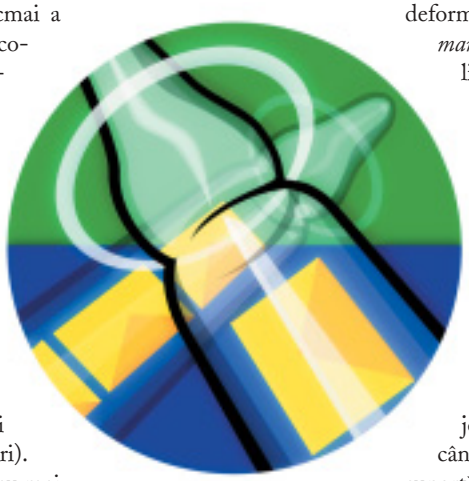
Fiola de SQL (22)

Duplicitate

—Marin Fotache

Prezența, într-o tabelă, a două linii absolut identice a fost considerată o erezie de la bun început, atât de către fondatorul (F.F. Codd), cât și de cei mai cunoscuți exponenți (precum C.J. Date) ai modelului relațional al bazelor de date. La urma urmei, lucrurile erau firești: spre deosebire de modelele anterioare, relaționalul permite accesul la o anumită înregistrare exclusiv prin valorile atributelor de pe linia respectivă. Ceea ce nu-i rău deloc, deoarece utilizatorul/programatorul nu mai este interesat de modul navigare printre înregistrări, de adrese și pointeri, ci exclusiv de conținut.

Membrii comitetului de standardizare a SQL au fost însă mult mai largi la inimă. După cum mărturisește Joe Celko, un argument important în acest sens a fost „problema mâncării de pisică" formulată de Len Gallagher și David Beech care au luat în discuție cazul bonului de la casa unui supermagazin pe care apare repetat codul unei cutii de mâncare pentru pisici, atunci când stăpânul pisicii cumpără două sau multe (cutii, bineînțeles). Reflectarea întocmai a datelor de la casa de marcat presupune repetarea codului fiecărui produs de un număr de ori care indică în ce cantitate a fost cumpărat.



Dincolo de teorie, eliminarea liniilor-duplicat sau valorilor identice ale unor atribute din cheia primară sau candidată a unei relații reprezintă o problemă întâlnită frecvent pe grupurile de discuții dedicate limbajului SQL. Iar cei care au avut ocazia să porteze baze de date de pe un SGBD xBase, gen Foxpro 2.x, pe un server de bază de date știu de ce...

Să presupunem că lucrăm la o firmă care-și vinde produsele exclusiv prin agenți itineranți (pentru un plus de idilic, toți agenții au laptop-uri). În timp, este posibil ca, în funcție de ofertă, doi sau mai mulți distribuitori ai firmei să vândă produse aceluiași client. Periodic, agenții transmit bazele de date proprii la sediul local, regional sau central, unde are loc fuziunea înregistrărilor. Dacă în schema bazei există tabela `CLIENTI_PF` (`PF` vine de la persoane fizice, pentru a le deosebi de clienții care sunt organizații sau firme, adică persoanele juridice):

```
CREATE TABLE clienti_pf (
    cnp VARCHAR2(14),
    numepren VARCHAR2(40),
    adresa VARCHAR2(70),
    codpostal CHAR(5),
    loc VARCHAR2(15),
    judet VARCHAR2(20),
    telefon CHAR(9),
    data_nast DATE,
    serie_nr_bi CHAR(8)
    )
```

atunci este evident că apariția pe două sau mai multe linii a unuia și aceluiași client (persoană) este iminentă. Aflarea liniilor identice, în condițiile în care `CNP` este corect, presupune banala interogare:

```
SELECT cnp, numepren
FROM clienti_pf
GROUP BY cnp, numepren
HAVING COUNT(*) > 1
```

Pe noi ne interesează însă formularea a câteva soluții în vederea eliminării tuturor tuplurilor care se repetă.

