

Capitolul 11. Declanșatoare

Declanșatoarele (trigger-e, în originalul englezesc) reprezintă un tip deosebit 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 operațiunile pentru care au fost definite. Inițial, erau vizate doar trei acțiuni: inserare, modificare și ștergere de linii. La această familie a declanșatoarelor, unele SGBD-uri au mai adăugat și alte tipuri, asociate, spre exemplu, actualizărilor tabelelor derivate, conectării unei aplicații/utilizator, deschiderea și închiderea instanței baze de date etc.

Cele mai importante avantaje ale declanșatoarelor sunt:

- permit instituirea unor reguli de validare cu mult mai complexe decât ceea ce permite clauza 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;
- pot institui un mecanism riguros de jurnalizare a modificărilor bazei.

11.1. Tipologia declanșatoarelor Oracle

Aria de utilizarea a triggerelor în Oracle este cât se poate de extinsă: restricții complexe, asigurarea integrității BD, îmbunătățirea securității, respectarea integrității referențiale între nodurile unei BD distribuite, jurnalizarea operațiunilor (tranzacțiilor), sincronizarea replicilor unei BD etc.

Totuși, documentația Oracle 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, SET DEFAULT etc. În plus, tot Application Developer's Guide specifică un număr maxim de linii ce ar trebui să alcătuiască un trigger - 60, la nevoie fiind indicată crearea unor proceduri stocate apelabile din trigger.

11.1.1. Clasificare

În Oracle există 15 tipuri diferite de declanșatoare DML - vezi tabelul 11.1. Odată cu versiunea 8i, în Oracle au mai fost introduse o serie de declanșatoare lansabile în execuție la deschiderea instanței bazei, închiderea instanței bazei,

conectarea și deconectarea utilizatorilor/aplicațiilor etc. pe care o să le discutăm spre finalul capitolului.

Tabel 8.1. Tipologia “clasică” a declanșatoarelor DML Oracle

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

11.1.2. 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 singură dată pentru o comandă de actualizare, indiferent de câte linii sunt afectate. Să luăm, spre exemplu, comanda:

```
UPDATE personal SET marca = marca + 100 WHERE marca > 1100
```

Presupunând că 50 de angajați au marca mai mare decât 1100, un trigger de modificare la nivel de linie se execută de 50 de ori, în timp ce unul la nivel de comandă o singură dată. Dacă nici un angajat nu are marca mai mare de 1100, un trigger de modificare la nivel de linie nu s-ar putea executa deloc, în timp ce declanșatorul la nivel de comandă este oricum lansat (evident, o singură dată).

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

Un trigger de tip BEFORE intră în acțiune înainte de modificarea propriu-zisă, și 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.). Declanșatorul de tip AFTER se execută după momentul producerii actualizării, servind la verificarea corectitudinii operațiunii (nu a fost scoasă din magazie o cantitate de material mai mare decât stocul dinaintea operațiunii), inserarea unei linii într-o tabelă de tip jurnal (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.

Declanșatoarele la nivel de linie/comadă pot fi combinate cu cele de tip BEFORE/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).

11.1.4. Trigger de tip în-loc-de (INSTEAD OF)

Acest gen de declanșator constituie un excelent mijloc de propagare a modificărilor dintr-o tabelă derivată în tabelele de bază din care provine. Amânăm această discuție pentru capitolul următor.

Un alt element foarte important este că în trigger-ele la nivel de linie, și în cele 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 sau :NEW.

11.2. Declanșatoare pentru generarea valorilor implicite

Nucleul SQL din Oracle nu permite specificarea, nici pentru valorile implicite (clauza DEFAULT din CREATE/ALTER TABLE), nici pentru regulile de validare (clauza CHECK), a funcțiilor utilizator. Ambele probleme își au rezolvarea în folosirea declanșatoarelor.

11.2.1. Valoare implicită calculată printr-o consultare SELECT MAX()...

Spre exemplu, pentru ca un angajat nou să primească drept marcă numărul următor, se poate folosi un declanșator la nivel de linie, prin care înainte inserării noii înregistrări, valoarea atributului *Marca* să fie obținută printr-o frază SQL – vezi listing 11.1.

Listing 11.1. Declanșatorul pentru generarea unei mărci noi - varianta 1

```
CREATE OR REPLACE TRIGGER trg_personal_ins_befo_row
  BEFORE INSERT ON personal
  REFERENCING OLD AS OLD NEW AS NEW
  FOR EACH ROW
BEGIN
  SELECT NVL(MAX(marca),1000) + 1 INTO :NEW.marca FROM personal ;
END ;
```

Soluția pare, și este, cel puțin la prima vedere, corectă. Din tabela *PERSONAL*, declanșatorul extrage valoarea maximă a atributului *Marca* pe care o incrementează și o alocă noii valori a atributului (adică valorii atributului *Marca* de pe linia curentă). Toate declanșatoarele declarate la nivel de linie (*ROW*) au acces la valorile atributelor dinaintea (*BEFORE*) și după actualizarea (*AFTER*) înregistrării, cum ar fi *:NEW.numepren*, *:OLD.compart*. La declanșatoarele corespunzătoare inserării unei înregistrări, valorile “vechi” (*:OLD.atribut*) sunt nule.

Chiar dacă în clauza *VALUES* a comenzii *INSERT*, pentru atributul *Marca* valoarea indicată este alta, urmându-se logica declanșatorului, înregistrarea va conține valoarea generată prin trigger. Spre exemplu, se lansează următorul *INSERT*:

```
INSERT INTO personal (marca, numepren)
VALUES (2111, 'Angajat 2111') ;
```

După această comandă se execută fraza SQL care afișează linia proaspăt introdusă – vezi figura 11.1. Valoarea care ne interesează nu este 2111, cea specificată, ci 1010, cea calculată și atribuită prin declanșator.

```
SQL> INSERT INTO personal (marca, numepren) VALUES (2111, 'Angajat 2111') ;
```

```
1 row created.
```

```
SQL> COMMIT ;
```

```
Commit complete.
```

```
SQL> SELECT * FROM personal WHERE marca IN
2 (SELECT MAX(marca) FROM personal)
3 /
```

MARCA	NUMEPREN	COMPA	DATASU	SALORAR	SALORARCO	C
1010	Angajat 2111	PROD	13-APR-03	75000	75000	N

```
SQL>
```

Figura 11.1. Valoarea mărcii generate prin declanșator

Acest gen de lucru, foarte popular în o serie de SGBD-uri precum Visual FoxPro, nu este prea gustat în lumea Oracle. Motivul principal apare din scenariul următor: la baza de date sunt conectați doi utilizatori, noi și un altul (pe care-l cunoaștem de câteva luni). Fiecare avem deschisă propria sesiune. La momentul t1 celălalt utilizator lansează comanda:

```
INSERT INTO personal (marca, numepren)
VALUES (1111, 'Angajat X') ;
```

Firește, până la “comiterea” tranzacției, nu putem lua act de prezența acestei noi înregistrări. La momentul t2, nefiind mai prejos, lansăm și noi o inserare:

```
INSERT INTO personal (marca, numepren) VALUES
(1111, 'Angajat Y') ;
```

O să constatăm ca operațiunea noastră durează destul de mult. Chiar când eram pe punctul de a ne pierde răbdarea, celălalt utilizatorul lansează COMMIT-ul (acesta este momentul t3). Ei bine, din inserarea noastră nu ne alegem decât cu un mesaj de eroare, ca în figura 11.2.

```
SQL> -- in acest timp un alt utilizator lanseaza comanda INSERT:
SQL> -- INSERT INTO personal (marca, numepren) VALUES (1111, 'Angajat X')
SQL> -- fara a executa si comanda COMMIT
SQL> -- noi lansam comanda INSERT INTO personal (marca, numepren) VALUES (1111, 'Angajat Y');
SQL> -- care va bloca SQL*Plus pina celalt utilizator va "comite" inserarea
SQL> -- celalt utilizator lanseaza COMMIT
SQL> INSERT INTO personal (marca, numepren) VALUES (1111, 'Angajat Y');
INSERT INTO personal (marca, numepren) VALUES (1111, 'Angajat Y')
*
ERROR at line 1:
ORA-00001: unique constraint (FOTACHEM.PK_PERSONAL) violated
```

SQL> |

Figura 11.2. Eșecul declanșatorului la inserarea concurrentă

Care este explicația ? Fraza SELECT din declanșator nu “vede” înregistrările inserate de alți utilizatori până în comentul “comiterii” lor, așa încât, atunci când două sau mai multe sesiuni operează simultan inserări, valorile noilor mărci sunt egale și, astfel, se violează cheia primară. De aceea, cvasi-totalitatea dezvoltatorilor de aplicații recurg la secvențe pentru generarea valorilor implicite ale cheilor surogat.

11.2.2. Soluție bazată pe secvențe

Secvența permite unei aplicații să preia numere consecutive unice pe un anumit interval. Gestiunea secvenței se face la nivelul central; fiecare apel care “cere” o valoare din secvență poate fi absolut sigur că nici un alt apel nu va primi o valoare identică, deci nu vor apărea conflicte. O secvență se creează prin comanda CREATE SEQUENCE. Pentru ca noii angajați să primească mărci unice consecutive, în ordinea preluării în tabelă, se crează secvența SEQ_MARCA:

```
CREATE SEQUENCE seq_marca INCREMENT BY 1
```

```
MINVALUE 1010 MAXVALUE 5555 NOCYCLE NOCACHE ORDER
```

Odată creată, referința la o secvență se face prin `NextVal`, caz în care se obține valoarea următoare a secvenței (incrementarea se face automat), în timp ce valoarea curentă a secvenței se obține prin `CurrVal`. Astfel încât pentru ca un angajat nou să aibă marca imediat superioară ultimei persoane introduse în tabelă, se modifică declanșatorul anterior ca în listing 11.2.

Listing 11.2. Folosirea secvenței în declanșator

```
CREATE OR REPLACE TRIGGER TRG_PERSONAL_INS_BEFO_ROW
  BEFORE INSERT ON PERSONAL
  REFERENCING OLD AS OLD NEW AS NEW
  FOR EACH ROW
BEGIN
  SELECT seq_marca.NEXTVAL INTO :NEW.marca FROM dual ;
END ;
/
```

Grație secvențelor, nu vor exista niciodată situații de violare a cheii primare în tabela `PERSONAL`. Orice apel la secvență prin clauza `NextVal` “mușcă” o valoare din secvență, “consumând-o”. Dincolo de avantaje, există însă și neplăceri. Astfel, dacă un utilizator execută o comandă `INSERT`, dar nu o validează (în loc de `COMMIT` va introduce `ROLLBACK`), valoarea preluată din secvență se pierde. Astfel, deseori, valorile cheilor surrogat prezintă “găuri”. Dacă cerințele firmei impun ca mărcile se fie alocate consecutiv, fără goluri, trebuie imaginată o soluție care să “recicleze” valorile nefolosite.

Completarea “găurilor” din secvențe

Soluția pe care v-o propunem în continuare se bazează pe o funcție inclusă în pachetul `PAKET_SALARIZARE` – a cărei nouă versiune este prezentată în listing 11.3. În semn de respect față de gropile presărate pe străzile Bucureștiului, funcția a fost supranumită `F_PRIMA_GAURĂ_MARCĂ`, iar pentru a respecta adevărul istoric, se cuvine de spus că și lașul începe să devină un concurent redutabil al capitalei în materie de consum de pivoți, bușe, amortizoare și alte delicatessuri *heavy-metal*.

Funcția apelează la un vector public, numit `v_marci` în care sunt stocate toate mărcile alocate, adică valorile atributului `Marcă` din `PERSONAL`. La primul apel al procedurii dintr-o sesiune, se apelează procedura `P_INITIALIZARE_V_MARCI` care “citește” valorile `PERSONAL.Marcă` și le depozitează în tabloul de manevră. De aici mărcile sunt trecute în tabloul `v_marci` pentru care indexul este chiar marca respectivă, ceea ce asigură o viteză de acces ridicată.

Listing 11.3. Noutăți în pachetul `PAKET_SALARIZARE`

```
CREATE OR REPLACE PACKAGE paket_salarizare AS
  -- declarăm un vector pentru stocarea mărcilor alocate
  TYPE t_v_marci IS TABLE OF personal.marca%TYPE INDEX BY PLS_INTEGER ;
  v_marci t_v_marci ;
```

```

PROCEDURE p_initalize_v_marci ;
FUNCTION f_prima_gaura_marca RETURN personal.marca%TYPE ;

-- în continuare, specificațiile pachetului sunt exact ca în Listing 9.30
...
END pachet_salarizare ;
/

CREATE OR REPLACE PACKAGE BODY pachet_salarizare AS

-----
PROCEDURE p_initalize_v_marci IS
    -- lat-un vector de manevră !
    v2_marca t_v_marci ;
BEGIN
    pachet_salarizare.v_marci.DELETE ;
    -- se stochează toate mărcile în vectorul de manevră
    SELECT marca BULK COLLECT INTO v2_marca FROM personal ORDER BY marca ;

    -- mărcile se trec din vectorul de manevră în cel public, V_MARCI, pentru care
    -- indexul este chiar marca respectivă (pentru acces rapid)
    FOR i IN 1..v2_marca.COUNT LOOP
        pachet_salarizare.v_marci (v2_marca(i)) := v2_marca (i) ;
    END LOOP ;
END p_initalize_v_marci ;

-----

FUNCTION f_prima_gaura_marca RETURN personal.marca%TYPE IS
    v_i personal.marca%TYPE ; -- prima valoare
    v_f personal.marca%TYPE ;
BEGIN
    -- dacă vectorul public V_MARCI nu este inițializat, se lansează procedura de profil
    IF v_marci.COUNT = 0 THEN
        p_initalize_v_marci ;
    END IF ;

    -- se determină valorile de start și actuală din secvența SEQ_MARCA
    SELECT min_value, last_number INTO v_i, v_f FROM USER_SEQUENCES
    WHERE sequence_name = 'SEQ_MARCA' ;

    -- se verifică, pe rând, dacă toate valorile din secvență au fost alocate
    FOR i IN v_i..v_f - 1 LOOP
        IF v_marci.EXISTS(i) THEN
            NULL ;
        ELSE
            -- s-a găsit o valoare nealocată ! se verifică dacă, între timp, o altă sesiune
            -- a beneficiat de această valoare
            IF pachet_exista.f_exista(i) THEN
                -- valoarea a fost deja preluată de alt utilizator
                pachet_salarizare.v_marci(i) := i ;
            ELSE
                -- valoarea poate (aparent) fi alocată
                RETURN i ;
            END IF ;
        END IF ;
    END LOOP ;

    -- în acest punct se ajunge dacă vectorul a fost parcurs în întregime și nu a fost găsită
    -- nici o valoare nealocată, caz în care se apelează la secvență

```

```

SELECT seq_marca.NextVal INTO v_f FROM DUAL ;
RETURN v_f ;

END f_prima_gaura_marca ;

...
-- în continuare, corpul pachetului este exact ca în Listing 9.31
...

END pachet_salarizare ;
/

```

Funcția-cheie a soluției, după verificarea inițializării tabloului `v_marci`, încarcă în două variabile valoarea inițială și următoare din secvența `SEQ_MARCA`. Cele două valori constituie reperele unei secvențe repetitive care verifică existența componentei `n` a tabloului (`v_marci.EXISTS`), ceea ce echivalează cu verificarea folosirii fiecărei valori din secvență. Dacă, la un moment dat, o valoare nu a fost alocată, aceasta se va returna ca marcă ce poate fi atribuită prin declanșatorul de inserare prezentat). Aici apare un artificiu care diminuează din riscuri: variabila-colecție `v_marca` este publică la nivel de sesiune (locală). Dacă sesiunea durează ore întregi, este posibil ca, din momentul inițializării colecției, până la cel al apelării funcției, un alt utilizator (altă sesiune) să fi “reciclat” deja din valorile pierdute ale secvenței sau să fi apelat la valorile următoare ale `SEQ_MARCA`. Așa că, înainte de a se returna valoarea `i`, se mai efectuează un control de rutină (folosind “supraîncărcata” funcție `F_EXISTA` din pachetul `PACHET_EXISTA`), dacă `i` a fost folosit de altcineva mai iute.

