Capitolul 10. Temporalitate şi acces în baze de date

Un criteriu esențial în evaluarea calității unei baze de date este cel al timpului. Validitatea temporală, mai precis, lipsa de adecvare a schemei bazei la modificările ce apar ulterior în sistem, reprezintă coşmarul multor administratori şi dezvoltatori de aplicații. Ideal ar fi ca în momentul proiectării schemei bazei să se cunoască direcțiile în care evoluează domeniul, legislația şi procedurile modulului/sistemului pentru care se dezvoltă aplicația, astfel încât structura să fie suficient de flexibilă pentru adaptări ulterioare. Echipele norocoase au câte un Brucan al lor, sau, după caz, o Mama Omida. Celelalte trebuie să procedeze ca noi, adică să-şi bată capul identificând eventualele probleme, iminente sau mai îndepărtate. Şi, după cum vedea, problemele sunt mult mai aproape decât credem...

10.1. Validitatea temporală a unei scheme

Baza de date dedicată vânzărilor, mai precis triada: PRODUSE {CodPr, DenPr, UM, ProcTVA, Grupa}, FACT {NrFact, CodCl, DataFact, Obs} şi LINII_FACT {NrFact, Linie, CodPr, Cantitate, PreţUnit} (vezi paragraful 5.2), este mai mult decât cunoscută, este celebră. Puține sunt cărțile dedicate bazelor de date care să reziste tentației de a nu da un exemplu de acest gen. Fireşte, autorii americani sunt „scutiți" de TVA, chiar dacă le rămân suficient de multe alte taxe. Ei bine, pentru mediul economic european, şi mai ales pentru mediul de afaceri românesc, această schemă, celebră cum e ea, este inadecvată pe termen lung, ba chiar mediu !

De ce ? Simplu ! Valoarea oricărei facturi se calculează, în lipsa unor atribute redundante, cu binecunoscuta interogare:

```
SELECT SUM(cantitate * pretunit) AS ValFactFaraTVA,
    SUM(cantitate * pretunit * (1 * procTVA)) AS ValFactTotala
FROM fact f
    INNER JOIN linii_fact lf ON f.nrfact=lf.nrfact
    INNER JOIN produse p ON lf.codpr=p.codpr
WHERE f.nrfact = 1
```

TVA-ul, pentru fiecare linie din factură, se determină pornindu-se de la ProcTVA din produse, dar acest atribut indică doar procentul *curent* al TVA pentru produsul cu pricina. Să presupunem că un super-market deschis în 1998 şi destul de bine automatizat (case de marcat, stocuri on-line) vinde, prin altele, şi cărți. Procentul de TVA a cunoscut variantii sensibile în ultimii ani. Atunci, dacă în luna februarie 2005, dorim să analizăm vânzările de cărți pe ultimii cinci ani, ce relevanță mai au valorile furnizate prin interogarea de mai sus ?

Rezolvarea acestei probleme esențiale pentru bunul mers al aplicației presupune stocarea procentului sau TVA-ului din momentul preluării facturii în baza de date. Prima soluție ar ține cont de modul în care are loc, la intervale mai mari sau mai mici, modificarea procentelor de TVA ale produselor – vezi figura 10.1.

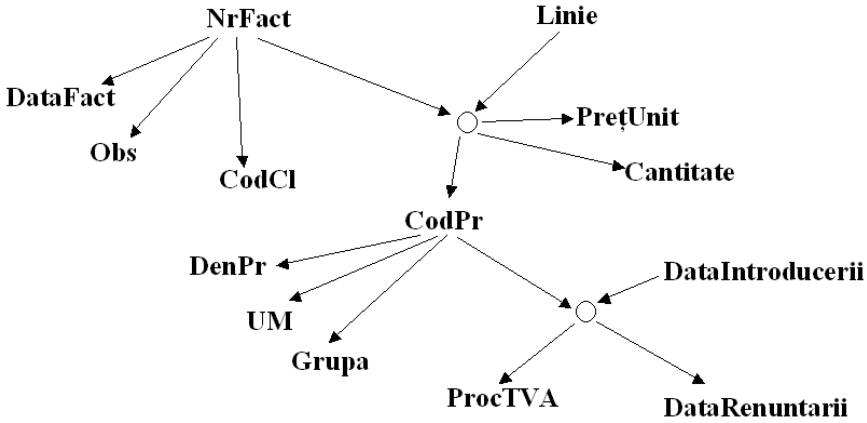

Figura 10.1. Procentul TVA pentru orice produs se poate schimba oricât de des

Pe baza acestui graf schema devine:

PRODUSE {CodPr, DenPr, UM, Grupa}

TVA_PRODUSE {CodPr, DataIntroducerii, DataRenunţarii, ProcTVA },

iar FACT şi LINII_FACT rămân neschimbate. Salvarea validității temporale se face cu pierderi grele de viteză, întrucât orice recompunere a valorii unei facturi este mult mai laborioasă (vezi figura 10.2):

```
SELECT f.NrFact, DataFact, Linie, DenPr,
       cantitate * pretunit AS ValFaraTVA,
       t.ProcTVA,
       cantitate * pretunit * (1 + t.procTVA/100) AS ValCuTVA
FROM fact f
   INNER JOIN linii_fact lf ON f.nrfact=lf.nrfact
   INNER JOIN produse p ON lf.CodPr=p.CodPr
   INNER JOIN tva_produse t ON lf.CodPr=t.CodPr AND
       f.datafact BETWEEN DataIntroducerii AND
             NVL(DataRenuntarii, SYSDATE)
ORDER BY f.NrFact
```

NRFACT	DATAFACT	LINIE	DENPR	VALFARATVA	PROCTVA	VALCUTVA
1	30-DEC-01	1	P1	1500000	22	1830000
1	30-DEC-01	2	P2	700000	19	833000
2	26-JUL-04	1	P3	780000	19	928200
3	30-DEC-01	2	P2	1280000	19	1523200
3	30-DEC-01	3	P3	1530000	19	1820700
4	26-JUL-04	2	P1	840000	19	999600
4	26-JUL-04	1	P1	1000000	19	1190000
5	31-JAN-99	1	P4	1710000	9	1863900
5	31-JAN-99	2	P5	754000	9	821860
6	12-DEC-03	1	P3	913500	19	1087065
7	10-JAN-05	3	P2	1425000	0	1425000
7	10-JAN-05	1	P6	1185000	19	1410150
7	10-JAN-05	2	P2	1530000	0	1530000
8	05-JAN-05	1	P1	1232500	9	1343425

Figura 10.2. Calculul corect al valorii produselor facturate

iar, pe de altă parte, şi triggerele de inserare/modificare/ştergere trebuie să „cotrobăie" prin TVA_PRODUSE, pentru a determina „procentul corect" în vederea actualizărilor corespunzătoare a atributelor calculate din FACT: `ValTotală` şi `TVAFact`. Listingul de creare a noii tabele şi de actualizare a PRODUSE, FACT şi LINI_FACT (listing 10.1) este disponibil pe web.

O parte dintre dvs. s-ar gândi ca, în loc să operăm modificările de TVA la toate produsele pe care le comercializăm, mai bine definim grupe de TVA. De exemplu, definim grupa *cărți*, iar o revenire la cota de 9% s-ar traduce prin modificarea (şi jurnalizarea) procentului numai la această grupă. Schema ar fi:

GRUPE_TVA {`IdGrupaTVA`, `DenGrupaTVA`, `Observaţii`}

PRODUSE {`CodPr`, `DenPr`, `UM`, `Grupa`, `IdGrupaTVA`}

TVA_Grupe {`IdGrupaTVA`, `DataIntroducerii`, `DataRenunţarii`, `ProcTVA`}.

Avantajul ar fi economia de timp la modificări repetate ale cotei de TVA ale diferitelor categorii de produse. Există, însă, şi două dezavantaje majore. Unul ar fi că mai apare o relație/restricție referențială. Iar, în al doilea şi cel mai important rând, soluția este compromisă dacă un produs este mutat, în timp, dintr-o categorie de TVA în alta. Nu ştiu dacă e cel mai nimerit exemplu (sigur, nu cel mai moral cu putință), dar ar fi posibil ca anumite categorii de publicații să fie considerate cam deocheate şi, mergând pe ideea că totul (mai ales plăcerea) se plăteşte, legiuitorul român stabileşte un procent de TVA diferit pentru o parte dintre cărți/reviste.

A treia variantă ar fi una dintre cele evocate în capitolul 9 – folosirea de atribute redundante, sintetice. Astfel, în tabela PRODUSE păstrăm atributului `ProcTVA` a cărui valoare indică procentul curent de TVA aplicat la vânzarea produsului, iar în tabela LINII_FACT introducem atributul `TVALinie`:

PRODUSE {`CodPr`, `DenPr`, `UM`, `Grupa`, `ProcCurentTVA` }

TVA_PRODUSE {CodPr, DataIntroducerii, DataRenunţarii, ProcTVA }

LINII_FACT {NrFact, Linie, CodPr, Cantitate, PreţUnit, TVALinie}

Deşi redundantă în bună măsură, structura este una acceptabilă şi „rezistentă" în timp. Firește, se poate merge şi mai departe cu discuția pentru această bază de date. Ține de regulile unei economii libere ca orice firmă să anunțe perioade de reduceri de preț, fie pentru scăpa de stocuri, fie pentru a impulsiona vânzările la un sortiment nou, necunoscut. Pe perioada promoțională a unui produs, prețul său „normal" de vânzare este PreţPromoţional, în timp ce în lipsa promoției prețul de vânzare este PreţCurent. Cu toate acestea, graful DF din figura 10.3 mai conține şi atributului PreţUnitar, şi aceasta deoarece atunci când un client cumpără o cantitate mai mare sau ne oferă nişte condiții superavantajoase (sau are cunoştințe în parlament/guvern), anumite produse pot fi vândute sub nivelul prețului curent.

