanșatorul de tip BEFORE-INSERT-ROW pentru tabela CLIENTI este cel din listing 17.13.

Listing 17.13. Declanșator PL/SQL pentru inserarea unei linii în tabela CLIENTI

```
CREATE OR REPLACE TRIGGER trg_clienti_ins_before_row
BEFORE INSERT ON clienti FOR EACH ROW
BEGIN
    :NEW.CodCl := seq_codcl.NEXTVAL;
END ;
```

Extragerea următoarei valori unice disponibile din secvență se realizează ușor diferit față de PL/pgSQL: *seq_codcl.NEXTVAL*. Până la versiunea 11g, în PL/SQL comanda de atribuire a valorii din secvență atributului *CodCl* se putea realiza numai în formatul:

```
SELECT seq_codcl.NEXTVAL INTO :NEW.CodCl FROM dual
```

⁴ Vezi și [Fotache s.a.2003]

Oracle are alergie la opțiunea ON UPDATE CASCADE în comenzile CREATE TABLE și ALTER TABLE. Printr-un declanșator de modificare a unei linii putem remedia acest lucru. Notăm ca plus al PL/SQL-ului că un declanșator de modificare poate fi asociat, chiar la creare, numai unui atribut (sau unei părți dintre atributele) din tabelă. Spre exemplu, dorim ca orice modificare a numărului unei facturi să se propage automat în tabelele copil, LINIFACT și INCASFAC. Declanșatorul care va rezolva această problemă este cel din listing 17.14.

Listing 17.14. Declanșator PL/SQL pentru substituiea opțiunii ON UPDATE CASCADE

```
CREATE OR REPLACE TRIGGER trg_facturi_upd_a_r_NrFact
AFTER UPDATE OF NrFact ON facturi FOR EACH ROW
BEGIN
    UPDATE liniifact SET NrFact = :NEW.NrFact WHERE NrFact = :OLD.NrFact ;
    UPDATE incasfact SET NrFact = :NEW.NrFact WHERE NrFact = :OLD.NrFact ;
END ;
```

Declanșatorul pentru inserarea de înregistrări în tabela FACTURI (listing 17.15) este constuit de o manieră similară și nu ridică nicio problemă.

Listing 17.15. Declanșatorul PL/SQL pentru inserarea unei înregistrări în FACTURI

```
CREATE OR REPLACE TRIGGER trg_facturi_ins
BEFORE INSERT ON facturi FOR EACH ROW
BEGIN
    :NEW.valtotala := 0 ;
    :NEW.tva := 0 ;
    :NEW.valincasata := 0 ;
END ;
```

Nici pentru declanșatorul *trg liniifact_ins* dedicat inserării unei înregistrări în tabela LINIIFACT nu apar modificări spectaculoase – vezi listing 17.16.

Listing 17.16. Declanșatorul PL/SQL pentru inserarea unei înregistrări în LINIIFACT

```
CREATE OR REPLACE TRIGGER trg liniifacturi_ins
BEFORE INSERT ON liniifact FOR EACH ROW
BEGIN
    :NEW.tvalinie := COALESCE(:NEW.cantitate,0) * COALESCE(:NEW.pretunit,0) *
        f_proctva2(:NEW.codpr) ;
    SELECT COALESCE(MAX(Linie),0) + 1 INTO :NEW.linie FROM liniifact
    WHERE NrFact=:NEW.NrFact ;

    UPDATE facturi SET tva = COALESCE(tva,0) + :NEW.tvalinie,
        valtotala = COALESCE(valtotala,0) + :NEW.cantitate * :NEW.pretunit + :NEW.tvalinie
    WHERE nrfact = :NEW.nrfact ;
END ;
```

Testarea cu aceleași trei comenzi INSERT din paragraful anterior a celor două declanșatoare este încununată de succes:

```
INSERT INTO facturi (nrfact, datafact, codcl, valtotala)
VALUES (3500, CURRENT_DATE, 1002, 33333333) ;
INSERT INTO liniifact VALUES (3500, 1, 2, 200, 100, 8989989) ;
INSERT INTO liniifact VALUES (3500, 2, 3, 100, 100, 0) ;
```

Necazurile încep odată cu tentativa de portare a declanșatorului din listing 17.4 – *trg_liniiifact_del*. Transcrierea din listingul 17.17 respectă aceeași idee, cu câteva diferențe, cea mai vizibilă fiind înlocuirea comenzii *RAISE EXCEPTION* cu mai durdulia *RAISE_APPLICATION_ERROR*.

Listing 17.17. Declanșatorul PL/SQL pentru ștergerea unei înregistrări din LINIIFACT

```
CREATE OR REPLACE TRIGGER trg_liniiifact_del
BEFORE DELETE ON liniiifact FOR EACH ROW
DECLARE
    v_nrlinii liniiifact.linie%TYPE ;
BEGIN
    SELECT MAX(linie) INTO v_nrlinii FROM liniiifact WHERE nrfact = :OLD.nrfact ;
    IF :OLD.linie <> COALESCE(v_nrlinii,0) THEN
        RAISE_APPLICATION_ERROR(-20567,
            'Nu puteti sterge decit ultima linie dintr-o factura ' ) ;
    END IF ;

    UPDATE facturi
    SET tva = tva - :OLD.tvalinie,
        valtotala = valtotala - :OLD.cantitate * :OLD.pretunit - :OLD.tvalinie
    WHERE nrfact = :OLD.nrfact ;
END ;
```

La compilare, PL/SQL se comportă pașnic, însă la verificarea sa prin comanda:

DELETE FROM liniiifact WHERE NrFact=3500 AND linie=1

suntem luați pe nepusă masă de mesajul de eroare din figura 17.6. Ni se spune că tabela LINIIFACT este schimbătoare (ca și omul, dealtminteri). Din mesaj și din documentație deducem că Oracle nu permite ca în declanșatoarele de modificare și ștergere (și cele de inserare, dacă se folosește sintaxa *INSERT INTO... SELECT...*) să apară o interogare (*SELECT*) chiar pe tabela în declanșatorul căreia suntem.

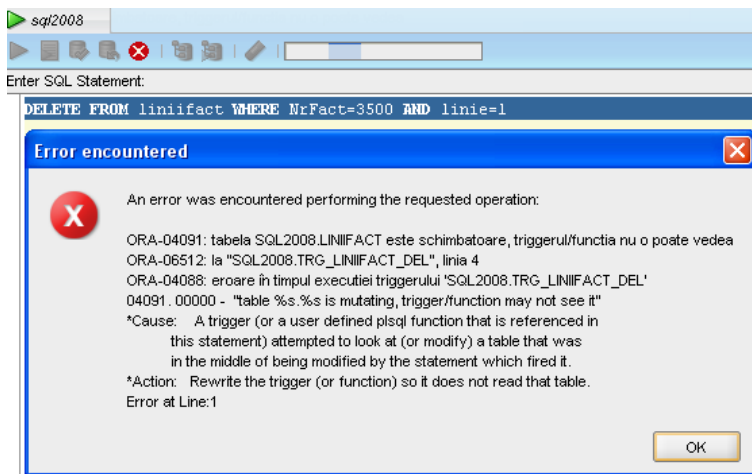


Figura 17.6. O problemă „clasică” în declanșatoarele de tip ROW în Oracle PL/SQL

Această restricție este valabilă numai pentru declanșatoarele la nivel de linie. Așa că ne-am putea gândi să evităm problema, pasând SELECT-ul în declanșatorul de tip AFTER STATEMENT, ultimul executat. Dar în declanșatoarele la nivel de comandă nu avem acces la valorile vechi (:OLD) și noi (:NEW) de pe linia ștersă sau modificată. O soluție anemică este să folosim variabile publice pe care le „umplem” în declanșatorul la nivel de linie și pe care le valorificăm în declanșatorul AFTER STATEMENT. Oricum, în ultimul paragraf din capitolul 16 am creat pachetul *pac_vinzari* în specificațiile căruia apar, ca din întâmplare, patru variabile (publice) *v_trg_linii_fact*, *v_trg_incasfact*, *v_nrfact* și *v_linie*.

Soluția din listing 17.18 este articulată în trei declanșatoare. Primul - *trg_linii_fact_del1* - este de tip BEFORE STATEMENT și NULL-iează cele două variabile publice. Al doilea (în ordinea execuției) - *trg_linii_fact_del2* - este tip AFTER ROW și, pe lângă decrementarea celor două atribute calculate din tabela FACTURI, stochează numărul facturii și numărul liniei șterse în cele două variabile publice. Al treilea - *trg_linii_fact_del3* - este ultimul executat (AFTER STATEMENT) și face verificarea propriu-zisă.

