Fiola de SQL (20)

Declanșatoare generalizate dedicate restricțiilor referențiale. Cazul Visual FoxPro 6

– Marin Fotache

Fiola de astăzi este mai diluată în materie de SQL și vizează mai degrabă „cârpeli procedurale“ ale unor goluri evidente în SGBD-uri folosite pe scară largă, goluri în ceea ce privește opțiunile de instituire a restricțiilor referențiale. Stardardul SQL-92 prevede ca la crearea unei tabele să poată fi definit orice set de reguli pentru respectarea restricțiilor referențiale. Operațiunile care ar putea periclita o asemenea restricție sunt:

- ștergerea unei înregistrări din tabela părinte;
- modificarea cheii primare sau candidat (ce este atribut de legătură într-o restricție referențială) într-o tabelă părinte, în condițiile în care există înregistrări copil;
- inserarea unei linii într-o tabelă copil;
- modificarea valorii cheii străine (atributul de legătură) în tabela copil.

Prezervarea (îmi place termenul, nu m-am putut abține să nu-l folosesc) restricției referențiale presupune declararea opțiunii `RESTRICT` – pentru interzicerea sau `CASCADE` – pentru propagarea în cascadă a modificărilor în fiecare din situațiile de mai sus. Să luăm cvasi-cunoscutele tabele `CLIENTI`, `FACTURI`, `PRODUSE` și `LINIIFACT`, create după sintaxa SQL-92 astfel:

```
CREATE TABLE clienti (
     codcl NUMBER(6) CONSTRAINT pk_clienti PRIMARY KEY,
     dencl VARCHAR(30),
     codfiscal CHAR(9),
     localit VARCHAR(25) ) ;

CREATE TABLE produse (
     codpr NUMBER(6) CONSTRAINT pk_produse PRIMARY KEY,
     denpr VARCHAR(30),
     um VARCHAR2(10),
     procTVA NUMBER(2,2) DEFAULT .19 ) ;
```

Listing 1. Creare în VFP a celor patru tabele

```
CREATE TABLE clienti ( codcl NUMBER(6) PRIMARY KEY, dencl
       CHAR(30), ;
    codfiscal CHAR(9) NULL, localit CHAR(25) )

CREATE TABLE facturi ( nrfact NUMBER(8) PRIMARY KEY,
       datafact DATE DEFAULT DATE(), ;
     codcl NUMBER(6), FOREIGN KEY codcl TAG codcl REFER-
          ENCES clienti TAG codcl      ) ;

CREATE TABLE liniifact (nrfact NUMBER(8), linie NUMBER(2)
       CHECK (linie > 0) ;
       ERROR 'Atributul linie trebuie sa fie mai mare ca
             zero !', ;
     codpr NUMBER(6), cantitate NUMBER(10), pretunit NUMBER
          (12), ;
     PRIMARY KEY STR(nrfact,8)+STR(linie,2) TAG primaru, ;
    FOREIGN KEY nrfact TAG nrfact REFERENCES facturi TAG n-
          rfact, ;
    FOREIGN KEY codpr TAG codpr REFERENCES produse TAG
          codpr  )
```

```
CREATE TABLE facturi (
     nrfact NUMBER(8) CONSTRAINT pk_facturi PRIMARY KEY,
     datafact DATE,
     codcl NUMBER(6)   CONSTRAINT fk_facturi_clienti
           REFERENCES clienti(codcl)
ON DELETE RESTRICT ON UPDATE CASCADE  ) ;

CREATE TABLE liniifact (
     nrfact NUMBER(8) CONSTRAINT fk_liniifact_facturi
            REFERENCES facturi(nrfact)
ON DELETE RESTRICT ON UPDATE CASCADE ,
     linie NUMBER(2) CONSTRAINT ck_linie CHECK (linie > 0),
     codpr NUMBER(6) CONSTRAINT fk_liniifact_produse
           REFERENCES produse(codpr)
ON DELETE RESTRICT ON UPDATE CASCADE,
     cantitate NUMBER(10),
     pretunit NUMBER (12),
     CONSTRAINT pk_liniifact PRIMARY KEY (nrfact,linie)   );
```

În virtutea celor declarate, se interzice ștergerea unui client (o înregistrare părinte din `CLIENTI`) dacă există măcar o linie-copil în `FACTURI`. Dacă se modifică valoarea atributului `CodCl` din `CLIENTI`, atunci modificarea se va propaga în cascadă în toate liniile copil din `FACTURI`. Din oficiu, în virtutea clauzei `REFERENCES`, orice valoare a `CodCl` din `FACTURI` care nu există în `CLIENTI` va fi respinsă de SGBD-ul respectiv.