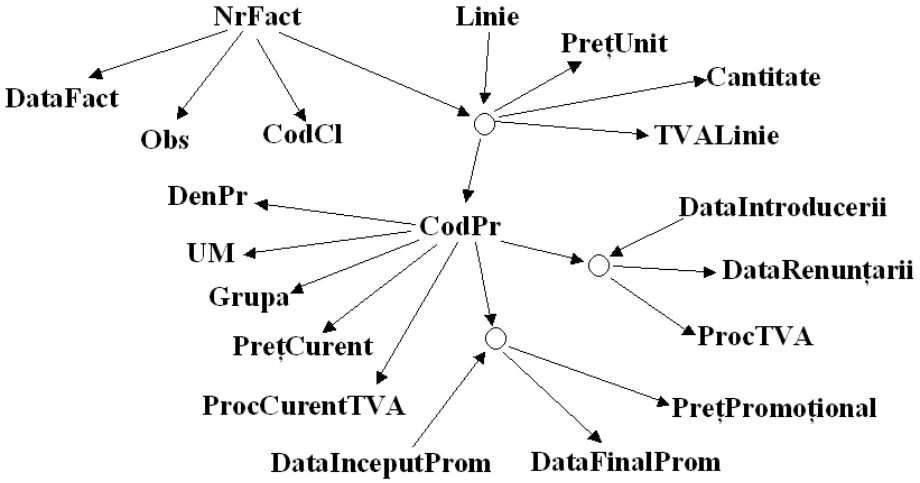

Figura 10.3. Preţuri şi campanii promoţionale

Schema rezultată din graf ar fi:

PRODUSE {CodPr, DenPr, UM, Grupa, PreţCurent, ProcCurentTVA}

TVA_PRODUSE {CodPr, DataIntroducerii, DataRenunţarii, ProcTVA }

PROMOȚII {CodPr, DataÎnceputProm, DataFinalProm, PreţPromoţional}

FACT {NrFact, CodCl, DataFact, Obs}

LINII_FACT {NrFact, Linie, CodPr, Cantitate, PreţUnit, TVALinie}

Listingul 10.2 s-a prelungit peste ceea ce era de aşteptat, deoarece, faţă de versiunea anterioară a tabelei PRODUSE, atributul `ProcTVA` este „rebotezat" `ProcCurentTVA`, iar în modicarea trebuie să se propage în toate blocurile PL/SQL care face apel la acest atribut.

Listing 10.2. Modificări ale schemei, unor declanşatoare şi noi funcţii pentru determinarea valorii implicite a preţului de vânzare

```
ALTER TABLE produse RENAME COLUMN ProcTVA TO ProcCurentTVA ;

ALTER TABLE produse ADD PretCurent NUMBER(12) ;

UPDATE produse SET PretCurent=1500 WHERE CodPr=1 ;
UPDATE produse SET PretCurent=1400 WHERE CodPr=2 ;
UPDATE produse SET PretCurent=1300 WHERE CodPr=3 ;
UPDATE produse SET PretCurent=1200 WHERE CodPr=4 ;
UPDATE produse SET PretCurent=1450 WHERE CodPr=5 ;
UPDATE produse SET PretCurent=1500 WHERE CodPr=6 ;

DROP TABLE promotii ;
CREATE TABLE promotii (
    CodPr NUMBER(6) NOT NULL REFERENCES produse (CodPr),
    DataIncputProm DATE,
    DataFinalProm DATE,
    PretPromotional NUMBER(12)
    ) ;

INSERT INTO promotii VALUES (1, DATE'2005-01-01', NULL, 1475) ;
INSERT INTO promotii VALUES (3, DATE'2005-10-20', DATE'2005-02-02', 1325) ;
INSERT INTO promotii VALUES (5, DATE'2005-02-01', NULL, 1325) ;

-- NOUL declanşator de stergere in LINII_FACT - la nivel de LINIE
CREATE OR REPLACE TRIGGER trg_lf_del
    BEFORE DELETE ON linii_fact FOR EACH ROW
BEGIN
    pac_vinzari.v_trg_liniifact := TRUE ;
    pac_vinzari.v_nrfact := :OLD.NrFact ;
    pac_vinzari.v_liniestearsa := :OLD.Linie ;
    UPDATE fact SET
        valtotala = valtotala - :OLD.cantitate *
            :OLD.pretunit * (1 +
            (SELECT ProcCurentTVA FROM produse WHERE codpr=:OLD.codpr)),
        TVAfact = TVAfact - :OLD.cantitate * :OLD.pretunit *
            (SELECT ProcCurentTVA FROM produse WHERE codpr=:OLD.codpr)
    WHERE nrfact=:OLD.nrfact ;
    pac_vinzari.v_trg_liniifact := FALSE ;
END ;
/

DELETE FROM linii_fact WHERE NrFact >= 10 ;
DELETE FROM fact WHERE NrFact >= 10 ;

INSERT INTO fact (nrfact, datafact, codcl) VALUES (10, SYSDATE,1001);
INSERT INTO fact (nrfact, datafact, codcl) VALUES (11, SYSDATE,1001);
INSERT INTO fact (nrfact, datafact, codcl) VALUES (12, SYSDATE,1002);
INSERT INTO fact (nrfact, datafact, codcl) VALUES (13, SYSDATE,1003);
INSERT INTO fact (nrfact, datafact, codcl) VALUES (14, SYSDATE,1002);
```

```
-------------------------------------------------------------
CREATE OR REPLACE FUNCTION f_datafact (nrfact_ fact.NrFact%TYPE)
     RETURN DATE
IS
     v_data DATE ;
BEGIN
     SELECT DataFact INTO v_data FROM fact
     WHERE NrFact=nrfact_ ;
     RETURN v_data ;
END f_datafact ;
/

-------------------------------------------------------------
CREATE OR REPLACE FUNCTION f_pret (codpr_ produse.codpr%TYPE,
     nrfact_ fact.nrfact%TYPE)
          RETURN produse.PretCurent%TYPE
IS
     v_pret   produse.PretCurent%TYPE ;
BEGIN
     SELECT PretPromotional INTO v_pret FROM promotii
     WHERE CodPr = codpr_ AND f_datafact (nrfact_) BETWEEN
          DataInceputProm        AND NVL(DataFinalProm,SYSDATE) ;
     RETURN v_pret ;
EXCEPTION
     WHEN NO_DATA_FOUND THEN
          SELECT PretCurent INTO v_pret FROM produse
               WHERE CodPr=codpr_ ;
          RETURN v_pret ;
END ;
/

-- modificam declansatorul BEFORE-INSERT-ROW in LINII_FACT
--    (versiunea anterioara era in Listing 9.6 (partea a II-a)
CREATE OR REPLACE TRIGGER trg_lf_ins0
       BEFORE INSERT ON linii_fact FOR EACH ROW
BEGIN
     pac_vinzari.v_trg_liniifact := TRUE ;
     :NEW.Linie := pac_vinzari.f_nrlinii (:NEW.NrFact) + 1 ;
     UPDATE fact SET NrLinii = NrLinii + 1
          WHERE NrFact = :NEW.NrFact ;
     :NEW.PretUnit := f_pret (:NEW.CodPr, :NEW.NrFact) ;
     pac_vinzari.v_trg_liniifact := FALSE ;
END ;
/

-----------------------------------------------
CREATE OR REPLACE TRIGGER trg_lf_ins
AFTER INSERT ON linii_fact FOR EACH ROW
BEGIN
     pac_vinzari.v_trg_liniifact := TRUE ;
     UPDATE fact SET
       valtotala = NVL(valtotala,0) +
          :NEW.cantitate *:NEW.pretunit * (1 +
          (SELECT ProcCurentTVA FROM produse WHERE codpr=:NEW.codpr)),
       TVAfact = NVL(TVAfact,0) + :NEW.cantitate * :NEW.pretunit *
          (SELECT ProcCurentTVA FROM produse WHERE codpr=:NEW.codpr)
     WHERE nrfact=:NEW.nrfact ;
     pac_vinzari.v_trg_liniifact := FALSE ;
END ;
```

```
INSERT INTO linii_fact VALUES (10, 1, 4, 950, NULL) ;
INSERT INTO linii_fact VALUES (11, 2, 5, 580, NULL) ;
INSERT INTO linii_fact VALUES (11, 1, 3, 630, NULL) ;
INSERT INTO linii_fact VALUES (12, 1, 6, 790, NULL) ;
INSERT INTO linii_fact VALUES (12, 2, 2, 850, NULL) ;
INSERT INTO linii_fact VALUES (12, 3, 2, 950, NULL) ;
INSERT INTO linii_fact VALUES (13, 1, 1, 850, NULL) ;
```

Cele câteva date de test, precum şi rezultatele interogărilor din figura 10.4 ne dau garanția că sistemul este funcționabil.

```
SQL> SELECT * FROM PRODUSE ;

     CODPR DENPR          UM   GRUPA    PROCCURENTTVA PRETCURENT
---------- -------------- ---- -------- ------------- ----------
         1 P1             kg                    .19       1500
         2 P2             kg                    .09       1400
         3 P3             kg                    .19       1300
         4 P4             mc   Textile          .19       1200
         5 P5             ml   Cafea            .19       1450
         6 P6             buc  Bere             .19       1500

6 rows selected.

SQL> SELECT * FROM PROMOTII;

     CODPR DATAINCEP DATAFINAL PRETPROMOTIONAL
---------- --------- --------- ---------------
         1 01-JAN-05                     1475
         3 20-OCT-05 02-FEB-05           1325
         5 01-FEB-05                     1325

SQL> SELECT f.NrFact, f.DataFact,Linie, CodPr, Cantitate, PretUnit
  2  FROM fact f INNER JOIN linii_fact lf ON f.NrFact=lf.NrFact
  3  WHERE f.NrFact >= 8
  4  /

    NRFACT DATAFACT       LINIE      CODPR  CANTITATE   PRETUNIT
---------- --------- ---------- ---------- ---------- ----------
         8 05-JAN-05          1          1        850       1450
        10 03-FEB-05          1          4        950       1200
        11 03-FEB-05          1          5        580       1325
        11 03-FEB-05          2          3        630       1300
        12 03-FEB-05          1          6        790       1500
        12 03-FEB-05          2          2        850       1400
        12 03-FEB-05          3          2        950       1400
        13 03-FEB-05          1          1        850       1475
```

Figura 10.4. Calculul valorii implicite a preţului de vânzare

Şi cum acest capitol va fi presărat cu întrebări „istorice", să ne gândim la problema: putem şti, pentru ultimii şapte ani, care au fost prețurile unitare „curente" ale produsului X ? Pentru a răspunde corect, ar trebui să lămurim dacă întrebarea are rost sau nu. Credem că, în cazul acesta, nu prea ! Totuşi, celor insistenți le-am putea pune la dispoziție toate prețurile la care au fost vândute produsele pe baza înregistrărilor stocate în LINII_FACT. Aici sunt consemnate, însă, şi prețurile mai mici datorate reducerilor făcute clienților fideli sau care au

cumpărat cantități foarte mari din acest produs, iar, pe de altă parte, s-ar putea ca, dacă produsul este foarte scump, şi vânzările foarte rare, o parte dintre prețuri să nu fi fost stocate. Pentru această a două speță, să luăm un exemplu. La un supermagazin, există un televizor cu cristale lichide şi toate „bunătățurile" la un preț exorbitant de 170 miloane lei. După şase luni, magazinul mai lasă din preț – 160 milioane lei, dar degeaba ! După alte două reduceri, la 155 şi 147 milioane, urmează o depreciere severă a leului față de euro, iar prețul ajunge la 159 milioane. Abia după o altă ieftinire, la 153 de milioane, produsul este vândut. Ei bine, din toată această evoluție a prețului, în schema propusă va fi preluat doar ultima valoare.