Listing 17.18. Trei declanșatoare PL/SQL pentru ștergerea unei înregistrări din LINIIFACT

```
DROP TRIGGER trg_linii_fact_del ;

-- primul declansator - BEFORE STATEMENT- "curata" variabilele publice
CREATE OR REPLACE TRIGGER trg_linii_fact_del1 BEFORE DELETE ON liniifact
BEGIN
    pac_vinzari.v_nrfact := NULL ;
    pac_vinzari.v_linie := NULL ;
END ;

-- al doilea declansator - AFTER ROW - "seteaza" variabilele publice
-- si actualizeaza FACTURI.tva si FACTURI.ValTotala
CREATE OR REPLACE TRIGGER trg_linii_fact_del2
AFTER DELETE ON liniifact FOR EACH ROW
BEGIN
    pac_vinzari.v_nrfact := :OLD.NrFact ;
    pac_vinzari.v_linie := :OLD.Linie ;
    UPDATE facturi
        SET tva = tva - :OLD.tvalinie,
            valtotala = valtotala - :OLD.cantitate * :OLD.pretunit - :OLD.tvalinie
    WHERE nrfact = :OLD.nrfact ;
END ;

-- al treilea declansator - AFTER STATEMENT - face verificarea
CREATE OR REPLACE TRIGGER trg_linii_fact_del3
AFTER DELETE ON liniifact
DECLARE
    v_nrlinii liniifact.linie%TYPE ;
BEGIN
    SELECT MAX(linie) INTO v_nrlinii FROM liniifact WHERE nrfact = pac_vinzari.v_nrfact ;
    IF pac_vinzari.v_linie < COALESCE(v_nrlinii,0) THEN
        RAISE_APPLICATION_ERROR(-20567,
            'Nu puteti sterge decit ultima linie dintr-o factura ');
    END IF ;
END ;
```

Acum comanda de ștergere funcționează. Cu o mică observație, esențială ! Ce se întâmplă dacă am șterge mai multe linii cu o singură comanda DELETE? Comanda:

DELETE FROM liniifact WHERE NrFact=3500 AND Linie=1 OR

NrFact = 3119 AND linie IN (2,3)

ar trebui blocată, întrucât din factura 3500 se șterge prima linie, iar factura are două. Comanda, însă, se execută fără obiecții, ceea ce, evident, este o eroare. Explicația ține de faptul că declanșatoarele păstrează numai ultima linie din ultima factură din care a avut loc ștergerea.

Prin urmare, în loc de variabile simple, va trebui să folosim colecții. Dintre cele trei tipuri de masive/colecții disponibile în PL/SQL vor recurge la vectori asociativi, utilizabili doar în programe (nu și în tabele). Listing-ul 17.19 conține specificațiile pachetului *pac_vinzari* în care definim două tipuri vectori asociativi – *t_nrfacturi* și *t_linii* – și două variabile de acest tip – *v_nrfacturi* și *v_linii*.

Listing 17.19. Două variabile masiv (publice) în PL/SQL

```
CREATE OR REPLACE PACKAGE pac_vinzari AS
  -- variabile publice
  v_trg_liniifact BOOLEAN := FALSE ;
  v_trg_incasfact BOOLEAN := FALSE ;
  v_unde_sunt VARCHAR2(30) ;

  TYPE t_nrfacturi IS TABLE OF liniifact.nrfact%TYPE INDEX BY PLS_INTEGER ;
  TYPE t_linii IS TABLE OF liniifact.linie%TYPE INDEX BY PLS_INTEGER ;
  v_nrfacturi t_nrfacturi;
  v_linii t_linii;

  FUNCTION f_proctva (codpr_ produse.codpr%TYPE) RETURN produse.proctva%TYPE ;
  FUNCTION f_proctva (denpr_ produse.denpr%TYPE) RETURN produse.proctva%TYPE ;
  FUNCTION f_valtot (nrfact_ facturi.nrfact%TYPE) RETURN NUMERIC ;
  FUNCTION f_val_cumul (nrfact_ facturi.nrfact%TYPE) RETURN NUMERIC ;
  PROCEDURE p_cumul ;
  PROCEDURE p_cumul2 ;
END ; -- pac_vinzari
/
```

Cele trei declanșatoare trebuie rescrise ca în listing 17.20. Cel de-al treilea devine un pic mai corpolent întrucât este nevoie de o buclă care să parcurgă toate componentele celor două variabile-colecție.

Listing 17.20. Noile versiuni ale celor trei declanșatoare de ștergere în LINIIFACT ce utilizează variabilele masiv

```
-- refacem primul declansator - BEFORE STATEMENT
CREATE OR REPLACE TRIGGER trg_liniifact_del1 BEFORE DELETE ON liniifact
BEGIN
  pac_vinzari.v_nrfacturi.DELETE ;
  pac_vinzari.v_linii.DELETE ;
END ;

-- al doilea declansator - AFTER ROW - rescris
CREATE OR REPLACE TRIGGER trg_liniifact_del2
BEFORE DELETE ON liniifact FOR EACH ROW
```

```

BEGIN
pac_vinzari.v_nrfacturi (pac_vinzari.v_nrfacturi.COUNT + 1) := :OLD.NrFact ;
pac_vinzari.v_linii (pac_vinzari.v_linii.COUNT + 1) := :OLD.Linie ;

UPDATE facturi
SET tva = tva - :OLD.tvalinie,
    valtotala = valtotala - :OLD.cantitate * :OLD.pretunit - :OLD.tvalinie
WHERE nrfact = :OLD.nrfact ;
END ;

-- al treilea declansator - AFTER STATEMENT - face verificarea
CREATE OR REPLACE TRIGGER trg_liniiifact_del3
AFTER DELETE ON liniifact
DECLARE
    v_nrlinii liniifact.linie%TYPE := 0 ;
BEGIN
FOR i IN 1..pac_vinzari.v_nrfacturi.COUNT LOOP
    SELECT MAX(linie) INTO v_nrlinii FROM liniifact WHERE nrfact = pac_vinzari.v_nrfacturi(i) ;
    IF pac_vinzari.v_linii(i) < COALESCE(v_nrlinii,0) THEN
        RAISE_APPLICATION_ERROR(-20567,
            'Nu puteti sterge decit ultima linie dintr-o factura ' ) ;
    END IF ;
END LOOP ;
END ;

```

În noua versiune cele trei declanșatoare par a-și face bine treaba. Dar, după cum meditam în precedentul paragraf, această soluție este mult prea rigidă pentru a fi acceptabilă, așa că ne propunem să redactăm varianta după care în facturile în care s-a șters măcar o linie se numerotează automat liniile rămase. Ca să nu mai umblăm în corpul pachetului *pac_vinzari* (ar trebui să compilăm toate funcțiile și procedurile deja existente), creăm unul nou-nouț pe care îl vom denumi *pac_colecții* – vezi listing 17.21.

Listing 17.21. Specificațiile și corpul pachetului ***pac_colecții***

```

CREATE OR REPLACE PACKAGE pac_colecții AS
    TYPE t_nrfacturi IS TABLE OF liniifact.nrfact%TYPE INDEX BY PLS_INTEGER ;
    TYPE t_linii IS TABLE OF liniifact.linie%TYPE INDEX BY PLS_INTEGER ;
    v_nrfacturi t_nrfacturi;
    v_linii t_linii;
    FUNCTION f_apare_in_nrfacturi (nrfact_ liniifact.nrfact%TYPE) RETURN BOOLEAN ;
END ; -- pac_colecții
/

CREATE OR REPLACE PACKAGE BODY pac_colecții AS
-----
FUNCTION f_apare_in_nrfacturi (nrfact_ liniifact.nrfact%TYPE) RETURN BOOLEAN
IS
    v_este BOOLEAN := FALSE ;
BEGIN
    FOR i IN 1..v_nrfacturi.COUNT LOOP
        IF v_nrfacturi(i) = nrfact_ THEN
            v_este := TRUE ;
            EXIT ;
        END IF ;
    END LOOP ;
    RETURN v_este ;

```

```
END f_apare_in_nrfacturi ;
```

```
END ; -- pac_colectii  
/
```

Pe lângă cele două tipuri și variabile vectori asociativi, am creat și o funcție de căutare a unui număr de factură în vectorul *v_nrfacturi*. Dintre cele trei declanșatoare din listing-ul 17.22, ultimul este cel mai diferit față de varianta anterioară, fiind cel care renumerează toate liniile în facturile în care avem măcar o linie ștearsă.

Listing 17.22. (Și) mai noile declanșatoare de ștergere în LINIIFACT

```
-- primul declansator - BEFORE STATEMENT
CREATE OR REPLACE TRIGGER trg_liniiifact_del1 BEFORE DELETE ON liniiifact
BEGIN
    pac_colectii.v_nrfacturi.DELETE ;
    pac_colectii.v_linii.DELETE ;
END ;
/

-----

-- al doilea declansator - AFTER ROW
CREATE OR REPLACE TRIGGER trg_liniiifact_del2
BEFORE DELETE ON liniiifact FOR EACH ROW
BEGIN
    IF pac_colectii.f_apare_in_nrfacturi(:OLD.NrFact) = FALSE THEN
        pac_colectii.v_nrfacturi (pac_colectii.v_nrfacturi.COUNT + 1) := :OLD.NrFact ;
    END IF;

    UPDATE facturi
        SET tva = tva - :OLD.tvalinie,
            valtotala = valtotala - :OLD.cantitate * :OLD.pretunit - :OLD.tvalinie
        WHERE nrfact = :OLD.nrfact ;
END ;
/

-----

-- al treilea declansator - AFTER STATEMENT
CREATE OR REPLACE TRIGGER trg_liniiifact_del3
AFTER DELETE ON liniiifact
DECLARE
    j NUMBER(2) ;
    CURSOR c_liniiifact (nrfact_ liniiifact.nrfact%TYPE) IS
        SELECT * FROM liniiifact WHERE NrFact = nrfact_ ORDER BY Linie FOR UPDATE ;
BEGIN
    FOR i IN 1..pac_colectii.v_nrfacturi.COUNT LOOP
        j := 1 ;
        FOR rec_liniiifact IN c_liniiifact (pac_colectii.v_nrfacturi(i)) LOOP
            UPDATE liniiifact SET linie = j WHERE CURRENT OF c_liniiifact ;
            j := j + 1 ;
        END LOOP ;
    END LOOP ;
END ;
/
```

Figura 17.7 ilustrează situația liniilor facturii 3119 înaintea și după ștergerea celor cu numerele 1 și 3. Totul pare a fi în regulă. Atenție, însă ! Declanșatorul *trg_liniiifact_del3* va modifica în unele înregistrări din LINIIFACT numărul liniei

ceea ce poate atrage lansarea eventualei declanșator de modificare al aceleași tabele.

