Fiola de SQL (18)

Exporturi FoxPro – Oracle/PostgreSQL – *Marin Fotache*

Exagerând (ca de obicei), se poate spune că scurta, dar tumultoasa viață a oricărui dezvoltator/(i)responsabil de aplicații are o singură constantă: schimbarea. Când un tânăr l-a întrebat pe Socrate ce să facă - să se însoare sau să rămână holtei - acesta i-ar fi răspuns: Tinere, oricum ai face, tot o să regreți !

Știu că nu o să fiți de acord cu ideea, dar lucrurile sunt, într-o oarecare măsură, similare și cu schimbarea. Dacă e prea abruptă, e sursă de stres, depresii, nevroze și belșug pentru farmacii. Cum să treci aplicația din FoxPro 2.5 în Oracle sau SQL Server în câteva luni, doar cu un teanc de cărți și câteva sfaturi ale unor amici/corespondenți (care, pe deasupra, mai și întâzie cu răspunsurile – orice asemănare cu persoana de față este pur întâmplătoare) ? Pe de altă parte, lipsa schimbării e și mai toxică. Aceleași ecrane în mod text, aceleași probleme presante lună de lună, an de an, senzația că nu faci nimic nou, că te plafonezi cu „bătrânele“ @SAY...GET-uri nu sunt cu nimic mai înălțătoare.

Așa că, dacă puteți, migrați ! Nu neapărat în Canada/SUA/UE, ci spre Oracle, SQL Server, DB2, Sybase, Informix – unde viza e cam scumpă, dar și câștigurile ulterioare sunt grase - sau Postgres, MySQL unde viza e chiar gratis !

Ceea ce ne propunem să discutăm astăzi ține de exportul datelor dintr-un SGBD de categorie ușoară (PC) într-un server de baze de date. Exemplul tipic (pentru noi) de SGBD „micro“ este FoxPro în care au fost dezvoltate majoritatea covârșitoare a aplicațiilor până în a doua parte a anilor ‘90. Cât privește serverele de baze de date, lucrurile sunt mai complicate. Noi ne vom opri la sintaxele SQL Oracle - pentru cei bogați și Postgres – pentru ceilalți. Sigur, alegerea celor trei produse este nedreaptă și cu totul arbitrară. Am recunoscut lucrul acesta și cu alte ocazii, și nu mi-a folosit la nimic...

Trebuie precizat de la bun început că mai toate serverele de baze de date au module proprii (loadere) de import a datelor din alte produse, iar sarcina transferului cade în sarcina administratorului bazei de date. De partea cealaltă, Visual FoxPro are un modul de migrare a bazei de date în SQL Server și Oracle. Noi însă vom face abstracție de aceste lucruri, încercând să formulăm soluții mai mult mai puțin generale pe baza unor comenzi SQL și proceduri.

Să presupunem că în urma utilizării unei aplicații FoxPro dispunem de o tabelă FACTURI cu următoarea structură:

```
CREATE TABLE facturi ( ;
nrfact NUMBER(8), ;
datafact DATE, ;
gestiune CHAR(4) ;
codcl NUMBER(6), ;
obs CHAR(50) ;
) ;
```

Am trecut cu vederea restricțiile de cheie primară, referențiale și alte reguli de validare (la nivel de câmp sau înregistrare), deoarece e posibil ca tabela să fi fost creată în versiuni FoxPro 2.x ce nu dispuneau de container (dicționar) de date.

Pe serverul BD (Oracle, PostgreSQL) schema bazei este deja creată, în sensul că există o tabelă cu același nume – FACTURI – ce trebuie populată cu toate înregistrările tabelei din FoxPro. Prima soluție, prezentată în listing 1, folosește o facilitate FoxPro de redirectare a comenzilor de ieșire într-un fișier text indicat prin comanda `SET TEXMERGE TO`. Backslash-ul (\) este similar semnului întrebării (?), în sensul că șirul, atributul sau funcția ce urmează sunt plasate la începutul unei linii noi în fișierul ASCII `listing_0.txt`. Backslash-ul dublu (\\) are funcțiune similară ??, iar parantezele unghiulare duble (<<...>>) determină includerea în fișierul-destinație a rezultatului evaluării funcției sau atributului.

Listing 1. Prima varianta FoxPro de generare a scriptului de populare a tabelei FACTURI în Oracle/PostgreSQL

```
*-----------------------
USE facturi
SET TEXTMERGE ON
SET TEXTMERGE TO 'listing_0.txt'
SCAN
   \INSERT INTO FACTURI VALUES (
      \\<<ALLT(STR(facturi.nrfact))>>,
      \\TO_DATE('<<DTOC(facturi.datafact)>>',
         'DD/MM/YYYY'),
      \\'<<ALLT(facturi.gestiune)>>'
         '<<ALLT(STR(facturi.codcl))>>'
      \\<<STR(facturi.valtotala)>>) ;
ENDSCAN
SET TEXTMERGE TO
SET TEXTMERGE OFF
RETURN
*-----------------------
```

Conținutul fișierului ASCII este în genul celui din figura 1.

