Capitolul 9. Denormalizare şi postnormalizare

Scopul prioritar al proiectării bazei de date este găsirea unei scheme de stocare care să preia toate informaţiile relevante pentru aplicaţie, să permită declararea restricţiilor pe care trebuie să le respecte datele în bază şi să ofere suport pentru prelucrările ulterioare (grupare, însumare, analize încrucişate) ale datelor în vederea identificării informaţiilor relevante fiecărui grup de utilizatori. Cu toate acestea, proiectarea anticipează şi o serie de probleme vitale pentru bunul mers al aplicaţiei, cum ar fi viteza de acces, siguranţa datelor, politica de acces etc. Chiar dacă acestea ţin preponderent de *proiectarea fizică* a bazei, câteva chestiuni pot fi abordate şi în faza de proiectare logică.

Problema denormalizării este una de "tradiţie" în proiectarea bazelor de date relaţionale, şi destul de agitată, dacă luăm în considerare diferenţele de opinii în definirea şi mecanismele sale. Definită încă din anii `80 şi disputată în a doua parte a anilor `90, denormalizarea şi-a stabilizat propria-i „piaţă", alcătuită din două tabere ireductibile, una a teoreticienilor de tip Fabian Pascal[1], fundamentalişti gata de a salva oricând teoria dând vina exclusiv pe piaţă, mai precis, pe slăbiciunile SGBD-urilor, iar cealaltă a practicienilor, uneori ignoranţi de-a binelea în materie chiar de noţiuni fundamentale despre bazele de date, pentru care orice, dar absolut orice, este posibil şi permis într-o schemă de baze de date, atâta vreme cât binele aplicaţiei sau programatorilor o cere sau cerşeşte.

Ca şi până acum, noi vom face în acest capitol naveta între fundamentalişti şi ignoranţi, încercând pe cât posibil să conturăm câteva atuuri şi pericole ale denormalizării şi, fireşte, atrăgându-ne dezaprobarea şi antipatia ambelor categorii prin „întinarea nobilelor idealuri"[2] de o manieră „imparţială".

9.1. Denormalizarea - rău-rău, dar chiar necesar ?

Una din primele definiţii ale denormalizării fost formulată de Candace Fleming şi Barbara von Halle: procesul prin care, după definirea unei structuri de date stabile, deplin normalizate, se introduc selectiv date redundante pentru facilitarea unor cerinţe specifice legate de performanţă[3]. Într-o lucrare mai recentă, Robert Muller definea denormalizarea ca procesul combinării de tabele sau obiecte pentru a creşte viteza de acces, de obicei prin evitarea joncţiunilor[4]. Ceva mai analitică,

---

[1] Vezi, spre exemplu, [Pascal01]

[2] Nu intenţionez să plătesc drepturi de autor pentru aceste trei cuvinte lemnoase ale unui personagiu clădit din aceeaşi esenţă (precum cuvintele).

[3] [Fleming&vonHalle 89], p.440

[4] [Muller99], p.6

Toby Teorey definea denormalizarea ca un proces alcătuit din următoarele activități[5]:

- selectarea proceselor dominante, pe baza frecvenței, volumului, priorității;
- definirea unor extensii ale tabelelor existente (atribute sau chiar tabele noi) care să amelioreze performanțele de interogare a bazei;
- evaluarea costului total al interogării, stocării şi actualizării;
- evaluarea efectelor colaterale, cum ar fi riscul pierderii integrității datelor.

Hoffer *s.a.* leagă denormalizarea strict de nivelul fizic al schemei unei baze de date, definind-o destul de exotic drept procesul transformării relațiilor normalizate în specificații privind înregistrări fizice nenormalizate[6]. Din contră, pentru Elmasri şi Navathe, procesul denormalizării se desfăşoară preponderent la nivel logic: procesul stocării joncțiunilor dintre relații aflate în forme normale avansate ca relații aflate, inevitabil, în forme normale mai „joase" este cunoscut sub numele de denormalizare[7]. Amuzant este şi faptul că, dacă pentru Elmasri şi Navathe denormalizarea este o „cădere" din forme normale înalte în altele mai joase, Connoly şi Begg definesc denormalizarea ca o rafinare a schemei relaționale[8].

Probabil cel mai des invocat motiv pentru denormalizare este timpul necesar efectuării joncțiunilor între tabelele fragmentate rezultate în urma normalizării. Dacă numărul tabelelor este mare, numărul de rapoarte/informații solicitate de utilizatori este uriaş şi presupune joncționări numeroase, utilizatorii sunt numeroşi, iar cerințele de viteză sunt vitale, tentația este de a introduce atribute redundante sau de a duplica atribute în tabele diferite, astfel încât numărul joncțiunilor necesare obținerii informațiilor frecvente să fie cât mai mic cu putință. Prin urmare, uneori, deviațiile de la o bună normalizare par logice şi practice[9]. Chis Date sintetizează foarte bine argumentul în trei părți al denormalizatorilor[10]:

1. Normalizarea deplină înseamnă multe relații separate logic.
2. Mai multe relații logic separate înseamnă mai multe fişiere de stocare separate fizic.
3. Mai multe fişiere de stocare separate fizic înseamnă un mai mare volum de operații de intrare/ieşire, deci consum de resurse.

Craig Mullins sugerează că există câteva simptome care trebuie să atragă atenția asupra schemelor susceptibile de denormalizare[11]:

i. rapoarte şi interogări critice extrag informațiile prin interogarea simultană a multor tabele;

---

[5] [Teorey99], p.8

[6] [Hoffer s.a. 02], p.215

[7] [Elmasri & Navathe 00], p.484

[8] [Connolly & Begg 02], p.507

[9] [Poole 00]

[10] [Date04], p.393

[11] [Mullins98]

ii. există grupuri repetitive care sunt nu procesate atât individual, cât la nivel de set (grup);
iii. interogările/rapoarte reclamă calcule aplicabile multor atribute aflate în tabele diferite;
iv. utilizatori diferiți accesează simultan aceeaşi tabelă sau grup de tabele după criterii diferite;
v. cheile primare ale unor tabele sunt prea mari şi consumă timp la joncționări;
vi. există diferențe majore, din punct de vedere statistic, între gradul de folosire în interogări a unor atribute față de altele.

Bock şi Schrage remarcau, pe bună dreptate, că puține sunt cărțile care să prezinte în detaliu cum anume trebuie făcută denormalizarea[12], dând, totuşi, ca exmplu, lucrarea Gillette s.a.[13] care recomandă trei linii directoare:

I. identificarea aplicațiilor-program care conțin un număr excesiv de joncțiuni, desemnând drept excesive cazurile în care o tabelă virtuală, tabelă-obiect necesită mai mult de trei joncțiuni;
II. reducerea numărului de chei străine, care ar atrage descongestionarea indecşilor ce trebuie gestionați la actualizările tabelelor; reducerea cheilor străine se traduce, de fapt, prin reducerea numărului de tabele din bază;
III. menținerea mecanismului de integritate pentru evitarea anomaliilor datelor din bază.

Cele trei linii directoare nu suferă doar de generalitate, ci chiar sunt discutabile, în sine, întrucât aplicarea lor trebuie raportată permanent la situația efectivă „din teren".

După cum punctau Fleming şi von Halle, problemele de performanță în utilizarea bazelor de date nu țin de structura relațională obținută prin normalizare, ci de mecanismul deficient de stocare şi acces la date specific fiecărui produs comercial. Dacă SGBD-urile ar fi perfecte, atunci schema obținută în urma proiectării bazei de date nu ar prezenta probleme de performanță[14]. Ideea este susținută şi de Bock şi Schrage pentru care, slabele performanțe ale sistem depind şi de[15]:

- platforme hardware inadecvate;
- slaba optimizare (acordare – tuning) a SGBD-ului;
- tehnici de programare rudimentare, stufoase sau neglijente;
- proiectare conceptuală/logică inadecvată;
- schemă fizică nepusă bine la punct.

[12] [Bock & Schrage 02]
[13] [Gillette s.a. 95]
[14] [Fleming&vonHalle 89], p.435
[15] [Bock & Schrage 02]

Fireşte, pentru lucrarea de faţă ne interesează doar aspectele ce ţin de schemele (logică, în mai mică măsură, fizică) bazei.

Chris Date demontează argumentele denormalizării, deoarece acestea pornesc de la premisa falsă că o tabelă corespunde unui fişier fizic. Or, nicăieri în modelul relaţional acest lucru nu este nici măcar sugerat. Paradoxul sesizat de autor este că, dacă este necesară, denormalizarea ar putea fi efectuată la nivel fizic, dar la nivelul fizic nu putem discuta de relaţii, ci de fişiere fizice[16]. De fapt, Date (ca şi Fabian Pascal, dealtminteri) nu scapă prilejul de a da vina pe SQL şi actualele SGBD-uri bazate pe SQL care nu operează cum trebuie separarea nivelului logic de cel fizic într-o bază de date.

Fleming şi von Halle sunt de părere că deviaţiile de la structura obţinută prin normalizare sunt cauţionabile numai dacă[17]: (1) alte mecanisme de optimizare sunt inadecvate sau indisponibile; (2) cerinţele de performanţă sunt imperioase; (3) există posibilitatea implementării unui mecanism care, chiar în condiţiile duplicării informaţiilor, asigură integritatea şi consistenţa datelor; (4) orice deviere de la modelul logic al datelor este documentată şi explicată, astfel încât să existe o cunoaştere deplină asupra artificiilor folosite în schemă. Ideea este susţinută şi de Craig Mullins care sugerează ca, înainte de a porni la denormalizarea unei scheme, să se zăbovească asupra a trei chestiuni[18]:

sistemul poate atinge performanţe acceptabile fără denormalizare ?

după denormalizare, performanţa sistemului va continua să fie inacceptabilă ?

va deveni sistemul, în urma denormalizării, mai puţin sigur ?

Dacă răspunsul la măcar una dintre aceste întrebări este afirmativ, atunci denormalizarea trebuie evitată.