Left Screenshot:

Enter SQL Statement:

```
SELECT * FROM liniifact WHERE NrFact = 3119
```

Results:

	NRFACT	LINE	CODPR	CANTITATE	PRETUNIT	TVALINE
1	3119	1	2	35	1090	3433,5
2	3119	2	3	40	700	5320
3	3119	3	4	55	1410	6979,5
4	3119	4	5	755	6300	903735

Right Screenshot:

Enter SQL Statement:

```
DELETE FROM liniifact WHERE NrFact = 3119 AND linie IN (1,3);
```

```
SELECT * FROM liniifact WHERE NrFact = 3119
```

Results:

	NRFACT	LINE	CODPR	CANTITATE	PRETUNIT	TVALINE
1	3119	1	3	40	700	5320
2	3119	2	5	755	6300	903735

Figura 17.7. Renumerotarea liniilor prin declanșatoare în factura 3119

Ne propunem să rezolvăm și problema protejării atributelor calculate de modificările neautorizate. Pentru aceasta vom pune la treabă o altă variabilă publică din pachetul *pac_vinzari* (listing 17.19) – *v_trg_liniiifact*. Această variabilă va avea valoarea logică TRUE doar pe parcursul execuției declanșatoarelor de inserare, modificare și ștergere ale tablei LINIIFACT. Prin urmare, în trei dintre declanșatoare – *trg_liniiifact_ins*, *trg_liniiifact_del2* și *trg_liniiifact_upd_br* – variabilă va fi setată pe TRUE imediat înainte de *UPDATE facturi ...* (comanda care modifică valorile atributelor ValTotala și TVA) și modificată pe FALSE imediat după UPDATE.

Listing 17.23. Noile declanșatoare ale LINIIFACT care asigură „protecția” atributelor calculate din FACTURI

```
CREATE OR REPLACE TRIGGER trg_liniiifact_ins
BEFORE INSERT ON liniifact FOR EACH ROW
BEGIN
    pac_vinzari.v_trg_liniiifact := TRUE ;
    :NEW.tvalinie := COALESCE(:NEW.cantitate,0) * COALESCE(:NEW.pretunit,0) *
        f_proctva2(:NEW.codpr) ;
    SELECT COALESCE(MAX(Linie),0) + 1 INTO :NEW.linie FROM liniifact
    WHERE NrFact=:NEW.NrFact ;

    UPDATE facturi SET tva = COALESCE(tva,0) + :NEW.tvalinie,
        valtotala = COALESCE(valtotala,0) + :NEW.cantitate * :NEW.pretunit + :NEW.tvalinie
    WHERE nrfact = :NEW.nrfact ;
    pac_vinzari.v_trg_liniiifact := FALSE ;
END ;
/

-----
-- se modifica al doilea declansator de stergere - AFTER ROW
CREATE OR REPLACE TRIGGER trg_liniiifact_del2
BEFORE DELETE ON liniifact FOR EACH ROW
BEGIN
    IF pac_colectii.f_apare_in_nrfacturi(:OLD.NrFact) = FALSE THEN
        pac_colectii.v_nrfacturi (pac_colectii.v_nrfacturi.COUNT + 1) := :OLD.NrFact ;
    END IF;
    pac_vinzari.v_trg_liniiifact := TRUE ;
    UPDATE facturi
    SET tva = tva - :OLD.tvalinie,
```

```

        valtotala = valtotala - :OLD.cantitate * :OLD.pretunit - :OLD.tvalinie
    WHERE nrfact = :OLD.nrfact ;
    pac_vinzari.v_trg liniifact := FALSE ;
END ;
/

-----

-- il modificam usor si pe al treilea declansator de stergere
CREATE OR REPLACE TRIGGER trg_liniifact_del3
AFTER DELETE ON liniifact
DECLARE
    j NUMBER(2) ;
    CURSOR c_liniifact (nrfact_ liniifact.nrfact%TYPE) IS
        SELECT * FROM liniifact WHERE NrFact = nrfact_ ORDER BY Linie FOR UPDATE ;
BEGIN
    FOR i IN 1..pac_colectii.v_nrfacturi.COUNT LOOP
        j := 1 ;
        FOR rec_liniifact IN c_liniifact (pac_colectii.v_nrfacturi(i)) LOOP
            pac_vinzari.v_unde_sunt := 'trg_liniifact_del3' ;
            UPDATE liniifact SET linie = j WHERE CURRENT OF c_liniifact ;
            pac_vinzari.v_unde_sunt := NULL ;
            j := j + 1 ;
        END LOOP ;
    END LOOP ;
END ;
/

-----

CREATE OR REPLACE TRIGGER trg_liniifact_upd_bs
BEFORE UPDATE OF NrFact, CodPr, Cantitate, PretUnit, TVALinie ON liniifact
BEGIN
    IF COALESCE(pac_vinzari.v_unde_sunt, ' ') NOT IN
        ('trg_liniifact_upd_as', 'trg_liniifact_del3') THEN
        pac_colectii.v_nrfacturi.DELETE ;
    END IF ;
END ;
/

-----

CREATE OR REPLACE TRIGGER trg_liniifact_upd_br_linie
BEFORE UPDATE OF Linie ON liniifact FOR EACH ROW
BEGIN
    IF COALESCE(pac_vinzari.v_unde_sunt, ' ') NOT IN
        ('trg_liniifact_del3', 'trg_liniifact_upd_as') THEN
        RAISE_APPLICATION_ERROR(-20456,
            'Nu aveti permisiunea de a modifica valoarea atributului Linie !') ;
    END IF ;
END ;
/

-----

CREATE OR REPLACE TRIGGER trg_liniifact_upd_br
BEFORE UPDATE OF NrFact, CodPr, Cantitate, PretUnit, TVALinie ON liniifact
FOR EACH ROW
BEGIN
    pac_vinzari.v_unde_sunt := 'trg_liniifact_upd_br' ;
    pac_vinzari.v_trg_liniifact := TRUE ;
    :NEW.tvalinie := COALESCE(:NEW.cantitate,0) * COALESCE(:NEW.pretunit,0) *
        f_proctva2(:NEW.codpr) ;
    UPDATE facturi SET tva = COALESCE(tva,0) - :OLD.tvalinie,
        valtotala = COALESCE(valtotala,0) - :OLD.cantitate * :OLD.pretunit + :OLD.tvalinie
    WHERE nrfact = :OLD.nrfact ;

```

```

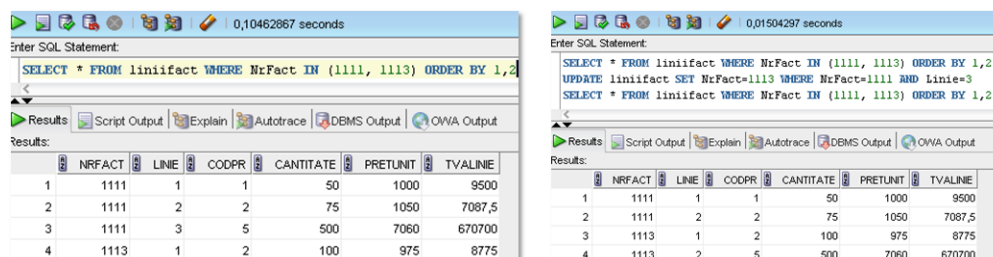
UPDATE facturi SET tva = COALESCE(tva,0) + :NEW.tvalinie,
    valtotala = COALESCE(valtotala,0) + :NEW.cantitate * :NEW.pretunit + :NEW.tvalinie
WHERE nrfact = :NEW.nrfact ;

IF :NEW.NrFact <> :OLD.NrFact THEN
    pac_colectii.v_nrfacturi(pac_colectii.v_nrfacturi.COUNT+1) := :OLD.NrFact ;
    pac_colectii.v_nrfacturi(pac_colectii.v_nrfacturi.COUNT+1) := :NEW.NrFact ;
END IF ;
pac_vinzari.v_trg_liniiifact := FALSE ;
pac_vinzari.v_unde_sunt := NULL ;
END ;
/

-----
CREATE OR REPLACE TRIGGER trg_liniiifact_upd_as
AFTER UPDATE OF NrFact, CodPr, Cantitate, PretUnit, TVALinie ON liniiifact
DECLARE
    j NUMBER(2) ;
    CURSOR c_liniiifact (nrfact_ liniiifact.nrfact%TYPE) IS
        SELECT * FROM liniiifact WHERE NrFact = nrfact_ ORDER BY Linie FOR UPDATE ;
BEGIN
    IF COALESCE(pac_vinzari.v_unde_sunt, ' ') IN
        ('trg_liniiifact_upd_as', 'trg_liniiifact_del3') THEN
        NULL ;
    ELSE
        FOR i IN 1..pac_colectii.v_nrfacturi.COUNT LOOP
            j := 1 ;
            FOR rec_liniiifact IN c_liniiifact (pac_colectii.v_nrfacturi(i)) LOOP
                pac_vinzari.v_unde_sunt := 'trg_liniiifact_upd_as' ;
                UPDATE liniiifact SET linie = j WHERE CURRENT OF c_liniiifact ;
                pac_vinzari.v_unde_sunt := NULL ;
                j := j + 1 ;
            END LOOP ;
        END LOOP ;
    END IF ;
END ;
/