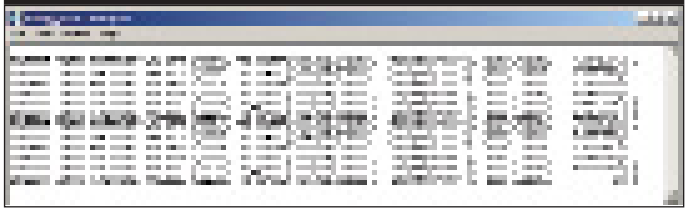
Programul din listing 1 poate fi adaptat sintaxei SQL a oricărui server BD în care se exportă liniile din FoxPro.

Această primă variantă are cel puțin două dezavantaje. Mai întâi, este procedurală. Nu că ar fi dezavantaj în sine. Dar din acesta decurge un al doilea neajuns – specificitatea, adică imposibilitatea folosirii și la alte SGBD-uri client. Este drept, cam orice SGBD are comenzi pentru implementarea unui asemenea mecanism.

Mult mai tentantă este însă generarea fișierului-script doar printr-o simplă comandă SQL:

```
SELECT "INSERT INTO FACTURI VALUES
         ("+ALLT(STR(facturi.nrfact))+","+;
      "TO_DATE('"+DTOC(facturi.datafact)+"',
         'DD/MM/YYYY')"+;
```

Figura 1. Scriptul pentru „importul“ conținutului tabelei FACTURI din FoxPro în Oracle/PostgreSQL



```
      ",'"+ALLT(facturi.gestiune)+"',
         "+ALLT(STR(facturi.codcl))+",
         "+STR(facturi.valtotala)+") ;" ;
FROM FACTURI ;
TO FILE listing_0 NOCONSOLE PLAIN
```

Persistând în greșeala „specificității" (FoxPro), încercăm să generalizăm gradual lucrile, și vom începe cu discutarea unei proceduri care generalizează exportul liniilor din FoxPro, indiferent de câte atribute conține tabela și tipologia lor. Relația SQL-generalizare ne duce cu gândul la variabile, eventual macrosubstuție. Așa vă n-ar strica să lămurim câteva chestiuni în acest sens. Secvența următoare:

```
sir_ = "INSERT INTO FACTURI VALUES
         ("+ALLT(STR(facturi.nrfact))+","+;
   "TO_DATE('"+DTOC(facturi.datafact)+"', 'DD/MM/YYYY')"+
         ", "+ ;
   "'"+ALLT(facturi.gestiune)+"',
      "+ALLT(STR(facturi.codcl))+","+;
STR(facturi.valtotala)+") ;"

SELECT sir_  FROM FACTURI  TO FILE listing_0
         NOCONSOLE PLAIN ;
```

ar trebui, aparent, să obțină un conținut identic pentru fișierul ASCII. Cu toate acestea, rezultatul este cu totul altul – vezi figura 2.

Explicația este simplă: evaluarea variabilei `sir_` are loc înaintea de fraza `SELECT`. De fapt, dacă am scoate alias-ul tabelei `FACTURI` din prima comandă, sau dacă tabela `FACTURI` n-ar fi deschisă, am recepționa un mesaj de eroare. Pentru ca evaluarea expresiei să se facă la fiecare linie a tabelei, putem recurge un artificiu – o funcție care să întoarcă tocmai șirul clauzei `SELECT`:

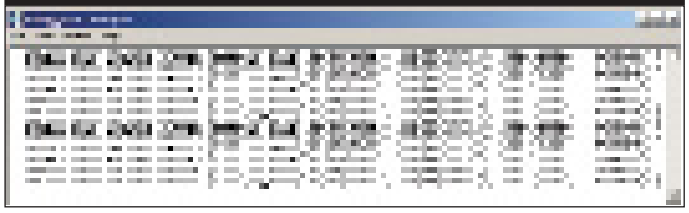
```
SELECT f() ;
FROM facturi ;
TO FILE listing_0 NOCONSOLE PLAIN  WHERE nrfact < 1130

FUNCTION f
sir_ = "INSERT INTO FACTURI VALUES
         ("+ALLT(STR(facturi.nrfact))+","+;
"TO_DATE('"+DTOC(facturi.datafact)+"', 'DD/MM/YYYY')"+
         ", "+;
"'"+ALLT(facturi.gestiune)+"',"+ALLT(STR(facturi.codcl))+;
","+STR(facturi.valtotala)+") ; "
RETURN sir_
```

Exportul liniilor tabelei FACTURI indiferent de structura acesteia

Varianta pe care v-o propunem în continuare a fost realizată în VFP6, însă poate fi ușor adaptată la versiunile 2.x ale FoxPro. Ideea este de copia structura tabelei de exportat într-o altă tabelă ad-hoc numită TEMP și construirea dinamică a clauzei `SELECT` generatoare a fișierului ASCII ce va servi drept script de populare a tabelei cu același nume în Oracle/PostgreSQL.