Chiar şi adepţii necondiţionaţi ai denormalizării atrag atenţia asupra faptului că denormalizarea are un cost, tradus, de cele mai multe ori, în pierderea flexibilităţii, diminuarea scalabilităţii, performanţe mai slabe ale aplicaţiei/bazei de date sau chiar ameninţări la adresa integrităţii datelor. Plus, efortul necesitat de mecanismele laborioase de păstrare a corectitudinii informaţiilor paralele, redudandante[19].

9.2. Categorii de operaţii legate de denormalizare

Există o largă tipologie a operaţiunilor ce pot fi subsumate denormalizării. De fapt, după cum spunea şi Date, una dintre problemele uşor de conştientizat ţine de faptul că, în genera, este destul de uşor de identificat momentul în care trebuie începută denormalizarea şi aproape imposibil de ştiut când trebuie oprită[20]. Ironic, autorul plusează afirmând că, dacă în cazul normalizării, obiectivul era aducerea

[16] [Date04], p.393

[17] [Fleming&vonHalle 89], p.438

[18] [Mullins98]

[19] Vezi, spre exemplu [Hoberman02]

[20] [Date04], p.394

bazei într-o formă normală cât mai „înaltă", denormalizarea ar putea să-şi propună drept țel o formă normală cât mai joasă.

Dick Root identifică şapte tehnici utilizabile în denormalizare[21]:

1. Date duplicate (copii integrale sau parțiale ale unor atribute sau grupuri de atribute).
2. Date derivate/calculate.
3. Chei surogate.
4. Supra-normalizare - partiționare/segmentare pe verticală: ruperea unei relații în două sau mai multe.
5. Partiționare/segmentare pe orizontală: linii diferite dintr-o relație sunt stocate în tabele diferite.
6. Joncțiuni stocate – joncționarea a două sau mai multe tabele şi stocarea rezultatului într-o tabelă de-sine-stătătoare.
7. Grupuri de date repetitive (date de tip vector).

Într-o cu totul altă abordare, Bock şi Schrage tratează câteva modalități de denormalizare aplicabile fiecărei forme normale[22]: 1NF, 2NF, 3NF cu câteva sugestii pentru formele normale „mai înalte". Acestei abordări i se poate reproşa faptul că amestecă denormalizarea cu normalizarea, în timp ce majoritatea autorilor recomandă ca discuțiile legate de normalizăre să fie purtate numai după ce schema bazei de date este adusă în 3NF sau o formă ulterioară acesteia.

Mullins recomandă folosirea, în cadrul denormalizării, a următoarelor artificii[23]:

- tabele pre-joncționate;
- tabele-raport, construite în funcție de rapoartele critice ale aplicației;
- tabele-oglindă, mai ales în mediile cu acces simultan al unui mare număr de utilizatori;
- tabele partiționate, în funcție de grupurile de utilizatori;
- tabele combinate, pentru consolidarea legăturilor *unu la mai mulți* într-o singură tabelă;
- date copiate (duplicate);
- grupuri repetitive;
- date derivate;
- tabele de mărire a vitezei, recomandabile mai ales în cazul ierarhiilor.

Atributele duplicate sunt, probabil, cea mai comună categorie a denormalizării. Un atribut este duplicat dacă se repetă, sub formă identică sau similară, într-una sau mai multe tabele din bază. Din această categorie sunt excluse atributele de tip cheie străină necesare a pune în legătură relațiile copil cu cele părinte. Fleming şi von Halle sunt de părere că există patru tipuri de atribute duplicate, şi nici unul nu este absolut necesar[24]:

---

[21] [Root01]

[22] [Bock & Schrage 02]

[23] Mullins98]

[24] [Fleming&vonHalle 89], p.440

1. copii exacte ale unui atribut (se exclud din discuție cheile străine);
2. atribute derivate din cele existente, sau sintetice (calculate prin însumare, numărare sau alte calcule aplicate altor atribute);
3. grupuri repetitive care, inițial, apar pe mai multe linii în tabelă;
4. substitute artificiale ale unor atribute existente.

Deşi nu o fac la modul cel mai explicit, cele două autoare lasă să se înțeleagă că doar primul tip (copiile exacte în tabele diferite) poate în încadrat la categoria *denormalizare*. Este şi părerea noastră, şi, în acelaşi timp, argumentul pentru tăcerea mormântală de până acum în privința denormalizării, deşi redundanțe am introdus încă din capitolul 7.

Țin, totuşi, cheile surogat de denormalizare ? Root spune că da, iar dacă interpretăm în sens larg al patrulea tip de atribute duplicate din clasificarea de mai sus, şi Fleming şi von Halle sunt pe-aproape. Eu, unul, mă raliez celor care spun că nu. Este drept, cheile surogat introduc un anumit grad de redundanță în relații, prin faptul că prezintă, aşa cum am văzut în capitolul 7, un grad de artificialitate, adăugându-se celorlalte atribute ce caracterizează „natural" o entitate, proces, fenomen. Însă, după cum spunea Date, nu orice redundanță înseamnă denormalizare !

Față de cele declarate în paragraful 7.1, s-ar mai putea spune că, uneori, tentația „surogării" unui grup de atribute (în sensul introducerii unui substitut artificial pentru un atribut/grup de atribute) în vederea simplificării restricțiilor referențiale şi a joncțiunilor este irezistibilă. Insist că este vorba de simplificarea joncțiunilor/restricțiilor referențiale, şi nu eliminarea lor. Să luăm un exemplu. De câteva capitole discutăm de baza de date dedicată gestionării studenților şi mai ales rezultatelor de la examene. În perspectiva procesului „Bologna", învățământul superior va fi organizat pe trei cicluri, licență, specializare+master şi doctorat. Aşa că structura tabelei STUD ar putea fi:

```
CREATE TABLE stud (
   matricol VARCHAR2(18) PRIMARY KEY,
   numepren VARCHAR2(50),
   cnp CHAR(13),
   ciclu_studii CHAR(1),
   forma_de_studii CHAR(1),
   an_studii NUMBER(1),
   modul_specializare VARCHAR2(30),
   serie_curs NUMBER(1),
   grupa NUMBER(2)
   );
```

Atributul `Ciclu_Studii` ar putea avea doar una dintre valorile 1, 2, 3, `Forma_De_Studii` este *Z* (învățământ de *zi*) sau *D* (învățământ la *distanță*), iar `Modul_Specializare` ar avea o dublă folosință: dacă studentul este în primii ani, în care cursurile sunt comune (în aceste semestre, studenții sunt afiliați unor module de studii), atunci valoarea atributului desemnează un modul, în timp ce pentru studenții înscrişi în ultimele semestre ale ciclului 1, sau în ciclurile 2 sau 3,

atunci valoarea semnalizează specializarea pe care o urmează. La fiecare modul/specializare, ciclu, formă şi an de studii pot fi mai multe serii de curs.

În privința disciplinelor studiate, n-ar fi prea multe de adăgat:

```
CREATE TABLE discipl (
   coddisc CHAR(6) PRIMARY KEY,
   dendisc VARCHAR2(60),
   nrcreddisc NUMBER(2)
   );
```

Planul de învătâmânt se stabileşte la nivel de ciclu/formă/an/ modul/specializare:

```
CREATE TABLE curricula (
   ciclu_studii CHAR(1),
   forma_de_studii CHAR(1),
   an_studii NUMBER(1),
   modul_specializare VARCHAR2(30),
   coddisc CHAR(6) REFERENCES discipline (coddisc),
   categorie_disc CHAR(1),
   nr_ore_curs NUMBER(3),
   nr_ore_seminar NUMBER(3),
   nr_ore_laborator NUMBER(3),
   PRIMARY KEY (ciclu_studii, forma_de_studii,an_studii,
   modul_specializare, coddisc)
   );
```

unde `Categorie_Disc` indică dacă disciplina respectivă este obligatorie, opțională sau facultativă, iar programarea examenelor se face tot pe cicluri, forme, ani şi module/specializări de studii:

```
CREATE TABLE examene (
   ciclu_studii CHAR(1),
   forma_de_studii CHAR(1),
   an_studii NUMBER(1),
   modul_specializare VARCHAR2(30),
   coddisc CHAR(6) REFERENCES discipline (coddisc),
   dataex DATE,
   salaex VARCHAR2(30),
   PRIMARY KEY (ciclu_studii, forma_de_studii,an_studii,
   modul_specializare, coddisc,dataex)
   );
```

Ce-i drept, nu întotdeauna un întreg an de studiu încape într-o sală, dar să presupunem că am avea voie ca valorile atributului `SalaEx` să nu fie scalare.

Nu putem să reproşăm prea multe acestei structuri. Însă, de fiecare dată când vom dori să aflăm ce discipline sunt în planul de învățământ al formației de studiu din care face parte studentul *Stroici Y Vasile*, altfel spus, care îi sunt, pentru anul în curs, disciplinele pe care trebuie să le urmeze (obligatorii) sau din care trebuie să aleagă (opționalele), ne-ar trebui o interogare de genul:

```
SELECT dendisc, categorie_disc, nrcreddisc
FROM stud
```

```
   INNER JOIN curricula
   ON stud.ciclu_studii = curricula.ciclu_studii
       AND stud.forma_de_studii = curricula.forma_de_studii
       AND stud.an_studii = curricula.an_studii AND
   stud.modul_specializare = curricula.modul_specializare
   INNER JOIN disc
       ON curricula.coddidsc = disc.coddisc
WHERE numepren = 'Stroici Y Vasile'
```

Celor care n-au obosit le putem recomanda călduros să afle în ce zi are programat examenul la *Baze de date* studentul *Pingică I George*. În asemenea condiții, să nu-i judecăm prea aspru pe cei care vor apela la un atribut care să substituie faimoasa combinație (`Ciclu_Studii, Forma_de_Studii, An_Studii, Modul_Specializare`). Atributul ar fi un surogat în toată regula, pe care l-am putea boteza `IdFormaţie`. Acesta ar fi o sursă nouă de dependențe funcționale pentru care constituim o relație distinctă. Iată noua structură:

```
CREATE TABLE formaţii (
   idformaţie NUMBER(6) PRIMARY KEY,
   ciclu_studii CHAR(1),
   forma_de_studii CHAR(1)
   an_studii NUMBER(1)
   modul_specializare VARCHAR2(30)
   );
CREATE TABLE stud (
   matricol VARCHAR2(18) PRIMARY KEY,
   numepren VARCHAR2(50),
   cnp CHAR(13),
   idformaţie NUMBER(6) REFERENCES formaţii(idformaţie),
   serie_curs NUMBER(1),
   grupa NUMBER(2)
   );
CREATE TABLE discipl (
   coddisc CHAR(6) NOT NULL PRIMARY KEY,
   dendisc VARCHAR2(60) NOT NULL,
   nrcreddisc NUMBER(2)
   );
CREATE TABLE curricula (
   idformaţie NUMBER(6) REFERENCES formaţii(idformaţie),
   coddisc CHAR(6) REFERENCES discipline (coddisc),
   categorie_disc CHAR(1)
   nr_ore_curs NUMBER(3),
   nr_ore_seminar NUMBER(3),
   nr_ore_laborator NUMBER(3),
   PRIMARY KEY (idformaţie, coddisc)
   );
CREATE TABLE examene (
   idformaţie NUMBER(6) REFERENCES formaţii(idformaţie),
   coddisc CHAR(6) REFERENCES discipline (coddisc),
   dataex DATE,
   salaex VARCHAR2(30),
   PRIMARY KEY (idformaţie, coddisc,dataex)
   );
```

Joncţiunile s-au simplificat cu preţul apariţiei unei relaţii noi şi a unei serii de restricţii referenţiale ale căror gestionare reclamă timp şi resurse. Rămâne ca dvs. să decideţi dacă merită sau nu efortul...