10.2. Consemnarea evoluției unor parametri. „Refacerea" unei valori anterioare

Pentru aplicațiile tranzacționale, consemnarea operațiunilor la momentul producerii lor, sau nu mult timp după aceea, este rezolvată în cea mai mare a cazurilor prin schema de stocare. Astfel, pentru baza de date VÎNZĂRI, preluarea în baza de date a vânzărilor şi încasărilor, deşi uneori cu un uşor decalaj temporal (mai ales la încasări), reflectă totuşi ordinea cronologică a evenimentelor. Analog stau lucrurile şi pentru centrul de închirieri din capitolul 7, pentru firma de transport călători din capitolul 8, „bancomatele" din capitolul 9 sau mini-aplicația de contabilitate generală tratată în ultimele trei capitole.

Din rațiuni de simplificare, şi, de cele mai multe, ori, datorită lipsei de relevanță pentru firmă, evoluția unor parametri nu este consemnată în bază. Bunăoară, în baza de date VÎNZĂRI, am fost foarte atenți, mai ales în acest capitol, cu evoluția procentului TVA şi prețului fiecărui produs, însă faptul că un client îşi poate schimba, în timp, sediul sau numărul de telefon, nu ne-a preocupat absolut deloc. La urma urmei, este chiar aşa de important să ştim pe ce stradă îşi avea clientul *X* sediul în decembrie 2000 (presupunând că aplicația noastră e implementată în iunie 1997) ? Ei, bine, există firme şi organizații, precum ROMTELECOM, DISTRIGAZ, S.C. ELECTRICA, bănci, şi mai ales Poliția şi serviciul de evidența populației pentru care cunoaşterea pe termen lung a tuturor datelor legate de domiciliul clienților are o mare importanță. Pentru aceste cazuri, graful DF poate fi cel din figura 10.5.

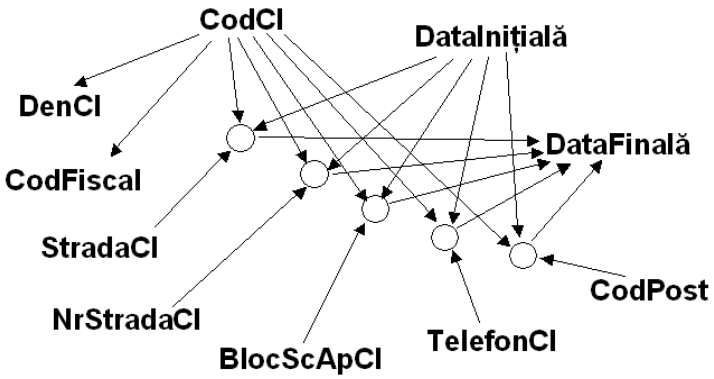

Figura 10.5. Preluarea oricărei modificări de adresă

`DataIniţială` şi `DataFinală` repezintă intervalul pentru care este valabilă acea componentă a adresei. Valoarea NULL a atributului `DataFinală` ar semnala faptul că respectiva componentă a adresei este valabilă în prezent. Codul, denumirea şi codul fiscal sunt proprietăţile cele mai constante ale clientului, aşa că modificarea lor este improbabilă. În sursele dependenţelor compuse a fost inclus şi atributul `DataIniţială`, deoarece, în timp, un client poate reveni la o adresă anterioară sau în apropierea unei adrese anterioare.

Structura se complică rău de tot:

CLIENŢI {`CodCl`, `DenCl`, `CodFiscal`}

CLIENŢI_STRĂZI {`CodCl`, `StradaCl`, `DataIniţială`, `DataFinală`}

CLIENŢI_NR {`CodCl`, `NrStradaCl`, `DataIniţială`, `DataFinală`}

CLIENŢI_BLOC {`CodCl`, `BlocScApCl`, `DataIniţială`, `DataFinală`}

CLIENŢI_TEL {`CodCl`, `TelefonCl`, `DataIniţială`, `DataFinală`}

CLIENŢI_CODPOST {`CodCl`, `CodPost`, `DataIniţială`, `DataFinală`}

Dacă ţinem cont că majoritatea interogărilor legate de clienţi privesc adresa curentă a acestora, n-ar strica, totuşi, să aplicăm ceea ce ne-a predicat capitolul 9, şi anume că o anumită doză de redundanţă nu strică pe la casa omului:

CLIENŢI2 {`CodCl`, `DenCl`, `CodFiscal`, `StradaCl`, `NrStradaCl`, `BlocScApCl`, `TelefonCl`, `CodPost`}

Listingul de creare a celor cinci tabele este disponibil pe web (listing 10.3). Probabil că cei mai mulţi dintre detractorii schemei stufoase la care am ajuns de dragul unei banale adrese, mai precis, a istoricului său, vor acuza complexitatea

presupusă de gestiunea celor şase tabele. Ei bine, lucrurile nu sunt chiar aşa. Putem ascunde multe dintre detalii lucrând „în fundal”, cu declanşatoarele. Astfel, orice modificare a vreunuia dintre atributele din adresă poate fi imortalizată de maniera prezentată în listingul 10.4.

Listing 10.4. O funcţie şi un declanşator pentru „contabilizarea” modificărilor adresei unui client

```
CREATE OR REPLACE FUNCTION f_data_ultimei_modif (
    codcl_ clienti2.CodCl%TYPE,
    atribut_ VARCHAR2) RETURN DATE
AS
    v_data DATE ;
    v_tabela VARCHAR2(20) ;
BEGIN
    CASE atribut_
    WHEN 'StradaCl' THEN
        v_tabela := 'CLIENTI_STRAZI';
    WHEN 'NrStradaCl' THEN
        v_tabela := 'CLIENTI_NR';
    WHEN 'BlocScApCl' THEN
        v_tabela := 'CLIENTI_BLOC';
    WHEN 'Telefon' THEN
        v_tabela := 'CLIENTI_TEL';
    WHEN 'CodPost' THEN
        v_tabela := 'CLIENTI_CODPOST';
    END CASE ;

    EXECUTE IMMEDIATE 'SELECT MAX(DataFinala)  FROM '
        || v_tabela || ' WHERE CodCl = :1 ' INTO v_data USING codcl_ ;
    RETURN v_data ;
END ;
/

-------------------------------------------------------------------------------------
-- modificarea unui element din adresa in tabele CLIENTI2
CREATE OR REPLACE TRIGGER trg_clienti_upd2
    AFTER UPDATE OF StradaCl, NrStradaCl, BlocScApCl,
    Telefon, CodPost ON clienti2 FOR EACH ROW
BEGIN
    -- s-a modificat strada ?
    IF NVL(:NEW.StradaCl,' ') <> NVL(:OLD.StradaCl,' ') THEN
        INSERT INTO clienti_strazi VALUES (:OLD.CodCl, NVL(:OLD.StradaCl,' '),
        NVL(f_data_ultimei_modif (:OLD.CodCl, 'StradaCl'), DATE'1990-01-01'),
        SYSDATE);
    END IF ;

    -- s-a modificat numarul ?
    IF NVL(:NEW.NrStradaCl,' ') <> NVL(:OLD.NrStradaCl,' ') THEN
        INSERT INTO clienti_nr VALUES (:OLD.CodCl,NVL(:OLD.NrStradaCl,' '),
        NVL(f_data_ultimei_modif (:OLD.CodCl, 'NrStradaCl'), DATE'1990-01-01'),
        SYSDATE);
    END IF ;

    -- s-a modificat blocul/scara/apartamentul ?
    IF NVL(:NEW.BlocScApCl,' ') <> NVL(:OLD.BlocScApCl,' ') THEN
        INSERT INTO clienti_bloc VALUES (:OLD.CodCl,NVL(:OLD.BlocScApCl,' ') ,
        NVL(f_data_ultimei_modif (:OLD.CodCl, 'BlocScApCl'), DATE'1990-01-01'),
```

```
        SYSDATE);
    END IF ;

    -- s-a modificat numarul de telefon ?
    IF NVL(:NEW.Telefon, ' ') <> NVL(:OLD.Telefon,' ') THEN
        INSERT INTO clienti_tel VALUES (:OLD.CodCl,NVL(:OLD.Telefon, ' '),
        NVL(f_data_ultimei_modif (:OLD.CodCl, 'Telefon'), DATE'1990-01-01'),
        SYSDATE);
    END IF ;

    -- s-a modificat codul postal ?
    IF NVL(:NEW.CodPost,0) <> NVL(:OLD.CodPost,0) THEN
        INSERT INTO clienti_codpost VALUES (:OLD.CodCl, NVL(:OLD.CodPost,0),
        NVL(f_data_ultimei_modif (:OLD.CodCl, 'CodPost'), DATE'1990-01-01'),
        SYSDATE);
    END IF ;
END ;
/
```

Funcţia determină care a fost data ultimei modificări, pentru clientul-parametru, a străzii, numărului... Dacă este vorba de prima modificare a unui client/atribut, atunci data returnată este 1 ianuarie 1990, deşi mai corectă ar fi fost valoarea de implementare a aplicaţiei – 1 ianuarie 1997. Pentru a ne mai dezmorţi un pic oasele în PL/SQL, am folosit un mic articiu – SQL dinamic, astfel încât să nu scriem câte un SELECT fiecăreia dintre cele cinci tabele. Declanşatorului nu-i mai rămâne decât să verifice care dintre atribute s-a modificat şi să facă inserarea în tabela „istoric" corespunzătoare.