O soluție brutală

Una dintre cele mai primitive variante constă în crearea unei tabele-oglindă în care fiecare linie este unică (deci oglinda e cam deformată, așa că mai bine îi spuneam *tabelă de manevră*), prin folosirea clauzei `DISTINCT`. Se șterg liniile tabelei sursă și apoi se re-populează cu liniile din tabela de manevră:

```
CREATE TABLE clienti_pf2 AS
  (SELECT DISTINCT * FROM clienti_pf) ;
 DELETE FROM clienti_pf ;
 INSERT INTO clienti_pf
  (SELECT * FROM clienti_pf2)     ;
COMMIT ;
```

Chiar dacă sintaxa este una Oracle / PostgreSQL, această variantă funcționează pe majoritatea SGBD-urilor. Probleme apar atunci când sintaxa comenzii `CREATE TABLE` sau `INSERT` nu suportă subconsultări. Este situația Visual FoxPro. Dacă tabela `CLIENȚI` nu este prea mare, în locul tabelei oglindă se poate folosi un vector. Iată succesiunea comenzilor:

```
SELECT DISTINCT * FROM clienti_pf INTO ARRAY vClienti_pf2
DELETE FROM clienti_pf
SELECT clienti_pf
PACK
INSERT INTO clienti_pf FROM ARRAY vClienti_pf2
```

Dacă însă volumul de date poate depășește capacitatea de memorare a unei variabile-masiv, se poate recurge la tabela `CLIENTI_PF2`, înregistrările acesteia fiind preluate cu `APPEND FROM`:

```
SELECT DISTINCT * FROM clienti_pf INTO TABLE clienti_pf2
DELETE FROM clienti_pf
SELECT clienti_pf
PACK
APPEND FROM clienti_pf2
```

Brutalitatea maximă a acestei variante poate fi atinsă prin ştergerea tabelei-sursă după crearea CLIENTI_PF2, şi redenumirea acesteia din urmă, ceva în genul miniscriptului:

```
CREATE TABLE clienti_pf2 AS
   (SELECT DISTINCT *  FROM clienti_pf) ;
DROP TABLE clienti_pf ;
RENAME clienti_pf2 TO clienti_pf2 ;
```

Ştergerea liniilor redundante

Soluţiile ce pot fi formulate pentru eliminarea liniilor redundante, fără salvarea intermediară a înregistrărilor într-o tabelă de manevră, diferă sensibil de la SGBD la SGBD. Aceasta deoarece modul de identificare a unei linii, altfel decât prin conţinutul atributelor, este diferit.

Să începem cu sintaxa Oracle. Fiecare linie a unei tabele are un identificator unic – ROWID - destul de criptic şi care care conţine date legate de adresa fizică, pe disc, a înregistrării. Ordinea cronologică în care au fost introduse înregistrările nu este cea a ROWID, de aceea acesta nu trebuie folosit precum RECNO()-ul din VFP. Totuşi la eliminarea dublurilor ne este de mare folos:

```
DELETE FROM clienti_pf WHERE ROWID NOT IN
    (SELECT MIN(ROWID) FROM clienti_pf  GROUP BY cnp)
```

În subconsultare, se constituie câte un grup pentru fiecare persoană (CNP). Fiecărui grup i se determină linia cu cel mai mic ROWID. DELETE-ul principal elimină toate înregistrările ale căror identificator nu se regăseşte în setul extras de subconsultare.

Firește, nu-i de lepădat nici varianta ce foloseşte o subconsultare corelată:

```
DELETE FROM clienti_pf c1 WHERE ROWID >
    (SELECT MIN(c2.ROWID)
     FROM clienti_pf c2
     WHERE c1.cnp=c2.cnp )
```

Pentru fiecare înregistrare din prima instanţă, c1, a CLIENTI_PF, se determină cel mai mic ROWID dintre liniile cu acelaşi c1.cnp. Dacă c1.ROWID este mai mare decât valoarea extrasă de subconsultare, linia respectivă din c1 este ştearsă. Întrucât unde-i corelare începe şi EXISTS, ultima comandă se poate scrie şi sub forma:

```
DELETE FROM clienti_pf c1 WHERE EXISTS
    (SELECT 1
     FROM clienti_pf c2
     WHERE c1.cnp=c2.cnp AND c1.ROWID > c2.ROWID)
```