```

Înainte de a trece la redactarea declanșatorului de „protecție”, punem la încercare aceste declanșatoare printr-o comandă UPDATE care să modifice, pentru o înregistrare din LINIIFACT, numărul facturii:



0,10462867 seconds

Enter SQL Statement:

```
SELECT * FROM liniiifact WHERE NrFact IN (1111, 1113) ORDER BY 1,2
```

Results:

	NRFAC	LINE	CODPR	CANTITATE	PRETUNIT	TVALINE
1	1111	1	1	50	1000	9500
2	1111	2	2	75	1050	7087,5
3	1111	3	5	500	7060	670700
4	1113	1	2	100	975	8775

0,01504297 seconds

Enter SQL Statement:

```
SELECT * FROM liniiifact WHERE NrFact IN (1111, 1113) ORDER BY 1,2
UPDATE liniiifact SET NrFact=1113 WHERE NrFact=1111 AND Linie=3
SELECT * FROM liniiifact WHERE NrFact IN (1111, 1113) ORDER BY 1,2
```

Results:

	NRFAC	LINE	CODPR	CANTITATE	PRETUNIT	TVALINE
1	1111	1	1	50	1000	9500
2	1111	2	2	75	1050	7087,5
3	1113	1	2	100	975	8775
4	1113	2	5	500	7060	670700

Figura 17.8. Trecerea unei linii din factura 1111 în factura 1113

```

SELECT * FROM liniiifact WHERE NrFact IN (1111, 1113) ORDER BY 1,2 ;
UPDATE liniiifact SET NrFact=1113 WHERE NrFact=1111 AND Linie=3 ;

```

```
SELECT * FROM liniifact WHERE NrFact IN (1111, 1113) ORDER BY 1,2 ;
```

Figura 17.8, ce suprinde liniile facturilor 1111 și 1113 înainte și după modificare, crește gradul de încredere în declanșatoarele tabeli LINIIFACT.

Nu ne-a rămas decât să redactăm declanșatorul de modificare a celor două atribute calculate din tabela FACTURI, declanșator care va testa valoarea variabilei *v_trg_liniifact* din pachetul *pac_vinzari* – vezi listing 17.24.

Listing 17.24. Declanșatorul care blochează modificarea directă a atributelor calculate

```
CREATE OR REPLACE TRIGGER trg_facturi_upd_br1
BEFORE UPDATE OF tva, valtotala ON facturi FOR EACH ROW
BEGIN
  IF pac_vinzari.v_trg_liniifact THEN
    -- e în regula, modificarile vin din declansatoarele LINIIFACT
    NULL ;
  ELSE
    RAISE_APPLICATION_ERROR(-26789,
      'Nu aveti permisiunea de a modifica interactiv atributele TVA si ValTotala') ;
  END IF ;
END ;
/
```

Figura 17.9 confirmă funcționarea declanșatoarelor celor două tabele. Încercarea de modificare directă a valorii atributului TVA lansează triggerul *trg_facturi_upd_br1*. Întrucât comanda UPDATE lasă „reci” declanșatoarele tabeli LINIIFACT variabila publică rămâne pe starea sa obișnuită – FALSE. Declanșatorul nu are încotro și va genera mesajul de eroare.

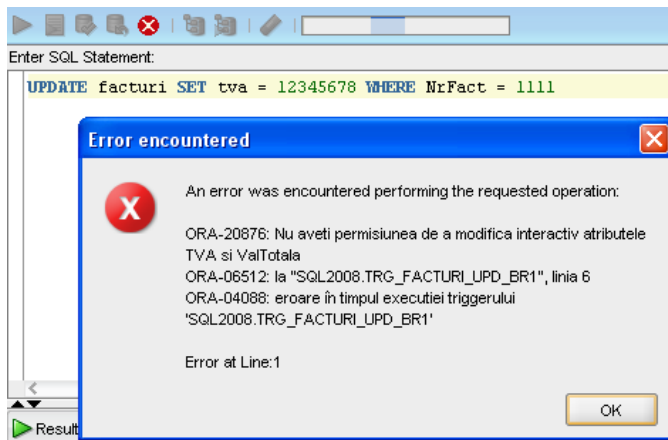


Figura 17.9. Tentativă blocată de modificare directă a TVA-ului unei facturi

Paragraful 14.3 făcea trimitere la problemele de actualizare a tabelelor sursă la modificarea conținutului tabelelor virtuale. Spuneam atunci că soluția elegantă în aceste situații o reprezintă declanșatoarele de tip INSTEAD OF. Pentru a exemplifi-

ca un asemenea gen de declanșator, ne vom folosi de exemplul din paragraful 14.3.3:

```
CREATE VIEW vFacturi AS
    SELECT lf.NrFact, DataFact, DenCl, f.CodCl, Linie, lf.CodPr, DenPr,
           UM, ProcTVA, Grupa,
           Cantitate, PretUnit,
           Cantitate * PretUnit AS ValFaraTVALinie,
           Cantitate * PretUnit * ProcTVA AS TVALinie,
           Cantitate * PretUnit * (1+ProcTVA) AS ValTotLinie
    FROM liniifact lf
    INNER JOIN produse p ON lf.CodPr=p.CodPr
    INNER JOIN facturi f ON lf.NrFact=f.NrFact
    INNER JOIN clienti c ON f.CodCl=c.CodCl
    ORDER BY lf.NrFact, Linie;
```

Ca element pregătitor, grupăm două funcții de căutare într-un pachet special pe care-l vom numi *pac_exista* – vezi listing 17.25. Prima, *f_exista_produș*, returnează TRUE dacă există în tabela PRODUSE un sortiment cu codul specificat, iar a doua, *f_exista_factura*, returnează valoarea logică TRUE dacă există o factură (în FACTURI) cu numărul specificat la invocare.

Listing 17.25. Un pachet nou

```
CREATE OR REPLACE PACKAGE pac_exista AS
    FUNCTION f_exista_produș (codpr_ produse.codpr%TYPE) RETURN BOOLEAN ;
    FUNCTION f_exista_factura (nrfact_ facturi.nrfact%TYPE) RETURN BOOLEAN ;
END ;
/
CREATE OR REPLACE PACKAGE BODY pac_exista AS
-----
    FUNCTION f_exista_produș (codpr_ produse.codpr%TYPE)
        RETURN BOOLEAN IS
        v_unu NUMBER(1) ;
    BEGIN
        SELECT 1 INTO v_unu FROM produse WHERE CodPr = codpr_ ;
        RETURN TRUE ;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RETURN FALSE ;
    END f_exista_produș ;
-----
    FUNCTION f_exista_factura (nrfact_ facturi.nrfact%TYPE)
        RETURN BOOLEAN IS
        v_unu NUMBER(1) ;
    BEGIN
        SELECT 1 INTO v_unu FROM facturi WHERE NrFact = nrfact_ ;
        RETURN TRUE ;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RETURN FALSE ;
    END f_exista_factura ;
END ;
```

/

Declanșatorul executat la inserarea unei linii în tabela virtuală – vezi listing 17.26 – testează, înainte de adăugarea înregistrării în tabela LINIIFACT, dacă înregistrarea respectivă nu se referă la un produs nou (PRODUSE) și/sau la o factură nouă (FACTURI).

Listing 17.26. Declanșatorul INSTEAD OF INSERT

```
CREATE OR REPLACE TRIGGER trg_vFacturi_oi INSTEAD OF INSERT ON vFacturi
BEGIN
  IF NOT pac_exista.f_exista_factura(:NEW.NrFact) THEN
    INSERT INTO facturi (NrFact, DataFact, CodCl)
      VALUES (:NEW.NrFact, :NEW.DataFact, :NEW.CodCl) ;
  END IF ;
  IF NOT pac_exista.f_exista_produs(:NEW.CodPr) THEN
    INSERT INTO produse
      VALUES (:NEW.CodPr, :NEW.DenPr, :NEW.UM, :NEW.Grupa, :NEW.ProcTVA) ;
  END IF ;
  INSERT INTO liniifact
    VALUES (:NEW.NrFact, :NEW.Linie, :NEW.CodPr, :NEW.Cantitate,
      :NEW.PretUnit, NULL) ;
END ;
```

The screenshot displays the SQL Developer interface during the execution of the trigger. The main window shows the trigger code being executed. Two smaller windows show the results of the queries: one for the 'facturi' table and one for the 'produse' table. The 'facturi' window shows a single row with NrFact = 3120. The 'produse' window shows a single row with CodPr = 8. The main window also shows the results of the 'INSERT INTO' statement, which includes the 'liniifact' table. The 'liniifact' table shows a single row with NrFact = 3120, Linie = 1, CodPr = 8, Cantitate = 400, PretUnit = 1200, and TVALINE = 91200.