Cu această schemă, aflarea străzilor pe care îşi aveau clienţii sediile la momentul 8 februarie 2005, ora 9:42, nu ridică probleme deosebite. O interogarea s-ar putea baza pe o facilitate mai recentă din dialectul SQL Oracle, şi anume posibilitatea de a „boteza" rezultatul unei interogări (STRĂZI) şi a-l folosi în fraza SELECT următoare ca o tabelă în toată regula:

```
WITH strazi AS
      (SELECT codcl, stradacl, datainitiala,
            datafinala - 1 /(24*60*60) AS datafinala
       FROM clienti_strazi
       UNION
       SELECT c.codcl, stradacl,
            NVL((SELECT MAX(datafinala) FROM clienti_strazi
                  WHERE codcl=c.codcl)
            ,TO_DATE('1990-01-01','YYYY-MM-DD')),
            SYSDATE + 1 / (24*60*60)
      FROM clienti2 c
      ORDER BY codcl, datainitiala
      )
SELECT dencl, s.stradacl
FROM clienti2 c INNER JOIN strazi s ON c.codcl=s.codcl AND
      TO_DATE('2005-02-08  09:42:00','YYYY-MM-DD  HH24:MI:SS')
      BETWEEN datainitiala AND datafinala
```

Pentru ca data inițială a unui interval să nu fie identică cu cea finală a intervalului precedent, datafinală a fost decrementată cu o secundă, adică 1 / (24*60*60) dintr-o zi (24 de ore, 60 de minute şi 60 de secunde). Deşi nu parcurgem o lucrare dedicată SQL-ului, vă mai propun o soluție:

```
SELECT dencl, c.codcl,
      NVL((SELECT stradacl FROM clienti_strazi
            WHERE codcl=c.codcl AND
            TO_DATE('2005-02-08 09:42:00',
            'YYYY-MM-DD HH24:MI:SS') BETWEEN
            datainitiala AND datafinala-1/(24*60*60)),
      c.stradacl) stradacl
FROM clienti2 c
```

Pe calapodul acestei secunde soluții, suntem în măsură să furnizăm un răspuns rezonabil la întrebarea: *Care era adresa completă a clientului CL1 la 8 februarie 2005, ora 9:42:00 („ora bazei de date") ?* Cele trei interogări sunt salvate în listingul 10.5 de pe web.

```
SELECT dencl, c.codcl,
      NVL((SELECT stradacl FROM clienti_strazi
      WHERE codcl=c.codcl AND
      TO_DATE('2005-02-08 09:42:00','YYYY-MM-DD HH24:MI:SS')
      BETWEEN datainitiala AND datafinala-1/(24*60*60)),
      c.stradacl) stradacl,
      NVL((SELECT nrstradacl FROM clienti_nr
      WHERE codcl=c.codcl AND
      TO_DATE('2005-02-08 09:42:00','YYYY-MM-DD HH24:MI:SS')
      BETWEEN datainitiala AND datafinala-1/(24*60*60)),
      c.nrstradacl) nrstradacl,
      NVL((SELECT blocscapcl FROM clienti_bloc
      WHERE codcl=c.codcl AND
      TO_DATE('2005-02-08 09:42:00','YYYY-MM-DD HH24:MI:SS')
      BETWEEN datainitiala AND datafinala-1/(24*60*60)),
      c.blocscapcl) blocscapcl,
      NVL((SELECT telefon FROM clienti_tel
      WHERE codcl=c.codcl AND
      TO_DATE('2005-02-08 09:42:00','YYYY-MM-DD HH24:MI:SS')
      BETWEEN datainitiala AND datafinala-1/(24*60*60)),
      c.telefon) telefon,
      NVL((SELECT codpost FROM clienti_codpost
      WHERE codcl=c.codcl AND
      TO_DATE('2005-02-08 09:42:00','YYYY-MM-DD HH24:MI:SS')
      BETWEEN datainitiala AND datafinala-1/(24*60*60)),
      c.codpost) codpost
FROM clienti2 c
WHERE dencl='CL1'
```

Urmărirea evoluției unor parametri poate avea o valoarea mai mult decât pur informativă, fiind uneori chiar vitală pentru contabilitatea şi finanțele unei firme. În acest sens continuăm cu un exemplu care a dat ceva bătăi de cap proiectanților de baze de date: gestiunea stocurilor. Intrările în gestiune ale materiilor prime, materialelor şi celorlalte elemente patrimoniale care se încadrează la categoria *stocuri* sunt contabilizate la prețul de cumpărare (dacă au fost cumpărate), prețul de producție (dacă au fost produse *in-house*) s.a.m.d. Stocurile sunt destinate consumului, adică producerii altor sortimente, sau vânzării[1]. Problema este că, indiferent de tipul ieşirii din gestiune, evaluarea stocurilor la ieşire se face după una dintre metodele: FIFO – *First Input – First Output*, adică *primul intrat – primul ieşit*, LIFO – *Last Input – Last Output*, adică *ultimul intrat – primul ieşit* sau CMP - *Cost Mediu Ponderat* (aici inițialele sunt „româneşti").

Să presupunem că Sortimentul X este cumpărat (de fapt, i se face recepția) pentru prima dată pe 2 februarie 2005, ora 10:00, prilej cu care se întocmeşte o notă de recepție, iar intrarea „primeşte" identificatorul 12345. Cantitatea intrată este 5, iar prețul unitar 1000 de lei (grei). Pe 3 februarie, ora 9:00 se recepționează încă 3 unități din acelaşi sortiment la prețul unitar de 1100 lei:

IdIntrare	CantIntrată	PreţUnitIntrare	ValoareIntrare	CantRămasă
12345	5	1000	5000	5
12467	3	1100	3300	3

Vorbim, practic, de două tranşe ale acestui produs, una la prețul unitar de 1000 lei şi o alta la 1100. După cele două intrări, are loc darea în consum a 6 unități din sortiment[2]. Întrucât firme aplică principiul FIFO, „descărcarea" va începe cu prima tranşă. Cum, însă, cantitatea ieşită este 6, iar prima tranşă are numai 5 unități, „se va muşca" o unitate şi din a doua tranşă:

IdIeşire	CantIeşire	ValoareIeşire	IdIntrare	CantDescărcată
14101	6	6100	12345	5
			12467	1

După această ieşire, situația celor două tranşe este următoarea:

IdIntrare	CantIntrată	PreţUnitIntrare	ValoareIntrare	CantRămasă
12345	5	1000	5000	0
12467	3	1100	3300	2

Schema bazei trebuie să consemneze nu numai intrările şi ieşirile de stocuri, ci şi corespondența ieşiri-intrări, astfel încât principiul FIFO să fie respectat. Mai mult,

[1] Trecem sub tăcere alte tipuri de operațiuni cu stocurile rezultate din inventarieri, degradări, calamități etc.

[2] Documentul care se întocmeşte este, de obicei, bonul de consum

la un eventual control, trebuie demonstrat că, la orice ieşire a material, descărcarea gestiunii s-a făcut la preţul[3] corect. În aceste condiţii, graful dependenţelor funcţionale ar putea fi, în linii mari, similar celui din figura 10.6.

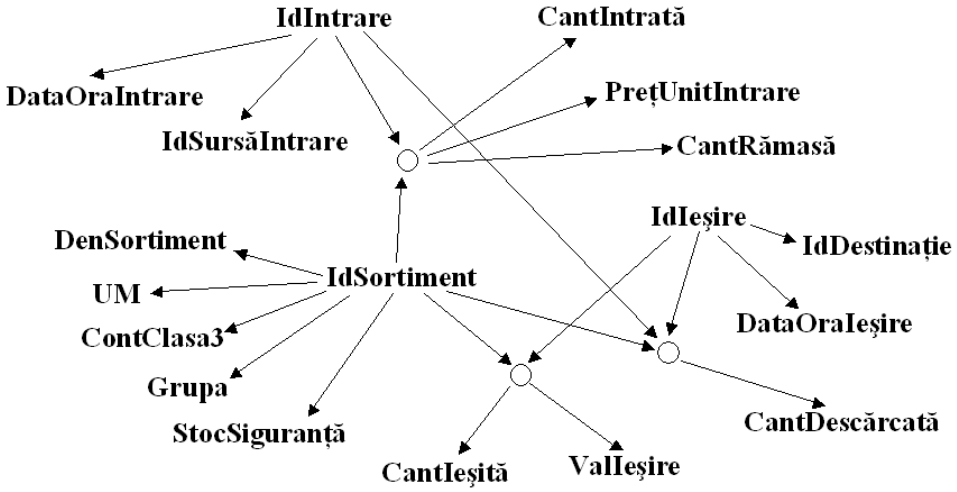

Figura 10.6. Gestiunea stocurilor prin metoda FIFO

Interesant e că tot acest mecanism de punere în corespondenţă a ieşirilor cu intrările de stocuri poate fi gestionat automat, la nivelul „logicii" bazei de date, fireşte, prin intermediul declanşatoarelor. Listingul 10.5 conţine comenzile de creare a tabelelor despinse din graful din figura 10.6 şi câteva comenzi de populare, în consonanţă cu exemplul de mai sus.

Listing 10.5. Implementarea metodei FIFO prin declanşator

```
--
DROP TABLE iesiri_intrari ;
DROP TABLE iesiri_materiale ;
DROP TABLE iesiri ;
DROP TABLE intrari_materiale ;
DROP TABLE intrari ;
DROP TABLE materiale ;

CREATE TABLE materiale (
    IdSortiment NUMBER(7) PRIMARY KEY,
    DenSortiment VARCHAR2(30) UNIQUE,
    UM VARCHAR2(10),
    ContClasa3 VARCHAR2(15),
    Grupa VARCHAR2(30),
    StocSiguranta NUMBER(10) DEFAULT 0 NOT NULL
    ) ;

CREATE TABLE intrari (
```