Lucrurile stau așa de frumos nu numai în standard, dar și câteva SGBD-uri, cum este cazul versiunii 7.1 a PostgreSQL. Pentru un SGBD *free*, situația este entuziasmantă, mai ales dacă ținem seama că mare parte din SGBD-urile existente pe piață și care ustură la pungă nu dispun de aceste facilități (nu sunt revoltat câtuși de puțin, deoarece, dacă nu existau aceste lacune ale SGBD-urilor în uz, trebuia să găsesc pentru acest număr al fiolei un alt subiect de discuție.

Declararea relațiilor permanente între tabelele unei baze de date VFP6

Astăzi ne ocupăm numai de restricțiile referențiale în Visual FoxPro. Aceasta deoarece sunt atât de multe de spus, încât pentru alte SGBD-uri trebuie să amânăm discuția, lucru la care, fie vorba între noi, mă pricep. VFP permite declararea interactivă (cu ajutorul mouse-ului) a regulilor implementate în vederea conformității cu restricțiile referențiale, însă pentru dezolvarea aplicațiilor profesionale acest lucru nu e de prea mare folos. Așa încât trebuie folosite comenzi `CREATE TABLE` și clauze `FOREIGN KEY` – vezi *listing 1*.

Din nefericire, clauza `FOREIGN KEY` instituie doar legături permanente între tabele și nu are nimic de a face cu restricția referențială ! Așa că singurul mod de a rezolva coerent problema este recurgerea la declanșatoare.

Practicienii preferă folosirea unui sistem uniform pentru denumirea declanșatoarelor. De exemplu, declanșatoarele pentru ștergerea, modificarea și inserarea liniilor în tabela `CLIENTI` pot fi denumite trg_del_clienti, `trg_upd_clienti` și `trg_ins_clienti`. Folosind funcția

Listing 2. Program pentru declararea declanşatoarelor fiecărei tabele din bază

```
DIME vTabele(1,1)
ADBOBJECTS(vTabele, "TABLE")
FOR i = 1 TO ALEN(vTabele)
   tab_ = vTabele(i)
   trigger_ = "trg_del_"+allt(tab_)+"()"
   CREATE TRIGGER ON &tab_ FOR DELETE AS &trigger_
   trigger_ = "trg_upd_"+allt(tab_)+"()"
   CREATE TRIGGER ON &tab_ FOR UPDATE AS &trigger_
   trigger_ = "trg_ins_"+allt(tab_)+"()"
   CREATE TRIGGER ON &tab_ FOR INSERT AS &trigger_
ENDFOR
```

ADBOBJECTS(), care stochează într-un vector informaţii despre tabelele bazei de date, programul din *listing 2* declară câte trei declanşatoare pentru toate tabelele bazei.

Procedură stocată de iniţializare a variabilelor

Din capul locului trebuie precizat că soluţia propusă în continuare funcţionează numai atunci când restricţiile referenţiale se stabilesc printr-un singur atribut, adică nu există nici o cheie străină compusă. În plus, indecşii simpli au acelaşi nume cu atributul de indexare în toate tabelele. De asemenea, nu sunt rezolvate restricţiile stabilite între două atribute ale aceleaşi tabele (de ex. în tabela PERSONAL atributul MarcaŞef este cheie străină, tabela părinte fiind tot PERSONAL, iar atributul de legătură al "părintelui" este Marca).

Începem cu o procedură stocată pentru deschiderea tabelelor şi iniţializarea variabilelor-tablou ce conţin informaţii despre tabelele bazei (vTabele) şi legăturile permanente dintre tabele – suportul restricţiilor referenţiale, astfel:

- vRelatii este un vector (masiv uni-dimensional) ce conţine câte o componentă pentru fiecare dintre relaţiile permanente stabilite între două tabele ale bazei;
- vCopii – tabela copil;
- vParinti – tabela părinte;
- vTaguriCopil – numele indexului elementar (tag-ului) al tabelei copil prin care se stabileşte legătura cu tabela părinte;
- vTaguriParinte - numele indexului elementar (tag-ului) al tabelei părinte prin care se stabileşte legătura cu tabela copil.