9.3. Copii „exacte" ale unui atribut în mai multe tabele

Plasarea unui atribut necheie într-o tabelă copil este cazul cel mai frecvent invocat şi, în egală măsură, cel mai puţin recomandat în denormalizare. Să luăm cazul bazei de date pentru evidenţierea notelor despre studenţi, discutată ultima dată în paragraful 7.2.3 (listingul 7.17). Probabil că informaţiile cele mai solicitate din bază se referă la notele obţinute de studenţi:

```
SELECT numepren, studenti.matricol, dendisc, dataex, nota
FROM note INNER JOIN studenti
ON note.matricol=studenti.matricol
 INNER JOIN discipline ON note.coddisc=discipline.coddisc
```

Dacă ne gândim că un asemenea gen de interogare o să se execute de sute de mii sau chiar milioane de ori pe an (la obţinerea a sute de rapoarte, dar şi consultări lansate de studenţi/părinţi folosind o aplicaţie web conectată la baza de date), ne-ar putea "încolţi" ideea introducerii atributelor `numepren` şi `dendisc` şi în tabela NOTE:

```
ALTER TABLE note ADD (numepren VARCHAR2(50)) ;
ALTER TABLE note ADD (dendisc VARCHAR2(60));
```

Aducerea la zi a valorilor celor două atribute presupune două comenzi UPDATE destul de simple:

```
UPDATE note SET numepren =
(SELECT numepren FROM studenti
 WHERE matricol=note.matricol)
şi
UPDATE note SET dendisc =
(SELECT dendisc FROM discipline
     WHERE coddisc=note.coddisc).
```

Avantajul ar fi că, din moment ce orice notă este furnizată dintr-o singură tabelă, fără a fi nevoie de joncţiuni:

```
SELECT numepren, matricol, dendisc, dataex, nota
FROM note
```

viteza este incomparabil mai mare. Însă nici dezavantajele nu sunt de neglijat. Unul este imensul spaţiu risipit. Dacă luăm în calcul faptul că tabela NOTE poate să numere, la un moment dat, milioane de înregistrări, introducerea celor două atribute costă un spaţiu deloc de neglijat. Cel mai deranjant este însă riscul incon-

sistenţelor care pot să apară între valorile NumePren şi DenDisc din NOTE şi cele din tabelele de "origine", STUDENŢI şi DISCIPLINE. De aceea, orice actualizarea a celor două atribute în tabelele de origine trebuie să se propage neapărat în NOTE. Lucru posibil doar prin intermediul declanşatoarelor.

Un alt exemplu tentant ar fi cazul practic tratat în paragraful 7.4. Astfel, în tabela DISTRIBUŢIE {IdFilm, Rol, Actor} s-ar economisi timp la obţinerea a o serie de informaţii dacă acestor atribute li s-ar adăuga atributele TitluOriginal şi TitluRo din FILME.

Fireşte, atributele duplicate pot fi preluate nu numai din tabelele părinte, ci şi din "bunici", "străbunici" etc. De exemplu, tabela LINII_FACT din baza de date VÎNZĂRI (vezi paragraful 5.2) este cea mai des folosită, atât la reconstituirea valorii facturilor (în lipsa unor atribute calculate FACTURI.ValFărăTVA, FACTURI.ValTotală). Pentru analiza multidimensională a datelor legate de produsele vândute, putem introduce atât atribute precum data facturii (FACTURI), numele produsului şi cota de TVA ale produsului (PRODUSE), precum şi denumirea clientului (CLIENŢI), localitatea (CODURI_POŞTALE) şi judeţul (JUDEŢE) în care clientul îşi are sediul: LINII_FACT {NrFact, DataFact, Linie, CodPr, DenPr, ProcTVA, Localitate, Judeţ}. La urma urmei, chiar ne apropiem de OLAP şi depozite de date - o tehnologie care a potenţat denormalizarea.

În această privinţă, mă raliez punctului de vedere potrivit căruia atributele duplicat trebuie evitate aproape întotdeauna ! Dacă o aplicaţie merge foarte lent la interogări, trebuie găsite alte explicaţii pentru proasta funcţionare sau, la o adică, trebuie schimbat SGBD-ul ! Ideea (schimbării SGBD-ului) nu este atât de neroadă cum pare la prima vedere, astăzi piaţa fiind foarte generoasă la orice categorie de SGBD-uri, inclusiv cele de tip *open-source*.

9.4. Atribute derivate/sintetice

Deşi induc un grad de redundanţă semnificativ, atributele derivate sunt privite cu mult mai mare îngăduinţă. Ba chiar încadrarea lor în tehnicile denormalizării este discutabilă. Pentru prima dată am recurs la atribute derivate în paragraful 7.2.2 în vederea simplificării implementării unor restricţii la nivelul bazei de date. Astfel, pentru baza de date BIBLIOTECĂ, întrucât un titlu (carte) nu poate fi cumpărat în mai mult de 10 exemplare, am convenit că o soluţie elegantă ar fi ca, în tabela TITLURI, să introducem atributul NrExemplare, atribut actualizabil exclusiv prin declanşatoarele tabelei EXEMPLARE, restricţia fiind implementabilă printr-o clauză CHECK declarată noului atribut. Iar în cazul bazei de date CONTABILITATE, numărul atributelor derivate introduse în tabela OPERAŢIUNI a fost de 4, utilitatea lor fiind evidentă, ca şi costurile actualizării şi păstrării consistenţei lor. Cele mai importante rezerve cu privire la acest tip de atribute sunt similare celor pomenite la atributele duplicat: volum suplimentar necesar stocării şi complexitatea mecanismului de menţinere a corectitudinii valorilor acestora.

Dacă unele atribute suplimentare calculate sunt necesare pentru declararea restricţiilor, alteori câştigul urmărit nu ţine decât de creşterea vitezei de acces la o

serie de informaţii din bază. Poate cel mai nimerit caz este, în acest sens, cel al bazei de date VÎNZĂRI, adusă în 3NF în finalul paragrafului 5.2 sub forma a opt relaţii: JUDEŢE {Jud, Judeţ, Regiune}, CODURI_POŞTALE {CodPost, Localitate, Comună, Jud}, PRODUSE {CodPr, DenPr, UM, ProcTVA, Grupa}, CLIENŢI2[25] {CodCl, DenCl, CodFiscal, StradaCl, NrStradaCl, BlocScApCl, TelefonCl, CodPost}, FACT {NrFact, CodCl, DataFact, Obs}, LINII_FACT {NrFact, Linie, CodPr, Cantitate, PreţUnit}, ÎNCASĂRI {CodÎnc, DataÎnc, CodDoc, NrDoc, DataDoc} şi ÎNCAS_FACT {CodÎnc, NrFact, Tranşa}. Scriptul de creare a acestor tabele poate fi descărcat sub forma listingului 9.1.

Schema de stocare este una acceptabilă, atâta vreme cât nu luăm în calcul aspectele temporale (vezi capitolul următor). Printre cele mai frecvent căutate informaţii din bază sunt şi:

- valoare unei facturi, fără şi cu TVA;
- valoarea încasată dintr-o factură;
- valoarea rămasă de încasat dintr-o factură (diferenţa dintre valoarea cu TVA a facturii şi valoarea deja încasată).

Determinarea oricăreia dintre cele trei informaţii presupune o serie de joncţiuni, după cum urmează:

```
SELECT SUM(cantitate * pretunit) AS ValoareFaraTVA,
 SUM(cantitate * pretunit * (1 * procTVA)) AS ValTotala,
 SUM(NVL(transa,0)) AS ValIncasata,
 SUM(NVL(transa,0)) - SUM(cantitate * pretunit *
 (1 * procTVA)) AS ValRamasaDeÎncasat
FROM fact f
 INNER JOIN linii_fact lf ON f.nrfact=lf.nrfact
 INNER JOIN produse p ON lf.codpr=p.codpr
 LEFT OUTER JOIN incas_fact if ON f.nrfact=if.nrfact
WHERE f.nrfact = 7545576
```

Fiind un formular cu regim special, din momentul confirmării, o factură nu poate suferi modificări. La fel stau lucrurile şi cu ordinele de plată, extrasele de cont şi alte documente ce atestă încasarea facturilor. Prin urmare, în timp, declanşatoarele de modificare şi de ştergere din FACT, LINII_FACT şi INCAS_FACT vor fi extrem de rar executate.