[3] Pentru firmă, preţul de descărcare la darea în consum sau vânzare este, de fapt, un cost, o cheltuială.

```
    IdIntrare NUMBER(10) NOT NULL PRIMARY KEY,
    DataOraIntrare DATE,
    IdSursaIntrare NUMBER(6)
    ) ;

CREATE TABLE intrari_materiale (
    IdIntrare NUMBER(10) NOT NULL REFERENCES intrari (IdIntrare),
    IdSortiment NUMBER(7) NOT NULL REFERENCES materiale (IdSortiment),
    CantIntrata NUMBER(10),
    PretUnitIntrare NUMBER(10,2),
    CantRamasa NUMBER(10) DEFAULT 0,
    PRIMARY KEY (IdIntrare, IdSortiment)
    ) ;

CREATE TABLE iesiri (
    IdIesire NUMBER(10) NOT NULL PRIMARY KEY,
    DataOraIesire DATE,
    IdDestinatie NUMBER(6)
    ) ;

CREATE TABLE iesiri_materiale (
    IdIesire NUMBER(10) NOT NULL REFERENCES iesiri (IdIesire),
    IdSortiment NUMBER(7) NOT NULL REFERENCES materiale (IdSortiment),
    CantIesita NUMBER(10),
    ValIesire NUMBER(12,2),
    PRIMARY KEY (IdIesire, IdSortiment)
    ) ;

CREATE TABLE iesiri_intrari (
    IdIesire NUMBER(10) NOT NULL REFERENCES iesiri (IdIesire),
    IdIntrare NUMBER(10) NOT NULL REFERENCES intrari (IdIntrare),
    IdSortiment NUMBER(7) NOT NULL REFERENCES materiale (IdSortiment),
    CantDescarcata NUMBER(10),
    PRIMARY KEY (IdIesire, IdIntrare, IdSortiment)
    ) ;

INSERT INTO materiale VALUES (1111, 'Sortiment X', 'kg', '3011.01', 'Materiale constructie', 0) ;
INSERT INTO materiale VALUES (1112, 'Sortiment Y', 'kg', '3011.01', 'Materiale constructie', 0) ;

INSERT INTO intrari VALUES
    (12345, TO_DATE('02/02/2005 10:00:00', 'DD/MM/YYYY HH24:MI:SS'), 123) ;
INSERT INTO intrari VALUES
    (12346, TO_DATE('02/02/2005 10:10:00', 'DD/MM/YYYY HH24:MI:SS'), 124) ;
INSERT INTO intrari VALUES
    (12467, TO_DATE('03/02/2005 09:00:00', 'DD/MM/YYYY HH24:MI:SS'), 123) ;

INSERT INTO intrari_materiale VALUES (12345, 1111, 5, 1000, 5) ;
INSERT INTO intrari_materiale VALUES (12345, 1112, 10, 1200, 10) ;
INSERT INTO intrari_materiale VALUES (12467, 1111, 3, 1100, 3) ;

INSERT INTO iesiri VALUES
    (14101, TO_DATE('03/02/2005 10:00:00', 'DD/MM/YYYY HH24:MI:SS'), 898) ;
INSERT INTO iesiri VALUES
    (14102, TO_DATE('03/02/2005 10:07:00', 'DD/MM/YYYY HH24:MI:SS'), 780) ;

CREATE OR REPLACE TRIGGER trg_iesiri_materiale
    BEFORE INSERT ON iesiri_materiale FOR EACH ROW
DECLARE
    CURSOR c_stoc (idsortiment_ materiale.IdSortiment%TYPE) IS
```

```
        SELECT * FROM intrari_materiale
        WHERE IdSortiment = idsortiment_ AND CantRamasa > 0
        ORDER BY IdIntrare ;
    v_cant_de_repartizat iesiri_materiale.CantIesita%TYPE ;
BEGIN
    v_cant_de_repartizat := :NEW.CantIesita ;
    :NEW.ValIesire := 0 ;
    FOR rec_stoc IN c_stoc (:NEW.IdSortiment) LOOP
        IF rec_stoc.CantRamasa >= v_cant_de_repartizat THEN
            INSERT INTO iesiri_intrari VALUES (:NEW.IdIesire, rec_stoc.IdIntrare,
                    :NEW.IdSortiment, v_cant_de_repartizat) ;
            UPDATE intrari_materiale SET CantRamasa = CantRamasa - v_cant_de_repartizat
            WHERE IdIntrare=rec_Stoc.IdIntrare AND IdSortiment=rec_stoc.IdSortiment ;
            :NEW.ValIesire := :NEW.ValIesire + rec_stoc.PretUnitIntrare * v_cant_de_repartizat ;
            v_cant_de_repartizat := 0 ;
            EXIT ;
        ELSE
            INSERT INTO iesiri_intrari VALUES (:NEW.IdIesire, rec_stoc.IdIntrare,
                    :NEW.IdSortiment, rec_stoc.CantRamasa) ;
            UPDATE intrari_materiale SET CantRamasa = 0
            WHERE IdIntrare=rec_Stoc.IdIntrare AND IdSortiment=rec_stoc.IdSortiment ;
            v_cant_de_repartizat := v_cant_de_repartizat - rec_stoc.CantRamasa ;
            :NEW.ValIesire := :NEW.ValIesire + rec_stoc.PretUnitIntrare * rec_stoc.CantRamasa ;
        END IF ;
    END LOOP ;
    IF v_cant_de_repartizat > 0 THEN
        RAISE_APPLICATION_ERROR (-20881, 'Stoc insuficient !') ;
    END IF ;
END ;
/

INSERT INTO iesiri_materiale VALUES (14101, 1111, 6, 0) ;
```

Punctul cheie al listingului este declanşatorul de inserare în IEŞIRI_MATERIALE. La introducerea unei tabele în această tabelă, cursorul C_STOC se populează cu toate tranşele de preţ ale sortimentului respectiv la care stocul curent (cantitatea rămasă) este mai mare ca zero. Parcurgerea înregistrărilor din cursor are loc până la momentul în care suma cantităţilor repartizate este egală cu `:NEW.CantIeşită` – cantitatea indicată în bonul de consum. La fiecare tranşă identificată, tranşă ce corespunde unei înregistrări din cursor, se decrementează valoarea variabilei `v_cant_de_repartizat`, iniţializată înainte de parcurgerea cursorului cu valoarea `:NEW.CantIeşită`. Dacă, după ieşirea din structura repetitivă, variabila nu are valoarea zero, înseamnă că stocul sortimentului a fost insuficient pentru onorarea cantităţii de pe bonul de consum şi, prin urmare, se va declanşa o eroare.

Un alt caz de calculare retrospectivă a unei valori poate fi articulat în jurul exemplului din pagraful 9.5 dedicat curselor din formula I, al cărui graf este reprezentat în figura 9.4. După cele discutate mai sus, putem reproşa schemei din paragraful respectiv lipsa de „retrospectivă", întrucât datele pot fi gestionate doar pentru sezonul curent. Introducem, deci, două atribute, `Sezon` şi `Etapă`, plus alte două, `PuncteCursă` şi `PunctePenalizare`, necesare preluării rezultatelor

fiecărei curse. În aceste condiţii, PunctePoziţie indică doar punctajul actual pentru plasarea pe un loc anume la finalul unei curse, în timp ce ultimele două atribute introduse sunt esenţiale pentru istoricul aplicaţiei. Graful este acum cel din figura 10.7, iar scriptul de creare şi populare a tabelelor este conţinut în listingul 10.6 (disponibil pe web).

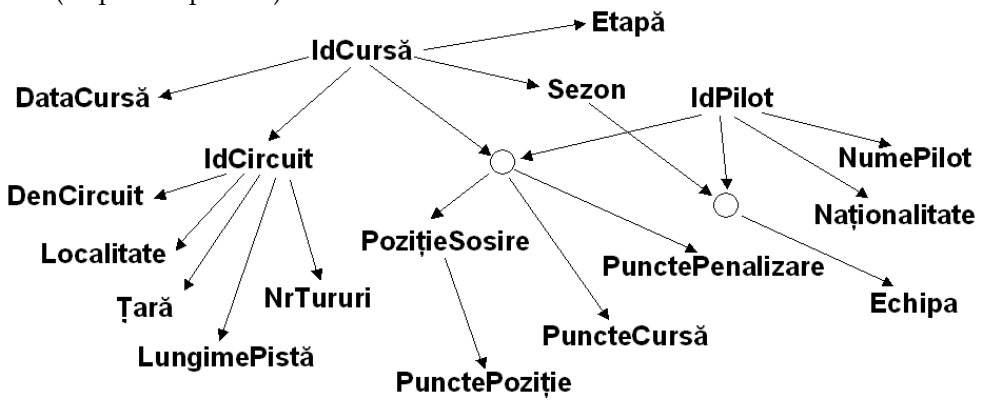

Figura 10.7. Rezultate – formula I

Să presupunem, nu fără realism, că o informaţie frecvent solicitată ar fi: *Care era clasamentul piloţilor (sau echipelor) după etapa a doua a sezonului 2002 ?* La drept vorbind, această informaţie poate fi obţinută fără prea mare efort fără a opera nici o modificare în schema bazei de date – vezi interogarea şi rezultatul din figura 10.8.

```
SQL> SELECT ROWNUM, t.*
  2  FROM
  3   (SELECT NumePilot, SUM(PuncteCursa) AS Puncte,
  4    SUM(PunctePenalizare) AS Penalizari,
  5     SUM(PuncteCursa - PunctePenalizare) AS Total
  6   FROM rezultate_curse rc INNER JOIN curse c ON rc.IdCursa=c.IdCursa
  7    INNER JOIN piloti p ON rc.IdPilot=p.IdPilot
  8   WHERE sezon=2002 AND Etapa <= 2
  9   GROUP BY NumePilot
 10   ORDER BY Total DESC) t
 11  /

    ROWNUM NUMEPILOT          PUNCTE PENALIZARI      TOTAL
---------- --------------- ---------- ---------- ----------
         1 Flash Hook              16          0         16
         2 Slash Gordon            20          8         12
         3 Nonaka John             11          0         11
         4 Benjamin Walter          9          0          9
         5 Yourdon Luke             6          0          6
```

Figura 10.8. Clasamentul după primele două etape ale sezonului 2002

Astfel încât recurgerea la denormalizare/postnormalizare, prin folosirea unei tabele redundante/sintetice de tipul CLASAMENTE_ETAPE {Sezon, Etapă, IdPilot, PuncteCumulate, PenalizăriCumulate} nu se prea justifică.

Cu totul altfel stau lucrurile în cazul aplicației de gestiune a competențelor pe posturi şi persoane, descrisă în paragraful 8.2 (vezi graful din figura 8.8 şi schema din figura 8.9). Lucrurile sunt cu atât mai interesante cu cât, atât posturile, cât şi competențele şi angajații, cunosc evoluții semnificative în timp. Astfel, pentru compartimentul *Resurse umane*, important este nu numai nivelul actual unei anumite competențe pentru un angajat, dar şi modul în care a evoluat acest nivel în ultimii *n* ani. Puteam declara astfel ca specificitate a acestei scheme faptul că interesează evoluția a aproape tuturor atributelor din tabelele bazei, aşa că aplicarea schemei de lucru de la componentele adresei (figura 10.5) pare prea stufoasă.