Acest stil de lucru la evita violarea cheii primare. Iată și noua versiune a declanșatorului, cea care beneficiază fără scrupule de funcție – listing 11.4.

Listing 11.4. Versiunea 3 a declanșatorului de inserare

```

CREATE OR REPLACE TRIGGER trg_personal_ins_befo_row
BEFORE INSERT ON personal FOR EACH ROW
DECLARE
    v_noua_marca personal.marca%TYPE ;
BEGIN
    v_noua_marca := pachet_salarizare.f_prima_gaura_marca() ;
    :NEW.marca := v_noua_marca ;
    pachet_salarizare.v_marci (v_noua_marca) := v_noua_marca ;
END ;

```

După apelul la funcție, valoarea preluată este și “operată” (ca fiind folosită) în variabila tablou `v_marci`. Singura problemă rămasă nerezolvată este violarea cheii primare atunci când comanda de inserare se lansează în condițiile în care alt utilizator nu a “comis” inserarea din sesiunea sa.

11.3. Declanșatoare destinate integrității referențiale

În Oracle, restricțiile referențiale pot fi gestionate prin opțiunile comenzilor `CREATE TABLE` sau `ALTER TABLE`. Astfel, ștergerea unei înregistrări părinte sau o valoare eronată a unei chei străine sunt sancționate prompt de SGBD. Principala problemă referențială rămasă nerezolvată este actualizarea în cascadă a unui atribut părinte în toate înregistrările copil. Implicit, la creare, pentru restricțiile referențiale opțiunile sunt `ON DELETE RESTRICT` și `ON UPDATE RESTRICT`. Opțiunea `UPDATE CASCADE` poate fi implementată, cel puțin deocamdată, numai cu ajutorul declanșatoarelor, ce-i drept, cu un efort destul de mic. Astfel, pentru ca modificarea unei mărci în tabela `PERSONAL` să se propage în cascadă în toate tabelele copil, ne folosim de un declanșator la nivel de linie, ca în listing 11.5.

Listing 11.5. Declanșator pentru propagarea modificărilor în înregistrările copil

```
CREATE OR REPLACE TRIGGER trg_personal_upd_after_row
  AFTER UPDATE OF marca ON personal
  REFERENCING OLD AS OLD NEW AS NEW
  FOR EACH ROW
BEGIN
  UPDATE pontaje SET marca = :NEW.marca WHERE marca = :OLD.marca ;
  UPDATE sporuri SET marca = :NEW.marca WHERE marca = :OLD.marca ;
  UPDATE retineri SET marca = :NEW.marca WHERE marca = :OLD.marca ;
  UPDATE salarii SET marca = :NEW.marca WHERE marca = :OLD.marca ;
END ;
```

Prin folosirea opțiunii `UPDATE OF marca ON personal` ne asigurăm că declanșatorul va fi lansat numai la modificarea atributului `Marca`, indiferent de ceea ce s-a întâmplat cu celelalte câmpuri ale tabelului în înregistrarea respectivă.

Atenție ! Simplitatea declanșatorului de mai sus are, totuși, un preț: comanda `UPDATE` ce modifică valoarea atributului `Marca` din `PERSONAL` trebuie se limiteze la o singură linie a tabelului, altfel spus, lucrurile iau o turnură dramatică prin acest trigger dacă se lansează, spre exemplu, `UPDATE personal SET marca = marca + 1`.

Complicăm problema încercând să modificăm declanșatorul, astfel încât utilizatorul să poată beneficia, la alegere, de ambele variante, `UPDATE CASCADE` și `UPDATE RESTRICT`. Precizarea modalității de acțiune se realizează folosind două variabile publice, declarate, unde altundeva ?, în `PACHET_SALARIZARE`: `v_regula_upd_personal` și `v_regula_del_personal`. Căutarea unei mărci în tabele părinte se va realiza prin intermediul unor funcții specializate, `F_MARCA_IN_PONTAJE`, `F_MARCA_IN_SPORURI`, `F_MARCA_IN_RETINERI` și `F_MARCA_IN_SALARII`, incluse în același pachet – vezi listing 11.6.

Listing 11.6. Noutățile din `PACHET_SALARIZARE`

```
CREATE OR REPLACE PACKAGE pachet_salarizare AS
-- regulile (CASCADE sau RESTRICT) de urmat la modificarea unei mărci sau ștergerea
-- unei linii în PERSONAL
```

```

v_regula_upd_personal CHAR(1) := 'C' ; -- implicit UPDATE CASCADE
v_regula_del_personal CHAR(1) := 'R' ; -- implicit DELETE RESTRICT

-- adăugăm și patru funcții pentru căutarea unei mărci în tabelele copil
FUNCTION f_marca_in_pontaje (marca_ personal.marca%TYPE) RETURN BOOLEAN ;
FUNCTION f_marca_in_sporuri (marca_ personal.marca%TYPE) RETURN BOOLEAN ;
FUNCTION f_marca_in_retineri (marca_ personal.marca%TYPE) RETURN BOOLEAN ;
FUNCTION f_marca_in_salarii (marca_ personal.marca%TYPE) RETURN BOOLEAN ;

...
--- restul ca în Listing 11.3
...
END pachet_salarizare ;
/

CREATE OR REPLACE PACKAGE BODY pachet_salarizare AS

-----
FUNCTION f_marca_in_pontaje (marca_ personal.marca%TYPE) RETURN BOOLEAN IS
    v_unu NUMBER(1) := 0;
BEGIN
    SELECT 1 INTO v_unu FROM dual WHERE EXISTS
        (SELECT 1 FROM pontaje WHERE marca = marca_);
    RETURN TRUE ;
EXCEPTION
WHEN NO_DATA_FOUND THEN
    RETURN FALSE ;
END f_marca_in_pontaje ;

-----
FUNCTION f_marca_in_sporuri (marca_ personal.marca%TYPE) RETURN BOOLEAN IS
    v_unu NUMBER(1) := 0;
BEGIN
    SELECT 1 INTO v_unu FROM dual WHERE EXISTS
        (SELECT 1 FROM sporuri WHERE marca = marca_);
    RETURN TRUE ;
EXCEPTION
WHEN NO_DATA_FOUND THEN
    RETURN FALSE ;
END f_marca_in_sporuri;

-----
FUNCTION f_marca_in_retineri (marca_ personal.marca%TYPE) RETURN BOOLEAN IS
    v_unu NUMBER(1) := 0;
BEGIN
    SELECT 1 INTO v_unu FROM dual WHERE EXISTS
        (SELECT 1 FROM retineri WHERE marca = marca_);
    RETURN TRUE ;
EXCEPTION
WHEN NO_DATA_FOUND THEN
    RETURN FALSE ;
END f_marca_in_retineri;

-----
FUNCTION f_marca_in_salarii (marca_ personal.marca%TYPE) RETURN BOOLEAN IS
    v_unu NUMBER(1) := 0;
BEGIN
    SELECT 1 INTO v_unu FROM dual WHERE EXISTS
        (SELECT 1 FROM salarii WHERE marca = marca_);
    RETURN TRUE ;
EXCEPTION

```

```

WHEN NO_DATA_FOUND THEN
    RETURN FALSE ;
END f_marca_in_salarii ;
...
--- restul ca în Listing 11.3
...
END pachet_salarizare ;
/

```

După operarea acestor modificări în pachet, se ajustează și declanșatorul de actualizare – vezi listing 11.7. În noua sa variantă, declanșatorul testează valoarea variabilei `v_regula_upd_personal`. Dacă aceasta este setată pe 'C', se vor aplica regulile actualizării în cascadă. Dacă însă este 'R', atunci se scotocesc, pe rând, toate tabelele copil. În caz că se găsește o singură înregistrare copil, se declanșează o eroare zgometoasă.

Listing 11.7. Noua versiune a declanșatorului de actualizare

```

-- declanșator cu dublă acțiune, CASCADE sau RESTRICT
CREATE OR REPLACE TRIGGER trg_personal_upd_after_row
AFTER UPDATE OF marca ON personal
REFERENCING OLD AS OLD NEW AS NEW
FOR EACH ROW
BEGIN
    IF pachet_salarizare.v_regula_upd_personal = 'C' THEN
        -- UPDATE CASCADE !
        UPDATE pontaje SET marca = :NEW.marca WHERE marca = :OLD.marca ;
        UPDATE sporuri SET marca = :NEW.marca WHERE marca = :OLD.marca ;
        UPDATE retineri SET marca = :NEW.marca WHERE marca = :OLD.marca ;
        UPDATE salarii SET marca = :NEW.marca WHERE marca = :OLD.marca ;
        UPDATE concedii SET marca = :NEW.marca WHERE marca = :OLD.marca ;
    ELSE
        -- UPDATE RESTRICT !
        IF pachet_salarizare.f_marca_in_pontaje (:OLD.marca) THEN
            RAISE_APPLICATION_ERROR (-20500,
                'Marca modificata are copii in PONTAJE');
        END IF ;
        IF pachet_salarizare.f_marca_in_sporuri (:OLD.marca) THEN
            RAISE_APPLICATION_ERROR (-20501,
                'Marca modificata are copii in SPORURI');
        END IF ;
        IF pachet_salarizare.f_marca_in_retineri (:OLD.marca) THEN
            RAISE_APPLICATION_ERROR (-20502,
                'Marca modificata are copii in RETINERI');
        END IF ;
        IF pachet_salarizare.f_marca_in_salarii (:OLD.marca) THEN
            RAISE_APPLICATION_ERROR (-20503, 'Marca modificata are copii in SALARII');
        END IF ;
    END IF ;
END ;
/

```

Modul de lucru propus este evident în secvența de comenzi SQL*Plus din figura 11.3. Mai întâi, se specifică, prin intermediul variabilei `v_regula_upd_personal`, regula UPDATE CASCADE. Comanda UPDATE următoare modifică valoarea mărcii 102 în 222. Operațiunea este încununată cu

succes, dacă este să luăm în considerare mesajul de confirmare a actualizării. După o scurtă pauză publicitară se schimbă variabila, în 'R'. Se observă că următoarea comandă UPDATE se va confrunta cu ostilitatea SGBD-ului, deoarece noua regulă instituită interzice această operațiune.

```
SQL> execute pachet_salarizare.v_regula_upd_personal := 'C'

PL/SQL procedure successfully completed.

SQL> UPDATE personal SET marca = 222 WHERE marca = 102 ;

1 row updated.

SQL> execute pachet_salarizare.v_regula_upd_personal := 'R';

PL/SQL procedure successfully completed.

SQL> UPDATE personal SET marca = 102 WHERE marca = 222 ;
UPDATE personal SET marca = 102 WHERE marca = 222
*
ERROR at line 1:
ORA-20500: Marca modificata are copii in PONTAJE
ORA-06512: at "FOTACHEM.TRG_PERSONAL_UPS_AFTER_ROW", line 11
ORA-04088: error during execution of trigger 'FOTACHEM.TRG_PERSONAL_UPS_AFTER_ROW'

SQL>
```

Figura 11.3. Verificarea modului de funcționare al declanșatorului de actualizare

Lucrurile sunt un pic mai interesante în cazul declanșatorului de ștergere. Implicit, în virtutea clauzei REFERENCES folosită în comenzile CREATE/ALTER TABLE, dacă se dorește ștergerea unei înregistrări din PERSONAL care conține o marcă pentru care există pontaje și/sau sporuri și/sau rețineri și/sau date centralizate privind drepturile salariate, Oracle se împotrivi pentru a putea păstra integritatea referențială.

Varianta declanșatorului TRG_PERSONAL_DEL_BEFORE_ROW, lansabil înaintea ștergerii unei linii din tabela PERSONAL este una flexibilă, de o manieră similară declanșatorului de modificare, în sensul că, manevrând variabila publică v_regula_del_personal, se va adopta "politica" DELETE RESTRICT sau DELETE CASCADE – vezi listing 11.8.

Listing 11.8. Declanșatorul de ștergere

```
-- declanșator de ștergere (tot) cu dublă acțiune, CASCADE sau RESTRICT
CREATE OR REPLACE TRIGGER trg_personal_del_before_row
  BEFORE DELETE ON personal
  REFERENCING OLD AS OLD NEW AS NEW
  FOR EACH ROW
BEGIN
  IF pachet_salarizare.v_regula_del_personal = 'C' THEN
    -- DELETE CASCADE !
    DELETE FROM salarii WHERE marca = :OLD.marca ;
    DBMS_OUTPUT.PUT_LINE ('Din SALARII s-au sters ' || SQL%ROWCOUNT || ' inregistrari ');
    DELETE FROM retineri WHERE marca = :OLD.marca ;
    DBMS_OUTPUT.PUT_LINE ('Din RETINERI s-au sters ' || SQL%ROWCOUNT || ' inregistrari ');
    DELETE FROM sporuri WHERE marca = :OLD.marca ;
```

```

        DBMS_OUTPUT.PUT_LINE ('Din SPORURI s-au sters ' || SQL%ROWCOUNT || ' inregistrari ');
        DELETE FROM pontaje WHERE marca = :OLD.marca ;
        DBMS_OUTPUT.PUT_LINE ('Din PONTAJE s-au sters ' || SQL%ROWCOUNT || ' inregistrari ');
    ELSE
        -- UPDATE RESTRICT !
        IF pachet_salarizare.f_marca_in_pontaje (:OLD.marca) THEN
            RAISE_APPLICATION_ERROR (-20510, 'Marca stearsa are copii in PONTAJE');
        END IF ;
        IF pachet_salarizare.f_marca_in_sporuri (:OLD.marca) THEN
            RAISE_APPLICATION_ERROR (-20511, 'Marca stearsa are copii in SPORURI');
        END IF ;
        IF pachet_salarizare.f_marca_in_retineri (:OLD.marca) THEN
            RAISE_APPLICATION_ERROR (-20512, 'Marca stearsa are copii in RETINERI');
        END IF ;
        IF pachet_salarizare.f_marca_in_salarii (:OLD.marca) THEN
            RAISE_APPLICATION_ERROR (-20513, 'Marca stearsa are copii in SALARII');
        END IF ;
    END IF ;
END ;
/

```

Prin comparație cu mecanismul nativ de urmărire a restricției referențiale, declanșatorul nostru afișează care este tabela copil în care vor apărea înregistrări orfane dacă ștergerea s-ar produce fără discernământ- vezi figura 11.4.

```

SQL> EXECUTE pachet_salarizare.v_regula_del_personal := 'R'

PL/SQL procedure successfully completed.

SQL> DELETE FROM personal WHERE marca = 1011 ;

1 row deleted.

SQL> DELETE FROM personal WHERE marca = 103 ;
DELETE FROM personal WHERE marca = 103
*
ERROR at line 1:
ORA-20510: Marca stearsa are copii in PONTAJE
ORA-06512: at "FOTACHEM.TRG_PERSONAL_DEL_BEFORE_ROW", line 15
ORA-04088: error during execution of trigger 'FOTACHEM.TRG_PERSONAL_DEL_BEFORE_ROW'

SQL> |

```

Figura 11.4. Testarea variantei DELETE RESTRICT

Partea cea mai interesantă este ștergerea în cascadă. Comanda DELETE din figura 11.5 încearcă din nou să șteargă angajatul cu marca 103, de data această în condițiile în care variabila de control al ștergerilor este setată pe 'C' (cascadă). Deoarece în corpul declanșatorului, după fiecare ștergere a liniilor copil din fiecare tabelă dependentă, a fost introdusă comanda de afișare a numărului de linii șterse, putem vizualiza cu mai mare claritate modul de derulare a ostilităților pe "ramura" DELETE CASCADE a declanșatorului.

```

SQL> EXECUTE pachet_salarizare.v_regula_del_personal := 'C'

PL/SQL procedure successfully completed.

SQL> DELETE FROM personal WHERE marca = 103
      2 /
Din SALARII s-au sters 1 inregistrari
Din RETINERI s-au sters 1 inregistrari
Din SPORURI s-au sters 1 inregistrari
Din PONTAJE s-au sters 41 inregistrari

1 row deleted.

SQL> |

```

Figura 11.5. Testarea variantei DELETE CASCADE

11.4. Reguli de validare complexe, tabele mutante

După cum bine de mai lamentam în capitolul 4, clauza `CHECK` dintr-o comandă `CREATE TABLE` (sau `ALTER TABLE`) nu poate conține nume de variabile, funcții, proceduri sau pachete. Cu atât mai puțin fraze `SELECT`. Așa încât, pentru declararea unei restricții de comportament ceva mai complexe, trebuie să folosim declanșatoarele.