Pe acest considerent, ne putem gândi să introducem în tabela FACT trei atribute, ValTotală, TVAFact şi ValÎncasată. Toate trei vor fi calculate automat prin declanşatoarele de inserare, modificare şi ştergere ale tabelelor LINII_FACT şi ÎNCAS_FACT:

```
ALTER TABLE fact ADD ValTotală NUMBER (15) ;
ALTER TABLE fact ADD TVAFact NUMBER (15) ;
ALTER TABLE fact ADD ValÎncasată NUMBER (15) ;
```

[25] Denumim tabela CLIENŢI2, şi nu CLIENŢI, pentru cei care descarcă şi testează listingurile cărţii, astfel încât, la execuţia în aceeaşi schemă, ar apărea o suprapunere cu tabela CLIENŢI de la cazul practic dedicat centrului de închirieri video din capitolul 7.

Cele două declanşatoare de inserare ar trebui să funcţioneze după următoarea logică prezentată în listingul 9.2:

Listing 9.2. Declanşatoarele de inserare în LINII_FACT şi INCAS_FACT

```
CREATE OR REPLACE TRIGGER trg_lf_ins
    AFTER INSERT ON linii_fact FOR EACH ROW
BEGIN
    UPDATE fact SET
        valtotala = NVL(valtotala,0) + :NEW.cantitate *:NEW.pretunit * (1 +
            (SELECT procTVA FROM produse WHERE codpr=:NEW.codpr)),
        TVAfact = NVL(TVAfact,0) + :NEW.cantitate * :NEW.pretunit *
            (SELECT procTVA FROM produse WHERE codpr=:NEW.codpr)
        WHERE nrfact=:NEW.nrfact ;
END ;
/
CREATE OR REPLACE TRIGGER trg_if_ins
    AFTER INSERT ON incas_fact FOR EACH ROW
BEGIN
    UPDATE fact
    SET ValIncasata = NVL(ValIncasata,0) + :NEW.transa
    WHERE nrfact=:NEW.nrfact ;
END ;
/
```

Declanşatoarele de modificare şi ştergere reclamă efort doar de scriere şi implementare, întrucât execuţia lor de către SGBD este foarte rară. Importanţa lor este însă esenţială în menţinerea corectitudinii valorilor pentru atributele calculate/sintetice - vezi listing 9.3.

Listing 9.3. Declanşatoarele de modificare şi ştergere ale tabelelor LINII_FACT şi INCAS_FACT

```
-- actualizare in LINII_FACT
CREATE OR REPLACE TRIGGER trg_lf_upd
    AFTER UPDATE ON linii_fact FOR EACH ROW
BEGIN
    IF :NEW.nrfact <> :OLD.nrfact THEN -- trebuie scazut de la vecheia factura
        UPDATE fact SET                -- si adaugat la noua
            valtotala = valtotala - :OLD.cantitate *:OLD.pretunit * (1 +
                (SELECT procTVA FROM produse WHERE codpr=:OLD.codpr)),
            TVAfact = TVAfact - :OLD.cantitate * :OLD.pretunit *
                (SELECT procTVA FROM produse WHERE codpr=:OLD.codpr)
        WHERE nrfact=:NEW.nrfact ;

        UPDATE fact SET
            valtotala = valtotala + :NEW.cantitate *:NEW.pretunit * (1 +
                (SELECT procTVA FROM produse WHERE codpr=:NEW.codpr)),
            TVAfact = TVAfact + :NEW.cantitate * :NEW.pretunit *
                (SELECT procTVA FROM produse WHERE codpr=:NEW.codpr)
        WHERE nrfact=:NEW.nrfact ;

    ELSE  -- nu s-a schimbat numarul facturii

        UPDATE fact SET
            valtotala = valtotala - :OLD.cantitate *:OLD.pretunit * (1 +
```

```
                    (SELECT procTVA FROM produse WHERE codpr=:OLD.codpr)) +
                    :NEW.cantitate *:NEW.pretunit * (1 +
                    (SELECT procTVA FROM produse WHERE codpr=:NEW.codpr)),
                TVAfact = TVAfact - :OLD.cantitate * :OLD.pretunit *
                    (SELECT procTVA FROM produse WHERE codpr=:OLD.codpr)
        WHERE nrfact=:NEW.nrfact ;
    END IF ;
END ;
/

-- stergere in LINII_FACT
CREATE OR REPLACE TRIGGER trg_lf_del
    BEFORE DELETE ON linii_fact FOR EACH ROW
BEGIN
    UPDATE fact SET
        valtotala = valtotala - :OLD.cantitate *:OLD.pretunit * (1 +
            (SELECT procTVA FROM produse WHERE codpr=:OLD.codpr)),
        TVAfact = TVAfact - :OLD.cantitate * :OLD.pretunit *
            (SELECT procTVA FROM produse WHERE codpr=:OLD.codpr)
    WHERE nrfact=:OLD.nrfact ;
END ;
/

-- actualizare in INCAS_FACT
CREATE OR REPLACE TRIGGER trg_if_upd
     AFTER UPDATE ON incas_fact FOR EACH ROW
BEGIN
    IF :NEW.nrfact <> :OLD.nrfact THEN -- trebuie scazut de la vecheia factura
        UPDATE fact SET                -- si adaugat la noua
            ValIncasata = ValIncasata - :OLD.transa
        WHERE nrfact=:OLD.nrfact ;

        UPDATE fact SET ValIncasata = ValIncasata + :NEW.transa
        WHERE nrfact=:NEW.nrfact ;

    ELSE
        UPDATE fact SET ValIncasata = ValIncasata - :OLD.transa - :NEW.transa
        WHERE nrfact=:NEW.nrfact ;
    END IF;
END ;
/

-- stergere in INCAS_FACT
CREATE OR REPLACE TRIGGER trg_if_del
    BEFORE DELETE ON Incas_fact FOR EACH ROW
BEGIN
    UPDATE fact
    SET ValIncasata = ValIncasata - :OLD.transa
    WHERE nrfact=:OLD.nrfact ;
END ;
/
```

Atenţie, însă ! Actualizarea atributelor sintetice pe baza declanşatoarelor este necesară, dar nu şi suficientă în asigurarea corectitudinii lor. Ce s-ar întâmpla dacă, după câtva timp de funcţionare a aplicaţiei, cineva (nu spunem cine !) ar „scăpa”o comandă de genul:

```
UPDATE fact
SET ValTotală = 555555 ;
```

Toate liniile tabelei FACT ar fi afectate, iar valoarea totală a fiecărei facturi ar fi cu totul aiurea. Oficial, pentru cei neiniţiaţi, s-ar putea lansa ideea atacului unui virus, dar soluţia aceasta are o viaţă scurtă. Trebuie găsit un mecanism prin care atributele calculate să fie modificate numai prin declaşatoare. În Oracle, o soluţie nu prea complicată ţine de folosirea unor variabile publice care să semnalizeze că modificarea se face din declanşatorul autorizat. Iată, în listingul 9.4, o asemenea variantă de lucru.

Listing 9.4. Pachetul, noile declanşatoare ale tabelelor LINII_FACT şi INCAS_FACT, plus declanşatoarele tabelei FACT pentru controlul actualizării atributelor sintetice

```
CREATE OR REPLACE PACKAGE pac_vinzari AS
    v_trg_liniifact BOOLEAN := FALSE ;
    v_trg_incas_fact BOOLEAN := FALSE ;
END pac_vinzari ;
/
-------------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_lf_ins
    AFTER INSERT ON linii_fact FOR EACH ROW
BEGIN
    pac_vinzari.v_trg_liniifact := TRUE ;
    UPDATE .... – comanda este identică celei din listing 9.3
    pac_vinzari.v_trg_liniifact := FALSE ;
END ;
/
-------------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_if_ins
     AFTER INSERT ON incas_fact FOR EACH ROW
BEGIN
    pac_vinzari.v_trg_incas_fact := TRUE ;
    UPDATE .... – comanda este identică celei din listing 9.3
    pac_vinzari.v_trg_incas_fact := FALSE ;
END ;
/
-------------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_lf_upd
    AFTER UPDATE ON linii_fact FOR EACH ROW
BEGIN
    IF :NEW.nrfact <> :OLD.nrfact THEN -- trebuie scazut de la vecheia factura
        pac_vinzari.v_trg_liniifact := TRUE ;
        UPDATE .... – comanda este identică celei din listing 9.3
        UPDATE .... – comanda este identică celei din listing 9.3
        pac_vinzari.v_trg_liniifact := FALSE ;
    ELSE  -- nu s-a schimbat numarul facturii
        pac_vinzari.v_trg_liniifact := TRUE ;
        UPDATE .... – comanda este identică celei din listing 9.3
        pac_vinzari.v_trg_liniifact := FALSE ;
    END IF ;
END ;
/
-------------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_lf_del
    BEFORE DELETE ON linii_fact FOR EACH ROW
```

```
BEGIN
    pac_vinzari.v_trg_liniifact := TRUE ;
    UPDATE .... – comanda este identică celei din listing 9.3
    pac_vinzari.v_trg_liniifact := FALSE ;
END ;
/
-------------------------------------------------------------------------------
-- actualizare in INCAS_FACT
CREATE OR REPLACE TRIGGER trg_if_upd
    AFTER UPDATE ON Incas_fact FOR EACH ROW
BEGIN
    IF :NEW.nrfact <> :OLD.nrfact THEN -- trebuie scazut de la vecheia factura
        pac_vinzari.v_trg_incas_fact := TRUE ;
        UPDATE .... – comanda este identică celei din listing 9.3
        pac_vinzari.v_trg_incas_fact := FALSE ;
    ELSE
        pac_vinzari.v_trg_incas_fact := TRUE ;
        UPDATE .... – comanda este identică celei din listing 9.3
        pac_vinzari.v_trg_incas_fact := FALSE ;
    END IF;
END ;
/
-------------------------------------------------------------------------------
-- stergere in INCAS_FACT
CREATE OR REPLACE TRIGGER trg_if_del
    BEFORE DELETE ON Incas_fact FOR EACH ROW
BEGIN
    pac_vinzari.v_trg_incas_fact := TRUE ;
    UPDATE .... – comanda este identică celei din listing 9.3
    pac_vinzari.v_trg_incas_fact := FALSE ;
END ;
/
-------------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_fact_ins
    BEFORE INSERT ON fact FOR EACH ROW
BEGIN
    :NEW.ValTotala := 0 ;
    :NEW.TVAFact  := 0 ;
    :NEW.ValIncasata := 0 ;
END ;
/
-------------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_fact_upd2
    AFTER UPDATE OF ValTotala, TVAFact, ValIncasata
    ON fact FOR EACH ROW
BEGIN
    IF :NEW.ValTotala <> :OLD.ValTotala OR :NEW.TVAFact <> :OLD.TVAFact THEN
        IF pac_vinzari.v_trg_liniifact = FALSE THEN
            RAISE_APPLICATION_ERROR(-20876,
            'ValTotala si TVAFact nu pot fi modificate interactiv !');
        END IF ;
    END IF ;

    IF :NEW.ValIncasata <> :OLD.ValIncasata THEN
        IF pac_vinzari.v_trg_incas_fact = FALSE THEN
            RAISE_APPLICATION_ERROR(-20876,
            'ValIncasata nu poate fi modificata interactiv !');
        END IF ;
```

```
    END IF ;
END ;
/
```

Pentru un plus de lizibilitate, elementele de noutate față de listingul 9.3 au fost puse în evidență prin îngroşare. Listingul descărcabil pe web este, însă, complet.