Deşi laborioasă, o soluție aplicabilă acestui gen de probleme ține de crearea unei replici-jurnal fiecărei tabele din bază, fără a schimba în vreun fel schema tabelelor existente. Ca exemplu, creăm tabela-jurnal pentru COMPETENȚE_PERSONAL pe care o vom denumi IST_COMPETENȚE_PERSONAL:

```
DROP TABLE ist_competente_personal ;
CREATE TABLE ist_competente_personal (
 marca NUMBER (6) NOT NULL,
 idcompfrunza NUMBER (10) NOT NULL,
 nivel NUMBER(5) NOT NULL,
 data_init DATE DEFAULT CURRENT_DATE NOT NULL,
 data_fin DATE DEFAULT NULL,
 PRIMARY KEY (marca, idcompfrunza, data_init)
 ) ;
```

Cele două atribute suplimentare, `Data_Init` şi `Data_Fin` vor indica perioada de valabilitate a înregistrării respective. Astfel, la inserarea unei linii în tabela-sursă se va face o inserare şi în „jurnalul" său, lucru destul de simplu de realizat prin declanşatorul operațiunii:

```
CREATE OR REPLACE TRIGGER trg_competente_personal_ins
      AFTER INSERT ON competente_personal
      FOR EACH ROW
BEGIN
      INSERT INTO ist_competente_personal VALUES (:NEW.marca,
            :NEW.idcompfrunza, :NEW.Nivel, SYSDATE, NULL) ;
END ;
/
```

La ştergerea liniei din tabela-sursă, atributul `Data_Fin` va prelua data şi ora curentă, denulizarea sa semnalizând expirarea înregistrării:

```
CREATE OR REPLACE TRIGGER trg_competente_personal_del
      AFTER DELETE ON competente_personal
      FOR EACH ROW
BEGIN
      UPDATE ist_competente_personal SET data_fin = SYSDATE
      WHERE marca=:OLD.marca AND
       idcompfrunza=:OLD.IdCompFrunza AND data_fin IS NULL ;
```

```
END ;
/
```

Modificarea unei linii în tabela-sursă (tabela-stăpân) ar presupune declararea ca expirată a vechii copii a liniei şi inserarea noii instanţe. O variabilă ne va asigura că noua instanţă va fi valabilă la o secundă (1/(24*60*60)) de la expirarea precedentei instanţe.

```
CREATE OR REPLACE TRIGGER trg_competente_personal_upd
      AFTER UPDATE ON competente_personal
      FOR EACH ROW
DECLARE
      v_data DATE := SYSDATE ;
BEGIN
      -- se declara "expirate" vechile valori
      UPDATE ist_competente_personal SET data_fin = v_data
      WHERE marca=:OLD.marca AND
       idcompfrunza=:OLD.IdCompFrunza AND data_fin IS NULL ;

      -- se insereaza noile valori
      INSERT INTO ist_competente_personal VALUES (:NEW.marca,
            :NEW.idcompfrunza, :NEW.Nivel,
                  v_data + 1 /(24*60*60), NULL) ;
END ;
/
```

Pentru corectitudinea soluţiei, ar trebui să folosim variabile publice, modificate în cele trei declanşatoare de mai sus, şi „controlate" în triggerele de inserare şi modificare ale tabelei IST_COMPETENŢE_PERSONAL, astfel încât vom evita alterarea jurnalizării prin comenzi directe (altfel decât prin declanşatoarele tabelei COMPETENŢE_PERSONAL). Cele trei declanşatoare de mai sus sunt grupate pe web în listingul 10.8.

Extragerea informaţiilor curente din bază presupune consultarea exclusivă a tabelelor stabilite încă din capitolul 8, în timp ce scormonirea prin arhive reclamă folosirea tabelelor-jurnal. Astfel, dacă se doreşte aflarea nivelul tuturor competenţelor la care era evaluată Vasilescu Georgeta la 16 februarie 2005 ora 9:42, interogarea ar putea avea forma:

```
SELECT dencompetenta, nivel
FROM
      (SELECT marca, numepren FROM ist_personal9
       WHERE numepren='Vasilescu Georgeta' AND
            TO_DATE('2005-02-16 09:42','YYYY-MM-DD HH24:MI')
            BETWEEN data_init AND NVL(data_fin, SYSDATE)) p
      INNER JOIN
      (SELECT * FROM ist_competente_personal
       WHERE TO_DATE('2005-02-16 09:42','YYYY-MM-DD HH24:MI')
            BETWEEN data_init AND NVL(data_fin, SYSDATE)) cp
      ON p.marca=cp.marca
      INNER JOIN
            (SELECT idcompetenta, dencompetenta
```

```
            FROM ist_competente
            WHERE TO_DATE('2005-02-16 09:42',
                  'YYYY-MM-DD HH24:MI') BETWEEN data_init AND
          NVL(data_fin, SYSDATE)) c
    ON cp.idcompfrunza=c.idcompetenta
```

Înainte de a ne freca palmele mulțumiți de isprava noastră, ar trebui să ne gândim dacă lucrurile sunt într-adevăr în regulă pentru toate situațiile de actualizare a tabelei-sursă. Veştile nu sunt din cale-afară de încurajatoare. Mecanismul este corect doar pentru atributele neimplicate în restricțiile referențiale. Ce se întâmplă dacă în tabela PERSONAL9 se modifica marca unui angajat care, prin mecanismul UPDATE CASCADE, determină actualizarea liniilor copil din COMPETENȚE_PERSONAL ? Declanşatorul nostru - `trg_competente_personal_upd` - va contabiliza conştiincios expirarea înregistrării referitoare la vechea marcă şi va insera o linie nouă care se va referi la noua valoare a mărcii. Anxietatea se leagă de răspunsul corect la întrebarea: *cum se poate şti, în timp, că linii diferite din tabela-jurnal, cu valori diferite ale atributului* `Marcă`*, se referă, de fapt, la una şi aceeaşi persoană* ?

Primul impuls ar fi să interzicem orice modificare a atributelor de tip cheie surogat sau chiar a tuturor atributelor părinte din schemă. Însă acest lucru nu este întotdeauna realist, mai ales dacă atributul părinte nu este surogat, sau dacă situația este una cu totul specială (două firme fuzionează, şi bazele lor de date la fel). Aşa că ne încercăm norocul cu o altă variantă, pezentată în listingul 10.9. Tabela istoric va avea încă două atribute – `IdJurnalizare_Init` şi `IdJurnalizare_Fin` –, prin care consemnăm tranzacția de inserare, modificare sau ştergere din COMPETENȚE_PERSONAL care a produs adăugarea sau înregistrarea instanței/liniei respective. Declanşatorul de inserare al tabelei-stăpân va aloca, pe baza secvenței, un identificator tranzacției - `IdJurnalizare_Init`, iar cel de ştergere va aloca, tot pe baza secvenței, o valoare pentru atributul `IdJurnalizare_Fin`. Interesul nostru se leagă preponderent de modificările din tabela principală; fiecare modificare generează în tabela jurnal o ştergere, urmată de o inserare. Or, cheia problemei este ca ştergerea şi inserare să aibă acelaşi identificator al tranzacției, adică, la ştergere, `IdJurnalizare_Fin` să fie identic cu `IdJurnalizare_Init` al noii instanțe inserate.

Listing 10.9. Un alt mod de jurnalizare

```
DROP TABLE ist_competente_personal ;

CREATE TABLE ist_competente_personal (
    idjurnalizare_init NUMBER(14) PRIMARY KEY,
    marca NUMBER (6) NOT NULL,
    idcompfrunza NUMBER (10) NOT NULL,
    nivel NUMBER(5) NOT NULL,
    data_init DATE DEFAULT CURRENT_DATE NOT NULL,
    data_fin DATE DEFAULT NULL,
    idjurnalizare_fin NUMBER(14) DEFAULT NULL
    ) ;

DROP SEQUENCE seq_idjurnalizare ;
```

```
CREATE SEQUENCE seq_idjurnalizare
    START WITH 1 MINVALUE 1 MAXVALUE 99999999999999
    NOCYCLE NOCACHE ORDER INCREMENT BY 1 ;

-------
CREATE OR REPLACE TRIGGER trg_competente_personal_ins
    AFTER INSERT ON competente_personal
    FOR EACH ROW
BEGIN
    INSERT INTO ist_competente_personal VALUES (seq_idjurnalizare.NextVal,
        :NEW.marca, :NEW.idcompfrunza, :NEW.Nivel, SYSDATE, NULL, NULL) ;
END ;
/

-----
CREATE OR REPLACE TRIGGER trg_competente_personal_del
    AFTER DELETE ON competente_personal
    FOR EACH ROW
BEGIN
    UPDATE ist_competente_personal
    SET data_fin = SYSDATE, idjurnalizare_fin = seq_idjurnalizare.NextVal
    WHERE marca=:OLD.marca AND idcompfrunza=:OLD.IdCompFrunza
        AND data_fin IS NULL ;
END ;
/

-----
CREATE OR REPLACE TRIGGER trg_competente_personal_upd
    AFTER UPDATE ON competente_personal
    FOR EACH ROW
DECLARE
    v_data DATE ;
BEGIN
    v_data := SYSDATE ;

    -- se declara "expirate" vechile valori
    UPDATE ist_competente_personal
    SET data_fin = v_data, idjurnalizare_fin = seq_idjurnalizare.NextVal
    WHERE marca=:OLD.marca AND idcompfrunza=:OLD.IdCompFrunza
        AND data_fin IS NULL ;

    -- se insereaza noile valori
    INSERT INTO ist_competente_personal VALUES (seq_idjurnalizare.CurrVal,
        :NEW.marca, :NEW.idcompfrunza,
        :NEW.Nivel, v_data + 1 /(24*60*60), NULL, NULL) ;

END ;
/
```

Referindu-ne la problema noastră, dacă marca angajatului 20002 va fi modificată în 22222 la un moment dat: `UPDATE personal9 SET marca=22222 WHERE numepren='Mihuleac Iulian'`, operaţiunile se vor derula astfel:

- se denulizează valorile atributelor `Data_Fin` şi `IdJurnalizare_Fin` pentru toate liniile din IST_COMPETENŢE_PERSONAL în care valoarea mărcii este 20002; Primul atribut ia valoarea datei/orei curente, iar al doilea „muşcă" valoarea următoare din secvenţă;

- se inserează un set de linii egal cu numărul celor modificate în COMPETENȚE_PERSONAL, în care valoarea `IdJurnalizare_Init` este identică cu cea a `IdJurnalizare_Fin` de la pasul anterior, iar `Data_Init` este valoarea `Data_Fin` de la pasul anterior plus o secundă.

Să presupunem că, după un timp, marca aceluiaşi angajat este modificată din nou, din 22222 în 22223: `UPDATE personal9 SET marca=22223 WHERE numepren = 'Mihuleac Iulian'`. În funcție de momentul pentru care interesează nivelul competențelor acestui angajat, valorile corecte vor fi furnizate fie de liniile în care marca este 20002, fie de cele în care marca este 222222, fie de cele în care marca este 22223. Ei bine, avem motive serioase de a privi languros oferta SQL de la Oracle, deoarece avem un excelent sprijin în astfel de situații. Indiferent de câte modificări s-au produs asupra mărcii, pornind de la marca actuală, putem şti toate valorile acestui atribut printr-o interogare ierarhică, în care legătura dintre niveluri se realizează prin valoarile identice ale identificatorilor de „expirare" din vechile linii şi de adăugare a noilor instanțe:

```
SELECT *
FROM ist_competente_personal
START WITH marca IN (SELECT marca
                     FROM personal9
                     WHERE numepren='Mihuleac Iulian')
CONNECT BY PRIOR idjurnalizare_init=idjurnalizare_fin
```

Dacă dorim să aflăm nivelul tuturor competențelor actuale (presupunem că nu s-a modificat nimic în nomenclatorul competențelor) pentru angajatul Mihuleac Iulian la 16 februarie 2005 ora 10:25 interogarea ce foloseşte artificiul de mai sus ar fi:

```
SELECT dencompetenta, nivel
FROM competente c INNER JOIN
      (SELECT *
       FROM
            (SELECT *
            FROM ist_competente_personal
            START WITH marca IN (SELECT marca
                  FROM personal9
                  WHERE numepren='Mihuleac Iulian')
            CONNECT BY PRIOR
                  idjurnalizare_init=idjurnalizare_fin
            )
      WHERE TO_DATE('2005-02-16 10:25','YYYY-MM-DD HH24:MI')
            BETWEEN data_init AND NVL(data_fin, SYSDATE)
      ) cp ON c.idcompetenta=cp.idcompfrunza
```

Varianta sută la sută corectă a acestei interogări ar trebui să lucreze cu tabela IST_COMPETENȚE, o subconsultare similară celei aplicate IST_COMPETENȚE_PERSONAL extrăgând elementele definitorii ale competențelor la momentul *16*

februarie 2005 ora 10:25. În acest stil putem rezolva orice problemă legată de temporalitate, chiar dacă preţul plătit nu este chiar neglijabil.