Figura 17.10. Verificarea funcționării declanșatorului INSTEAD OF INSERT

Analog pot fi create declanșatoare pentru modificări și ștergeri ale liniilor din tabela virtuală. Verificarea funcționării declanșatorului (figura 17.10) se face executând pe rând următoarele comenzi:

```
SELECT * FROM facturi WHERE NrFact = 3120
```

```
SELECT * FROM produse WHERE CodPr = 8
```

```
INSERT INTO vFacturi (NrFact, Linie, CodPr, Cantitate, PretUnit, DataFact,  
                    CodCl, DenPr, UM, Grupa, ProcTVA)  
VALUES (3120, 1, 8, 400, 1200, DATE'2007-10-01', 1001, 'Produs 8', 'buc',  
        'Cafea', 0.19) ;
```

```
SELECT * FROM produse WHERE CodPr = 8
```

```
SELECT * FROM liniifact WHERE NrFact = 3120
```

```
SELECT * FROM facturi WHERE NrFact = 3120
```

17.4. Declanșatoare în Transact-SQL

Și în T-SQL există atât declanșatoare de tip DML, cât și de tip DDL (CREATE/ALTER/DROP). Spre deosebire de PL/pgSQL și PL/SQL tipologia triggerelor DML este ceva mai săracă. Pentru o tabelă se pot defini numai declanșatoare de tip AFTER și INSTEAD OF la nivel de comenzi, iar pentru tabele virtuale declanșatoare de tip INSTEAD OF⁵. Să începem discuția cu un caz simplu, cel al trigger-ului pentru inserarea de înregistrări în tabela FACTURI – vezi listing 17.27.

Listing 17.27. Prima versiune a declanșatorului INSERT pentru tabela FACTURI

```
CREATE TRIGGER trg_facturi_ins ON facturi FOR insert  
AS  
    UPDATE facturi SET ValTotala = 0, TVA = 0, ValIncasata = 0  
    WHERE NrFact = (SELECT NrFact FROM inserted)
```

Poate cea mai importantă deosebire T-SQL ține de faptul că aici declanșatoarele nu sunt la nivel de linie, ci la nivel de comandă. De aceea, nu avem posibilitatea de a folosi opțiunile OLD și NEW pentru vechile și noile valori. Există însă două tabele virtuale în care SQL Server stochează toate liniile care ne interesează. Prima tabelă logică (după titulatura documentației SQL Server) virtuală este INSERTED și conține toate liniile inserate, sau – atenție – liniile modificate (valorile noi rezultate după modificare). A doua este DELETED și conține liniile șterse sau liniile vechi (dinaintea modificărilor). Prin urmare, tabela INSERTED poate fi folosită la inserări și modificări, iar DELETED la ștergeri și modificări.

Testăm funcționabilitatea declanșatorului:

⁵ Ne interesează doar declanșatoarele T-SQL. Potrivit documentației SQL Server, pot fi create și alte trigger, sub forma unor metode implementate în .NET

```
INSERT INTO facturi VALUES (3500, GETDATE(), 1002, NULL, 33333,
    NULL, NULL, NULL, NULL)
```

```
SELECT * FROM facturi WHERE NrFact = 3500
```

Figura 17.11 conține linia inserată în FACTURI prin comanda INSERT anterioară și confirmă funcționarea acestei prime versiuni a *trg_facturi_ins*.

The screenshot shows a SQL query window titled 'LAPTOPD.SQLENT...QLQuery1.sql*' with a 'Summary' tab. The query contains two statements: an INSERT INTO facturi VALUES statement and a SELECT * FROM facturi WHERE NrFact = 3500 statement. Below the query window, the 'Results' tab is active, displaying a table with 10 columns: NrFact, DataFact, CodCl, Obs, ValTotala, TVA, ValIncasata, Reduceri, and Penalizari. The first row of data shows NrFact 3500, DataFact 2008-12-17 10:22:00, CodCl 1002, Obs NULL, ValTotala 0.00, TVA 0.00, ValIncasata 0.00, Reduceri NULL, and Penalizari NULL.

	NrFact	DataFact	CodCl	Obs	ValTotala	TVA	ValIncasata	Reduceri	Penalizari
1	3500	2008-12-17 10:22:00	1002	NULL	0.00	0.00	0.00	NULL	NULL

Figura 17.11. Verificarea funcționării declanșatorului INSERT pentru tabela FACTURI

Continuăm discuția cu declanșatorul de inserare (de înregistrări) în tabela LINIIFACT - *trg liniifact_ins* - prezentat în listing 17.28. După tipicul paragrafelor anterioare, acest declanșator trebuie să calculeze valoarea atributului TVAlinie pentru fiecare înregistrare inserată, să crească valoarea totală și tva-ul facturii în care se inserează fiecare linie și, în plus, să aibă grijă ca liniile inserate în fiecare factură să fie numerotate consecutiv.

Listing 17.28. Declanșatorul INSERT pentru tabela LINIIFACT

```
CREATE TRIGGER trg_linifact_ins ON liniifact FOR insert
AS
    UPDATE liniifact
    SET TVAlinie = (
        SELECT COALESCE(Cantitate,0) * COALESCE(PretUnit,0) *
            dbo.f_proctva(CodPr)
        FROM inserted WHERE NrFact=liniifact.NrFact AND Linie=liniifact.Linie)
    WHERE CAST(NrFact AS CHAR(8)) + CAST(Linie AS CHAR(2)) =
        (SELECT CAST(NrFact AS CHAR(8)) + CAST (Linie AS CHAR(2))
        FROM inserted)

    UPDATE facturi
    SET TVA = (SELECT SUM(TVAlinie) FROM liniifact WHERE NrFact = facturi.NrFact),
        ValTotala = (SELECT SUM(Cantitate * PretUnit + TVAlinie) FROM liniifact
            WHERE NrFact = facturi.NrFact)
    WHERE NrFact IN (SELECT NrFact FROM inserted)

    DECLARE @i TINYINT ;
    DECLARE @linie TINYINT ;
    DECLARE @nrfact INT ;

    DECLARE c_facturi CURSOR FOR
        SELECT DISTINCT NrFact FROM inserted ORDER BY NrFact
    OPEN c_facturi
    FETCH NEXT FROM c_facturi INTO @nrfact
    WHILE @@FETCH_STATUS = 0 BEGIN
        SET @i = (SELECT COALESCE(MAX(Linie),0) FROM liniifact
            WHERE NrFact=@nrfact AND Linie NOT IN (
```

```

        ) + 1
    DECLARE c_linii CURSOR FOR
        SELECT Linie FROM inserted WHERE NrFact = @nrfact ORDER BY Linie
    OPEN c_linii
    FETCH NEXT FROM c_linii INTO @linie
    WHILE @@FETCH_STATUS = 0 BEGIN
        UPDATE liniifact SET Linie = @i WHERE NrFact=@nrfact AND Linie=@linie
        SET @i = @i + 1
        FETCH NEXT FROM c_linii INTO @linie
    END
    CLOSE c_linii
    FETCH NEXT FROM c_facturi INTO @nrfact
END
DEALLOCATE c_linii
CLOSE c_facturi
DEALLOCATE c_facturi

```

Am ținut cont de faptul că tabela virtuală INSERTED poate avea mai multe linii (comanda a avut forma *INSERT INTO liniifact SELECT ... FROM ...*), de aceea primele comenzi UPDATE folosesc clauzele WHERE în forma prezentată, iar pentru numerotare este necesară folosirea unui cursor – *c_facturi* - care avea câte o înregistrare pentru fiecare factură inserată. Fiecărei facturi în care s-a adăugat măcar o linie i se (re)numerotează toate liniile, folosindu-se în acest scop un al doilea cursor – *c_linii*. Iată și verificarea (vezi și figura 17.12).

LAPTOPD.SQLE20...QLQuery1.sql* Summary

```

INSERT INTO liniifact VALUES (3500, 1, 2, 200, 100, 8989989) ;
INSERT INTO liniifact VALUES (3500, 4, 3, 100, 100, 0) ;

SELECT * FROM liniifact WHERE NrFact = 3500

SELECT * FROM facturi WHERE NrFact = 3500

```

Results Messages

	NrFact	Linie	CodPr	Cantitate	PretUnit	TVAlinie
1	3500	1	2	200	100.00	1800.00
2	3500	2	3	100	100.00	1900.00

	NrFact	DataFact	CodCl	Obs	ValTotala	TVA	ValIncasata	Reduceri	Penalizari
1	3500	2008-12-17 10:22:00	1002	NULL	33700.00	3700.00	0.00	NULL	NULL

Figura 17.12. Verificarea funcționării declanșatorului INSERT pentru tabela LINIIFACT

```

INSERT INTO liniifact VALUES (3500, 1, 2, 200, 100, 8989989) ;
INSERT INTO liniifact VALUES (3500, 4, 3, 100, 100, 0) ;
SELECT * FROM liniifact WHERE NrFact = 3500
SELECT * FROM facturi WHERE NrFact = 3500

```

Chiar dacă a doua comandă INSERT a încercat să atribuie celei de-a doua linii numărul 4, trigger-ul și-a făcut datoria și a asigurat numerotarea corectă.

Declanșatorul de ștergere de-acum este unul banal – vezi listing 17.29. Trebuie recalculate valorile atributelor TVA și ValTotala din tabela FACTURI și renumerate liniile în care s-a șters măcar o linie. Firește, de data aceasta primadona este tabela virtuală DELETED în care sunt înregistrate toate liniile șterse.