Urmând această idee, ne mai putem gândi la câteva atribute consultate în mod frecvent, ce ar putea fi introduse în schemă şi calculate prin declanşatoare, în condițiile în care calculul lor ar implica joncțiuni:

în tabela PRODUSE, s-ar putea introduce atributul `Vânzări`, incrementabil la fiecare prezență a produsului respectiv pe linia vreunei facturi; actualizarea sa este operată din cele trei declanşatoare ale LINII_FACT;

în tabela ÎNCASĂRI, atributul `ValoareÎncasare`, actualizabil prin declanşatoarele tabelei INCAS_FACT;

în tabela CLIENȚI:

`Vânzări` ar cumula valorile facturilor emise destinate clientului respectiv, fiind actualizabil prin declanşatoarele tabelei FACT (în condițiile folosirii `ValTotală`, `TVAFact`);

`Plăţi` ar însuma toate încasările de la client, corectitudinea valorii sale fiind asigurată prin intermediul celor trei declanşatoare ale tabelei FACT, de data aceasta atributul împricinat fiind `ValÎncasată`.

Interesant este că o serie de asemenea atribute ar putea fi utile şi pentru implementarea unor reguli mai sofisticate, precum cele discutate în paragraful 7.2.2. Astfel, firma pentru care este proiectată schema bazei de date VÎNZĂRI decide să nu mai vândă nimic niciunui client care are datorii mai mari de 200 de milioane de lei, sau clienților la care datoriile sunt mai mari de 30% de valoarea vânzărilor. Cu aportul ultimelor două atribute introduse în CLIENȚI, această restricție devine o banalitate:

```
ALTER TABLE clienţi ADD CHECK
(vânzări - plăţi <= 200000000) ) ;
```

în cazul primei formulări, sau:

```
ALTER TABLE clienţi ADD CHECK
((vânzări - plăţi) / vânzări <= .3 ) ;
```

pentru a o onora pe cea de-a doua.

În concluzie, deşi nu suntem forțați să o facem, am putea delimita două categorii de atribute redundante derivate/sintetice: cea din paragraful 7.2.2, care grupează atributele necesare implementării unor restricții mai complexe, şi categoria atributelor utile în obținerea unor informații (punctuale sau rapoarte în toată regula) din bază cu un consum de resurse cât mai mic sau, în fine, cât mai judicios.

Gradul lor de utilitate trebuie raportat la costurile integrității şi coerenței valorilor lor, uneori meritând efortul, iar alteori nu. Un caz simplu pe care-l putem încadra în categoria celor care merită efortul este cel al tabelei LINII_FACT, în care introducem un atribut de genul `ValLinieCuTVA`, adică valoarea cu TVA a produsului de pe linia curentă a facturii, calculată prin expresia: `LINII_FACT.Cantitate * LINIIFACT.PreţUnit * (1 + PRODUSE.ProcTVA)`. Argumentul principal ține de regimul facturilor, care, după momentul introducerii, nu mai pot fi modificate, în schimb, numărul consultărilor ulterioare în care sunt implicate înregistrările respective este imens.

Vom vedea în capitolul următor că, prin ricoşeu, acest atribut redundant ne face un mare serviciu în materie de valabilitate în timp a schemei bazei, în condițiile în care, la câțiva ani, procentul de TVA al unui produs se poate modifica. Printre alte atribute relativ utile se numără cele trei propuse anterior în acest paragraf, `FACT.ValTotală`, `FACT.TVAFact`, `FACT.ValÎncasată`, `CLIENȚI.Vânzări`, `CLIENȚI.Plăți`.

Chiar dacă utile, majoritate acestor atribute au o doză a artificial şi sunt tratate, uneori, cu nemeritat dispreț. Am vrea să vă propunem şi o speță de redundață care este, pe cât de exagerată, pe atât de utilă. După repetate încercări, baza de date CONTABILITATE a fost adusă la o schemă rezonabilă, chiar dacă cu destule atribute redundante. Graful final al dependențelor este cel din figura 7.10. Ei bine, pentru a obține şi cele mai simple documente contabile (nu ştim care sunt acelea), ne lipseşte o informație esențială: soldul inițial a conturilor. Mulțumim pe această cale celor care se pricep la contabilitate şi au păstrat discreția până în acest punct al cărții. Trebuie degrabă introdus, aşadar, atributul `SoldIniţial` care depinde funcțional de atributul `ContElementar`. Necazul e că soldul inițial al unui cont poate fi debitor sau creditor.

Pentru conturile de activ, soldurile inițiale sunt debitoare, iar pentru cele de pasiv soldurile inițiale sunt creditoare. În schimb, pentru o serie de conturi bifuncționale soldul inițial poate fi debitor, în timp ce alte conturi bifuncționale prezintă sold inițial creditor. Drept care, deşi pare o crasă redundanță, introducem nu un atribut pentru soldul inițial, ci două: sold inițial debitor (`SoldInitBD`) şi sold inițial creditor (`SoldInitCR`). Ba, mai mult, pentru a avea în orice moment "la îndemână" soldul curent al contului, vom recurge la două atribute denumite pompos calculate/sintetizatoare/centralizatoare: rulaj debitor (`RulajDB`) şi rulaj creditor (`RulajCR`). Forma uşor schimbată a grafului de dependențe funcționale/de incluziune este cea din figura 9.1.

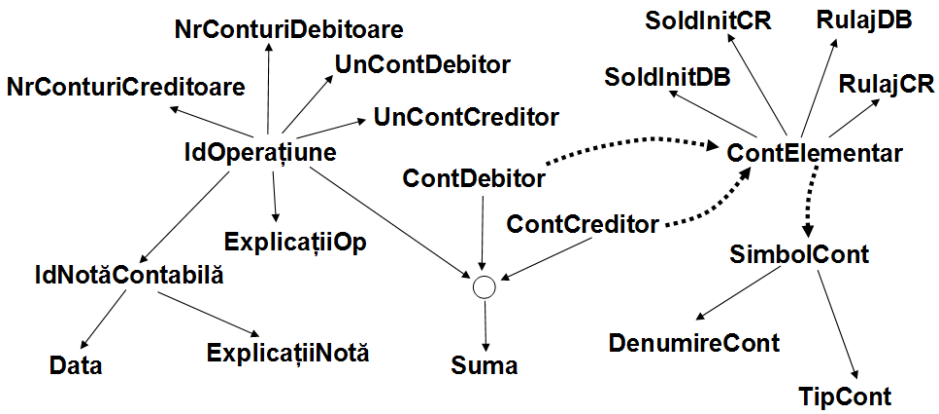

Figura 9.1. Noi atribute, care mai de care mai redundante

Noile atribute afectează doar tabela CONTURI_ELEMENTARE care, mai nou, are structura: CONTURI_ELEMENTARE {`ContElementar`, `SoldInitDB`, `SoldInitCR`, `RulajDB`, `RulajCR`}.

```
ALTER TABLE conturi_elementare ADD SoldInitDB NUMBER(14) ;
ALTER TABLE conturi_elementare ADD SoldInitCR NUMBER(14) ;
ALTER TABLE conturi_elementare ADD RulajDB NUMBER(14) ;
ALTER TABLE conturi_elementare ADD RulajCR NUMBER(14) ;
```

Un cont, oricare ar fi natura sa, nu poate avea simultan sold inițial şi pe debit şi pe credit, aşa încât avem nevoie de o restricție de genul:

```
ALTER TABLE conturi_elementare
ADD CHECK (NVL(SoldInitDB,0) * NVL(SoldInitCR,0) = 0) ;
```

Un alt exemplu în care folosirea unui atribut derivat/calculat este crucială priveşte o aplicație prin care o bancă gestionează conturile de card folosite la bancomate sau pentru plăți în supermagazine. Contul din care se pot face plăți sau retrage numerar cu ajutorul cardului este identificat prin `IdCont`, celelalte informații despre cont fiind: numărul (`NrCont`), tipul (`TipCont`), data la care a fost deschis (`DataDeschiderii`), soldul la deschidere (`SoldIniţial`) şi titularul (`CNPPosesor`). Odată deschis un asemena cont, există două mari categorii de operațiuni (`IdOperaţiune`, `DataOraOp`) cu acesta: *alimentări* (virarea unor sume în acest cont, operațiune care se numeşte, în termeni tehnici, de alimentare a contului - `IdAlimCont`, `SumaAlimCont`) şi *plăți* (`IdPlată`, `TipPlată`, `SumăPlată`, `ProcentComision`). În funcție de tipul plății, banca percepe un comision exprimat ca procent aplicat la suma plătită.

Categoria *plăți* cuprinde, la rândul său, trei tipuri de operațiuni:

Retragere de numerar de la bancomat (prilej, uneori, de cozi similare celor de la alimentarele de acum mai bine de 15 ani). Fiecare retragere de acest tip are un identificator (`IdRetrNumerar`), iar a doua informație vitală pentru consemnarea

sa este de la ce bancomat s-a făcut retragerea numerarului, fiecare bancomat fiind caracterizat printr-un identificator (`IdBancomat`), un număr "oficial" (`NrBcmt`) ce i-a fost alocat de banca de care aparține (`Bancă`), şi adresa la care este situat (`AdresaBcmt`).