10.3. Protejarea „trecutului". Închideri contabile şi nu numai

Mare parte dintre proiectanţii şi dezvoltatorii de aplicaţii economice nu iau prea în serios problemele ce ţin de legislaţia afacerilor, care interzice modificarea ulterioară a unora dintre datele privitoare la operaţiuni consemnate în documente oficiale. Mai precis, odată ce o firmă a depus la administraţiile financiare balanţa de verificare sau bilanţul, îi este interzis să mai modifice retroactiv datele respective, chiar dacă descoperă greşeli. Corectarea se realizează, de obicei, sub forma stornării urmată de preluarea sumelor corecte, ambele operaţiuni fiind consemnate în contul lunii sau lunilor următoare descoperirii erorii.

În ceea ce ne priveşte, vom examina în acest paragraf două tipuri de „interdicţii temporale". Prima priveşte eternele şi fascinantele facturi emise. O factură emisă consemnează o vânzare de produse şi servicii. Fiind un document fundamental în relaţiile dintre partenerii de afaceri, având regim de probă în justiţie, ca să nu mai vorbim de apetitul fiscului în materie de impozite şi taxe, factura este unul dintre cele mai sensibile documente întocmit de firmă, având un regim de întocmire şi circulaţie foarte strict. De aceea, după redactarea facturii şi listarea sa se face o verificare temeinică a datelor sale, după care factura îşi începe circuitul. Acesta este momentul după care factura nu trebuie să mai sufere nici o modificare !

Din perspectiva bazei de date, introducerea unei facturi înseamnă inserarea unei linii în tabela FACT, apoi a *n* tupluri în LINII_FACT, corespunzător celor *n* produse şi servicii vândute. După inserarea celor *n* linii în LINII_FACT, ar mai putea surveni modificări în ambele tabele, până la confirmarea facturii, care semnalizează definitivarea sa. Problema e că, în actuala structură, momentul definitivării facturii nu este preluat în tabelele bazei de date VÎNZĂRI. O soluţie destul de simplă ar fi să recurgem la un atribut special – `Definitivată` – pe care să-l introducem în tabela FACT:

```
ALTER TABLE fact ADD definitivata CHAR(1) DEFAULT 'N'
      NOT NULL CHECK (definitivata IN ('D','N')) ;
```

Prin interfaţa aplicaţiei, utilizatorul va semnaliza definitivarea şi „închiderea" facturii, valoarea atributului `Definitivată` devenind *D* (da). Interzicerea oricărei modificări în factură sau ştergerii facturii va cădea în seama declanşatoarelor tabelelor FACT şi LINII_FACT care vor verifica starea noului atribut. Listingul 10.10 conţine banalul declanşator de ştergere al tabelei FACT.

Listing 10.10. Blocarea ştergerii unei facturi definitivate

```
CREATE OR REPLACE TRIGGER trg_fact_del
    BEFORE DELETE ON fact FOR EACH ROW
```

```
BEGIN
    IF :OLD.definitivata='D' THEN
        RAISE_APPLICATION_ERROR(-20349, 'Nu puteti sterge o factura definitivata !');
    END IF ;
END ;
/
```

Lucrurile nu sunt prea diferite pentru declanşatorul de modificare, iar declanşatoarele de inserare, modificare şi ştergere în LINII_FACT ar putea beneficia de o funcție specială care să returneze valoarea atributului-stare. Atenție ! Dacă tabela FACT conține şi atributul calculat `ValÎncasată`, acesta are un regim special, putând fi editat până la încasarea totală a facturii.

Cei care cunosc, fie şi sumar, practica financiar-contabilă ar putea obiecta: bine-bine, dar dacă din momentul confirmării facturii nu mai putem face nici o modificare, cum consemnăm faptul că factura respectivă a fost, peste câteva ore, zile, săptămâni, chiar luni, stornată ? Mai ales că ar trebui să consemnăm atât momentul stornării, cât şi, eventual, noua factură ce reglează stornarea. Variante de lucru:

- în FACT introducem atributele `DataStornării` şi `NrFactCorectoare`, iar prin declanşatoare autorizăm modificarea post-definitivare numai a acestor două câmpuri;
- în FACT introducem atributul `EsteStornată` cu valori D/N, şi, în plus, creăm o tabelă specială FACT_STORNATE {`NrFact`, `DataStornării`, `NrFactCorectoare`};
- păstrăm neschimbată structura tabelei FACT, gestionând stornarea exclusiv prin tabela FACT_STORNATE {`NrFact`, `DataStornării`, `NrFactCorectoare`}.

Ultima variantă este cea mai puțin traumatizantă pentru declanşatoarele bazei, însă are marele dezavantaj că, la obținerea valorile exacte ale vânzărilor, trebuie făcută, de fiecare dată, joncțiunea externă a tabelei FACT cu FACT_STORNATE.

Al doilea tip de „interdicție temporală" este tocmai cel invocat în primele fraze ale paragrafului – închiderea contabilă. La finalul unei luni calendaristice, sau în primele zile ale lunii următoare, are loc operațiunea creatoare de frisoane pentru orice contabil care se respectă – închiderea de lună. Cei care au avut ocazia să lucreze într-o firmă/organizație ştiu că în anumite zile din lună în biroul *Contabilitate* se merge pe vârfuri, iar câteva persoane cu fețe congestionate nu pot fi deranjate nici în ruptul capului, deoarece „fac închiderea" ! Dincolo de legende şi datini populare contabile, închiderea presupune centralizarea operațiunilor pe module ale sistemului informațional financiar-contabil (imobilizări, stocuri, încasări/plăți, salarizare etc.), calculul dărilor către stat (TVA, impozite şi contribuții legate de salarizare), descărcarea conturilor de cheltuieli şi venituri etc.

Din momentul terminării închiderii, listării/redactării şi depunerii la fisc a tuturor documentelor, se interzice modificarea retroactivă a datelor conținute în rapoartele respective. Lucrurile sunt absolut normale, deoarece orice astfel de modificare ar putea antrena decalaje informaționale între datele din bază şi docu-

mentele oficiale. De aceea, este nevoie de o tabelă specială pe care o vom numi ÎNCHIDERI.

```
CREATE TABLE inchideri (
      an NUMBER(4) DEFAULT EXTRACT (YEAR FROM CURRENT_DATE)
            NOT NULL CHECK (an >= 2004),
      luna NUMBER(2) DEFAULT
            EXTRACT (MONTH FROM CURRENT_DATE)
            NOT NULL CHECK (luna BETWEEN 1 AND 12),
      datainchiderii DATE DEFAULT CURRENT_DATE NOT NULL,
      operator VARCHAR2(30) DEFAULT USER,
      PRIMARY KEY (an, luna)
      ) ;
```

La momentul finalizării închiderii pe luna februarie 2005, se va insera o linie în această tabelă:

```
INSERT INTO inchideri (an, luna) VALUES (2005,2);
```

Orice inserare, modificare sau ştergere care antrenează valori trebuie supusă controlului „temporal”; astfel, la modificarea valorii sau TVA-ului unei facturi, trebuie verificat dacă respectiva factură nu face parte dintr-o lună pentru care s-a făcut închiderea, caz în care inserarea/modificarea/ştergerea respectivă trebuie anulată. Pentru mărirea vitezei de lucru pachetul `pac_închideri` din listingul 10.11 foloseşte o variabilă publică iniţializată la începutul sesiunii şi la o inserare în tabela INCHIDERI. Banalul declanşator de ştergere în tabela FACT va verifica, suplimentar, dacă data facturii întocmite se situează pe o lună deja „închisă”, caz în care va declanşa o eroare.

Listing 10.11. Pachetul şi un declanşator dedicate închiderilor contabile

```
-------------------------------------------------------------
-- pachetul PAC_INCHIDERI
CREATE OR REPLACE PACKAGE pac_inchideri AS
    v_ultima_zi_inchisa DATE := NULL ;
    PROCEDURE init_v_ultima_zi_inchisa ;
END ;
/
CREATE OR REPLACE PACKAGE BODY pac_inchideri AS
----------------------------------
PROCEDURE init_v_ultima_zi_inchisa
IS
    v_data DATE ;
BEGIN
    IF v_ultima_zi_inchisa  IS NULL THEN
        SELECT MAX(TO_DATE('01/'|| luna || '/' || an, 'DD/MM/YYYY'))
        INTO v_data FROM inchideri ;
        v_ultima_zi_inchisa := LAST_DAY(v_data) ;
    END IF ;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        v_ultima_zi_inchisa := DATE'1999-12-31' ;
```

```
END ;
END ; -- pachet
/
--------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_inchideri_ins
    AFTER INSERT ON inchideri
BEGIN
    pac_inchideri.v_ultima_zi_inchisa := NULL ;
    pac_inchideri.init_v_ultima_zi_inchisa ;

END ;
/
--------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_fact_del
    BEFORE DELETE ON fact FOR EACH ROW
BEGIN
    IF :OLD.definitivata='D' THEN
        RAISE_APPLICATION_ERROR(-20349, 'Nu puteti sterge o factura definitivata !');
    END IF ;

    pac_inchideri.init_v_ultima_zi_inchisa ;

    IF pac_inchideri.v_ultima_zi_inchisa >= :OLD.DataFact THEN
        RAISE_APPLICATION_ERROR(-20899,
            'Factura apartine unei luni <<inchise>> !') ;
    END IF ;
END ;
/
```

10.4. Arhivări, fuziuni, compatibilități

Chiar şi în SGBD-urile generoase spațiul este unul limitat, iar viteza de interogare vitală pentru bunul mers al aplicației. Interesul unei scheme de stocare este de a prelua toate informațiile relevante în vederea consultării lor ulterioare. Cu cât datele preluate sunt mai analitice, şi numărul trazacțiilor companiei ce deține baza de date este mai ridicat, cu atât dimensiunea bazei devine tot mai mare, atingând, nu de puține ori, în câțiva ani, zeci şi sute de gigaocteți, poate chiar mai mult. Dacă în paragraful 9.7 ne puneam problema ruperii relațiilor „pe verticală" în vederea creşterii vitezei de acces prin gruparea atributelor în tabele diferite (funcție de frecvența folosirii atributelor), acum ne gândim să rupem pe orizontală tabelele „tranzacționale", prin mutarea liniilor referitoare la operațiuni finalizate cu ceva timp în urmă.