Un compartiment nu poate avea mai mult de zece angajați !

Începem cu o problemă care, la prima vedere, nu pare din cale-afară de pretențioasă: din rațiuni de management și personal, un compartiment poate să numere maxim 10 angajați. Această regulă poate fi încălcată în două situații, când de adaugă o linie nouă în tabela `PERSONAL` sau când se modifică valoarea atributului `Compart` într-o linie din aceeași tabelă. De aceea, se va crea un declanșator - `TRG_PERSONAL_COMPARTIMENT` - al cărui corp este cel din listing 11.9.

Listing 11.9. Tentativă de instituirii a regulii “un compartiment - maximum 10 suflete”

```

CREATE OR REPLACE TRIGGER trg_personal_compartiment
  AFTER INSERT OR UPDATE OF compart ON personal
  REFERENCING OLD AS OLD NEW AS NEW
  FOR EACH ROW
DECLARE
  v_citi NUMBER(4);
BEGIN
  dbms_output.put_line('trg_personal_compartiment');
  SELECT COUNT(*) INTO v_citi FROM personal WHERE compart = :NEW.compart;
  IF v_citi >= 10 THEN
    RAISE_APPLICATION_ERROR (-20520, 'Un compartiment nu poate depasi 10 angajati');
  END IF;
END;
/

```

Ca element de noutate, se observă că declanșatorul se lansează în cele două situații, inserare și modificarea atributului `Compart`. Fraza `SELECT` determină câți angajați numără compartimentul (`:NEW.compart`) la momentul inserării/modificării. Dacă valoarea variabilei `v_citi` este mai mare sau egală cu 10, atunci se declanșează eroarea care semnalizează încălcarea restricției.

Tabele mutante

Ei bine, să verificăm funcționalitatea declanșatorului. Ne propunem să modificăm compartimentul angajatului cu marca 102, adică să mutăm acest om al muncii în departamentul IT (în ciuda recesiunii din domeniu). Comanda `UPDATE` prin care s-ar realiza această operațiune se soldează, însă, cu un brutal mesaj de eroare, după cum arată figura 11.6.

```
SQL> update personal set compart = 'IT' WHERE marca=1012 ;
update personal set compart = 'IT' WHERE marca=1012
*
ERROR at line 1:
ORA-04091: table FOTACHEM.PERSONAL is mutating, trigger/function may not see it
ORA-06512: at "FOTACHEM.TRG_PERSONAL_COMPARTIMENT", line 5
ORA-04088: error during execution of trigger 'FOTACHEM.TRG_PERSONAL_COMPARTIMENT'
```

Figura 11.6. O tabelă mutantă...

Comanda de actualizare a adus tabela `PERSONAL` în postura de tabelă mutantă, ceea ce PL/SQL interzice fără drept de apel. O tabelă mutantă (*mutating table*) este o tabelă în curs de modificare printr-o comandă DML. Pentru un declanșator, mutantă este tabela pentru care a fost declarat. De asemenea, la folosirea opțiunii `ON DELETE CASCADE`, la ștergererea liniei părinte, tabele ce trebuie actualizate sunt, și ele, mutante. O altă noțiune este cea de tabelă restricționată (*constraining table*), categorie ce desemnează tabela care este accesată în virtutea unei restricții referențiale. La actualizarea tabelii `PONTAJE`, `PERSONAL` este o tabelă restricționată.

“Poruncile” PL/SQL sunt clare:

- într-un declanșator, nu se interoghează sau actualizează o tabelă mutantă;
- un declanșator nu poate consulta sau modifica atributele din componența cheilor primare, cheilor alternative și cheilor străine din tabelele restricționate ale tabelii curente (tabela pentru care s-a definit declanșatorul).

Acestea ar fi veștile proaste. Vestea bună că aceste două restricții sunt valabile numai pentru declanșatoarele la nivel de linie. Așa că putem imagina rapid o soluție pentru ieșirea din impas. Declarăm o variabilă publică căreia îi zicem `v_compart_nou` și o includem în pachetul `PACHET_SALARIZARE`. Apoi folosim două declanșatoare, unul la nivel de linie, în care memorăm (în variabila publică) valoarea compartimentului, și un altul la nivel de comandă în care interogăm tabela `PERSONAL` pentru a testa câți angajați are compartimentul. Listingul 11.10 conține cele două declanșatoare.

Listing 11.10. Rezolvarea “mutanței” prin două declanșatoare și o variabilă publică

```

-- se șterge declanșatorul anterior
DROP TRIGGER trg_personal_compartiment ;
/

-- declanșatorul la nivel de linie preia noua valoare a COMPARTimentului
CREATE OR REPLACE TRIGGER trg_personal_compart_linie
  AFTER INSERT OR UPDATE OF compart ON personal
  REFERENCING OLD AS OLD NEW AS NEW
  FOR EACH ROW
  BEGIN
    dbms_output.put_line('trg_personal_compartiment_linie');
    pachet_salarizare.v_compart_nou := :NEW.compart ;
  END ;
/

-- declanșatorul la nivel de comandă
CREATE OR REPLACE TRIGGER trg_personal_compart_comanda
  AFTER INSERT OR UPDATE OF compart ON personal
  REFERENCING OLD AS OLD NEW AS NEW
  DECLARE
    v_citi NUMBER(4) ;
  BEGIN
    dbms_output.put_line('trg_personal_compartiment_comanda') ;
    SELECT COUNT(*) INTO v_citi FROM personal
      WHERE compart = pachet_salarizare.v_compart_nou ;
    IF v_citi > 10 THEN
      RAISE_APPLICATION_ERROR (-20520, 'Un compartiment nu poate depasi 10 angajati');
    END IF ;
  END ;
/

```

Pe ici pe colo a fost strecurată câte o comandă de afișare pentru ca, la test, să verificăm secvența de execuție a declanșatoarelor. Figura 11.7 imortalizează dialogul SQL*Plus pentru verificarea modului în care se verifică restricția.

```

SQL> update personal set compart = 'CONTA' WHERE marca=1012 ;
trg_personal_compartiment_linie
trg_personal_compartiment_comanda

1 row updated.

SQL> update personal set compart = 'PROD' WHERE marca=1012 ;
update personal set compart = 'PROD' WHERE marca=1012
*
ERROR at line 1:
ORA-20520: Un compartiment nu poate depasi 10 angajati
ORA-06512: at "FOTACHEM.TRG_PERSONAL_COMPART_COMANDA", line 7
ORA-04088: error during execution of trigger 'FOTACHEM.TRG_PERSONAL_COMPART_COMANDA'

SQL> |

```

Figura 11.7. Verificarea limitei numărului angajați dintr-un compartiment

Compartimentul Contabilitate nu are un număr prea mare de membri, așa că dacă angajatul cu marca 1012 este mutat aici nu este nici o problemă. Altfel stau

lucrurile cu producția. Orice încercare de a mai afilia acestui departament vreun angajat depășește limita maxim admisă, așa că declanșatorul își face datoria.

Bucuria ne este potolită de următorul scenariu: în tabela PERSONAL, 7 angajați sunt afiliați compartimentului Contabilitate, iar 6 celui Financiar. Printr-o decizie nedemocratică, se hotărăște ca cele două compartimente să fuzioneze, titulatura păstrată fiind Contabilitate. Altfel spus, valoarea atributului `Compart` va fi 'CONTA' pentru toți cei pentru care, în prezent, este 'FIN'. În plus, toți cei care nu fac parte dintr-un compartiment vor fi automat afiliați compartimentului Producție ('PROD'). Zis și făcut. Numai că UPDATE-ul următor:

```
UPDATE personal SET compart =      CASE compart
  WHEN 'FIN' THEN 'CONTA'
  WHEN NULL THEN 'PROD'
  ELSE compart END
```

trece ca vodă prin SQL. Deși compartimenul contabilitate va fi supraponderal (13 angajați, cu 3 peste limita admisă), declanșatorul nu a sesizat samavolnicia. De ce? Pentru că variabila din pachet este una simplă (scalară), iar declanșatorul care face verificarea, fiind declarat la nivel de comandă (STATEMENT), va prelua doar compartimentul ultimei linii modificate care, dat fiind caracterul întâmplător, poate fi altul decât cel la care regula este încălcată.

De aceea, în locul variabilei simple `v_compart_nou`, declarăm în pachetul PACHET_SALARIZARE un vector asociativ, ca în listing 11.11, `v_compart_noi`. Acesta va stoca toate valorile modificare de UPDATE, indiferent de numărul acestora.

Listing 11.11. Declararea vectorului V_COMPART_NOI în pachet

```
CREATE OR REPLACE PACKAGE pachet_salarizare AS
  v_compart_nou personal.compart%TYPE ;
  TYPE t_v_compart IS TABLE OF personal.compart%TYPE INDEX BY PLS_INTEGER ;
  v_compart_noi t_v_compart ;

  --- restul este ca-n Listing 11.6.
  ....
```

Trebuie avut grijă, însă, pentru ca, pentru a nu apare interferențe, înainte de lansarea celor două declanșatoare, vectorul să fie golit. Aceasta presupune folosirea unui al treilea declanșator, de tip BEFORE STATEMENT - vezi listing 11.12.

Listing 11.12. Soluție ameliorată pentru rezolvarea problemei tabelii mutante

```
DROP TRIGGER trg_personal_compart_comanda ;
/

-- declanșatorul BEFORE STATEMENT golește vectorul
CREATE OR REPLACE TRIGGER trg_personal_compart_bef_stat
  BEFORE INSERT OR UPDATE OF compart ON personal
  REFERENCING OLD AS OLD NEW AS NEW
BEGIN
```

```

    pachet_salarizare.v_compart_noi.DELETE ;
END ;
/

-- declanșatorul la nivel de linie preia toate noile valori ale COMPARTimentului
CREATE OR REPLACE TRIGGER trg_personal_compart_linie
    AFTER INSERT OR UPDATE OF compart ON personal
    REFERENCING OLD AS OLD NEW AS NEW
FOR EACH ROW
BEGIN
    pachet_salarizare.v_compart_noi (pachet_salarizare.v_compart_noi.COUNT + 1) :=
        :NEW.compart ;
END ;
/

-- declanșatorul AFTER STATEMENT
CREATE OR REPLACE TRIGGER trg_personal_compart_comanda
    AFTER INSERT OR UPDATE OF compart ON personal
    REFERENCING OLD AS OLD NEW AS NEW
DECLARE
    v_citi NUMBER(4) ;
BEGIN
    FOR i IN 1..pachet_salarizare.v_compart_noi.COUNT LOOP
        SELECT COUNT(*) INTO v_citi FROM personal
            WHERE compart = pachet_salarizare.v_compart_noi (i) ;
        IF v_citi > 10 THEN
            RAISE_APPLICATION_ERROR (-20520, 'Compartimentul ' || ' ' depaseste 10 angajati !!!');
        END IF ;
    END LOOP ;
END ;
/

```

După crearea acestor trei declanșatoare, comanda UPDATE nu mai poate eluda restricția numărului maxim de angajați dintr-un compartiment.

Evitarea duplicării cheii primare

Să imaginăm un alt scenariu pesimist. La actualizare, un utilizator modifică marca unui angajat, noua valoare fiind, să zicem, superioară cu trei valori curente din secvența SEQ_MARCA. La una dintre următoarele inserări, este aproape sigur că valoarea returnată de secvență se va suprapune cu cea modificată, așa încât cheia primară va fi violată. De aceea, pentru a evita o asemenea neplăcere, în declanșatorul de actualizare la nivel de linie (AFTER UPDATE ROW) trebuie să verificăm poziția noii mărci în raport cu valoarea curentă din secvență. Dacă valoarea mărcii este peste starea curentă a secvenței putem alege fie să interzicem actualizarea, fie să “consumăm” valorile secvenței până ajungem la nivelul mărcii modificate.

Ne valorificăm vocația rabinică încercând un compromis: în declanșator se va compara valoarea nouă a mărcii (:NEW.Marca) cu valoarea curentă a secvenței - v_marca_curenta. Dacă :NEW.marca <= v_marca_curenta + 3, atunci “consumăm” secvența până la nivelul :NEW.marca. Dacă decalajul este mai mare de trei unități, atunci blocăm modificarea. Această logică este prezentată într-o

variantă nou-nouță a declanșatorului TRG_PERSONAL_UPD_AFTER_ROW, ce face obiectul listingului 11.3.

Listing 11.13. Protejarea secvenței pentru evitarea încălcării unicității cheii primare

```
CREATE OR REPLACE TRIGGER trg_personal_upd_after_row
  AFTER UPDATE OF marca ON personal REFERENCING OLD AS OLD NEW AS NEW
  FOR EACH ROW
DECLARE
  v_marca_curenta personal.marca%TYPE ;
  v_marca personal.marca%TYPE ;
BEGIN
  IF pachet_salarizare.v_regula_upd_personal = 'C' THEN -- UPDATE CASCADE
    ...
    ... partea dedicată restricției referențiale rămâne identică Listingului 11.7
    ...
  END IF ;

  IF :NEW.marca > :OLD.marca THEN
    -- riscul apare doar atunci când noua marcă e mai mare decât vechea

    -- se determină valoarea actuală din secvența SEQ_MARCA
    SELECT last_number INTO v_marca_curenta FROM USER_SEQUENCES
    WHERE sequence_name = 'SEQ_MARCA' ;

    IF :NEW.marca <= v_marca_curenta + 3 THEN
      -- consumăm secvența pentru a evita violarea cheii primare
      FOR i IN v_marca_curenta .. :NEW.marca LOOP
        SELECT seq_marca.NEXTVAL INTO v_marca FROM dual ;
      END LOOP ;
    ELSE
      RAISE_APPLICATION_ERROR (-20529,
        'Marca depășeste cu mai mult de 3 valoarea curenta a secvenței ');
    END IF ;
  END IF ;
END ;
/
```

Verificarea are loc numai dacă vechea valoare a mărcii este mai mare decât cea dinainte. Dacă da, se extrage valoarea următoare din secvență (LAST_NUMBER) și se stochează în variabila v_marca_curenta. Aceasta se compară cu noua valoare a mărcii (:NEW.marca <= v_marca_curenta + 3). Dacă marca nouă este peste valoarea curentă a secvenței, dar nu o depășește cu 3, atunci se execută bucla care consumă cele una, două sau trei NEXTVAL-uri ale secvenței, pentru ca aceasta să depășească pragul generator de eroare. În caz contrar se declanșează eroarea care blochează modificarea.