Figura 2. Primul eșec în folosirea unei variabile dedicată clauzei SELECT



Listing 2 Script pentru exportul înregistrărilor tabelei FACTURI – indiferent de structura acesteia

```
* se inschide tabela TEMP in care se memoreaza structura
         tabelei de exportat
IF USED('temp')
   SELECT temp
   USE
ENDIF

* se deschide tabela facturi
IF !USED('facturi')
   USE facturi IN 0
ENDIF

* se copiaza structura tabelei FACTURI in tabela TEMP
SELECT facturi
COPY STRUCTURE TO temp EXTENDED FIELDS nrfact, datafact,
         gestiune, codcl, valtotala

USE temp IN 0
SELECT temp

* fraza SELECT foloseste functia f2
SELECT f2() FROM FACTURI TO FILE listing_0 NOCONSOLE PLAIN

*-----------------
* continului functiei f2
FUNCTION f2
IF TYPE('sir_') = 'U'
   PUBLIC sir_
ENDIF
alias_ = ALIAS()

* partea de inceput din variabil sir_
sir_ = "INSERT INTO FACTURI VALUES ("

* se parcurg toate liniile din TEMP - fiecare reprezinta
* un atribut ce trebuie inclus in sir_
SELECT temp
SCAN
   SELE (alias_)
   IF LEN(sir_) > 31
      sir_ = sir_ + ", "
   ENDIF
   nume_ = alias_+"."+temp.field_name
   DO CASE
   CASE temp.field_type = "N"
      sir_ = sir_ + ALLT(STR( &nume_, temp.field_len,
            temp.field_dec) )
   CASE temp.field_type = "C"
      sir_ = sir_ + "'"+ALLT( &nume_ )+"'"
   CASE temp.field_type = "D"
      sir_ = sir_ + "TO_DATE('"+DTOC( &nume_ ) + "',
            'DD/MM/YYYY')"
   ENDCASE
   SELECT temp
ENDSCAN
sir_ = sir_ + ") ;"
SELE (alias_)
RETURN sir_
*-----------------
```

Pe calapodul soluției anterioare, conținutul clauzei `SELECT` va fi furnizat de o funcție – `F2`. Iată și procedura – vezi listing 2.

Deși poate iritantă și sigur consumatoare de timp, reconstruirea variabilei `sir_` la fiecare înregistrare a tabelei `FACTURI` este necesară pentru preluarea corectă a valorilor atributelor din înregistrarea respectivă. Un spor de viteză poate fi obținut prin copierea structurii tabelei într-un masiv de memorie.