Să luăm cazul bazei de date VÎNZĂRI. Aplicația a fost implementată în 1997 şi, dacă nu recurgem la arhivare, aflarea valorilor facturate şi încasate ale unui client în mai 2005 prespune consultatea unor tabele FACT, LINII_FACT, INCAS_FACT uriaşe. Chiar dacă SGBD-urile dispun de mecanisme optimizatoare ale accesului, este de aşteptat ca o tabelă de ordinul gigaocteților să presupună o viteză de consultare sensibil mai mică decât una de dimensiunea câtorva megaocteți.

Practic, datele din perioada 1997-2003 sunt mult mai rar consultate decât cele din ultimele 12 luni. Tocmai luând în calcul o viteză de acces cât mai bună, putem să apelăm la o serie de tabele pentru arhivare, cum ar fi FACT_ARHIVATE, LINII_FACT_ARHIVATE. Operaţiunea de arhivare ar presupune vărsarea tuturor liniilor până la o dată specificată – vezi listing 10.12.

Listing 10.12. Arhivarea facturilor

```
DROP TABLE linii_fact_arhivate ;
DROP TABLE incas_fact_arhivate ;
DROP TABLE incasari_arhivate ;
DROP TABLE fact_arhivate ;
CREATE TABLE fact_arhivate AS SELECT * FROM fact WHERE 1=2 ;
CREATE TABLE linii_fact_arhivate AS SELECT * FROM linii_fact WHERE 1=2 ;
CREATE TABLE incasari_arhivate AS SELECT * FROM incasari WHERE 1=2 ;
CREATE TABLE incas_fact_arhivate AS SELECT * FROM incas_fact WHERE 1=2 ;

ALTER TABLE fact_arhivate ADD DataArhivarii DATE ;
ALTER TABLE fact_arhivate ADD PRIMARY KEY (NrFact) ;

ALTER TABLE linii_fact_arhivate ADD DataArhivarii DATE ;
ALTER TABLE linii_fact_arhivate ADD PRIMARY KEY (NrFact, Linie) ;

ALTER TABLE incasari_arhivate ADD DataArhivarii DATE ;
ALTER TABLE incasari_arhivate ADD PRIMARY KEY (CodInc) ;

ALTER TABLE incas_fact_arhivate ADD DataArhivarii DATE ;
ALTER TABLE incas_fact_arhivate ADD PRIMARY KEY (CodInc, NrFact) ;


CREATE OR REPLACE PROCEDURE p_arhivare_fact (datafact_ DATE)
IS
BEGIN
    -- se adauga liniile de arhivat in tabele-arhiva
    INSERT INTO linii_fact_arhivate
        SELECT linii_fact.*, SYSDATE
        FROM linii_fact
        WHERE nrfact IN
            (SELECT nrfact FROM fact
                    WHERE datafact <= datafact_) ;
    INSERT INTO fact_arhivate
        SELECT fact.*, SYSDATE FROM fact WHERE datafact <= datafact_ ;

    INSERT INTO incasari_arhivate
        SELECT incasari.*, SYSDATE FROM incasari
        WHERE codinc IN
            (SELECT codinc FROM incas_fact WHERE nrfact IN
            (SELECT nrfact FROM fact WHERE datafact <= datafact_)
                    );
    INSERT INTO incas_fact_arhivate
        SELECT incas_fact.*, SYSDATE FROM incas_fact
        WHERE nrfact IN (SELECT nrfact FROM fact
                    WHERE datafact <= datafact_) ;

    -- se dezactivea declasatoarele, pentru a evita verificarile suplimentare
    --    si actualizarea atributelor calculate
    EXECUTE IMMEDIATE ' ALTER TABLE linii_fact DISABLE ALL TRIGGERS ' ;
    EXECUTE IMMEDIATE ' ALTER TABLE incas_fact DISABLE ALL TRIGGERS ' ;
```

```
    EXECUTE IMMEDIATE ' ALTER TABLE fact DISABLE ALL TRIGGERS ' ;
    EXECUTE IMMEDIATE ' ALTER TABLE incasari DISABLE ALL TRIGGERS ' ;

    -- se sterg liniile arhivate
    DELETE FROM linii_fact  WHERE nrfact IN
        (SELECT nrfact FROM fact WHERE datafact <= datafact_) ;
    DELETE FROM incas_fact WHERE nrfact IN
        (SELECT nrfact FROM fact WHERE datafact <= datafact_) ;
    DELETE FROM incasari WHERE codinc IN
        (SELECT codinc FROM incasari_arhivate) ;

    DELETE FROM fact WHERE datafact <= datafact_ ;

    -- se reactiveaza declasatoarele
    EXECUTE IMMEDIATE 'ALTER TABLE linii_fact ENABLE ALL TRIGGERS ' ;
    EXECUTE IMMEDIATE 'ALTER TABLE fact ENABLE ALL TRIGGERS ' ;
    EXECUTE IMMEDIATE 'ALTER TABLE incasari ENABLE ALL TRIGGERS ' ;
    EXECUTE IMMEDIATE 'ALTER TABLE incas_fact ENABLE ALL TRIGGERS ' ;
END ;
```

În condițiile în care tabelele-arhivă vor fi protejate la ştergeri şi modificări, rezultă un spor de viteză considerabil, deoarece tabele sunt mult mai „suple", păstrând numai informații recente. Atenție ! Arhivarea nu trebuie operată prea des, iar facturile arhivate trebuie să fi fost reglate (încasate) definitiv !

Marea problemă a acestui stil de lucru este că rapoartele şi interogările privind vânzările şi încasările trebuie să-şi extragă informațiile atât din tabelele FACT, LINII_FACT..., cât şi din arhivele lor. Se pot crea, în acest scop, tabele virtuale, după modelul:

```
CREATE VIEW vFact AS
       SELECT fact.*, SYSDATE AS DataCurenta
       FROM fact
             UNION
       SELECT *
       FROM fact_arhivate
```

În continuare să presupunem că două firme informatizate până în dinți decid să fuzioneze. În urma fuziunii, şi bazele de date se vor uni, ce-i drept nici dintr-o sorbire, nici dintr-un click de mouse. Dacă schemele sunt similare, şi sunt motive să credem că, pentru unele aplicații precum VÎNZĂRI, supoziția nu este chiar neacoperită în practică, ce se întâmplă cu entitățile comune ? Cu profile similare, cele două firme au, mai mult ca sigur, clienți comuni şi produse comune. Este evident că după fuziune, trebuie să rămână o singură instanță a fiecărei entități, altminteri identificarea ar fi imposibilă, iar informațiile extrase eronate.

Spre exemplu, cele două firme aveau fiecare câte o tabelă de clienți pe care le vom nota CLIENȚI1 şi CLIENȚI2. Codurile clienților nu se suprapun, însă firma partener ce este identificată prin codul 2011 în CLIENȚI1 este aceeaşi cu cea desemnată de codul 1001 în CLIENȚI2. Trebuie, aşadar, să renunțăm la una dintre instanțe. Scenariul n-ar fi prea complicat:

```
-- se "varsă" CLIENTI2 în CLIENTI1 şi procedăm analog cu
--           tabele copii, nepoţi etc: FACTURI, LINIFACT…
INSERT INTO clienti1 SELECT * FROM clienti2 ;
INSERT INTO facturi1 SELECT * FROM facturi2 ;
-- …
--

-- se modifică în tabelele copil valorile 2011 ale CodCl în 1001
UPDATE facturi1 SET codcl=1011 WHERE codcl=2011
-- … se continuă cu ceilalţi copii ai tabelei CLIENTI

-- se şterge din CLIENTI1 linia clientului 2011
DELETE FROM clienti1 WHERE codcl=2011
```

Bun, şi acum vestea proastă: cele două firme sunt, de fapt, lanţuri de supermagazine care şi-au implementat sistemele informaționale de ani buni, iar datele adunate sunt atât de voluminoase încât nu încap în întregime pe hard-disk, găsindu-se doar pe câteva sute de CD-ROM-uri. Şi, chiar dacă ar încăpea pe un întreg hard-disk, nu avem nici resurse şi nici timp pentru a le copia, actualiza şi rescrie.

Am luat acest exemplu pentru a vă intimida, însă avem şi un altul mult mai banal. Deseori, atunci când o aplicație este exploatată de mai mulți utilizatori, se întâmplă ca, din neatenție, unul dintre ei să nu sesizeze că clientul din factură există deja (poate nici nu se vede prea bine numele acestuia pe factură) în bază, alege butonul *Client nou* al formularului şi... gata, în câteva zeci de secunde de alegem cu un client aflat pe două linii ale tabelei CLIENȚI. Iar dacă samavolnicia e identificată mult mai târziu, după salvarea pe CD a bazei de date şi ştergerea înregistrărilor salvate, avem o situație ce nu diferă prea mult de cea descrisă anterior.

Fireşte cea mai bună pare a fi soluția de jurnalizare a modificărilor din paragraful 10.2, ce ar putea fi reexaminată şi rescrisă pentru a rezolva şi această problemă. Putem apela, însă, şi la o altă variantă mai simplă în care să stocăm corespondențele codurilor clienților : CODURI_CLIENTI_SCHIMBATE {CodClActual, UnCodClAnterior, DataConsemnarii}.

Grija cea mai mare a soluției este ca nici o valoarea a CodCl să nu se recicleze. Odată folosită, din momentul renunțării la ea, valoarea trebuie să rămână blocată, altminteri confuziile ar fi iminente. În ultima propoziție din paragraful 9.5 amenințam că vom reveni cu câteva detalii legate de folosirea de relații redundante pentru gestionarea corespondențelor dintre datele actuale şi cele anterioare. Or un asemenea tip de tabelă ne-ar putea rezolva problema evitării reciclărilor ulterioare, mai ales atunci când o parte dintre arhive nici nu mai sunt în baza de date curentă, ci arhivate pe CD-uri:

CLIENȚI_ARHIVAȚI {CodCl, DataArhivării}
FACT_ARHIVATE {NrFact, CodCl, DataArhivării}

Inserările şi modificările `CodCl` şi `NrFact` în CLIENŢI şi FACT vor trebui monitorizate, declanşatoarele celor două tabele consultând aceste două tabele arhivă, evitându-se, astfel, valorile ce au fost deja salvate şi arhivate.