Pentru un plus de claritate, figura 11.8 surprinde pașii necesari demonstrării funcționalității noii versiuni a declanșatorului. După (re)crearea declanșatorului se afișează valoarea curentă a secvenței (este 1029, adică LAST_NUMBER - 1). Prima comandă UPDATE încearcă să depășească serios limita de trei unități (noua valoare a mărcii se dorește a fi 1044, destul de mult peste 1030), așa că declanșatorul se grăbește să reacționeze cu mesajul de eroare. În schimb, a doua comandă de modificare este mai cuminte, noua valoare a mărcii (1031) înscriindu-se în limita

tolerată. Astfel încât declanșatorul este mult mai îngăduitor. Ultima interogare a secvenței pune în evidență că, întrucât cea mai mare marcă este acum 1031, secvența va genera, la următoarea apelare, valoarea 1032, deci restricția de cheie primară nu va fi încălcată.

```
SQL> @f:\oracle_carte\cap11_declansatoare\listing11_13.sql

Trigger created.

SQL> select * from user_sequences;

SEQUENCE_NAME                MIN_VALUE  MAX_VALUE  INCREMENT_BY  CYCLE  CACHE_SIZE  LAST_NUMBER
-----
SEQ_MARCA                     1010      5555        1  N  Y           0         1030

SQL> UPDATE personal SET marca = 1044 WHERE marca = 1028 ;
UPDATE personal SET marca = 1044 WHERE marca = 1028
*
ERROR at line 1:
ORA-20529: Marca depaseste cu mai mult de 3 valoarea curenta a secventei
ORA-06512: at "FOTACHEM.TRG_PERSONAL_UPD_AFTER_ROW", line 40
ORA-04088: error during execution of trigger 'FOTACHEM.TRG_PERSONAL_UPD_AFTER_ROW'

SQL> UPDATE personal SET marca = 1031 WHERE marca = 1028 ;

1 row updated.

SQL> select * from user_sequences;

SEQUENCE_NAME                MIN_VALUE  MAX_VALUE  INCREMENT_BY  CYCLE  CACHE_SIZE  LAST_NUMBER
-----
SEQ_MARCA                     1010      5555        1  N  Y           0         1032

SQL> |
```

Figura 11.8. Dialogul SQL*Plus care pune în valoare noua variantă a declanșatorului

11.5. Atribute actualizate prin declanșatoare

Bazele de date elaborate pentru aplicații complexe, cu zeci, sute sau chiar mii de utilizatori conectați simultan, folosesc din plin *denormalizarea*, ce poate fi definită laconic drept *redundanță controlată*, în vederea accelerării vitezei de acces sau protejării anumitor informații. Baza de date pe care o folosim din capitolul patru până în prezent este plină de redundanțe, mai mult sau mai puțin controlate. Astfel, în tabela SPORURI, OreNoapte însumează valorile atributului cu același nume din tabela PONTAJE, pentru un angajat și lună (dintr-un an) date. Nici SpVech (sporul vechime) sau SpNoapte (sporul acordat pentru orele de noapte prestate) nu se deosebesc prea mult, întrucât, ambele sunt calculate prin expresii în care apar atribute din tabelele PERSONAL și PONTAJE. Ce să mai zicem de tabela SALARII...

Chiar dacă poate părea un exces, ne propunem ca actualizarea atributelor calculate ale tabelelor SPORURI și SALARII să fie realizată exclusiv prin declanșatoare ale tabelii PONTAJE:

- inserarea unei linii în tabela PONTAJE semnifică o nouă zi lucrată (sau de concediu) pentru un angajat; de aceea, se cuvine să *incrementăm* atributele SPORURI.SpVech, SPORURI.OreNoapte, SPORURI.SpNoapte, SALARII.OreLucrete, SALARII.OreCO, SALARII.VenitBaza și SALARII.Sporuri;
- ștergerea unei linii în PONTAJE presune operațiunea inversă, de *decrementare*;
- modificarea este ceva mai complexă; în principiu, putem, însă vorbi, de o *decrementare* a atributelor pentru *vechile* valori ale mărcii și lunii (anului) urmată de o *incrementare* a atributelor pentru *noile* marcă și lună/an.

Ținând seama de cele de mai sus, să redactăm declanșatoarele tabelor PONTAJE și SPORURI. Înainte de aceasta, adăugăm pachetului încă două variabile despre care păstrăm momentan o discreție absolută și trei funcții care, pe baza mărcii, să furnizeze salariul orar, salariul orar pentru calculul concediului de odihnă – vezi listing 11.14.

Listing 11.14. Adăugarea în pachet a funcțiilor pentru aflarea salariilor și datei pentru calculul sporului de vechime

```
CREATE OR REPLACE PACKAGE pachet_salarizare AS
v_declansator_PONTAJE BOOLEAN := FALSE ;
v_declansator_SPORURI BOOLEAN := FALSE ;
v_declansator_ALTESP BOOLEAN := FALSE ;
FUNCTION f_salorar (marca_ personal.marca%TYPE) RETURN personal.salorar%TYPE ;
FUNCTION f_salorarco (marca_ personal.marca%TYPE) RETURN personal.salorarco%TYPE ;
FUNCTION f_datasv (marca_ personal.marca%TYPE) RETURN personal.datasv%TYPE ;
...
--- restul ca în Listing 11.11
...
END pachet_salarizare ;
/

-- corpul pachetului
CREATE OR REPLACE PACKAGE BODY pachet_salarizare AS

-----
FUNCTION f_salorar (marca_ personal.marca%TYPE) RETURN personal.salorar%TYPE
IS
    v_salorar personal.salorar%TYPE ;
BEGIN
    SELECT salorar INTO v_salorar FROM personal WHERE marca = marca_ ;
    RETURN v_salorar ;
EXCEPTION
WHEN NO_DATA_FOUND THEN
    RETURN 0 ;
END f_salorar ;

-----
FUNCTION f_salorarco (marca_ personal.marca%TYPE) RETURN personal.salorarco%TYPE
IS
    v_salorarco personal.salorarco%TYPE ;
BEGIN
    SELECT salorarco INTO v_salorarco FROM personal WHERE marca = marca_ ;
```

```

RETURN v_salorarco ;
EXCEPTION
WHEN NO_DATA_FOUND THEN
RETURN 0 ;
END f_salorarco ;

-----
FUNCTION f_datasv (marca_ personal.marca%TYPE) RETURN personal.datasv%TYPE
IS
v_datasv personal.datasv%TYPE ;
BEGIN
SELECT datasv INTO v_datasv FROM personal WHERE marca = marca_ ;
RETURN v_datasv ;
EXCEPTION
WHEN NO_DATA_FOUND THEN
RETURN SYSDATE ;
END f_datasv ;
...
--- restul ca în Listing 11.11
...
END pachet_salarizare ;
/

```

Listing-ul 11.15 aduce câteva premiere “triggeristice”. Mai întâi, declanșatorul este unul mai des lansat, deoarece va intra în acțiune și după inserarea, și după modificarea și după ștergerea unei linii din PONTAJE. Pentru a trata diferențiat ceea ce este de făcut la fiecare dintre cele trei operațiuni, la testare s-au folosit clauzele **INSERTING**, **UPDATING** și **DELETING**. La actualizare și inserare se verifică dacă e vorba de primul pontaj pe luna respectivă pentru angajatul de pe linia curentă (scop în care se apelează la serviciile funcției **F_EXISTA** din pachet). În caz că da, se adaugă o linie nouă în tabela **SPORURI** (și, analog, în **SALARII**), iar în caz contrar se actualizează linia corespunzătoare.

Listing 11.15. Declanșatorul tabelii **PONTAJE** pentru actualizarea tabelilor **SPORURI** și **SALARII**

```

CREATE OR REPLACE TRIGGER trg_pontaje_after_row
AFTER INSERT OR UPDATE OR DELETE ON pontaje
REFERENCING NEW AS NEW OLD AS OLD FOR EACH ROW
DECLARE
v_spvech sporuri.spvech%TYPE ;
v_sjnoapte sporuri.sjnoapte%TYPE ;
v_venitbaza salarii.venitbaza%TYPE ;
v_procent transe_sv.procent_sv%TYPE ;
BEGIN
-- la triggerele de inserare și actualizare, valorile atributelor trebuie incrementate
IF INSERTING OR UPDATING THEN
v_venitbaza := (:NEW.orelucrate * pachet_salarizare.f_salorar(:NEW.marca) +
:NEW.oreco * pachet_salarizare.f_salorarco(:NEW.marca) ) ;
v_sjnoapte := :NEW.orenoapte * pachet_salarizare.f_salorar(:NEW.marca) * .15 ;
v_procent := pachet_salarizare.f_procent_spor_vechime (
pachet_salarizare.f_anii_vechime (
pachet_salarizare.f_datasv(:NEW.marca), EXTRACT (YEAR FROM :NEW.data),
EXTRACT (MONTH FROM :NEW.data) ) ) ;
v_spvech := v_venitbaza * v_procent / 100 ;

```

```

-- ne ocupăm, mai întâi, de tabela SPORURI
IF pachet_exista.f_exista (:NEW.marca, EXTRACT (YEAR FROM :NEW.data),
    EXTRACT (MONTH FROM :NEW.data), 'SPORURI') THEN
    -- există o înregistrare în SPORURI, ce trebuie actualizată
    UPDATE sporuri SET spvech = spvech + v_spvech,
        orenoapte = orenoapte + :NEW.orenoapte, spnoapte = spnoapte + v_spnoapte
    WHERE marca = :NEW.marca AND an = EXTRACT (YEAR FROM :NEW.data) AND
        luna = EXTRACT (MONTH FROM :NEW.data) ;
ELSE
    -- nu există înregistrare în SPORURI, deci trebuie inserată
    INSERT INTO sporuri VALUES ( :NEW.marca, EXTRACT (YEAR FROM :NEW.data),
        EXTRACT (MONTH FROM :NEW.data), v_spvech, :NEW.orenoapte,
        v_spnoapte, 0 ) ;
END IF ;

-- apoi de tabela SALARII
IF pachet_exista.f_exista (:NEW.marca, EXTRACT (YEAR FROM :NEW.data),
    EXTRACT (MONTH FROM :NEW.data), 'SALARII') THEN
    -- există o înregistrare în SALARII, ce trebuie actualizată
    UPDATE salarii SET orelucrate = orelucrate + :NEW.orelucrate,
        oreco = oreco + :NEW.oreco, venitbaza = venitbaza + v_venitbaza,
        sporuri = sporuri + v_spvech + v_spnoapte
    WHERE marca = :NEW.marca AND an = EXTRACT (YEAR FROM :NEW.data)
        AND luna = EXTRACT (MONTH FROM :NEW.data) ;
ELSE
    -- nu există înregistrare în SALARII, deci trebuie inserată
    INSERT INTO salarii VALUES (:NEW.marca, EXTRACT (YEAR FROM :NEW.data),
        EXTRACT (MONTH FROM :NEW.data), :NEW.orelucrate, :NEW.oreco,
        v_venitbaza, v_spvech + v_spnoapte, 0, 0 ) ;
END IF ;
END IF ;

-- la trigerile de ștergere și actualizare, valorile atributelor trebuie DECREMENTATE
IF DELETING OR UPDATING THEN
    v_venitbaza := (:OLD.orelucrate * pachet_salarizare.f_salorar(:OLD.marca) +
        :OLD.oreco * pachet_salarizare.f_salorarco(:OLD.marca) ) ;
    v_spnoapte := :OLD.orenoapte * pachet_salarizare.f_salorar(:OLD.marca) * .15 ;
    v_procent := pachet_salarizare.f_procent_spor_vechime ( pachet_salarizare.f_ani_vechime (
        pachet_salarizare.f_datasv(:OLD.marca), EXTRACT (YEAR FROM :OLD.data),
        EXTRACT (MONTH FROM :OLD.data) ) ) ;
    v_spvech := v_venitbaza * v_procent / 100 ;

-- tabela SPORURI
UPDATE sporuri SET spvech = spvech - v_spvech, orenoapte = orenoapte + :OLD.orenoapte,
    spnoapte = spnoapte + v_spnoapte
WHERE marca = :OLD.marca AND an = EXTRACT (YEAR FROM :OLD.data)
    AND luna = EXTRACT (MONTH FROM :OLD.data) ;

-- tabela SALARII
UPDATE salarii SET orelucrate = orelucrate - :OLD.orelucrate,
    oreco = oreco - :OLD.oreco, venitbaza = venitbaza - v_venitbaza,
    sporuri = sporuri - v_spvech - v_spnoapte
WHERE marca = :OLD.marca AND an = EXTRACT (YEAR FROM :OLD.data)
    AND luna = EXTRACT (MONTH FROM :OLD.data) ;
END IF ;
END ;
/

```

Tot pentru a mai schimba un pic aerul, în locul funcțiilor de conversie `TO_NUMBER (TO_CHAR (data, ...))`, s-a folosit proaspăt introdusă funcție (în Oracle 9i) `EXTRACT`. În schimb, din rațiuni de comoditate a autorilor, s-a renunțat la funcțiile de rotunjire și trunchiere, astfel încât valorile sunt calculate la ultimul leuț. Asta în așteptarea leului greu.

Așadar, ceea ce fost inițial un bloc și a devenit procedură – actualizarea automată a tabelelor `SPORURI` și `SALARII` – a devenit subiect de declanșator. În acest moment, orice modificare a tabelii `PONTAJE` atrage automat după sine sincronizarea celor două tabele pline ochi de attribute redundante. Rămâne o singură sursă de incoerențe, și anume atributul `AlteSp` (alte sporuri) din tabela `SPORURI`. Modificarea acestui trebuie să fie urmată automat de modificarea valorii atributului `SALARII.sporuri`, pentru angajatul (`Marca`) și luna/anul de pe linia curentă din `SPORURI`. Iată, în listing 11.16 și rezolvarea acestei probleme.

Listing 11.16. Propagarea modificării în `SALARII` a atributului `SPORURI.altesp`

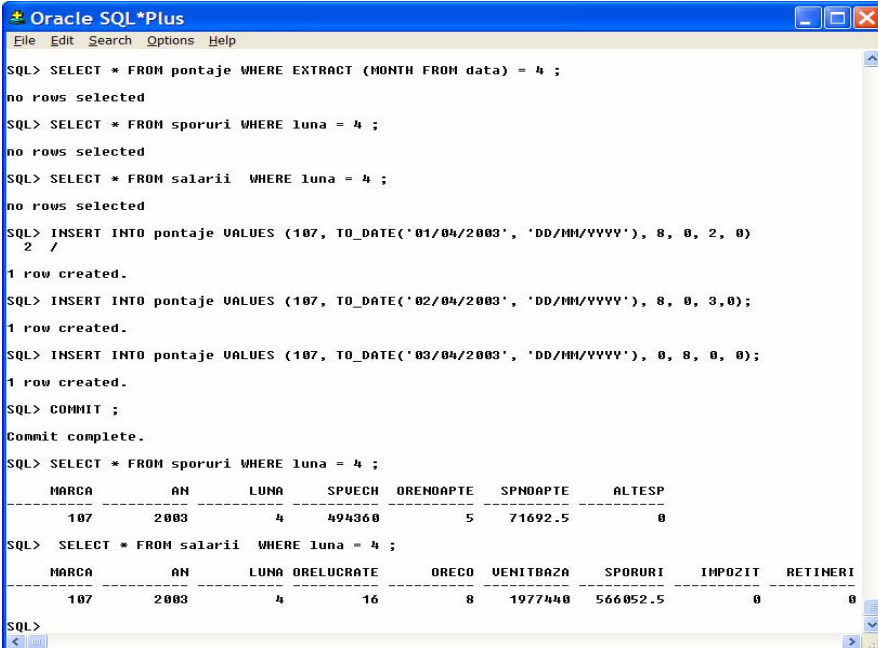
```
CREATE OR REPLACE TRIGGER trg_sporuri_after_row
  AFTER INSERT OR UPDATE OR DELETE ON sporuri
  REFERENCING NEW AS NEW OLD AS OLD FOR EACH ROW
BEGIN
  -- la triggerul de actualizare, valoarea SALARII.sporuri trebuie incrementată
  IF UPDATING AND :NEW.altesp <> :OLD.altesp THEN
    UPDATE salarii SET sporuri = sporuri + :NEW.altesp - :OLD.altesp
      WHERE marca = :NEW.marca AND an = :NEW.an AND luna = :NEW.luna ;
  ELSE
    -- deocamdată, nimic !
    NULL ;
  END IF ;
END ;
/
```

Figura 11.9 surprinde succesiunea de comenzi în `SQL*Plus` pentru testarea modului de funcționare a declanșatorului tabelii `PONTAJE`, ce-i drept, numai pentru operațiunea de inserare. Mai întâi se verifică dacă există vreo linie referitoare la luna aprilie (nu se specifică anul, deoarece ne-am apucat de treabă abia în 2003). Apoi se inserează trei zile de pontaj pentru angajatul cu marca 107, două de lucru efectiv (1 și 2 aprilie) și una de concediu (3 aprilie). Ultimele două `SELECT`-uri demonstrează că actualizarea celor două tabele pare a fi corectă.