O soluţie elegantă se bazează pe utilizarea unei subconsultări care extrage, pentru fiecare grup de înregistrări (valoare comună a CNP), liniile în care ROWID nu este cel minim, printr-o theta-joncţiune cu o tabelă ad-hoc numită LINII_MIN ce conţine liniile unice care interesează în final:

```
DELETE FROM clienti_pf WHERE ROWID IN
    (SELECT c1.ROWID
     FROM clienti_pf c1,
        (SELECT cnp, MIN(ROWID) AS rowid_min
         FROM clienti_pf
         GROUP BY cnp
         ) linii_min
     WHERE c1.cnp=linii_min.cnp AND c1.ROWID <>
                 linii_min.rowid_min )
```

Trucul este posibil datorită toleranţei Oracle la prezenţa frazelor SELECT în clauze FROM.

În Visual FoxPro fiecare înregistrare are un număr care reprezintă ordinea fizică a liniei respectice în tabelă şi care poate fi obţinut atât la listare (comenzile LIST, DISPLAY), cât şi prin funcţia RECNO(). Din nefericire, funcţia RECNO() nu e de mare folos pentru interogări de genul celor de mai sus. Spre exemplu, rezultatul interogării:

```
SELECT MIN(RECNO()), * FROM clienti_pf GROUP BY cnp
```

conţine aceeaşi valoare a minimului pentru toate coloanele. Trecerea peste acest impas vine de la folosirea unei funcţii – INREG_MIN_CNP - apelată din comanda DELETE, astfel:

```
DELETE FROM clienti_pf WHERE RECNO() <> inreg_min_cnp(cnp)

FUNCTION INREG_MIN_CNP
PARAMETER cnp_
LOCAL rec_min(1)
SELECT MIN(RECNO()) FROM clienti_pf INTO ARRAY rec_min WHERE
   cnp = cnp_
RETURN rec_min(1,1)
ENDFUNC
```

Practic, pentru fiecare înregistrare din CLIENTI_PF se verifică dacă numărul său este diferit ce numărul returnat de funcţie. Funcţia determină cel mai mic număr de înregistrare pentru un CNP dat.

PostgreSQL pune la dispoziţie o pseudo-coloană (sau coloană sistem) asemnănătoare ROWID-ului din Oracle, şi anume CTID care este o pereche de valori formată din numărul blocului şi indexul tuplului în cadrul blocului. Cu toate acestea, soluţiile pe care vi le supun atenţiei se bazează pe o altă coloană sistem – OID – care este un număr unic atribuit de SGBD fiecărei linii a unei tabele. OID poate fi comparat pseudo-coloanei ROWNUM din Oracle. Iată şi transpunerea primei soluţii Oracle în PostgreSQL:

```
DELETE FROM clienti_pf WHERE OID NOT IN
(SELECT MIN(OID) FROM clienti_pf GROUP BY cnp)
```

şi a ultimeia:

```
DELETE FROM clienti_pf WHERE OID IN
 (SELECT c1.OID
 FROM clienti_pf c1,
        (SELECT cnp, MIN(OID) AS rowid_min
         FROM clienti_pf
         GROUP BY cnp
         ) linii_min
WHERE c1.cnp=linii_min.cnp AND c1.OID <> linii_min.rowid_min)
```

Paradoxal, deşi între Oracle şi PostgreSQL se pot face analogiile pe care le menţionam, ROWID – CTID şi ROWNUM – OID, interogările care funcţionează corect în cele două SGBD-uri folosesc numai ROWID, respectiv OID.

Firește, mai sunt sunt şi alte variante pe care nu le-am discutat. Spre exemplu, în DB2 se poate crea o tabelă virtuală actualizabilă în care numerotarea liniilor în cadrul unui unui CNP (grup) comun se realizează prin funcţia OLAP cu numele ROWNUMBER(). În etapa a doua, din această tabelă virtuală (şi, implicit, din tabela de bază) se vor şterge toate liniile cu număr mai mare ca 1.

Dar despre numerotarea liniilor într-un rezultat avem un rendez-vous special. Pe data viitoare!

Marin Fotache este conferenţiar dr. la Catedra de Informatică Economică, UAIC Iaşi, Facultatea de Economie şi Administrarea Afacerilor. Poate fi contactat pe email la: fotache@uaic.ro. ■ 98