Plăți ale unor facturi direct de la bancomat (ex. facturile Orange sau Connex), identificate printr-un atribut special (`IdPlatăBancomat`), plăți care, spre deosebire de retragerile de numerar, presupun virarea banilor într-un cont destinație (`IdContDest`) ce aparține beneficiarului plății. Despre contul-destinație, în afara identificatorului, mai interesează: numărul său (`NrContDest`) şi mai recentul identificator internațional IBAN (`IBAN_ContDest`), precum şi banca la care a fost deschis.

Plăți efectuate de la puncte de vânzare (*Point of Sale*) amplasare frecvent în supermagazine, stații de benzină etc. prevăzute cu un dispozitiv special în care se introduce cardul, se tastează codul-pin şi se face transferul banilor. Acest tip de plată este identificat, firește, printr-un atribut special (`IdPlPunctVînz`), iar elementele care interesează în mod deosebit privesc id-ul punctului de vânzare (`IdPunctVînzare`), denumirea (`DenPV`) şi adresa sa (`AdresaPV`), precum şi contul în care are loc virarea banilor (`IdContDest`).

După sistematizare, graful dependențelor funcționale poate arăta precum cel din figura 9.2.

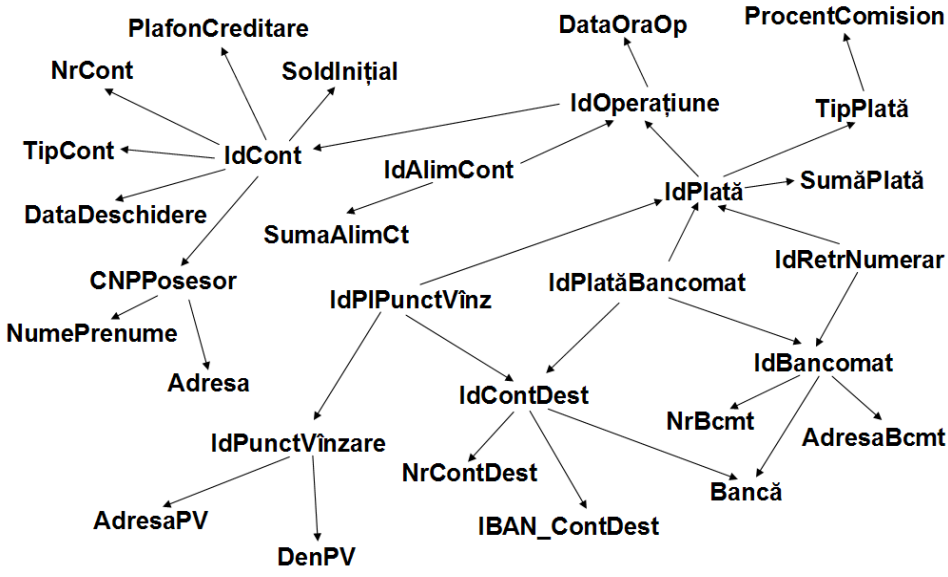

Figura 9.2. Graful DF pentru baza de date BANCOMATE

Relațiile decupate pe baza acestui graf, precum şi câteva dintre restricțiile ce pot fi instituite, sunt prezentate în scriptul Oracle din listingul 9.5.

Listing 9.5. Scriptul de creare a tabelelor bazei de date BANCOMATE

```
DROP TABLE plati_puncte_vinzare ;
```

```
DROP TABLE puncte_vinzare ;
DROP TABLE plati_prin_bancomat ;
DROP TABLE conturi_destinatie ;
DROP TABLE retrageri_numerar ;
DROP TABLE bancomate ;
DROP TABLE plati ;
DROP TABLE tipuri_plati  ;
DROP TABLE alimentare_conturi ;
DROP TABLE operatiuni_card ;
DROP TABLE conturi_card ;
DROP TABLE titulari_card ;

CREATE TABLE titulari_card (
    CNPPosesor CHAR(13) NOT NULL PRIMARY KEY,
    NumePrenume VARCHAR2(45) NOT NULL,
    Adresa VARCHAR2(60)
);

CREATE TABLE conturi_card (
    IdCont NUMBER(14) NOT NULL PRIMARY KEY,
    CNPPosesor CHAR(13) NOT NULL REFERENCES titulari_card (CNPPosesor),
    DateDeschidere DATE DEFAULT CURRENT_DATE NOT NULL,
    TipCont VARCHAR2(15) NOT NULL,
    NrCont NUMBER(16) NOT NULL,
    PlafonCreditare NUMBER(14) NOT NULL,
    SoldInitial NUMBER(14) NOT NULL
);

CREATE TABLE operatiuni_card (
    IdOperatiune NUMBER(16) NOT NULL PRIMARY KEY,
    DataOraOp DATE DEFAULT CURRENT_DATE NOT NULL,
    IdCont NUMBER(14) NOT NULL REFERENCES conturi_card (IdCont)
);

CREATE TABLE alimentare_conturi (
    IdAlimCont NUMBER(15) NOT NULL PRIMARY KEY,
    SumaAlimCont NUMBER (14) NOT NULL CHECK (SumaAlimCont > 0),
    IdOperatiune NUMBER(16) NOT NULL REFERENCES operatiuni_card (IdOperatiune)
);

CREATE TABLE tipuri_plati (
    TipPlata VARCHAR2(15) NOT NULL PRIMARY KEY,
    ProcentComision NUMBER(4,4)
);

CREATE TABLE plati (
    IdPlata NUMBER(15) NOT NULL PRIMARY KEY,
    TipPlata VARCHAR2(15) NOT NULL REFERENCES tipuri_plati (TipPlata),
    SumaPlata NUMBER(14),
    IdOperatiune NUMBER(20) NOT NULL REFERENCES operatiuni_card (IdOperatiune)
);

CREATE TABLE bancomate (
    IdBancomat NUMBER(7) NOT NULL PRIMARY KEY,
    Banca VARCHAR2(30) NOT NULL,
    NrBcmt NUMBER(5) NOT NULL,
    AdresaBcmt VARCHAR2(50)
);
```

```
CREATE TABLE retrageri_numerar (
    IdRetrNumerar NUMBER(15) NOT NULL PRIMARY KEY,
    IdPlata NUMBER(15) NOT NULL REFERENCES plati (IdPlata),
    IdBancomat NUMBER(7) NOT NULL REFERENCES bancomate (IdBancomat)
);

CREATE TABLE conturi_destinatie (
    IdContDest NUMBER(10) NOT NULL PRIMARY KEY,
    NrContDest NUMBER(20) NOT NULL,
    Banca VARCHAR2(30) NOT NULL,
    IBAN_ContDest VARCHAR2(20) NOT NULL
);

CREATE TABLE plati_prin_bancomat (
    IdPlataBancomat NUMBER(15) NOT NULL PRIMARY KEY,
    IdPlata NUMBER(15) NOT NULL REFERENCES plati (IdPlata),
    IdBancomat NUMBER(7) NOT NULL REFERENCES bancomate (IdBancomat),
    IdContDest NUMBER(10) NOT NULL REFERENCES conturi_destinatie (IdContDest)
);

CREATE TABLE puncte_vinzare (
    IdPunctVinzare NUMBER(10) NOT NULL PRIMARY KEY,
    DenPV VARCHAR2(30),
    AdresaPV VARCHAR2(50)
 );

CREATE TABLE plati_puncte_vinzare (
    IdPlPunctVinz NUMBER(14) NOT NULL PRIMARY KEY,
    IdPlata NUMBER(15) NOT NULL REFERENCES plati (IdPlata),
    IdPunctVinz NUMBER(10) NOT NULL REFERENCES puncte_vinzare (IdPunctVinzare),
    IdContDest NUMBER(10) NOT NULL REFERENCES conturi_destinatie (IdContDest)
);
```

Ei bine, una dintre cele mai importante reguli ale aplicației este: în urma alimentărilor şi plăților, un cont nu poate avea un deficit mai mare decât plafonul de creditare. Spre exemplu, un client deschide un cont pentru card în care depune 10 milioane de lei (`SoldInițial`) şi stabileşte cu banca un plafon de creditare de 5 milioane de lei. După o primă retragere de numerar de 7 milioane lei, soldul contului rămâne de 3 milioane. Totuși, după o alimentare cu două milioane, alimentare în urma careia soldul ajunge la 5 milioane, clientul poate plăti/retrage din cont 10 miloane lei. Soldul său este de minus 5 milioane lei, iar din acest moment, până la o nouă alimentare a contului, nu mai poate face nici o plată/retragere.

La fiecare plată, aplicația trebuie, deci, neapărat să verifice dacă suma ce ar urma a fi plătită nu depăşeşte soldul curent plus plafonul de creditare. Or, verificarea presupune următorul calcul: soldul inițial + toate alimentările - toate plățile + plafonul de creditare > suma ce se vrea a fi plătită. Dacă banca are sute de mii de asemenea conturi, în câțiva ani numărul operațiunilor se ridică la milioane sau chiar zeci-sute de miloane, iar verificarea de mai sus devine extrem de consumatoare de resurse. Situația impune recurgerea la un nou atribut, `SoldCurent`, dependent funcțional de `IdCont`, care va fi adăugat tabelei CONTURI_CARD:

```
ALTER TABLE conturi_card ADD (SoldCurent NUMBER(14)) ;
```

Gestionarea sa se va face similar similar atributelor sintetice din schema bazei de date VÎNZĂRI, prin mecanismul în doi timpi, actualizarea pe baza declanşatoarelor altor tabele, plus protejarea la modificările interactive folosind declanşatorul de modificare al tabelei CONTURI_CARD.