Următorul calup de comenzi – vezi figura 11.10 – pune în valoare același declanșator, însă în condițiile actualizării și ștergerii de înregistrări în tabela `PONTAJE`. Astfel, pentru angajatul 107 și ziua de 2 aprilie 2003, valoarea corectă a orelor lucrate este 12 (în loc de 8), iar numărul orelor de noapte este 5 (în loc de 3). Așadar, dacă luăm în considerație datele afișate în figura anterioară, ar trebui ca noul total al orelor lucrate să fie 7 (tabela `SPORURI`), iar al celor lucrate 20 (înainte era 16). Cele două `SELECT`-uri ce urmează comenzii `UPDATE` certifică corectitudinea actualizărilor.

După afișarea liniilor celor două tabele, se șterge linia corespunzătoare pontajului angajatului 107 pe ziua de 1 aprilie (era, oricum, o păcăleală). În această zi

fuseseră pontate pe nedrept 8 ore, din care 2 de noapte. După ștergere, cele două interogări afișează noile valori ale liniilor din SPORURI și SALARII.



```

SQL> SELECT * FROM pontaje WHERE EXTRACT (MONTH FROM data) = 4 ;
no rows selected
SQL> SELECT * FROM sporuri WHERE luna = 4 ;
no rows selected
SQL> SELECT * FROM salarii WHERE luna = 4 ;
no rows selected
SQL> INSERT INTO pontaje VALUES (107, TO_DATE('01/04/2003', 'DD/MM/YYYY'), 8, 0, 2, 0)
2 /
1 row created.
SQL> INSERT INTO pontaje VALUES (107, TO_DATE('02/04/2003', 'DD/MM/YYYY'), 8, 0, 3, 0);
1 row created.
SQL> INSERT INTO pontaje VALUES (107, TO_DATE('03/04/2003', 'DD/MM/YYYY'), 0, 8, 0, 0);
1 row created.
SQL> COMMIT ;
Commit complete.
SQL> SELECT * FROM sporuri WHERE luna = 4 ;

```

MARCA	AN	LUNA	SPUECH	ORENOAPTE	SPNOAPTE	ALTESP
107	2003	4	494360	5	71692.5	0

```

SQL> SELECT * FROM salarii WHERE luna = 4 ;

```

MARCA	AN	LUNA	ORELUCRATE	ORECO	VENITBAZA	SPORURI	IMPOZIT	RETINERI
107	2003	4	16	8	1977440	566052.5	0	0

```

SQL>

```

Figura 11.9. Funcționarea declanșatoarelor tabeli PONTAJE - cazul inserării

```

Oracle SQL*Plus
File Edit Search Options Help

SQL> UPDATE pontaje SET ore lucrata=12, ore noapte=5 WHERE marca=107 AND
2 data=TO_DATE('02/04/2003', 'DD/MM/YYYY')
3 /

1 row updated.

SQL> COMMIT ;

Commit complete.

SQL> SELECT * FROM sporuri WHERE luna = 4 ;

MARCA      AN  LUNA      SPUECH  ORENOAPTE  SPNOAPTE  ALTESP
-----
107      2003      4      589950      7      100369.5      0

SQL> SELECT * FROM salarii WHERE luna = 4 ;

MARCA      AN  LUNA  ORELUCRATE  OREC0  VENITBAZA  SPORURI  IMPOZIT  RETINERI
-----
107      2003      4      20      8      2359800      690319.5      0      0

SQL> DELETE FROM pontaje WHERE marca=107 AND data=TO_DATE('01/04/2003', 'DD/MM/YYYY') ;

1 row deleted.

SQL> COMMIT;

Commit complete.

SQL> SELECT * FROM sporuri WHERE luna = 4 ;

MARCA      AN  LUNA      SPUECH  ORENOAPTE  SPNOAPTE  ALTESP
-----
107      2003      4      398770      5      71692.5      0

SQL> SELECT * FROM salarii WHERE luna = 4 ;

MARCA      AN  LUNA  ORELUCRATE  OREC0  VENITBAZA  SPORURI  IMPOZIT  RETINERI
-----
107      2003      4      12      8      1595080      470462.5      0      0

SQL> |

```

Figura 11.10. Funcționarea declanșatoarelor tabeli PONTAJE – cazurile modificării și ștergerii

Cât privește declanșatorul TRG_SPORURI_AFTER_ROW, vă lăsăm dvs. plăcerea de a-l testa în prezenta formă, urmând să ne ocupăm de dumnealui în chiar paragraful următor.

11.6. Declanșatoare pentru controlul actualizărilor

Deși impresionant pentru unii, mecanismul de actualizare automată a atributelor prin declanșatoarele discutate în paragraful anterior are cel puțin un călcâi al lui Ahile: ce se întâmplă când un utilizator modifică direct, prin comenzi DML (INSERT/UPDATE/DELETE) tabelele SPORURI și SALARII, iar valorile atributelor nu mai sunt sincronizate cu cele obținute prin centralizarea pontajelor? Evident, este vorba de o eroare gravă, de fapt, putem vorbi de pierderea consistenței bazei de date.

Tot prin declanșatoare și apel la cele două variabile publice din listing 11.14 putem reduce riscul unei asemenea erori. Adăugăm în declanșatorul TRG_PONTAJE_AFTER_ROW (listing 11.15) două linii, la începutul și sfârșitul secțiunii executabile – vezi listing 11.17. Prin modificarea variabilei publice pachet_salarizare.v_declansator_PONTAJE vom semnaliza altor declanșatoare că modificarea decurge din acțiunea acestui trigger.

Listing 11.17. Folosirea variabilei publice în declanșator

```

CREATE OR REPLACE TRIGGER trg_pontaje_after_row ....
DECLARE
...
BEGIN
    pachet_salarizare.v_declansator_PONTAJE := TRUE ;
    ...
    ....identic Listingului 11.15
    ...
    pachet_salarizare.v_declansator_PONTAJE := FALSE ;
END ;
/

```

Pentru tabela SPORURI definim două declanșatoare, unul mai mare și altul mai mic. Primul, TRG_SPORURI_AFTER_ROW, va fi lansat după inserarea și la ștergerea unei linii, precum și după modificarea valorii unuia dintre atributele: Marca, An, Luna, SpVech, OreNoapte și SpNoapte. După cum se observă, AlteSp face opinie separată, deoarece acest atribut este singurul care se actualizează altfel decât prin declanșatorul tabelii PONTAJE, așa încât în cinstea modificării acestuia a fost creat un declanșator special – TRG_SPORURI_ALTESP.

Corpul ambelor este conținut de listingul 11.18. În TRG_SPORURI_AFTER_ROW, inserarea și modificarea atributelor specificate vor fi permise doar dacă variabila publică pachet_salarizare.v_declansator_PONTAJE are valoarea TRUE, ceea ce înseamnă că editarea se face din declanșatorul tabelii PONTAJE. La ștergere, însă, e o altă logică. Întrucât în TRG_PONTAJE_AFTER_ROW nu s-a prevăzut ștergerea de linii din SPORURI, ștergerea se poate face interactiv, fie dintr-o aplicație, fie printr-o comandă DELETE, dar cu o condiție: la momentul ștergerii din sporuri, să nu existe nici o oră lucrată sau de concediu de către angajatul respectiv pentru luna de referință.

Listing 11.18. Două declanșatoare pentru tabela SPORURI

```

/* SPORURI - declanșator pt. INSERARE/STERGERE și modificarea tuturor atributelor,
cu excepția SPORURI.altesp */
CREATE OR REPLACE TRIGGER trg_sporuri_after_row
    AFTER INSERT OR UPDATE OF marca, an, luna, spvech, orenoapte, spnoapte
    OR DELETE ON sporuri
    REFERENCING NEW AS NEW OLD AS OLD FOR EACH ROW
DECLARE
    v_unu NUMBER(1) := 0 ;
BEGIN
    -- se modifică variabila publică
    pachet_salarizare.v_declansator_SPORURI := TRUE ;

    -- daca modificarea nu provine de la declanșatorul tabelii PONTAJE,
    -- inserarea și modificarea sunt interzise !
    IF pachet_salarizare.v_declansator_PONTAJE = FALSE THEN
        IF INSERTING THEN
            RAISE_APPLICATION_ERROR (-20332,
                'In SPORURI nu se pot insera intregirari interactiv !!!') ;
        END IF ;
        IF UPDATING THEN
            RAISE_APPLICATION_ERROR (-20333,
                'Nu puteti opera aceste modificari in mod interactiv !!!') ;
        END IF ;
    END IF ;

```

```

END IF ;

-- cu ștergerea e altă poveste; dacă nu există nici un pontaj, atunci linia se poate șterge
IF DELETING THEN
  BEGIN
    SELECT 1 INTO v_unu FROM DUAL WHERE EXISTS
      (SELECT 1 FROM pontaje WHERE marca=:OLD.marca AND
        EXTRACT (YEAR FROM data) = :OLD.an AND
        EXTRACT (MONTH FROM data) = :OLD.luna AND
        orelucrate + oreco > 0 );
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      v_unu := 0 ;
  END ;
  IF v_unu = 1 THEN
    RAISE_APPLICATION_ERROR (-20334,
      'Nu aveti permisiunea de a șterge aceasta linie din SPORURI !!!') ;
  END IF ;
END IF ;
pachet_salarizare.v_declansator_SPORURI := FALSE ;
END trg_sporuri_after_row ;
/

-- declanșator special pentru atributul SPORURI.altesp
CREATE OR REPLACE TRIGGER trg_sporuri_altesp
AFTER UPDATE OF altesp ON sporuri
REFERENCING NEW AS NEW OLD AS OLD FOR EACH ROW
BEGIN
  pachet_salarizare.v_declansator_ALTESP := TRUE ;
  UPDATE salarii SET sporuri = sporuri + :NEW.altesp - :OLD.altesp ;
  pachet_salarizare.v_declansator_ALTESP := FALSE ;
END trg_sporuri_altesp ;
/

```

Dacă primul declanșator modifică, la începutul și finalul său, variabila publică `v_declansator_SPORURI`, cel de-al doilea încadrează actualizarea tabelului `SALARII` de setarea unei alte variabile din pachet – `v_declansator_ALTESP`. Iată, în figura 11.11, modalitățile în care reacționează cele două declanșatoare la modificarea interactivă a câmpurilor `SPORURI.SpVech` și `SPORURI.AlteSp`. Primul `UPDATE` este sancționat, pe bună dreptate, în timp ce al doilea decurge fără probleme.

```

SQL> UPDATE sporuri SET spvech = 9999999 WHERE marca=107 AND an=2003 AND luna=4;
UPDATE sporuri SET spvech = 9999999 WHERE marca=107 AND an=2003 AND luna=4
*
ERROR at line 1:
ORA-20333: Nu puteti opera aceste modificari in mod interactiv !!!
ORA-06512: at "FOTACHEM.TRG_SPORURI_AFTER_ROW", line 14
ORA-04088: error during execution of trigger 'FOTACHEM.TRG_SPORURI_AFTER_ROW'

```

```

SQL> UPDATE sporuri SET altesp=2540000 WHERE marca=107 AND an=2003 AND luna=4;

1 row updated.

```

```

SQL>

```

Figura 11.11. "Comportarea" declanșatoarelor tabeli SPORURI

Pe același calapod se construiesc cele două declanșatoare ale tabeli SALARII. Din nou am apelat la două triggere, deoarece atributele marca, an, luna, orelucrate, oreco și venitbază pot fi editate doar din declanșatorul tabeli PONTAJE (la fel și inserarea unei înregistrări în SALARII). În schimb, atributul sporuri poate fi modificat atât din PONTAJE, cât și din SPORURI (atributul AlteSp). Conținutul celor două declanșatoare este afișat în listing 11.19.

Listing 11.19. Două declanșatoare pentru tabela SALARII

```

/* SALARII - declanșator pentru INSERARE/STERGERE și MODIFICAREA atributelor
   marca, an, luna, orelucrate, oreco, venitbaza */
CREATE OR REPLACE TRIGGER trg_salarii_after_row
  AFTER INSERT OR UPDATE OF marca, an, luna, orelucrate, oreco, venitbaza
  OR DELETE ON salarii
  REFERENCING NEW AS NEW OLD AS OLD FOR EACH ROW
DECLARE
  v_unu NUMBER(1) := 0 ;
BEGIN
  -- se interzice inserarea/modificarea ce NU provine de la declanșatorul tabeli PONTAJE
  IF pachet_salarizare.v_declansator_PONTAJE = FALSE THEN
    IF INSERTING THEN
      RAISE_APPLICATION_ERROR (-20335,
        'In SALARII nu se pot insera intregirari interactiv !!!') ;
    END IF ;
    IF UPDATING THEN
      RAISE_APPLICATION_ERROR (-20336,
        'Nu puteti opera aceste modificari in mod interactiv !!!') ;
    END IF ;
  END IF ;

  /* în caz că nu există înregistrări corespondente în PONTAJE sau SPORURI,
   linia se poate șterge */
  IF DELETING THEN
    BEGIN
      SELECT 1 INTO v_unu FROM DUAL WHERE EXISTS
        (SELECT 1 FROM pontaje WHERE marca=:OLD.marca AND
          EXTRACT (YEAR FROM data) = :OLD.an AND
          EXTRACT (MONTH FROM data) = :OLD.luna AND
          orelucrate + oreco > 0 )
        OR EXISTS (SELECT 1 FROM sporuri
          WHERE marca=:OLD.marca AND an = :OLD.an
            AND luna = :OLD.luna AND
            spvech + spnoapte + altesp > 0) ;
    EXCEPTION
      WHEN NO_DATA_FOUND THEN
        v_unu := 0 ;
    END ;

    IF v_unu = 1 THEN
      RAISE_APPLICATION_ERROR (-20337,
        'Nu aveti permisiunea de a șterge aceasta linie din SALARII !!!') ;
    END IF ;
  END IF ;
END trg_salarii_after_row ;
/

```

```

/* declanșator special pentru atributul Sporuri, ce poate fi modificat atât prin declanșatorul
   tabelii PONTAJE, cât și prin declanșatorul tabelii SPORURI (AlteSp) */
CREATE OR REPLACE TRIGGER trg_salarii_altesp
  AFTER UPDATE OF sporuri ON salarii
  REFERENCING NEW AS NEW OLD AS OLD FOR EACH ROW
BEGIN
  IF pachet_salarizare.v_declansator_PONTAJE
    OR pachet_salarizare.v_declansator_ALTESP THEN
    -- e-n regula
    NULL ;
  ELSE
    RAISE_APPLICATION_ERROR (-20338,
      'In SALARII, atributul Sporuri nu se poate edita interactiv !') ;
  END IF ;
END trg_salarii_altesp ;
/

```

Tot pentru elocvență, figura 11.12 suprinde dialogul SQL*Plus prin care cele două declanșatoare ale tabelii SALARII sunt puse la lucru.