Listing 17.29. Declanșatorul DELETE pentru tabela LINIIFACT

```
CREATE TRIGGER trg_liniiifact_del ON liniiifact FOR delete
AS
    UPDATE facturi
    SET TVA = (SELECT SUM(TVAlinie) FROM liniiifact WHERE NrFact = facturi.NrFact),
        ValTotala = (SELECT SUM(Cantitate * PretUnit + TVAlinie)
                     FROM liniiifact WHERE NrFact = facturi.NrFact)
    WHERE NrFact IN (SELECT NrFact FROM deleted)

    DECLARE @i TINYINT ;
    DECLARE @linie TINYINT ;
    DECLARE @nrfact INT ;

    DECLARE c_facturi CURSOR FOR
        SELECT DISTINCT NrFact FROM deleted ORDER BY NrFact
    OPEN c_facturi
    FETCH NEXT FROM c_facturi INTO @nrfact
    WHILE @@FETCH_STATUS = 0 BEGIN
        SET @i = 1
        DECLARE c_linii CURSOR FOR
            SELECT Linie FROM liniiifact WHERE NrFact = @nrfact ORDER BY Linie
        OPEN c_linii
        FETCH NEXT FROM c_linii INTO @linie
        WHILE @@FETCH_STATUS = 0 BEGIN
            UPDATE liniiifact SET Linie = @i WHERE NrFact = @nrfact AND Linie = @linie
            SET @i = @i + 1
            FETCH NEXT FROM c_linii INTO @linie
        END
        CLOSE c_linii
        FETCH NEXT FROM c_facturi INTO @nrfact
    END
    DEALLOCATE c_linii
    CLOSE c_facturi
    DEALLOCATE c_facturi
```

Înainte de a verifica modul în care se comportă declanșatorul, îl redactăm și pe cel de modificare – vezi listing 17.30 -, folosind deopotrivă tabelele virtuale INSERTED și DELETED. Avem și două premiere. Mai întâi, vrem să interzicem modificarea directă (printr-o comandă UPDATE) a valorilor atributelor Linie și TVAlinie. Pentru aceasta ne vom folosi de funcția *UPDATE()* care furnizează TRUE dacă valoarea atributului-argument a fost modificată și FALSE în caz contrar, și de funcția *TRIGGER_NESTLEVEL()* care se incrementează cu valoarea 1 la fiecare declanșator lansat (care poate lansa, la rândul său, un alt declanșator și, astfel, se mai crește cu 1 valoarea furnizată de funcție). Dacă modificarea s-ar face direct, atunci valoarea furnizată de funcție ar fi 1.

Listing 17.30. Declanșatorul UPDATE pentru tabela LINIIFACT

```

CREATE TRIGGER trg_liniiifact_upd ON liniiifact FOR update AS
IF (UPDATE(Linie) OR UPDATE(TVAlinie) ) AND (SELECT TRIGGER_NESTLEVEL() ) <= 1
BEGIN
    RAISERROR('Nu aveti voie sa modificati interactiv valorile atributelor Linie si TVAlinie',16, 1)
    ROLLBACK TRANSACTION
END

IF NOT UPDATE(Linie)
BEGIN
    PRINT TRIGGER_NESTLEVEL()
    UPDATE facturi
    SET TVA = (SELECT SUM(TVAlinie) FROM liniiifact WHERE NrFact = facturi.NrFact),
        ValTotala = ( SELECT SUM(Cantitate * PretUnit + TVAlinie)
                     FROM liniiifact WHERE NrFact = facturi.NrFact)
    WHERE NrFact IN (SELECT NrFact FROM deleted UNION
                    SELECT NrFact FROM DELETED )

    DECLARE @i TINYINT ;
    DECLARE @linie TINYINT ;
    DECLARE @nrfact INT ;
    DECLARE c_facturi CURSOR FOR
        SELECT DISTINCT NrFact FROM inserted UNION
        SELECT DISTINCT NrFact FROM deleted ORDER BY NrFact
    OPEN c_facturi
    FETCH NEXT FROM c_facturi INTO @nrfact
    WHILE @@FETCH_STATUS = 0 BEGIN
        PRINT @nrfact
        SET @i = 1
        DECLARE c_linii2 CURSOR FOR
            SELECT Linie FROM liniiifact WHERE NrFact = @nrfact ORDER BY Linie
        OPEN c_linii2
        FETCH NEXT FROM c_linii2 INTO @linie
        WHILE @@FETCH_STATUS = 0
        BEGIN
            PRINT @linie
            UPDATE liniiifact SET Linie = @i WHERE NrFact=@nrfact AND Linie=@linie
            SET @i = @i + 1
            FETCH NEXT FROM c_linii2 INTO @linie
        END
        CLOSE c_linii2
        DEALLOCATE c_linii2
        FETCH NEXT FROM c_facturi INTO @nrfact
    END
    CLOSE c_facturi
    DEALLOCATE c_facturi
END

```

De asemenea, mai luăm în calcul faptul că renumerotarea liniilor, efectuată de acest declanșator sau de *trg_liniiifact_del*, ar lansa, recursiv, *trg_liniiifact_upd*. De aceea, toate prelucrările din acest declanșator nu vor fi efectuate dacă singura valoare (sau valori) modificată ar fi cea a atributului Linie. Cursorul *c_facturi* va fi populat cu numerele facturilor care au suferit modificări, numere obținute prin reunirea valorilor NrFact din tabelele INSERTED și DELETED.

Pentru o mini-testare a declanșatoarelor, să ne concentrăm pe factura cu numărul 7028 (vezi figura 17.13):

```
SELECT * FROM liniifact WHERE NrFact = 7028
```

```
SELECT * FROM facturi WHERE NrFact = 7028
```

```
SELECT * FROM liniifact WHERE NrFact = 7028
|
SELECT * FROM facturi WHERE NrFact = 7028
```

	NrFact	Linie	CodPr	Cantitate	PretUnit	TVAlinie
1	7028	1	2	35	1090.00	3433.50
2	7028	2	3	40	700.00	5320.00
3	7028	3	4	55	1410.00	6979.50
4	7028	4	5	755	6300.00	903735.00

	NrFact	DataFact	CodCl	Obs	ValTotala	TVA	ValIncasata	Reduceri	Penalizari
1	7028	2007-12-21 00:00:00	1003	NULL	5819668.00	NULL	NULL	NULL	NULL

Figura 17.13. Conținutul „inițial” al facturii 7028

Factura are patru linii în care apar produsele 2, 3, 4 și 5. Le ștergem pe prima și a treia:

```
DELETE FROM liniifact WHERE NrFact = 7028 AND Linie IN (1,3)
```

După cum o demonstrează factura 17.14, declanșatorul de ștergere recalculează valoarea totală și TVA-ul (în tabela FACTURI) și renumerează corect cele două linii rămase.

```
SELECT * FROM liniifact WHERE NrFact = 7028
SELECT * FROM facturi WHERE NrFact = 7028

BEGIN TRAN
DELETE FROM liniifact WHERE NrFact = 7028 AND Linie IN (1,3)

SELECT * FROM liniifact WHERE NrFact = 7028
SELECT * FROM facturi WHERE NrFact = 7028
```

	NrFact	Linie	CodPr	Cantitate	PretUnit	TVAlinie
1	7028	1	3	40	700.00	5320.00
2	7028	2	5	755	6300.00	903735.00

	NrFact	DataFact	CodCl	Obs	ValTotala	TVA	ValIncasata	Reduceri	Penalizari
1	7028	2007-12-21 00:00:00	1003	NULL	5693555.00	909055.00	NULL	NULL	NULL

Figura 17.14. Conținutul facturii 7028 după ștergerea liniilor 1 și 3

Verificăm și vigilența declanșatorului *trg liniifact_upd* încercând să modificăm „direct” numărul primei linii din factura 7028:

UPDATE liniifact SET Linie = 6 WHERE NrFact = 7028 AND Linie = 1

Spre meritul lui, trigger-ul se opune acestei samavolnicii – vezi figura 17.15.

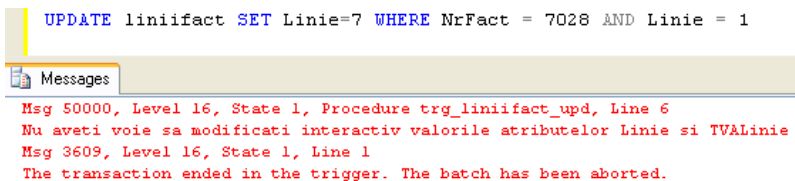


Figura 17.15. Blocarea modificării interactive a valorii atributului Linie din LINIIFACT

Piatra de încercare a declanșatorului o reprezintă modul de comportare la trecerea unei linii dintr-o factură în alta. Să examinăm situația inițială a facturilor 7013 și 7014 – vezi figura 17.16.

```
SELECT * FROM facturi WHERE NrFact IN (7013, 7014)
SELECT * FROM liniifact WHERE NrFact IN (7013, 7014) ORDER BY 1,2
```

	NrFact	DataFact	CodCl	Obs	ValTotala	TVA	ValIncasata	Reduceri	Penalizari
1	7013	2007-12-07 00:00:00	1003	NULL	5774498.50	912848.50	NULL	NULL	NULL
2	7014	2007-12-07 00:00:00	1001	NULL	97664.00	8064.00	NULL	NULL	NULL

	NrFact	Linie	CodPr	Cantitate	PretUnit	TVAlinie
1	7013	1	2	35	1090.00	3433.50
2	7013	2	3	40	700.00	5320.00
3	7013	3	4	50	1410.00	6345.00
4	7013	4	5	750	6300.00	897750.00
5	7014	1	2	80	1120.00	8064.00