Schimbând un pic registrul, facem apel la neuitatul capitol 7, mai precis la paragraful dedicat centrului de închirieri de casete video. În listingul 7.29 ne-am încordat forţele pentru a destiona automat, prin declanşatoare, valoarea atributului `FilmNr` din tabela CASETE_FILME. Acest atribut trebuia să se incrementeze automat la adăugarea unui film pe o casetă. Asemănarea cu atributul `Linie` din tabela LINII_FACT de la începutul acestui paragraf, sau cu atributul `OrdineCopertă` din tabela CĂRŢI_AUTORI (vezi figura 8.27 din paragraful 8.4), sau cu `LocalitNr` din LOC_RUTE (vezi figura 8.23 din paragraful 8.3.3), este absolut tulburătoare. Dificultatea acestor atribute ţine de faptul că valorile lor trebuie să fie ca un număr curent într-un tabel obişnuit, iar ştergerea unei linii din tabelele menţionate impune reaşezarea valorilor acestor numere curente.

Lucrurile s-ar putea simplifica în oarecare măsură dacă în tabelele părinte: CASETE, FACT, CĂRŢI şi RUTE am adăuga un atribut, fireşte, sintetic, care să conţină numărul liniilor din respectivele casete, facturi, cărţi şi rute. Să continuăm cu tandemul FACT – LINII_FACT din acest paragraf. Mai întâi, adăugăm atributul `NrLinii` în tabela FACT:

```
ALTER TABLE fact ADD (NrLinii NUMBER(2)) ;
```

Rescriem pachetul PAC_VÎNZĂRI, adăugând o funcţie care primeşte drept parametru numărul facturii şi returnează valoarea atributului `NrLinii`. Apoi, în declanşatorul de inserare pentru FACT, avem grijă ca valoarea lui `NrLinii` de pe linia nou introdusă să fie zero, iar în cel de modificare se verifică dacă eventuala modificare a acestui atribut provine din declanşatoarele tabelei LINII_FACT, semnalizarea fiind posibilă prin variabila publică V_TRG_LINIIFACT – vezi listing 9.6 – partea I.

Listing 9.6 (partea I). Pachetul, declanşatoarele actualizate de inserare şi modificare ale tabelei FACT

```
-- ==== Pachetul PAC_VINZARI
-----------------------------------------------
CREATE OR REPLACE PACKAGE pac_vinzari AS
    v_trg_liniifact BOOLEAN := FALSE ;
    v_trg_incas_fact BOOLEAN := FALSE ;
    FUNCTION f_nrlinii (nrfact_ fact.NrFact%TYPE)
        RETURN fact.NrLinii%TYPE ;
    v_nrfact fact.NrFact%TYPE ;
    v_liniestearsa linii_fact.Linie%TYPE ;
    v_trg_liniifact_del BOOLEAN := FALSE ;
END pac_vinzari ;
/
```

```
-----------------------------------------------
CREATE OR REPLACE PACKAGE BODY pac_vinzari AS
    FUNCTION f_nrlinii (nrfact_ fact.NrFact%TYPE)  RETURN fact.NrLinii%TYPE
IS
    v_nrlinii fact.NrLinii%TYPE ;
BEGIN
    SELECT NrLinii INTO v_nrlinii FROM fact WHERE NrFact = nrfact_ ;
    RETURN v_nrlinii ;
END f_nrlinii ;
END pac_vinzari ;
/

-- ==== Declansatoarele de inserare si modificare pentru FACT
-----------------------------------------------
CREATE OR REPLACE TRIGGER trg_fact_ins
    BEFORE INSERT ON fact FOR EACH ROW
BEGIN
    :NEW.ValTotala := 0 ;
    :NEW.TVAFact  := 0 ;
    :NEW.ValIncasata := 0 ;
    :NEW.NrLinii := 0 ;
END ;
/
-----------------------------------------------
CREATE OR REPLACE TRIGGER trg_fact_upd2
    AFTER UPDATE OF ValTotala, TVAFact,  ValIncasata, NrLinii
    ON fact FOR EACH ROW
BEGIN
    -- atribute modificate din LINII_FACT
    IF :NEW.ValTotala <> :OLD.ValTotala OR :NEW.TVAFact <> :OLD.TVAFact
        OR :NEW.NrLinii <> :OLD.NrLinii THEN
        IF pac_vinzari.v_trg_liniifact = FALSE THEN
            RAISE_APPLICATION_ERROR(-20876,
            'NrLinii, ValTotala si TVAFact nu pot fi modificate interactiv !');
        END IF ;
    END IF ;
    -- atribute modificate din INCAS_FACT
    IF :NEW.ValIncasata <> :OLD.ValIncasata THEN
        IF pac_vinzari.v_trg_incas_fact = FALSE THEN
            RAISE_APPLICATION_ERROR(-20876, 'ValIncasata nu poate fi modificata interactiv !');
        END IF ;
    END IF ;
END ;
/
```

De acum începe partea ceva mai interesantă. Păstrând declanşatorul de inserare la nivel de linie, de tip AFTER (TRG_LF_INS), se crează, tot pentru inserare, un altul - TRG_LF_INS0 – de tip BEFORE – ROW dedicat incrementării valorii atributului `NrLinii` din FACT – vezi listing 9.6 - partea a II-a. Mecanismul de gestionare a ştergerilor din LINII_FACT şi actualizarea corespunzătoare a atributelor sintetice din FACT este alcătuit din două declanşatoare de ştergere, unul la nivel de linie (BEFORE – ROW) şi altul la nivel de comandă (AFTER – STATEMENT), cu numele TRG_LF_DEL, respectiv TRG_LF_DEL2.

Listing 9.6 (partea a II-a). Un nou declanşator de inserare şi două de ştergere pentru tabela LINII_FACT

```
-- == Declansatoarele tabelei LINII_FACT
-----------------------------------------------
-- se pastreaza TRG_LF_INS

-----------------------------------------------
-- nou declansator de inserare (BEFORE-ROW)
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 ;
    pac_vinzari.v_trg_liniifact := FALSE ;
END ;
/
-----------------------------------------------
-- 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 procTVA FROM produse WHERE codpr=:OLD.codpr)),
        TVAfact = TVAfact - :OLD.cantitate * :OLD.pretunit *
            (SELECT procTVA FROM produse WHERE codpr=:OLD.codpr)
    WHERE nrfact=:OLD.nrfact ;
    pac_vinzari.v_trg_liniifact := FALSE ;
END ;
/
-----------------------------------------------
-- stergere in LINII_FACT - la nivel de COMANDA
CREATE OR REPLACE TRIGGER trg_lf_del2
    AFTER DELETE ON linii_fact
BEGIN
    IF pac_vinzari.v_nrfact IS NULL OR pac_vinzari.v_liniestearsa IS NULL THEN
        -- nu s-a sters nici o linie
        NULL ;
    ELSE
        pac_vinzari.v_trg_liniifact := TRUE ;
        IF pac_vinzari.f_nrlinii(pac_vinzari.v_nrfact) =
                    pac_vinzari.v_liniestearsa THEN
            -- linia stearsa din factura este ultima
            NULL ;
        ELSE
            -- liniei sterse i se da un numar mai mare pentru a nu se viola cheia primara
            pac_vinzari.v_trg_liniifact_del := TRUE ;
            UPDATE linii_fact
            SET linie =  pac_vinzari.f_nrlinii(pac_vinzari.v_nrfact) + 1
            WHERE NrFact = pac_vinzari.v_nrfact AND linie = pac_vinzari.v_liniestearsa ;

            -- se scade valoarea atributului LINIE pentru liniile ulterioare celei sterse
            UPDATE linii_fact
            SET linie =  linie - 1
            WHERE NrFact = pac_vinzari.v_nrfact AND linie > pac_vinzari.v_liniestearsa ;
            pac_vinzari.v_trg_liniifact_del := FALSE ;
        END IF ;
```

```
        UPDATE fact
        SET NrLinii =  NrLinii - 1
        WHERE NrFact = pac_vinzari.v_nrfact ;

        pac_vinzari.v_nrfact := NULL ;
        pac_vinzari.v_liniestearsa := NULL ;
        pac_vinzari.v_trg_liniifact := FALSE ;
    END IF ;
END ;
/
```

Declanşatorul de ştergere la nivel de linie stochează numărul facturii şi pe cel al liniei în două variabile publice, iar declanşatorul la nivel de comandă renumerotează liniile în LINII_FACT şi actualizează atributele calculate în FACT. Atenție ! Mecanismul este gândit pentru ştergerea unei singure linii la o comandă DELETE; dacă se introduce un DELETE mai generos, de genul: `DELETE FROM linii_fact WHERE NrFact >= 3`, lucrurile se încâlcesc rău de tot !

Ultima parte a listingului 9.6 prezintă cele două declanşatoare de modificare pentru tabela LINII_FACT. Sunt necesare două, la nivel de linie (BEFORE – ROW) şi la nivel de comandă (AFTER – STATEMENT) întrucât ținem morțiș ca valoarile atributului linie să fie corecte, deci re-corectate după fiecare modificare a `NrFact`.