Firește, ceea ce am expus în acest paragraf reprezintă doar un aspect privind controlul actualizării înregistrărilor în tabelele bazei. Un redutabil mecanism poate fi instituit prin alocarea de drepturi de acces/actualizare diferențiate pe tipuri de utilizatori, ceea ce va discutat într-un capitol viitor dedicat elementelor de administrare avansată.

```

SQL> UPDATE salarii SET venitbaza=9999999 WHERE luna=4 AND an=2003 AND marca=107;
UPDATE salarii SET venitbaza=9999999 WHERE luna=4 AND an=2003 AND marca=107
*
ERROR at line 1:
ORA-20336: Nu puteti opera aceste modificari in mod interactiv !!!
ORA-06512: at "FOTACHEM.TRG_SALARII_AFTER_ROW", line 10
ORA-04088: error during execution of trigger 'FOTACHEM.TRG_SALARII_AFTER_ROW'

SQL> UPDATE salarii SET sporuri=9999999 WHERE luna=4 AND an=2003 AND marca=107;
UPDATE salarii SET sporuri=9999999 WHERE luna=4 AND an=2003 AND marca=107
*
ERROR at line 1:
ORA-20338: In SALARII, atributul Sporuri nu se poate edita interactiv !
ORA-06512: at "FOTACHEM.TRG_SALARII_ALTESP", line 6
ORA-04088: error during execution of trigger 'FOTACHEM.TRG_SALARII_ALTESP'

SQL> UPDATE salarii SET retineri=9999999 WHERE luna=4 AND an=2003 AND marca=107;

1 row updated.

SQL>

```

Figura 11.12. "Comportarea" declanșatoarelor tabelii SALARII

De asemenea, se pot folosi tabele pentru alocarea de drepturi "pe orizontală", cum ar fi, de exemplu, situația în care un utilizator poate edita doar înregistrările din PERSONAL, PONTAJE, SPORURI și SALARII corespunzătoare numai salariaților dintr-un anumit compartiment.

11.7. Declanșatoare sistem

Declanșatoarele sistem au fost introduse ceva mai recent în arsenalul PL/SQL (Oracle 8i). În cadrul acestei categorii se poate opera o delimitare între declanșatoare DDL (*Data Definition Language*) și declanșatoare la nivelul baze de date. Primele privesc comenzi precum CREATE, ALTER și DROP, iar cele din a doua categorie sunt legate de operațiunile de pornire și oprire a serverului, conectarea/deconectarea unui utilizator și apariția unei erori. Formatul este ceva mai simplu, prin comparație de declanșatoarele DML:

```
CREATE [OR REPLACE] TRIGGER [schema].nume_declanșator
  BEFORE | AFTER eveniment DDL/bază ON DATABASE | SCHEMA
  [WHEN ... ]
  corpul_declanșatorului;
```

Dat fiind caracterul lor cu totul special, declanșatoarele sistem pot fi create numai de utilizatori care dețin privilegiul-sistem ADMINISTER DATABASE TRIGGER. Tabelul 11.2 sistematizează tipologia declanșatoarelor sistem.

Tabel 11.2. Tipuri de declanșatoare-sistem¹

Eveniment	Moment	Condiții de declanșare
STARTUP	AFTER	Pornirea unei instanțe
SHUTDOWN	BEFORE	Oprirea unei instanțe. Excepție fac cazurile de oprire anormală (ex. căderi de tensiune)
SERVERERROR	AFTER	Apariția unei erori pe server
LOGON	AFTER	Conectarea reușită a unui utilizator
LOGOFF	BEFORE	Delogarea unui utilizator
CREATE	BEFORE, AFTER	Înainte și după crearea unui obiect din schemă
DROP	BEFORE, AFTER	Înainte și după ștergerea unui obiect din schemă
ALTER	BEFORE, AFTER	Înainte și după modificarea unui obiect din schemă

Dacă în comanda de creare apare clauza DATABASE, atunci declanșatorul va acoperi toate schemele definite ale bazei, iar dacă se folosește opțiunea SCHEMA declanșatorul va fi valabil numai pentru schema curentă.

Pentru a exemplifica un trigger din această clasă, ne propunem să evităm o situație destul de neplăcută în Oracle - cea în care, datorită dependențelor dintre obiectele bazei, sau chiar din senin, o serie de proceduri, funcții sau blocuri devin invalide, fără a li se schimba nimic în definiție sau corp. De cele mai multe ori soluția constă în simpla recompilare a obiectului procedural respectiv. În ceea ce ne privește, blocul anonim din listing 11.20, odată lansat, preia în doi vectori asociați, `v_nume_obiecte` și `v_tipuri_obiecte`, numele și tipul următoarelor categorii de blocuri din schema curentă: proceduri (PROCEDURE), funcții (FUNCTION), specificații de pachet (PACKAGE), corpuri ale pachetelor

¹ Preluare din [Urman02], pp.481-482

(PACKAGE BODY) și declanșatoare (TRIGGER). Ordinea în care sunt încărcate în vectori este cea a `object_id`-ului, ceea ce echivalează cu ordinea creării, așa încât o serie de probleme legate de dependențele dintre blocuri este rezolvată.

Folosind o structură CASE în care testează tipul obiectului procedural, bucla generează dinamic comanda ALTER *bloc* COMPILE *procedură/funcție/declanșator* sau ALTER *pachet* COMPILE SPECIFICATION, respectiv ALTER *pachet* COMPILE BODY pentru specificațiile și corpurile pachetelor.

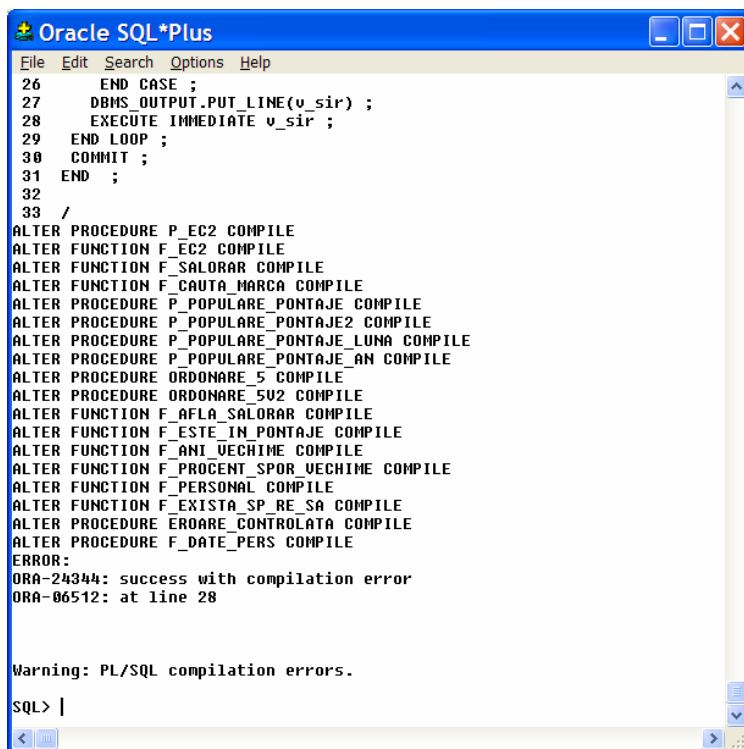
Listing 11.20. Bloc anonim ce recompilează blocuri procedurale din schema curentă

```
DECLARE
TYPE t_numo_obiecte IS TABLE OF user_objects.object_name%TYPE
INDEX BY PLS_INTEGER ;
v_numo_obiecte t_numo_obiecte ;
TYPE t_tipuri_obiecte IS TABLE OF user_objects.object_type%TYPE
INDEX BY PLS_INTEGER ;
v_tipuri_obiecte t_tipuri_obiecte ;
v_sir VARCHAR2(200) ;

BEGIN
-- stocăm în cei doi vectori numele și tipul fiecărui bloc de (re)compilat
SELECT object_name, object_type
BULK COLLECT INTO v_numo_obiecte, v_tipuri_obiecte
FROM user_objects
WHERE object_type IN ('PROCEDURE', 'FUNCTION', 'PACKAGE',
'PACKAGE BODY', 'TRIGGER')
ORDER BY object_id ;

-- le compilăm "dinamic"
FOR i IN 1..v_numo_obiecte.COUNT LOOP
v_sir := 'ALTER' ;
CASE v_tipuri_obiecte(i)
WHEN 'PACKAGE' THEN v_sir := v_sir || ' PACKAGE ' || v_numo_obiecte(i) ||
' COMPILE SPECIFICATION ' ;
WHEN 'PACKAGE BODY' THEN v_sir := v_sir || ' PACKAGE ' || v_numo_obiecte(i) ||
' COMPILE BODY ' ;
ELSE v_sir := v_sir || v_tipuri_obiecte(i) || ' ' || v_numo_obiecte(i) ||
' COMPILE ' ;
END CASE ;
DBMS_OUTPUT.PUT_LINE(v_sir) ;
EXECUTE IMMEDIATE v_sir ;
END LOOP ;
COMMIT ;
END ;
```

Dacă lansăm în execuție acest bloc, avem șanse apreciabile să facem cunoștință sau să ne reîntâlnim cu eroarea Oracle -24344, una dintre cele mai amuzante, deoarece ne spune, negru pe alb, *succes cu eroare la compilare*, aceasta, probabil, pentru moral - vezi figura 11.13. Lucrurile sunt însă ușor de explicat. Comanda de afișare inserată înainte de EXECUTE IMMEDIATE ne furnizează numele blocurilor recompilate. În cazul nostru, funcția F_DATE_PERS prezintă erori la compilare (rușine autorilor !), ceea ce declanșează celebra excepție și, firește, abandonează blocul anonim.



```

26      END CASE ;
27      DBMS_OUTPUT.PUT_LINE(v_sir) ;
28      EXECUTE IMMEDIATE v_sir ;
29    END LOOP ;
30  COMMIT ;
31  END ;
32
33  /
ALTER PROCEDURE P_EC2 COMPILE
ALTER FUNCTION F_EC2 COMPILE
ALTER FUNCTION F_SALORAR COMPILE
ALTER FUNCTION F_CAUTA_MARCA COMPILE
ALTER PROCEDURE P_POPULARE_PONTAJE COMPILE
ALTER PROCEDURE P_POPULARE_PONTAJE2 COMPILE
ALTER PROCEDURE P_POPULARE_PONTAJE_LUNA COMPILE
ALTER PROCEDURE P_POPULARE_PONTAJE_AN COMPILE
ALTER PROCEDURE ORDONARE_5 COMPILE
ALTER PROCEDURE ORDONARE_5U2 COMPILE
ALTER FUNCTION F_AFLA_SALORAR COMPILE
ALTER FUNCTION F_EST_E_IN_PONTAJE COMPILE
ALTER FUNCTION F_ANTI_VECHIME COMPILE
ALTER FUNCTION F_PROCENT_SPOR_VECHIME COMPILE
ALTER FUNCTION F_PERSONAL COMPILE
ALTER FUNCTION F_EXISTA_SP_RE_SA COMPILE
ALTER PROCEDURE EROARE_CONTROLATA COMPILE
ALTER PROCEDURE F_DATE_PERS COMPILE
ERROR:
ORA-24344: success with compilation error
ORA-06512: at line 28

Warning: PL/SQL compilation errors.

SQL> |

```

Figura 11.13. Depistarea erorii -24344 la compilare

Firește, trebuie să preluăm eroarea, astfel încât execuția să se deruleze până la ultimul bloc procedural din schemă, indiferent de câte dintre acestea au probleme de sintaxă. Profitând de conjunctură, transformăm blocul anonim în procedură pe care o botezăm RE_COMPILARE - vezi listing 11.21.

Erorii -24344 îi este asociată acum o excepție - `erori_la_compilare`, asociere realizată prin directiva de compilare `PRAGMA EXCEPTION_INIT`. De asemenea, folosim tabela TEMP care are un singur atribut VARCHAR2, însă destul de lung. Ne servim de această tabelă pentru a stoca numele blocurilor cu erori pentru o eventuală corecție a lor. Comanda `EXECUTE IMMEDIATE` este plasată într-un bloc secundar tocmai pentru tratarea excepției, tratare care presupune consemnarea în tabela TEMP.

Listing 11.21. Blocul anonim de recompilare transformat (și pus la punct) în procedură

```

CREATE OR REPLACE PROCEDURE re_compilare
IS
  TYPE t_num_e_obj IS TABLE OF user_objects.object_name%TYPE
    INDEX BY PLS_INTEGER ;
  v_num_e_obj t_num_e_obj ;
  TYPE t_tipuri_obj IS TABLE OF user_objects.object_type%TYPE
    INDEX BY PLS_INTEGER ;
  v_tipuri_obj t_tipuri_obj ;
  v_sir VARCHAR2(200) ;

```

```

erori_la_compilare EXCEPTION ;
PRAGMA EXCEPTION_INIT (erori_la_compilare, -24344) ;

BEGIN
  -- stocăm în cei doi vectori numele și tipul fiecărui bloc de (re)compilat
  SELECT object_name, object_type
    BULK COLLECT INTO v_nume_obiecte, v_tipuri_obiecte
  FROM user_objects
  WHERE object_type IN ('PROCEDURE', 'FUNCTION', 'PACKAGE', 'PACKAGE BODY',
    'TRIGGER') AND object_name NOT IN ('RE_COMPILARE', 'LA_LOGARE')
  ORDER BY CASE WHEN object_type ='PACKAGE BODY' THEN 1 ELSE 0 END || object_id ;

  -- pregătim tabela TEMP
  INSERT INTO temp VALUES (' Logarea din: ' || SYSTIMESTAMP || ' . Iata erorile:');

  -- re-compilarea "dinamică"
  FOR i IN 1..v_nume_obiecte.COUNT LOOP
    v_sir := ' ALTER ' ;
    CASE v_tipuri_obiecte(i)
      WHEN 'PACKAGE' THEN v_sir := v_sir || ' PACKAGE ' || v_nume_obiecte(i) ||
        ' COMPILE SPECIFICATION ' ;
      WHEN 'PACKAGE BODY' THEN v_sir := v_sir || ' PACKAGE ' || v_nume_obiecte(i) ||
        ' COMPILE BODY ' ;
      ELSE v_sir := v_sir || v_tipuri_obiecte(i) || ' ' || v_nume_obiecte(i) ||
        ' COMPILE ' ;
    END CASE ;

    BEGIN
      EXECUTE IMMEDIATE v_sir ;
    EXCEPTION
      WHEN erori_la_compilare THEN
        INSERT INTO temp VALUES ('Blocul ' || v_nume_obiecte(i) || ' prezinta erori !' ) ;
      WHEN OTHERS THEN
        INSERT INTO temp VALUES ('La blocul ' || v_nume_obiecte(i) || ' apare o eroare ciudata !' )
    ;
  END ;
END LOOP ;
END re_compilare ;
/

```

Un alt element care ține de sănătatea procedurii este eliminarea, dintre obiectele procedurale din schemă a chiar procedurii RE_COMPILARE și iminentului declanșator LA_LOGARE. Se evită astfel re-compilarea procedurii/triggerului din ea/el însuși, ceea ce blochează, dacă nu definitiv, măcar o bună bucată de timp a sesiunii Oracle.