Toate aceste variabile sunt declarate publice, pentru a putea fi folosite în procedurile stocate. Codul sursă al procedurii setup este prezentat în *listing 3*.

Lucrul cu variabile de memorie este grozav de indicat, deoarece căutarea şi sortarea este foarte rapidă. Să parcurgem pe rând declanşatoarele generalizate.

Declanşatorul general pentru ştergere

Începem prezentarea cu declanşatorul pentru ştergerea unei linii din orice tabelă a bazei, al cărui cod sursă este cel din *listing 4*. La început se verifică dacă variabilele ce conţin informaţii despre relaţiile dintre tabele au fost iniţializate, scop pentru care se foloseşte funcţia TYPE(). La primul apel al vreunui declanşator se execută procedura setup prezentată mai sus.

În continuare, se scanează vectorul vParinti, identificându-se toate restricţiile referenţiale în care tabela din care se şterge înregistrarea este părinte. Pentru fiecare situaţie, se caută dacă în tabela copil există măcar o linie ce prezintă valoarea cheii străine identică cu cea a înregistrării de şters, caz în care se afişează un mesaj lămuritor şi se returnează valoarea FALSE (.F.), ştergerea fiind, deci, interzisă. Căutarea este rapidă deoarece se foloseşte funcţia INDEXSEEK() fără a se modifica pointerul tabelei copil.

Un alt avantaj net al variantei declanşatorului prezentată în listing ţine de faptul că toate cazurile „ligioase" pentru referenţialitate sunt afişate explicit, prin comparaţie cu secul „Trigger failed" ce apare în cazul declanşatoarelor create asistat (interactiv), facilitate importantă pentru aplicaţiile „neaoşe".

Declanşatorul general pentru inserare

Inserarea unei linii într-o tabelă a bazei de date poate periclita o restricţie referenţială numai dacă respectiva tabelă se află în postura de copil. *Listing 5* conţine corpul declanşatorului de inserare. Se verifică dacă tabela în care se adaugă noua înregistrare apare drept copil în restricţii referenţiale. În caz că da, se testează dacă valoarea cheii străine este egală cu valoarea de pe linia curentă a tabelei părinte sau dacă există o linie părinte (se foloşte din nou funcţia INDEXSEEK()). De exemplu, valoarea cheii străine (LINIIFACT.NrFact) poate fi, la un moment dat, egală cu valorea curentă a cheii primare (FACTURI.NrFact) dacă, după adăugarea unei înregistrări în părinte urmează adăugarea înregistrărilor copil, iar înregistrarea-părinte nu a fost "comisă" (scrisă efectiv din buffer); lucrurile depind atât de modul de adăugare (APPEND BLANK sau INSERT INTO...), cât şi de tipul buffering-ului ales.

Declanşatorul general pentru modificare

În fine, ultimul dintre cele trei declanşatoare - cel de actualizare - este şi cel mai dificil, deoarece, pe de o parte, trebuie să trateze ambele spete

Listing 3. Procedura stocată de iniţializare a variabilelor ce conţin informaţii despre relaţiile dintre tabele

```
*====================
PROCEDURE SETUP
*--------------------
* se deschide baza de date
IF !DBUSED('vinzari')
   OPEN DATABASE vinzari EXCLUSIVE
ENDIF

PUBLIC vTabele
DIME vTabele(5,1)
ADBOBJECTS(vTabele, "TABLE")
FOR i = 1 TO ALEN(vTabele)
   tab_ = vTabele(i)
   * daca tabela nu este deja deschisa, se asociaza unei
          zone de lucru
   IF !USED((tab_))
      USE (tab_) IN 0 EXCLU
   ENDIF
   * daca exista macar un index, primul (dintre indecsi)
          devine principal
   SELECT (tab_)
   IF TAGCOUNT() > 0
      SET ORDER TO TAG 1
   ENDIF
   PUBLIC trg_ins_&tab_, trg_upd_&tab_, trg_del_&tab_
ENDFOR

PUBLIC vRelatii, vCopii, vParinti, vTaguriCopil, vTaguri-
       Parinte
DIME vRelatii(4,4), vCopii(1), vParinti(1), vTaguri-
     Copil(1), vTaguriParinte(1)
ADBOBJECTS(vRelatii, 'Relation')
* prima coloana din vRelatii - tabela COPIL
* a doua coloana din vRelatii - tabela PARINTE
* a treia coloana din vRelatii - index (tag) COPIL
* a patra coloana din vRelatii - index(tag) PARINTE
DIME vCopii(ALEN(vRelatii,1)), vParinti(ALEN(vRelatii,1)),;
   vTaguriCopil(ALEN(vRelatii,1)),
          vTaguriParinte(ALEN(vRelatii,1))
FOR i = 1 TO ALEN(vRelatii,1)
   vCopii(i) = vRelatii(i,1)
   vParinti(i) = vRelatii(i,2)
   vTaguriCopil(i) = vRelatii(i,3)
   vTaguriParinte(i) = vRelatii(i,4)
ENDFOR

ENDPROC
```