Listing 3 Procedură generală pentru pentru exportul înregistrărilor din toate tabelele unei baze de date VFP în Oracle/PostgreSQL

```
*******************************************
** exportul tuturor tabelelor
*******************************************

* se sterge fisierul ASCII destinatie
IF FILE('listing_0.txt')
   DELETE FILE listing_0.txt
ENDIF

* declararea variabilelor publice
IF TYPE('sir_') = 'U'
   PUBLIC sir_
ENDIF
PUBLIC ARRAY vTabele(5,1), mStructura (1,1), vAtribute(1),
          vTip(1), ;
   vLungime(1), vPozFractionare(1)

* deschiderea bazei
IF !DBUSED('vinzari')
   OPEN DATABASE vinzari
ENDIF

* se preiau numele tabelelor
ADBOBJECTS(vTabele, "TABLE")
FOR i = 1 TO ALEN(vTabele)
   tab_ = vTabele(i)
   IF !USED((tab_))
      USE (tab_) IN 0 EXCLU
   ENDIF
   DO export_tabela  WITH tab_
ENDFOR
RETURN
*******************************************

*=========================================
PROCEDURE export_tabela
PARAMETER tabela_
AFIELDS(mStructura, (tabela_))
nr_atrib = ALEN(mStructura,1)
DIMENSION vAtribute(nr_atrib), vTip(nr_atrib), ;
   vLungime(nr_atrib), vPozFractionare(nr_atrib)

FOR j = 1 TO nr_atrib
   vAtribute(j) = mStructura(j,1)
   vTip(j) = mStructura(j,2)
   vLungime(j) = mStructura(j,3)
   vPozFractionare(j) = mStructura(j,4)
ENDFOR

* fraza SELECT foloseste functia CLAUZA_SELECT
SELECT clauza_select(tabela_) FROM (tabela_) ;
   TO FILE listing_0 ADDITIVE NOCONSOLE PLAIN

ENDPROC
*=========================================

*-----------------------------------
* functia CLAUZA_SELECT
*-----------------------------------
FUNCTION clauza_select
parameter tabela_
alias_ = ALIAS()

* partea de inceput din variabila sir_
sir_ = "INSERT INTO "+tabela_+ " VALUES ("

lungime_ = 30 + len(tabela_)

* se parcurg toate componentele vectorului vAtribute
FOR k = 1 TO ALEN(vAtribute)
   SELE (alias_)
   IF k > 1
      sir_ = sir_ + ", "
   ENDIF
   nume_ = alias_+"." + vAtribute(k)
   DO CASE
   CASE vTip(k) = "N"
      sir_ = sir_ + NVL( ALLT(STR( &nume_, vLungime(k), ;
         vPozFractionare(k) ) ) , "NULL")
   CASE vTip(k) = "C"
      sir_ = sir_ + NVL( "'"+ALLT( &nume_ )+"'" , "NULL" )
   CASE vTip(k) = "D"
      sir_ = sir_ + NVL ( "TO_DATE('"+DTOC( &nume_ ) + ;
          "', 'DD/MM/YYYY')" , "NULL" )
   ENDCASE
   lungime_ = lungime_ + vLungime(k) + 2  && 2 pt. virgula
              si spatiu
ENDFOR
sir_ = sir_ + ");"
SELE (alias_)
* lungimea maxim posibila pentru un INSERT al acestei
              tabele
RETURN PADR(sir_,lungime_ + 2)
*-----------------------------------
```

Exportul tuturor liniilor tabelelor dintr-o bazei de date

Pentru versiunile mai recente ale FoxPro – de la VFP 3 (lansată în 1995) încoace – se poate construi o procedură care să automatizeze întreg procesul de export al înregistrărilor – vezi *listing 3*. Funcția `ADBOBJECTS (vTabele, "TABLE")` stochează în variabila (vector) de memorie `vTabele` numele tuturor tabelelor aflate în bază. Programul principală lansează, pentru fiecare tabelă în parte, procedura `export_tabela`.

Din rațiuni de viteză, structura unei tabele nu mai este stocată în tabela `TEMP`, ci într-un masiv de memorie - `mStructura` – cu ajutorul funcției `AFIELDS()`. Pentru comoditate, din acest masiv sunt create vectori cu nume mai sugestive – `vAtribute`, `vTip`, `vLungime` și `vPozFractionare` care conțin câte un element pentru fiecare atribut al tabelei curente.

Funcția care întoarce comanda `INSERT` pentru fiecare linie este numită mai sugestiv `clauza_select`, iar salvarea în fișierul ASCII se realizează folosind clauza `ADDITIVE`. Cel mai important câștig de viteză este legat de parcurgerea vectorului `vAtribute` (în locul scanării tabelei `TEMP`).

În fine, ar mai fi de subliniat două „finețuri“, prin comparație cu versiunea anterioară a funcției. Mai întâi, sunt avute în vedere și eventualele valori `NULL`e ce pot apărea și care, netratate, compromit exportul înregistrărilor din tabela respectivă, scop în care s-a folosit funcția `NVL` (vezi și cele două fiole „închinate“ exclusiv `NULL`ităților SQL). În al doilea rând, VFP prezintă un neajuns ce ar prejudicia iremediabil conținutul fișierului-script `listing_0.txt`: lungimea șirului extras printr-o frază `SELECT` depinde de lungimea șirului din prima înregistrare returnată. Pentru o justă dimensionare a lungimii comenzii `INSERT`, se folosește variabila `lungime_`, care adună 30 de caractere (cam atât ar lua cuvintele `INSERT INTO` și `VALUES`, cu spațiile și paranteza deschisă) plus lungimea numelui tabelei, lungimea maximă a fiecărui atribut (cu virgula și spațiul dintre atribute) plus ultimele două caractere care „sfârșesc“ comanda `INSERT` – paranteza închisă și punct-virgula.

Marin Fotache este Conferențiar Dr. la Catedra de Informatică Economică, UAIC Iași, Facultatea de Economie și Administrarea Afacerilor. Poate fi contactat pe email la: fotache@uaic.ro. ■ 87