În fine, pentru a "automatiza" recompilarea, apelăm procedura de mai sus dintr-un declanșator - LA_LOGARE - ce va fi lansat la fiecare conectare la schema curentă - vezi listing 11.22.

Listing 11.22. Declanșator de logare

```

CREATE OR REPLACE TRIGGER la_logare
AFTER LOGON ON FOTACHEM.SCHEMA
BEGIN
  re_compilare ;
END la_logare ;
/

```

Din nefericire, declanșatorul nu funcționează ! Suspiciunea planează asupra mecanismului tranzacțional. Ca orice comandă DDL, ALTER PROCEDURE, ALTER FUNCTION etc. închid automat o tranzacție. Ori, în procedură apar comenzi INSERT în tabela TEMP care ar trebui "commit-uite" la fiecare ALTER, ceea ce prin declanșatoare (nici) Oracle nu prea înghite.

Înainte de a gândi la un artificiu gen tranzacții autonome, modificăm logica ostilităților: mai întâi, trecem peste puseul de tensiune provocat de eșecul soluției anterioare, și, în loc să eliminăm consemnarea blocurilor procedurale cu probleme, folosim un vector ca substitut al tabeli TEMP. Cum tot am folosit un pachet pentru activități administrative - PAC_ADMINISTRARE - în capitolul trecut (vezi listingurile 10.4 - specificațiile și listing 10.5 - corpul), îl refolosim pentru a declara vectorul asociativ v_temp - vezi listing 11.23.

Listing 11.23. Pachetul de administrare reciclat

```
CREATE OR REPLACE PACKAGE pac_administrare
AUTHID CURRENT_USER
AS
TYPE t_temp IS TABLE OF VARCHAR2(3000) INDEX BY PLS_INTEGER ;
v_temp t_temp ;

-- restul ca in listing 10.4
...

END pac_administrare ;
```

Astfel, după cum se observă și în listing 11.24, noua variantă a procedurii RE_COMPILE stocază toate necazurile de la compilare în noul vector. Prima linie din secțiunea executabilă golește tabloul asociativ.

Listing 11.24. Procedura RE_COMPILE - versiunea 2.0.0.1

```
CREATE OR REPLACE PROCEDURE re_compile
IS
    TYPE t_numo_obiecte IS TABLE OF user_objects.object_name%TYPE
        INDEX BY PLS_INTEGER ;
    v_numo_obiecte t_numo_obiecte ;
    TYPE t_tipuri_obiecte IS TABLE OF user_objects.object_type%TYPE
        INDEX BY PLS_INTEGER ;
    v_tipuri_obiecte t_tipuri_obiecte ;
    v_sir VARCHAR2(200) ;
    erori_la_compilare EXCEPTION ;
    PRAGMA EXCEPTION_INIT (erori_la_compilare, -24344) ;

BEGIN
    pac_administrare.v_temp.DELETE ;

    -- stocăm în cei doi vectori numele și tipul fiecărui bloc de (re)compilat
    SELECT object_name, object_type
        BULK COLLECT INTO v_numo_obiecte, v_tipuri_obiecte
    FROM user_objects
    WHERE object_type IN ('PROCEDURE', 'FUNCTION', 'PACKAGE', 'PACKAGE BODY',
        'TRIGGER') AND object_name NOT IN
```

```

        ('RE_COMPILARE', 'LA_LOGARE', 'PAC_ADMINISTRARE')
ORDER BY CASE WHEN object_type = 'PACKAGE BODY' THEN 1 ELSE
        CASE WHEN object_type = 'TRIGGER' THEN 2 ELSE 0 END END , object_id ;

-- inițializarea primei componente a tabloului
pac_administrare.v_temp(1) := ' Logarea din ' || SYSTIMESTAMP || ' . Iata erorile:' ;

-- re-compilarea "dinamica"
FOR i IN 1..v_num_e_obiecte.COUNT LOOP
    v_sir := ' ALTER ' ;
    CASE v_tipuri_obiecte(i)
    WHEN 'PACKAGE' THEN v_sir := v_sir || ' PACKAGE ' || v_num_e_obiecte(i) ||
        ' COMPILE SPECIFICATION ' ;
    WHEN 'PACKAGE BODY' THEN v_sir := v_sir || ' PACKAGE ' || v_num_e_obiecte(i) ||
        ' COMPILE BODY ' ;
    ELSE v_sir := v_sir || v_tipuri_obiecte(i) || ' ' || v_num_e_obiecte(i) || ' COMPILE ' ;
    END CASE ;

    BEGIN
        EXECUTE IMMEDIATE v_sir ;
    EXCEPTION
    WHEN erori_la_compilare THEN
        pac_administrare.v_temp(pac_administrare.v_temp.COUNT + 1) :=
            SYSTIMESTAMP ||
            ' - Blocul ' || v_num_e_obiecte(i) || ' prezinta erori ! ' ;
    WHEN OTHERS THEN
        pac_administrare.v_temp(pac_administrare.v_temp.COUNT + 1) :=
            SYSTIMESTAMP ||
            ' - La blocul ' || v_num_e_obiecte(i) || ' apare o eroare ciudata ! ' ;

    END ;
END LOOP ;
END re_compilare ;

```

Corpul declanșatorului LA_LOGARE rămâne identic listingului 11.22. Figura 11.14 prezintă dialogul SQL*Plus pentru demonstrarea funcționalității celor expuse mai sus. Prin delogare și logare se lansează declanșatorul. Prin blocul anonim ce urmează se salvează conținutul vectorului în tabela TEMP care este, în final, listată.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> DISCONNECT
Disconnected from Personal Oracle9i Release 9.2.0.1.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production
SQL> CONNECT FOTACHEM
Enter password: *****
Connected.
SQL>
SQL> -- printr-un bloc anonim incarcam vectorul in tabela TEMP
SQL> BEGIN
2  FORALL i IN 1..pac_administrare.v_temp.COUNT
3      INSERT INTO temp VALUES (pac_administrare.v_temp(i));
4  END;
5  /

PL/SQL procedure successfully completed.

SQL> SELECT * FROM TEMP;

X

-----
Logarea din 12-AUG-03 09:35:42.310000000 AM -07:00. Iata erorile:
12-AUG-03 09:35:42.330000000 AM -07:00 - Blocul P_EC2 prezinta erori !
12-AUG-03 09:35:42.490000000 AM -07:00 - Blocul P_POPULARE_PONTAJE_AN prezinta erori !
12-AUG-03 09:35:42.500000000 AM -07:00 - Blocul ORDONARE_5 prezinta erori !
12-AUG-03 09:35:42.510000000 AM -07:00 - Blocul ORDONARE_5U2 prezinta erori !
12-AUG-03 09:35:42.620000000 AM -07:00 - Blocul F_DATE_PERS prezinta erori !
12-AUG-03 09:35:42.651000000 AM -07:00 - Blocul ORDONARE_5U3 prezinta erori !
12-AUG-03 09:35:42.891000000 AM -07:00 - La blocul PACHEI_EXISTA apare o eroare ciudata !
12-AUG-03 09:35:42.931000000 AM -07:00 - Blocul P_INSERT_PERSONAL prezinta erori !
12-AUG-03 09:35:43.041000000 AM -07:00 - Blocul PAC_INSERT prezinta erori !
12-AUG-03 09:35:43.081000000 AM -07:00 - Blocul TRG_PERSONAL_COMPART_LINIE prezinta erori !
12-AUG-03 09:35:43.091000000 AM -07:00 - Blocul TRG_PERSONAL_COMPART_BEI_STAT prezinta erori !
12-AUG-03 09:35:43.101000000 AM -07:00 - Blocul TRG_PONTAJE_AFTER_ROW prezinta erori !
12-AUG-03 09:35:43.141000000 AM -07:00 - Blocul TRG_INCHIDERI_INS prezinta erori !
12-AUG-03 09:35:43.141000000 AM -07:00 - Blocul TRG_PERSONAL_COMPART_COMANDA prezinta erori !

15 rows selected.

SQL> |

```

Figura 11.14. Funcționalitatea declanșatorului de logare la schema curentă

Utilitatea declanșatorului sistem care ne-a și dat ceva bătaie de cap în această variantă este una îndoielnică. La o adică, atunci când sesizăm neregularități, putem apela și singuri procedura de recompilare. Cu totul altfel stau lucrurile atunci când vrem să restricționăm accesul unui utilizator de la un anumit calculator. Să presupunem că aplicația client/server la care lucrăm este disponibilă câtorva zeci de utilizatori. Fiecare are grup are drepturi proprii de acces la obiectele bazei de date. Ca element suplimentar de securitate se hotărăște ca fiecare utilizator să se poată conecta la schemă numai de la anumite calculatoare, identificate prin adresele IP. În acest scop se crează o tabelă USERI_IPEURI:

```

CREATE TABLE useri_ipeuri (
    useru VARCHAR2(30),
    IP_authorized VARCHAR2(20),
    CONSTRAINT pk_uip PRIMARY KEY (useru, IP_authorized)
);

```

Această tabelă se populează astfel prin comenzi de genul:

```

INSERT INTO useri_ipeuri VALUES ('FOTACHEM', ' ');

```

Creăm un nou declanșator de logare, de data aceasta la baza de date. Îi spunem tot LA_LOGARE, deoarece la cel dinainte renunțăm. Corpul său - vezi listing 11.25

- testează dacă utilizatorul curent (furnizat de funcția `ORA_LOGIN_USER`) poate lucra de la adresa IP curentă (funcția `ORA_CLIENT_IP_ADDRESS`), prin consultarea tabelului `USERI_IPEURI`.

Listing 11.25. Declanșatorul de restricționare a accesului

```
CREATE OR REPLACE TRIGGER la_logare
AFTER LOGON ON DATABASE
DECLARE
    v_unu NUMBER(1) := 0 ;
BEGIN
    BEGIN
        SELECT 1 INTO v_unu FROM dual WHERE EXISTS
            (SELECT * FROM useri_ipeuri WHERE useru=ora_login_user AND
             ip_authorized=NVL(ora_client_ip_address,''));
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20510,
                'Nu aveti permisiunea de a va conecta de la acest calculator !');
    END ;
END ;
```

Verificăm funcționalitate declanșatorului creând un nou utilizator - `FOTACHEM2`. După creare și acordarea de drepturi, nu-l introducem în tabela `USERI_IPEURI`. La prima tentativă de logare declanșatorul va refuza sesiunea Oracle, după cum se observă în figura 11.15.

```
SQL> CREATE USER fotachem2 IDENTIFIED BY fotache2
2  DEFAULT TABLESPACE USERS TEMPORARY TABLESPACE TEMP
3  /

User created.

SQL> GRANT CONNECT, RESOURCE TO fotachem2 ;

Grant succeeded.

SQL> CONNECT FOTACHEM
Enter password: *****
Connected.
SQL> CONNECT FOTACHEM2
Enter password: *****
ERROR:
ORA-00604: error occurred at recursive SQL level 1
ORA-20510: Nu aveti permisiunea de a va conecta de la acest calculator !
ORA-06512: at line 11

Warning: You are no longer connected to ORACLE.
SQL> |
```

Figura 11.15. Funcționalitatea declanșatorului de logare la baza de date

11.8. Informații despre declanșatoare. Salvarea sub formă de script

O primă tabelă virtuală a dicționarului de date în care sunt informații generale despre declanșatoare (nume, id, data creării, data ultimei modificări, stare) este USER_OBJECTS. Interogarea următoare extrage numele și starea fiecărui trigger:

```
SELECT object_name, status
FROM user_objects
WHERE object_type = 'TRIGGER'
```

O altă tabelă virtuală, de data aceasta special dedicată acestui tip obiect procedural este USER_TRIGGERS - vezi figura 11.16.

SQL> desc user_triggers		
Name	Null?	Type
TRIGGER_NAME		VARCHAR2(30)
TRIGGER_TYPE		VARCHAR2(16)
TRIGGERING_EVENT		VARCHAR2(227)
TABLE_OWNER		VARCHAR2(30)
BASE_OBJECT_TYPE		VARCHAR2(16)
TABLE_NAME		VARCHAR2(30)
COLUMN_NAME		VARCHAR2(4000)
REFERENCING_NAMES		VARCHAR2(128)
WHEN_CLAUSE		VARCHAR2(4000)
STATUS		VARCHAR2(8)
DESCRIPTION		VARCHAR2(4000)
ACTION_TYPE		VARCHAR2(11)
TRIGGER_BODY		LONG

Figura 11.16. Atributele tabelii virtuale USER_TRIGGERS

Trigger_type se referă la momentul lansării BEFORE/AFTER EACH ROW/STATEMENT/EVENT, triggering_event conține comanda (INSERT/UPDATE/DELETE) sau evenimentul (LOGON...) asociat, base_object_type indică dacă declanșatorul este declarat pentru o tabelă, tabelă virtuală sau schemă, obiect al cărui nume este furnizat de următorul atribut, table_name, iar status precizează dacă declanșatorul este activat sau dezactivat - vezi și figura 11.17.

Un atribut interesant este description, deoarece furnizează o sinteză a informațiilor despre fiecare trigger, sinteză pe care o putem folosi la salvarea sub formă de script a unora sau tuturor declanșatoarelor. Similar scripturilor SQL*Plus de salvarea a tabelelor și restricțiilor prezentate în finalul capitoului 4, putem încropi un script pentru salvarea într-un fișier ASCII a tuturor declanșatoarelor din schema curentă - vezi listing 11.26.

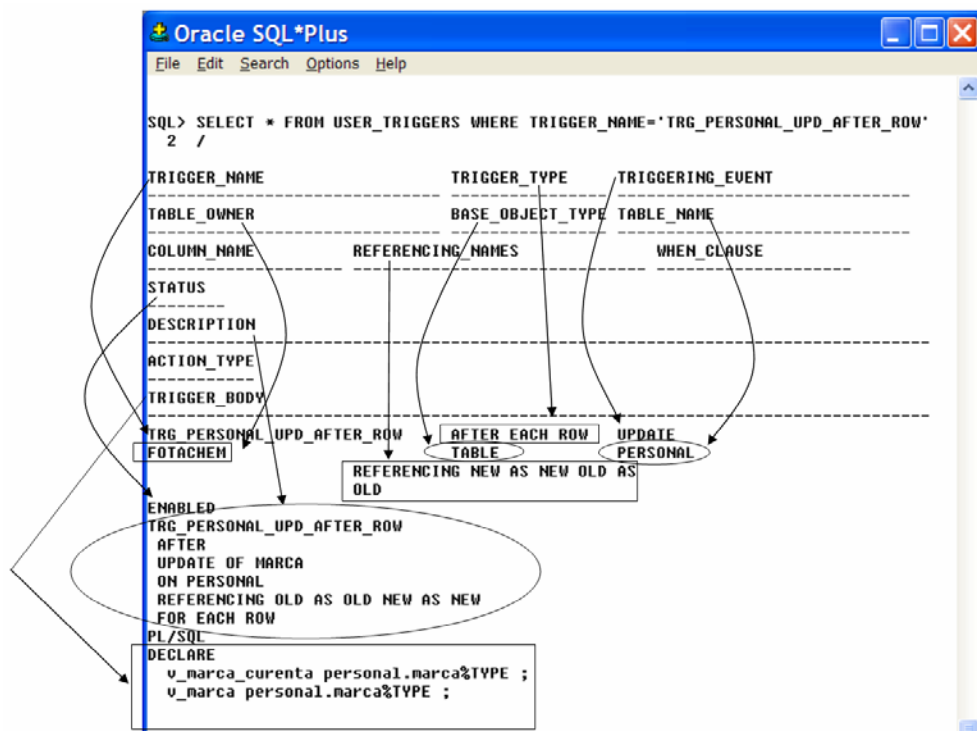


Figura 11.17. Informații despre declanșatorul TRG_PERSONAL_UPD_AFTER_ROW

Listing 11.26. Script SQL*Plus pentru salvarea declanșatoarelor sub formă de fișier text (re_creare_triggere.sql)