Listing 4. Declanșatorul de ștergere, valabil pentru toate tabelele bazei

```
*-------------------------------------
*     triggeru' general de stergere
*-------------------------------------
PROCEDURE trg_del
parameter tabela__
LOCAL i, copil_, tagcopil_, tabela_, curent_, tagparinte_,
         nrindexparinte_, ;
   cheieparinte_, valoare_cheieparinte_,
                  sir_valoare_cheieparinte_

tabela_ = allt(upper(tabela__))
IF TYPE('vTabele') = "U" OR TYPE('vRelatii') = "U"
   do setup
ENDIF
trg_del_&tabela_ = .t.

set_deleted_ = SET('DELETE')
SET DELETE ON

*** cautam daca tabela curenta este una parinte
curent_ = 1
i = ASCAN(vParinti, tabela_, curent_)

DO WHILE i > 0
   * s-a gasit o noua tabela-copil pentru acest parinte
   tagparinte_ = ALLT(vTaguriParinte(i))
   nrindexparinte_ = TAGNO(tagparinte_, vParinti(i),
                     vParinti(i))
   cheieparinte_ = KEY(nrindexparinte_,vParinti(i))
   valoare_cheieparinte_ =
           EVALUATE(vParinti(i)+"."+cheieparinte_)

   copil_ = ALLT(vCopii(i))
   tagcopil_ = ALLT(vTaguriCopil(i))

   * se cauta in indexul (tag-ul) copil daca exista o
        inregistrare
   * care sa corespunda inegistrarii curente din parinte
   IF INDEXSEEK(valoare_cheieparinte_, .F., copil_,
      tagcopil_)
      DO CASE
      CASE TYPE(cheieparinte_) = 'C'
         sir_valoare_cheieparinte_ = valoare_cheieparinte_
      CASE TYPE(cheieparinte_) = 'N'
         sir_valoare_cheieparinte_ =
                     ALLT(STR(valoare_cheieparinte_))

      *** vezi si alte tipuri de date !!!!!!!!

      ENDCASE
      MESSAGEBOX('Nu puteti sterge aceasta linia din
   '+vParinti(i)+chr(13)+ ;
         ' in care cheia de indexare este
                   '+sir_valoare_cheieparinte_+chr(13)+ ;
         'deoarece exista linii copil in '+vCopii(i))
      SET DELETE &set_deleted_
      RETURN .F.
   ENDIF
   curent_ = i + 1
   i = ASCAN(vParinti, tabela_, curent_)
ENDDO

trg_del_&tabela_ = .f.
SET DELETE &set_deleted_
RETURN .T.
ENDPROC
```