Figura 17.16. Situația de pornire a facturilor 7013 și 7014

Prima are patru linii în care apar produsele 2, 3, 4 și 5, iar a doua doar o linie ce indică vânzarea produsului 2. Încercăm să trecem a treia linie din factura 7013 în factura 7014. Comanda nu ar fi complicată:

UPDATE liniifact

SET NrFact=7014, Cantitate=500

WHERE NrFact=7013 AND Linie=3

Ceea ce așteptăm noi de la declanșator este recalcularea valorii totale și a tva pentru cele două facturi, dar și renumerotarea liniilor. Figura 17.17 demonstrează că lucrurile au decurs conform planului.

```

BEGIN TRAN
UPDATE liniifact SET NrFact=7014, Cantitate=500 WHERE NrFact=7013 AND Linie=3

SELECT * FROM facturi WHERE NrFact IN (7013, 7014)
SELECT * FROM liniifact WHERE NrFact IN (7013, 7014) ORDER BY 1,2

```

	NrFact	DataFact	CodCl	Obs	ValTotala	TVA	ValIncasata	Reduceri	Penalizari
1	7013	2007-12-07 00:00:00	1003	NULL	5697653.50	906503.50	NULL	NULL	NULL
2	7014	2007-12-07 00:00:00	1001	NULL	97664.00	8064.00	NULL	NULL	NULL

	NrFact	Linie	CodPr	Cantitate	PretUnit	TVALinie
1	7013	1	2	35	1090.00	3433.50
2	7013	2	3	40	700.00	5320.00
3	7013	3	5	750	6300.00	897750.00
4	7014	1	2	80	1120.00	8064.00
5	7014	2	4	500	1410.00	6345.00

Figura 17.17. Situația facturilor 7013 și 7014 după mutarea unei linii

Încheiem cu un declanșator de tip INSTEAD OF. Folosim același exemplu cu tabela virtuală vFacturi:

```

CREATE VIEW vFacturi AS
SELECT lf.NrFact, DataFact, DenCl, f.CodCl, Linie, lf.CodPr, DenPr,
       UM, ProcTVA, Grupa, Cantitate, PretUnit,
       Cantitate * PretUnit AS ValFaraTVALinie,
       Cantitate * PretUnit * ProcTVA AS TVALinie,
       Cantitate * PretUnit * (1+ProcTVA) AS ValTotLinie
FROM liniifact lf
      INNER JOIN produse p ON lf.CodPr=p.CodPr
      INNER JOIN facturi f ON lf.NrFact=f.NrFact
      INNER JOIN clienti c ON f.CodCl=c.CodCl

```

Spre deosebire de PL/pgSQL și PL/SQL, în SQL Server clauza ORDER BY nu poate apărea în interogarea-definiție a tabeli virtuale. Declanșatorul INSTEAD OF din listing 17.31 este croit după logica tabeli „logice” INSERT.

Listing 17.31. Declanșatorul UPDATE pentru tabela LINIIFACT

```

CREATE TRIGGER trg_vFacturi_oi ON vFacturi INSTEAD OF INSERT
AS
INSERT INTO facturi (NrFact, DataFact, CodCl)
SELECT NrFact, DataFact, CodCl
FROM inserted
WHERE NrFact NOT IN (SELECT NrFact FROM facturi)

INSERT INTO produse (DenPr, UM, Grupa, ProcTVA)
SELECT DenPr, UM, Grupa, ProcTVA
FROM inserted
WHERE CodPr NOT IN (SELECT CodPr FROM produse)

INSERT INTO liniifact

```

```

SELECT NrFact, Linie, (SELECT CodPr FROM produse WHERE DenPr = ins.DenPr),
      Cantitate, PretUnit, NULL
FROM inserted ins
WHERE CAST(NrFact AS CHAR(8)) + CAST (Linie AS CHAR(2))
      NOT IN (SELECT CAST(NrFact AS CHAR(8)) + CAST (Linie AS CHAR(2))
              FROM liniifact)

```

Verificarea acestuia se realizează cu aceeași comandă de inserare, urmată de două interogări necesare vizualizării (vezi figura 17.18)

```

INSERT INTO vFacturi (NrFact, Linie, CodPr, Cantitate, PretUnit, DataFact,
                    CodCl, DenPr, UM, Grupa, ProcTVA)
VALUES (3120, 1, 8, 400, 1200, '2007-10-01', 1001, 'Produs 8', 'buc', 'Cafea', 0.19) ;

```

```
SELECT * FROM produse WHERE CodPr = 8
```

```
SELECT * FROM liniifact WHERE NrFact = 3120
```

```
SELECT * FROM facturi WHERE NrFact = 3120
```

```

INSERT INTO vFacturi (NrFact, Linie, CodPr, Cantitate, PretUnit, DataFact,
                    CodCl, DenPr, UM, Grupa, ProcTVA)
VALUES (3120, 1, 8, 400, 1200, '2007-10-01', 1001, 'Produs 8',
      'buc', 'Cafea', 0.19) ;

SELECT * FROM produse WHERE CodPr = 8
SELECT * FROM liniifact WHERE NrFact = 3120
SELECT * FROM facturi WHERE NrFact = 3120

```

	CodPr	DenPr	UM	Grupa	ProcTVA
1	8	Produs 8	buc	Cafea	0.19

	NrFact	Linie	CodPr	Cantitate	PretUnit	TVAlinie
1	3120	1	8	400	1200.00	91200.00

	NrFact	DataFact	CodCl	Obs	ValTotala	TVA	ValIncasata	Reduceri	Penalizari
1	3120	2007-10-01 00:00:00	1001	NULL	571200.00	91200.00	0.00	0.00	0.00

Figura 17.18. Verificarea declanșatorului INSTEAD OF

17.5. Declanșatoare în SQL PL

Tipologia declanșatoarelor este comparabilă, ca nivel de detaliere, cu cea din Oracle. Ca și în T-SQL, trecem peste secvența și declanșatorul necesare cheilor surogat (cum a fost CodCl din CLIEȚI pe care l-am tratat în PL/SQL), întrucât pentru toate cheile surogat am folosit în capitolul 3 opțiunea GENERATED

ALWAYS. Poposim, deci, la trigger-ul de inserare în tabela FACTURI. Iată în listing-ul 17.32 sintaxa SQL PL pentru acesta.

Listing 17.32. Prima versiune a declanșatorului SQL PL de inserare în FACTURI

```
CREATE TRIGGER trg_facturi_ins
NO CASCADE BEFORE INSERT ON facturi
REFERENCING NEW AS n
FOR EACH ROW
MODE DB2SQL
vs: BEGIN ATOMIC
      SET n.TVA = 0 ;
      SET n.ValTotala = 0 ;
      SET n.ValIncasata = 0 ;
END vs
```

Pentru comparație cu celelalte paragrafe, folosim același INSERT:

```
INSERT INTO facturi VALUES (3500, CURRENT_DATE, 1002, NULL,
                             123456, 234567, 345678, NULL, NULL)
```

Similar Oracle, suntem obligați să redactăm declanșatorul care să țină loc de ON UPDATE CASCADE – vezi listing 17.33.

Listing 17.33. Declanșator SQL PL pentru substituția opțiunii ON UPDATE CASCADE

```
CREATE TRIGGER trg_facturi_upd_a_r_nrfect
AFTER UPDATE OF NRFACT ON facturi
REFERENCING OLD AS OLD NEW AS NEW FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
UPDATE liniifact SET NrFact = NEW.NrFact WHERE NrFact = OLD.NrFact ;
UPDATE incasfact SET NrFact = NEW.NrFact WHERE NrFact = OLD.NrFact ;
END
```

Pentru declanșatorul dedicat inserării unei înregistrări în tabela LINIIFACT apar câteva modificări. Nu putem proceda ca în PL/SQL unde aveam un singur declanșator BEFORE ROW în care modificam valorile atributelor Linie și TVALinie din LINIIFACT și incrementam ValTotala și TVA în FACTURI. SQL PL ne obligă să mutăm actualizarea atributelor calculate în declanșatorul AFTER ROW. Așa că vom avea două declanșatoare pentru inserare *trg_liniiifact_ins1* și *trg_liniiifact_ins2* - vezi listing 17.34.

Listing 17.34. Două declanșatoare SQL PL pentru inserarea unei înregistrări în LINIIFACT

```
-- declansatorul la nivel de linie de tipul BEFORE
CREATE TRIGGER trg_liniiifact_ins1 NO CASCADE
BEFORE INSERT ON liniifact
REFERENCING NEW AS NEW FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
  DECLARE v_tvalinie DECIMAL(9,2) ;
  SET v_tvalinie = COALESCE(NEW.cantitate,0) * COALESCE(NEW.pretunit,0) *
    f_proctva(CAST (NEW.codpr AS SMALLINT)) ;
  SET NEW.tvalinie = v_tvalinie ;
  SET NEW.linie = (SELECT COALESCE(MAX(Linie),0) + 1
```

```

                FROM liniifact
                WHERE NrFact = NEW.NrFact) ;

END

-- declansatorul la nivel de linie de tipul AFTER
CREATE TRIGGER TRG_LINIIFACTURI_INS2
AFTER INSERT ON liniifact
REFERENCING NEW AS NEW FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
    UPDATE facturi
    SET tva = COALESCE(tva,0) + NEW.tvalinie,
        valtotala = COALESCE(valtotala,0) + NEW.cantitate * NEW.pretunit + NEW.tvalinie
    WHERE NrFact = NEW.NrFact ;
END

```

Testăm aceste două declanșatoare cu aceleași două comenzi INSERT din paragrafele anterioare :

```
INSERT INTO liniifact VALUES (3500, 1, 2, 200, 100, 8989989) ;
```

```
INSERT INTO liniifact VALUES (3500, 2, 3, 100, 100, 0) ;
```

„Portăm” acum declanșatorul de ștergere pentru LINIIFACT în varianta sa rigidă, de interzicere a ștergerii unei alte linii decât ultima pentru o factură – listing 17.35. Declanșarea erorii se realizează prin comanda SIGNAL.