```
DROP TABLE temp1;

CREATE TABLE temp1 (
  descriere VARCHAR2(500),
  corp_trigger CLOB
);

INSERT INTO temp1
  SELECT description, TO_LOB(trigger_body)
FROM user_triggers;

COMMIT;

SPOOL f:\oracle_carte\cap11_declansatoare\re_creare_triggere.sql

SELECT 'CREATE OR REPLACE TRIGGER ' || descriere || ' ' ||
  CAST (corp_trigger AS VARCHAR2(3000)) || ' ';
AS "--Re-creare declansatoare"
FROM temp1;

SPOOL OFF
```


Ar mai fi de adăugat că putem reconstrui corpul declanșatoarelor și din tabela virtuală sistem USER_SOURCE. Mai mult, lucrurile se simplifică vizibil, întrucât nu există atribute de tip LONG. Scriptul din listing 11.27 salvează dintr-o singură frază SELECT corpul declanșatoarelor.

Listing 11.27. Reconstruirea declanșatoarelor pe baza tabeli virtuale USER_SOURCE

```
SPOOL f:\oracle_carte\cap11_declansatoare\re_creatre_triggere.sql
SELECT text
FROM
  (SELECT name, line, text
   FROM user_triggers ut INNER JOIN user_source us ON ut.trigger_name=us.name
   UNION
   SELECT trigger_name, 0, 'CREATE OR REPLACE '
   FROM user_triggers
   UNION
   SELECT trigger_name, -1, '-----'
   FROM USER_triggers
   UNION
   SELECT trigger_name, -2, ''
   FROM USER_triggers
   ORDER BY 1,2
  );
SPOOL OFF
```

În plus, între declanșatoare apare un rând vid și un altul cu liniuțe pentru a face mai ușor demarcația - vezi figura 11.18.

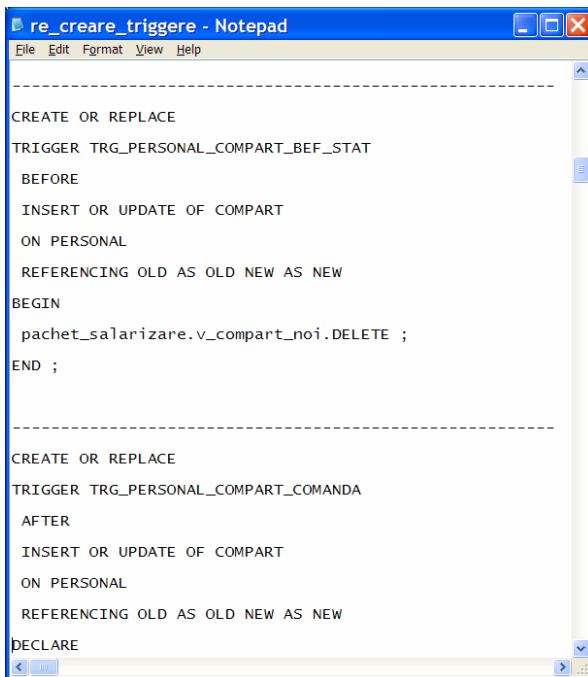


Figura 11.18. O porțiune din fișierul text ce conține corpul declanșatoarelor din schema curentă

11.9. Tipul de date LOB

Oracle oferă suport deplin pentru gestionarea fișierelor binare în baza de date prin intermediul atributelor de tip LOB (Large Objects). Fișierele pot conține date structurate (reprezentate sub formă de șiruri de caractere) sau date nestructurate (spre exemplu fișiere multimedia). În funcție de zona de păstrare a informațiilor, se disting două categorii de atribute LOB:

- *Cu stocare internă*, în interiorul bazei de date, a conținutului fișierului sursă (conținutul încărcat în baza de date va fi independent de modificările efectuate în fișierul sursă pe disc). În această categorie încadrăm următoarele tipuri de date: BLOB (Binary LOB-date nestructurate) și CLOB (Character LOB – date sub formă de șiruri de caractere).
- *Cu stocare externă* pe disc (în baza de date se păstrează doar adresa fișierului), categorie în care încadrăm tipul BFILE.

Notă. Pentru tipurile de date BLOB, tipul fișierului sursă nu are nici o relevanță atât timp cât în baza de date se va stoca o secvență de biți.

Secțiunea ce urmează este dedicată tipului BLOB, date nestructurate. Am amânat-o până în acest punct al lucrării deoarece, după cum veți vedea în finalul paragrafului, sunt necesare și câteva cunoștințe despre declanșatoare. Precizăm că tehnicile de manipulare sunt similare pentru tipul CLOB. Astfel, pornim de la presupunerea că se dorește a se păstra în baza de date, pentru fiecare angajat, fotografia și CV-ul (Curriculum Vitae). Fotografia va fi preluată dintr-un fișier de tip imagine (GIF, JPEG, etc.), iar CV-ul dintr-un fișier Word, Excel, PDF, ș.a.m.d. În acest sens vom adăuga la definiția tabelului PERSONAL (vezi capitolul 4 pentru detalii) două noi atribute de tip BLOB, după cum urmează:

```
ALTER TABLE Personal ADD ( foto BLOB, cv BLOB);
```

Valoarea (conținutul) unui atribut de tip BLOB este constituită din două elemente componente :

- **valoarea LOB** – șirul de biți ce formează conținutul fișierului încărcat;
- **adresa (pointer-ul) LOB** – indică localizarea fizică a conținutului LOB în baza de date.

Ca urmare, popularea cu date a unei tabeli ce conține atribute BLOB (sau CLOB) presupune parcurgerea a doi pași:

1. **crearea pointer-ului LOB** - se invocă funcția Oracle predefinită `empty_blob()` pentru atributele BLOB fie în cadrul unei fraze INSERT-SQL (pentru noile înregistrări) fie UPDATE-SQL pentru a modifica o adresă;
2. **încărcarea valorii LOB** la adresa creată în pasul anterior. Faza de încărcare a fișierului în baza de date se realizează cu ajutorul pachetului `DBMS_LOB` și presupune la rândul alți trei pași intermediari în următoarea ordine:

- a. blocarea înregistrării ce se dorește a fi actualizată;
- b. încărcarea conținutului fișierului prin intermediul procedurilor pachetului DBMS_LOB;
- c. comiterea tranzacției.

Important! Pentru ca server-ul Oracle sa poată accesa un fișier pe disc trebuie creat mai întâi obiectul DIRECTORY corespunzător directorului în care se găsesc fișierele ce trebuie încărcate. Astfel, presupunând că fișierele ce urmează a fi încărcate în baza de date se găsesc la adresa D:\TEMP\FISIERE_ANGAJATI\, construim un obiect DIRECTORY cu numele Fisiere_personal cu următoarea sintaxă (utilizatorul trebuie să dețină dreptul CREATE ANY DIRECTORY²):

```
CREATE DIRECTORY Fisiere_Personal  
AS 'D:\TEMP\FISIERE_ANGAJATI'
```

De asemenea, este foarte important ca, prin sistemul de operare, să acordăm **drept de citire** pentru directorul respectiv (în Win NT sau 2000 acest drept trebuie acordat utilizatorului *Everyone* pentru ca procesele de background ale server-ului Oracle să poată citi fișierele).

Listingul 11.28 prezintă un exemplu de încărcare a unei fotografii pentru angajatul cu marca 11119. Observăm că se respectă întocmai pașii precizați anterior:

1. se inserează o înregistrare nouă și se inițializează adresele pentru atributele BLOB cu funcția `empty_blob()`;
2. se extrage adresa atributului `foto` în variabila `b_lob` și blochează înregistrarea cu un `Select For Update`;
3. se inițializează o variabila `f_lob` (de tip BFILE) cu adresa fizică pe disc a fișierului `test.jpg` (fișierul se găsește pe disc la adresa `d:\temp\fisiere_angajati\`) prin intermediul funcției `BFILENAME()` ce preia drept argumente adresa fișierului identificată prin obiectul DIRECTORY creat anterior și numele fișierului (localizat în directorul respectiv)³;
4. se deschide fișierul de la adresa `f_lob` în mod `read-only` cu ajutorul procedurii `DBMS_LOB.FILEOPEN()`;
5. se încarcă conținutul fișierului în baza de date la adresa specificată de variabila `b_lob` (preluată anterior din atributul `foto`) cu ajutorul procedurii `DBMS_LOB.LOADFROMFILE()` (ultimul parametru specifică numărul de biți ce se dorește a fi încărcat și corespunde lungimii fișierului determinată cu funcția `DBMS_LOB.GETLENGTH(adresa_fisier)`);
6. se închide fișierul deschis de pe disc;

² Vezi capitolul de administrare pentru detalii privind acordarea drepturilor utilizatorilor

³ primul parametru al funcției `BFILENAME()` nu poate fi un șir de caractere reprezentând locația fizică a directorului pe disc ci numai un obiect DIRECTORY creat anterior

7. se "comite" tranzacția (fișierul trebuie închis înainte de commit)

Listing 11.28 Exemplu de încărcare a unui fișier în baza de date

```

INSERT INTO Personal VALUES(11119,'Test Blob', 'FIN', sysdate, 0,0,
'N', EMPTY_BLOB(), EMPTY_BLOB());

DECLARE
  f_lob BFILE;
  b_lob BLOB;
BEGIN
  SELECT foto INTO b_lob FROM Personal WHERE marca=11119 FOR UPDATE;

  f_lob := BFILENAME( 'Fisiere_Personal', 'test.jpg' );

  DBMS_LOB.FILEOPEN(f_lob, DBMS_LOB.FILE_READONLY);
  DBMS_LOB.LOADFROMFILE( b_lob, f_lob, DBMS_LOB.GETLENGTH(f_lob) );
  DBMS_LOB.FILECLOSE(f_lob);

  COMMIT;
END;
/

```

Trebuie precizat că un fișier BFILE trebuie întotdeauna închis pentru a nu ajunge în situația depășirii numărului maxim de fișiere deschise precizat prin parametrul `SESSION_MAX_OPEN_FILES` (în fișierul de inițializare `initSID.ora`). Din acest motiv este obligatorie definirea unui bloc de tratare a erorilor de genul:

```

EXCEPTION
  WHEN OTHERS THEN
    DBMS_LOB.FILECLOSE(f_lob);
    ROLLBACK;
    ...

```

Dacă totuși se întâmplă ca un bloc PL/SQL să se închidă anormal, putem apela procedura `DBMS_LOB.FILECLOSEALL()` ca ultimă modalitate de a închide toate fișierele deschise (o dată ce blocul PL/SQL s-a închis nu mai există posibilitatea recuperării pointer-ului BFILE utilizat în acea secvență).

Listingul 11.29 prezintă o procedură generalizată de încărcare a fișierelor corespunzătoare fotografiei și CV-ului unei persoane. O funcție nouă este utilizată pentru a verifica existența fișierelor la adresele specificate de parametri: `DBMS_LOB.FILEEXISTS(adresa_BFILE)` – returnează 1 dacă există un fișier pe disc la adresa specificată prin `adresa_BFILE` și 0 în caz contrar.

Listing 11.29. Procedura generalizată de încărcare a atributelor BLOB

```

CREATE OR REPLACE PROCEDURE Load_BLOB_From_FILE
(den_fisier_foto VARCHAR2, den_fisier_cv VARCHAR2, marcaAngajat INTEGER) IS
  adresa_foto BLOB;
  adresa_cv BLOB;
  fisier_foto BFILE;
  fisier_cv BFILE;

BEGIN

```

```
SELECT foto,cv INTO adresa_foto,adresa_cv
FROM Personal WHERE marca=marcaAngajat FOR UPDATE;
IF den_fisier_foto IS NOT NULL THEN
    fisier_foto:=BFILENAME('Fisiere_Personal',den_fisier_foto);
    IF DBMS_LOB.FILEEXISTS(fisier_foto)=0 THEN
        RAISE_APPLICATION_ERROR (-20001,'fisier foto inexistent pe disc');
    END IF;
    DBMS_LOB.FILEOPEN(fisier_foto,DBMS_LOB.FILE_READONLY);
    DBMS_LOB.LOADFROMFILE(adresa_foto,fisier_foto,DBMS_LOB.GETLENGTH(fisier_foto));
    DBMS_LOB.FILECLOSE(fisier_foto);
END IF;

IF den_fisier_cv IS NOT NULL THEN
    fisier_cv :=BFILENAME('Fisiere_Personal',den_fisier_cv);
    IF DBMS_LOB.FILEEXISTS(fisier_cv)=0 THEN
        RAISE_APPLICATION_ERROR (-20001,'fisier cv inexistent pe disc');
    END IF;
    DBMS_LOB.FILEOPEN(fisier_cv,DBMS_LOB.FILE_READONLY);
    DBMS_LOB.LOADFROMFILE(adresa_cv,fisier_cv,DBMS_LOB.GETLENGTH(fisier_cv));
    DBMS_LOB.FILECLOSE(fisier_cv);
END IF;

commit;

EXCEPTION
    WHEN OTHERS THEN
        DBMS_LOB.FILECLOSE(fisier_cv);
        DBMS_LOB.FILECLOSE(fisier_foto);
        rollback;

END;
/
```

Pentru a testa procedura de mai sus vom introduce o nouă înregistrare în tabela PERSONAL și vom încărca atributele foto și cv în două etape (vezi figura 11.19). În aceeași figură se observă că ultima încercare este sortită eșecului datorită faptului că nu există fișierul test pe calea specificată de obiectul Fisiere_Personal de tip DIRECTORY.

```
SQL> INSERT INTO Personal VALUES (2111111, 'Test2', 'FIN', sysdate, 0,0, 'N', EMPTY_BLOB(),EMPTY_BLOB());
1 row created.

SQL> EXECUTE Load_BLOB_From_FILE('Liviu_Cretu.jpg',null,21111);

PL/SQL procedure successfully completed.

SQL> EXECUTE Load_BLOB_From_FILE(null,'test_cv_excel.xls',21111);

PL/SQL procedure successfully completed.

SQL> EXECUTE Load_BLOB_From_FILE('test',null,21111);
BEGIN Load_BLOB_From_FILE('test',null,21111); END;

*
ERROR at line 1:
ORA-06502: PL/SQL: numeric or value error: invalid LOB locator specified:
ORA-22275
ORA-06512: at "SYS.DBMS_LOB", line 474
ORA-06512: at "LIVIU.LOAD_BLOB_FROM_FILE", line 34
ORA-20001: Fisier foto inexistent pe disc
ORA-06512: at line 1

SQL>
```

Figura 11.19. Apelul procedurii Load_BLOB_From_FILE

În final, precizăm că, de obicei, se construiește un trigger pentru inițializarea atributelor LOB (listing 11.30).

Listing 11.30. Declanșator pentru inițializarea adreselor LOB

```
CREATE TRIGGER init_Foto_CV BEFORE INSERT ON Personal FOR EACH ROW
BEGIN
    IF :NEW.foto IS NULL THEN
        :NEW.foto:= empty_blob();
    END IF;
    IF :NEW.cv IS NULL THEN
        :NEW.cv:=empty_blob();
    END IF;
END;
```