Listing 5. Declanșatorul general de inserare

```
*-------------------------------------
*  declansator general pentru inserare
*-------------------------------------
PROCEDURE trg_ins
parameter tabela__
LOCAL i, copil_, tagcopil_, nrindexcopil_, cheiecopil_,
         valoare_cheiecopil_,;
   tabela_, curent_, parinte_, tagparinte_,
            nrindexparinte_, ;
   cheieparinte_, valoare_cheieparinte_,
           sir_valoare_cheiecopil_

tabela_ = allt(upper(tabela__))
IF TYPE('vTabele') = "U" OR TYPE('vRelatii') = "U"
   do setup
ENDIF
trg_ins_&tabela_ = .t.

set_deleted_ = SET('DELETE')
SET DELETE ON

*** cautam daca tabela curenta este una copil
local curent_
curent_ = 1
i = ASCAN(vCopii, tabela_, curent_)

DO WHILE i > 0
   * s-a gasit o (noua) tabela-parinte pentru acest copil
   copil_ = ALLT(vCopii(i))
   tagcopil_ = ALLT(vTaguriCopil(i))
   nrindexcopil_ = TAGNO(tagcopil_, vCopii(i), vCopii(i))
   cheiecopil_ = KEY(nrindexcopil_,vCopii(i))
   valoare_cheiecopil_ = EVALUATE(copil_+"."+cheiecopil_)

   parinte_ = ALLT(vParinti(i))
   tagparinte_ = ALLT(vTaguriParinte(i))
   nrindexparinte_ = TAGNO(tagparinte_, vParinti(i),
                     vParinti(i))
   cheieparinte_ = KEY(nrindexparinte_,vParinti(i))
   valoare_cheieparinte_ =
           EVALUATE(parinte_+"."+cheieparinte_)

   DO CASE
   CASE TYPE(cheiecopil_) = 'C'
      sir_valoare_cheiecopil_ = valoare_cheiecopil_
   CASE TYPE(cheiecopil_) = 'N'
      sir_valoare_cheiecopil_ =
                  ALLT(STR(valoare_cheiecopil_))

   *** vezi si alte tipuri de date !!!!!!!!

   ENDCASE

   * se cauta in indexul (tag-ul) parinte daca exista o
              integistrare
   * care sa corespunda inegistrarii curente din parinte
   IF valoare_cheiecopil_ # valoare_cheieparinte_ AND ;
          !INDEXSEEK(valoare_cheiecopil_, .F., parinte_,
                     tagparinte_)
      MESSAGEBOX('Noua valoare a '+cheiecopil_+'
                -'+sir_valoare_cheiecopil_+;
         '- din tabela '+vCopii(i)+chr(13)+ ;
         ' nu se regaseste in tabela parinte
                '+vParinti(i))
      SET DELETE &set_deleted_
      RETURN .F.
   ENDIF
   curent_ = i + 1
   i = ASCAN(vCopii, tabela_, curent_)
ENDDO

trg_ins_&tabela_ = .f.
SET DELETE &set_deleted_
RETURN .T.
ENDPROC
```