Listing 17.35. Declanșatorul SQL PL pentru ștergerea unei înregistrări din LINIIFACT

```

CREATE TRIGGER trg_linifact_del
AFTER DELETE ON liniifact
REFERENCING OLD AS OLD FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
    DECLARE v_nrlinii SMALLINT ;
    SET v_nrlinii = (SELECT MAX(linie) FROM liniifact WHERE nrfact = OLD.nrfact) ;
    IF (OLD.linie <> COALESCE(v_nrlinii,0)) THEN
        SIGNAL SQLSTATE '80000'
        SET MESSAGE_TEXT='Nu puteti sterge decit ultima linie dintr-o factura' ;
    END IF ;
    UPDATE facturi
    SET tva = tva - OLD.tvalinie,
        valtotala = valtotala - OLD.cantitate * OLD.pretunit - OLD.tvalinie
    WHERE nrfact = OLD.nrfact ;
END

```

Spre deosebire de Oracle, în DB2 la verificarea declanșatorului prin comanda:

```
DELETE FROM liniifact WHERE NrFact=3500 AND linie=1
```

nu se declanșează eroarea de mutanță a tabelului, ci eroarea datorată ștergerii dintr-o factură a altei linii decât ultima (figura 17.19).

```
DELETE FROM liniifact WHERE NrFact=3500 AND linie=1
```

```
1 row(s) returned successfully.
```

```
----- Commands Entered -----  
DELETE FROM liniifact WHERE NrFact=3500 AND linie=1;
```

```
DELETE FROM liniifact WHERE NrFact=3500 AND linie=1  
DB21034E The command was processed as an SQL statement because it was not a  
valid Command Line Processor command. During SQL processing it returned:  
SQL0438N Application raised error with diagnostic text: "Nu puteti sterge  
deci ultima linie dintr-o factura". SQLSTATE=80000
```

Figura 17.19. Funcționarea primei versiuni a declanșatorului de ștergere în LINIIFACT

Varianta după care în facturile în care s-a șters măcar o linie se numerotează automat liniile rămase este mult mai puțin traumatizantă în DB2 SQL PL, prin comparație cu Oracle PL/SQL. Nu avem nevoie de variabile publice, ci doar de două declanșatoare, unul la nivel de linie, *trg_liniiifact_del*, care să decrementeze valoarea celor două attribute calculate din FACTURI și unul la nivel de comandă, *trg_liniiifact_del2*, care renumerotează liniile – vezi listing 17.36 (mai întâi ștergem *trg_liniiifact_del* și apoi lansăm scriptul).

Listing 17.36. Două declanșatoare SQL PL de ștergere în LINIIFACT

```
CREATE TRIGGER trg_liniiifact_del  
AFTER DELETE ON liniifact  
REFERENCING OLD AS OLD FOR EACH ROW MODE DB2SQL  
BEGIN ATOMIC  
    UPDATE facturi  
    SET tva = tva - OLD.tvalinie,  
        valtotala = valtotala - OLD.cantitate * OLD.pretunit - OLD.tvalinie  
    WHERE nrfact = OLD.nrfact ;  
END  
  
-----  
CREATE TRIGGER trg_liniiifact_del2  
AFTER DELETE ON liniifact  
REFERENCING OLD TABLE AS oldtab  
FOR EACH STATEMENT MODE DB2SQL  
BEGIN ATOMIC  
    DECLARE i SMALLINT ;  
    FOR rec1 AS SELECT DISTINCT NrFact FROM oldtab DO  
        SET i = 1 ;  
        FOR rec_linii AS SELECT * FROM liniifact WHERE NrFact = rec1.NrFact ORDER BY Linie DO  
            UPDATE liniifact SET linie = i  
            WHERE NrFact = rec_linii.NrFact AND linie=rec_linii.linie ;  
            SET i = i + 1 ;  
        END FOR ;  
    END FOR ;  
END
```

În schimb, protejarea atributelor calculate din tabela FACTURI de modificările neautorizate reprezintă o sarcină mult mai dificilă. Există în SQL PL variabile publice ce pot fi gestionate la nivel de sesiune de lucru, sintaxa fiind una simplă:

```
CREATE VARIABLE v_trg_liniifact CHAR(1) DEFAULT NULL
```

Din păcate, variabilele publice nu pot fi modificate în declanșatoare, așa că nu ne sunt de folos. Soluția tabelor temporare (globale) reclamă cunoștințe de SQL dinamic, și abia capitolul următor o să încercăm o soluție.

Așa că singura variantă ar fi să creăm o tabelă în care fiecărei variabile publice și utilizator să le alocăm câte o linie (înregistrare), urmând ca declanșatoarele tabelor LINIIFACT și FACTURI să lucreze cu această tabelă substituit al variabilelor publice.

Așa că încheiem paragraful cu exemplul de declanșatoare din categoria INSTEAD OF. Comanda CREATE VIEW nu acceptă în DB2 clauza ORDER BY, așa că sintaxa comenzii CREATE VIEW pentru vFacturi este:

```
CREATE VIEW vFacturi AS
SELECT lf.NrFact, DataFact, DenCl, f.CodCl, Linie, lf.CodPr, DenPr,
UM, ProcTVA, Grupa,
Cantitate, PretUnit,
Cantitate * PretUnit AS ValFaraTVALinie,
Cantitate * PretUnit * ProcTVA AS TVALinie,
Cantitate * PretUnit * (1+ProcTVA) AS ValTotLinie
FROM liniifact lf
INNER JOIN produse p ON lf.CodPr=p.CodPr
INNER JOIN facturi f ON lf.NrFact=f.NrFact
INNER JOIN clienti c ON f.CodCl=c.CodCl
```

Față de versiunea PL/SQL, în afara diferențelor de sintaxă, în redactarea declanșatorului INSTEAD OF INSERT în DB2 SQL PL trebuie avut în vedere că atributul CodPr din tabela PRODUSE are valori generate automat, deci în afara oricărui control. Ne bazăm pe faptul că deumirea produsului, deși nu este cheie alternativă, nu se repetă în tabela PRODUSE - vezi listing 17.37.

Listing 17.37. Declanșatorul INSTEAD OF INSERT

```
CREATE TRIGGER trg_vFacturi_ioi
INSTEAD OF INSERT ON vFacturi
REFERENCING NEW AS NEW FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
INSERT INTO facturi (NrFact, DataFact, CodCl)
SELECT NEW.NrFact, NEW.DataFact, NEW.CodCl
FROM sysibm.dual
WHERE NEW.NrFact NOT IN (SELECT NrFact FROM facturi) ;

INSERT INTO produse (DenPr, UM, Grupa, ProcTVA)
SELECT NEW.DenPr, NEW.UM, NEW.Grupa, NEW.ProcTVA
```

```

FROM sysibm.dual
WHERE NEW.CodPr NOT IN (SELECT CodPr FROM produse) ;

INSERT INTO liniifact VALUES (NEW.NrFact, NEW.Linie,
    (SELECT CodPr FROM produse WHERE DenPr=NEW.DenPr) ,
    NEW.Cantitate,
    NEW.PretUnit, NULL) ;
END

```

Chiar dacă în INSERT-ul de mai jos codul produsului specificat este 8, în tabelele LINIIFACT și PRODUSE a fost inserat codul 22 – vezi figura 17.20:

```

INSERT INTO vFacturi (NrFact, Linie, CodPr, Cantitate, PretUnit, DataFact,
    CodCl, DenPr, UM, Grupa, ProcTVA)
VALUES (3120, 1, 8, 400, 1200, '2007-10-01', 1001, 'Produs 8', 'buc', 'Cafea', 0.19) ;
SELECT * FROM produse WHERE CodPr = 8 ;
SELECT * FROM liniifact WHERE NrFact = 3120 ;
SELECT * FROM facturi WHERE NrFact = 3120 ;

```

```
-----
SELECT * FROM liniifact WHERE NrFact = 3120

```

NRFACT	LINIE	CODPR	CANTITATE	PRETUNIT	TVALINIE
3120.	1	23.	400.	1200.00	91200.00

1 record(s) selected.

```
SELECT * FROM produse

```

CODPR	DENPR	UM	GRUPA	PROCTVA
1	Produs 1	buc	Tigari	.19
2	Produs 2	kg	Bere	.09
3	Produs 3	kg	Bere	.19
4	Produs 4	l	Dulciuri	.09
5	Produs 5	buc	Tigari	.19
6	Produs 6	p250g	Cafea	.19
23	Produs 8	buc	Cafea	.19

7 record(s) selected.

Figura 17.20. Testarea declanșatorului INSTEAD OF vFacturi