Listing 6. Declanșatorul general de modificare

```
*-------------------------------------
*  declansator general ptr. modificare
*-------------------------------------
PROCEDURE trg_upd
parameter tabela__
LOCAL i, copil_, tagcopil_, nrindexcopil_, cheiecopil_,
      valoare_veche_cheiecopil_,;
   valoare_noua_cheiecopil_, tabela_, curent_, parinte_,
      tagparinte_, nrindexparinte_, ;
   cheieparinte_, valoare_cheieparinte_,
      valoare_veche_cheieparinte_, ;
   valoare_noua_cheieparinte_, sir_valoare_cheiecopil_

tabela_ = allt(upper(tabela__))
IF TYPE('vTabele') = "U" OR TYPE('vRelatii') = "U"
   do setup
ENDIF
trg_upd_&tabela_ = .t.

set_deleted_ = SET('DELETE')
SET DELETE ON

*** cautam daca tabela curenta este una copil
local curent_
curent_ = 1
i = ASCAN(vCopii, tabela_, curent_)

DO WHILE i > 0

   * s-a gasit o (noua) tabela-parinte pentru acest copil
   copil_ = ALLT(vCopii(i))
   tagcopil_ = ALLT(vTaguriCopil(i))
   nrindexcopil_ = TAGNO(tagcopil_, vCopii(i), vCopii(i))
   cheiecopil_ = KEY(nrindexcopil_,vCopii(i))
   valoare_veche_cheiecopil_ = OLDVAL(cheiecopil_, copil_)
   valoare_noua_cheiecopil_ =
           EVALUATE(copil_+"."+cheiecopil_)

   IF valoare_veche_cheiecopil_  # valoare_noua_cheiecopil_
              AND _TRIGGERLEVEL <= 1
      parinte_ = ALLT(vParinti(i))
      tagparinte_ = ALLT(vTaguriParinte(i))
      nrindexparinte_ = TAGNO(tagparinte_, vParinti(i),
                        vParinti(i))
      cheieparinte_ = KEY(nrindexparinte_,vParinti(i))
      valoare_cheieparinte_ =
              EVALUATE(parinte_+"."+cheieparinte_)

      DO CASE
      CASE TYPE(cheiecopil_) = 'C'
         sir_valoare_cheiecopil_ =
                     valoare_noua_cheiecopil_
      CASE TYPE(cheiecopil_) = 'N'
         sir_valoare_cheiecopil_ =
                     ALLT(STR(valoare_noua_cheiecopil_))

      *** vezi si alte tipuri de date !!!!!!!!

   ENDCASE

      * se cauta in indexul (tag-ul) parinte daca exista o
           integistrare
      * care sa corespunda inegistrarii curente din
           parinte
      IF valoare_noua_cheiecopil_ # valoare_cheieparinte_
         AND ;
         !INDEXSEEK(valoare_noua_cheiecopil_, .F.,
                    parinte_, tagparinte_)
         MESSAGEBOX('Noua valoare a '+cheiecopil_+'
                    '+sir_valoare_cheiecopil_+;
            '- din tabela '+vCopii(i)+chr(13)+ ;
            ' nu se regaseste in tabela parinte
                     '+vParinti(i))
         SET DELETE &set_deleted_
         RETURN .F.
      ENDIF
   ENDIF
   curent_ = i + 1
   i = ASCAN(vCopii, tabela_, curent_)
ENDDO


*** daca tabela curenta este una parinte si se modifica
         cheia primara
** trebuie propagata modificarea in toti copiii
curent_ = 1
i = ASCAN(vParinti, tabela_, curent_)

DO WHILE i > 0

   * s-a gasit o (noua) tabela copil pentru acest parinte
   parinte_ = ALLT(vParinti(i))
   tagparinte_ = ALLT(vTaguriParinte(i))
   nrindexparinte_ = TAGNO(tagparinte_, vParinti(i),
                     vParinti(i))
   cheieparinte_ = KEY(nrindexparinte_,vParinti(i))
   valoare_veche_cheieparinte_ = OLDVAL(cheieparinte_,
                                 parinte_)
   valoare_noua_cheieparinte_ =
               EVALUATE(parinte_+"."+cheieparinte_)

   * daca s-a modificat atributul-cheie, atunci
         actualizarea
   * se propaga in tabela copil
   IF valoare_veche_cheieparinte_  #
            valoare_noua_cheieparinte_
      copil_ = ALLT(vCopii(i))
      tagcopil_ = ALLT(vTaguriCopil(i))
      nrindexcopil_ = TAGNO(tagcopil_, vCopii(i),
                      vCopii(i))
      cheiecopil_ = KEY(nrindexcopil_,vCopii(i))
      valoare_veche_cheiecopil_ = OLDVAL(cheiecopil_,
                                  copil_)
      valoare_noua_cheiecopil_ =
                  EVALUATE(copil_+"."+cheiecopil_)

      UPDATE &copil_ ;
         SET &tagcopil_ = valoare_noua_cheieparinte_ ;
         WHERE &tagcopil_ = valoare_veche_cheiecopil_

   ENDIF
   curent_ = i + 1
   i = ASCAN(vParinti, tabela_, curent_)
ENDDO

ENDIF trg_upd_&tabela_ = .f.
SET DELETE &set_deleted_
ENDPROC
```

posibile ale unei tabele – cea de copil și cea de părinte – în restricțiile referențiale în care este implicată și, în al doilea rând, pentru tabelele părinte, să propage în cascadă modificările cheilor primare în toate tabelele copil corespondente – vezi *listing 6*.

Din fericire, în SGBD-urile de care ne vom ocupa peste câteva fiole, standardul SQL în materie de declarare a restricțiilor referențiale este mult mai respectat. Singura problemă ce trebuie rezolvată prin declanșatoare rămâne, de obicei, actualizarea în cascadă (`UPDATE CASCADE`). Dar despre lucrurile acestea (și nu numai) *La anul și la mulți ani!*

Marin Fotache este ConferențiarDr. la Catedra de Informatică Economică, UAIC Iași, Facultatea de Economie și Administrarea Afacerilor. Poate fi contactat pe email la: fotache@uaic.ro. ■ 98