Listing 9.6 (partea a III-a). Declanşatoarele de modificare pentru LINII_FACT

```
-----------------------------------------------
-- declansatorul de actualizare in LINII_FACT de tip BEFORE ROW
CREATE OR REPLACE TRIGGER trg_lf_upd
    BEFORE UPDATE ON linii_fact FOR EACH ROW
BEGIN
    IF :NEW.Linie <> :OLD.Linie AND pac_vinzari.v_trg_liniifact = FALSE THEN
        RAISE_APPLICATION_ERROR (-20870, 'Atributul Linie nu poate fi modificat direct !');
    END IF ;

    IF pac_vinzari.v_trg_liniifact_del = FALSE THEN
        pac_vinzari.v_trg_liniifact := TRUE ;
        IF :NEW.nrfact <> :OLD.nrfact THEN
            -- linia a fost trasferata in alta factura
            --                  mai intii, se scade valoarea, TVA-ul si se renumeroteaza
            --                              liniile facturii din care "a plecat"
            pac_vinzari.v_nrfact := :OLD.NrFact ;
            pac_vinzari.v_liniestearsa := :OLD.Linie ;
            UPDATE fact SET
                    NrLinii = NrLinii - 1,
                    valtotala = valtotala - :OLD.cantitate * :OLD.pretunit * (1 +
                        (SELECT procTVA FROM produse WHERE codpr=:OLD.codpr)),
                    TVAfact = TVAfact - :OLD.cantitate * :OLD.pretunit *
                        (SELECT procTVA FROM produse WHERE codpr=:OLD.codpr)
            WHERE nrfact=:OLD.nrfact ;

            --      restul face declansatorul UPD AFTER STATEMENT (trg_lf_upd2)

            --                  apoi, se atribute numarul liniei facturii in care "a ajuns"
            --                              si se actualizeaza valoarea si TVA-ul
            :NEW.Linie := pac_vinzari.f_nrlinii (:NEW.NrFact) + 1 ;
            UPDATE fact SET
                    NrLinii = NrLinii + 1,
```

```
                    valtotala = valtotala + :NEW.cantitate *:NEW.pretunit * (1 +
                      (SELECT procTVA FROM produse WHERE codpr=:NEW.codpr)),
                    TVAfact = TVAfact + :NEW.cantitate * :NEW.pretunit *
                    (SELECT procTVA FROM produse WHERE codpr=:NEW.codpr)
            WHERE nrfact=:NEW.nrfact ;

        ELSE
            -- nu s-a modificat nici NrFact, nici Linie
            pac_vinzari.v_nrfact := NULL ;
            pac_vinzari.v_liniestearsa := NULL ;
            UPDATE fact SET
                    valtotala = valtotala - :OLD.cantitate *:OLD.pretunit *
                            (1 + (SELECT procTVA FROM produse
                                    WHERE codpr=:OLD.codpr)) +
                        :NEW.cantitate *:NEW.pretunit * (1 +
                            (SELECT procTVA FROM produse
                                    WHERE codpr=:NEW.codpr)),
                    TVAfact = TVAfact - :OLD.cantitate * :OLD.pretunit *
                            (SELECT procTVA FROM produse WHERE codpr=:OLD.codpr)
            WHERE nrfact=:NEW.nrfact ;
        END IF ;
        pac_vinzari.v_trg_liniifact := FALSE ;
    END IF ;

END ;
/

----------------------------------------------------------------
-- declansatorul de actualizare in LINII_FACT de tip AFTER STATEMENT
CREATE OR REPLACE TRIGGER trg_lf_upd2 AFTER UPDATE ON linii_fact
BEGIN
    IF pac_vinzari.v_trg_liniifact_del = FALSE THEN
        IF pac_vinzari.v_nrfact IS NULL OR pac_vinzari.v_liniestearsa IS NULL THEN
            -- nu s-a facut trasferul liniei in alta factura, deci stam linistiti !
            NULL ;
        ELSE
            pac_vinzari.v_trg_liniifact := TRUE ;
            IF pac_vinzari.f_nrlinii(pac_vinzari.v_nrfact) + 1 =
                        pac_vinzari.v_liniestearsa THEN
                -- linia transferata din factura este ultima
                NULL ;
            ELSE
                -- se scade valoarea atributului LINIE pentru liniile ulterioare celei transferare
                pac_vinzari.v_trg_liniifact_del := TRUE ;
                UPDATE linii_fact
                SET linie =  linie - 1
                WHERE NrFact = pac_vinzari.v_nrfact AND linie >= pac_vinzari.v_liniestearsa ;
                pac_vinzari.v_trg_liniifact_del := FALSE ;
            END IF ;
            pac_vinzari.v_nrfact := NULL ;
            pac_vinzari.v_liniestearsa := NULL ;
            pac_vinzari.v_trg_liniifact := FALSE ;
        END IF ;
    END IF ;
END;
/
```

Derularea comenzilor şi rezultatele obţinute, aşa după cum sunt prezentate în figura 9.3, ne îndreptăţesc să credem că mecanismul funcţionează. În figură, mai întâi se afişează conţinutul tabelelor FACT şi LINII_FACT pentru facturile cu numerele 3 şi 4. Iniţial, factura 3 conţine trei linii, valoarea sa totală este 4215500, iat TVA-ul este de 565500, în timp ce factura cu numărul 4 are doar o linie, valoarea sa este 1190000, iar TVA-ul 190000. Comanda UPDATE pune la încercare declanşatoarele într-una dintre cele mai dificile speţe: linia 1 din factura 3 trece în factura 4. Astfel, pe de o parte, această linie ar trebui să devină linia a doua din factura 4, iar în factura 3 linia a doua ar trebui să devină prima şi a treia să devină a doua.

```
SQL> SELECT * FROM fact WHERE NrFact IN (3,4) ;

    NRFACT DATAFACT       CODCL OBS    VALTOTALA    TVAFACT VALINCASATA    NRLINII
---------- --------- ---------- ----- ---------- ---------- ----------- ----------
         4 28-JAN-05       1003          1190000     190000           0          1
         3 28-JAN-05       1002          4215500     565500      800000          3

SQL> SELECT * FROM linii_fact WHERE NrFact IN (3,4) ;

    NRFACT      LINIE      CODPR  CANTITATE   PRETUNIT
---------- ---------- ---------- ---------- ----------
         4          1          1        800       1250
         3          1          1        700       1200
         3          2          2        800       1600
         3          3          3        900       1700

SQL> update linii_fact set nrfact=4 where nrfact=3 and linie=1;

1 row updated.

SQL> SELECT * FROM fact WHERE NrFact IN (3,4) ;

    NRFACT DATAFACT       CODCL OBS    VALTOTALA    TVAFACT VALINCASATA    NRLINII
---------- --------- ---------- ----- ---------- ---------- ----------- ----------
         4 28-JAN-05       1003          2189600     349600           0          2
         3 28-JAN-05       1002          3215900     405900      800000          2

SQL>  SELECT * FROM linii_fact WHERE NrFact IN (3,4) ;

    NRFACT      LINIE      CODPR  CANTITATE   PRETUNIT
---------- ---------- ---------- ---------- ----------
         4          1          1        800       1250
         4          2          1        700       1200
         3          1          2        800       1600
         3          2          3        900       1700
```

Figura 9.3. Ilustrarea modului de funcţionare a declanşatoarelor

Cele două SELECT-uri succesoare comenzii UPDATE ne arată că, după actualizare, ambele facturi au două linii, valoarea facturii 3 a scăzut de la 4215500 le 3215900, în timp ce valoarea facturii 4 a crescut cu aceeaşi sumă. În plus, ordinea liniilor din tabela LINII_FACT este cea corectă. Ca şi în cazul ştergerii de tupluri din relaţia LINII_FACT, funcţionalitatea declanşatoarelor din listingul 9.6 este asigurată numai când printr-un UPDATE este afectată o singură linie.

9.5. Relații redundante/sintetice

O situația ceva mai rar întâlnită este cea în care, din rațiuni de minimizare a timpului de acces, se introduc nu numai atribute suplimentare, redundante, ci chiar relații întregi ! Chiar dacă economia de joncțiuni şi timpul de răspuns scurtat reprezintă deziderate generale ale tuturor relațiilor redundante/sintetice, se poate încerca o tipologizare a după cum urmează:

- relații necesare implementării unor restricții mai complexe, greu implementabile (numai) prin clauze CHECK şi declanşatoare;
- relații necesare obținerii mai rapide unor informații punctuale foarte frecvent solicitate;
- relații ce vizează accelerarea unor calcule pentru operațiuni frecvente;
- relații-semnal, pentru consemnarea sau semnalizarea unor situații limită, sau depăşirea unor plafoane etc.
- relații necesare la raportare, uneori chiar relații-raport;
- relații necesare re-grupărilor de date, pentru analize multidimensionale, OLAP etc.;
- relații destinate corespondențelor dintre datele actuale şi cele mai vechi.

Fireşte, diferențele dintre aceste categorii nu sunt întotdeauna evidente, destule cazuri pliindu-se pe două sau mai multe situații dintre cele enumerate. Prima categorie a fost ilustrată în paragraful 7.2.3. Pentru a doua categorie, primul exemplu pe care îl avem la îndemână priveşte tot baza de date EXAMENE. Toate notele obținute de un student sunt contabilizate meticulos în relația NOTE {Matricol, CodDisc, DataEx, Nota}. Structura este vitală pentru cunoaşterea studenților care trebuie să plătească a treia prezentare la un examen, să refacă activitatea din cursul anului (obligatorie la a patra prezentare) etc. Dezavantajul major al acestei scheme este că în zeci sau chiar sute de cazuri în care interesează media studentului sau numărul său de credite, trebuie luată în calcul doar *ultima notă obținută la fiecare obiect* !

Lucrul acesta reclamă destul de mult timp, deoarece presupune execuția unei interogări de genul:

```
SELECT matricol, coddisc, nota
FROM note
WHERE (matricol, coddisc, dataex) IN
     (SELECT matricol, coddisc, MAX(dataex)
      FROM note
      GROUP BY matricol, coddisc)
```

care, vă asigur, este costisitoare când NOTE are sute de mii sau milioane de înregistrări.

Deşi pare greu acceptabilă, putem propune soluția folosirii unei relații speciale, NOTE_FINALE {Matricol, CodDisc, NotaFinală} în care fiecare student are, pentru fiecare disciplină la care a susținut măcar un examen, numai ultima notă

(cea finală), obținută la prima, a doua, ... prezentare (vezi listing 9.7 de pe web). La inserarea unei linii în tabela NOTE, inserare care respectă ordinea cronologică a examinărilor, trebuie verificat dacă aceasta ar fi prima examinare de la disciplina respectivă, caz în care trebuie făcută o inserare în NOTE_FINALE, iar în caz contrar trebuie făcută modificarea valorii atributului `NOTE_FINALE.notafinală` - vezi listing 9.8.

Listing 9.8. Declanşatorul de inserare în NOTE

```
CREATE OR REPLACE TRIGGER trg_note_ins
    AFTER INSERT ON note FOR EACH ROW
BEGIN
    UPDATE note_finale SET notafinala = :NEW.nota
    WHERE matricol=:NEW.matricol AND coddisc=:NEW.coddisc ;

    IF SQL%ROWCOUNT = 0 THEN -- nr.liniilor modificate este 0
            INSERT INTO note_finale VALUES (:NEW.matricol, :NEW.coddisc, :NEW.nota) ;
    END IF;
END ;
```

Marele pericol al soluției decurge din supoziția că operarea notelor respectă ordinea cronologică a susținerii examenelor. Dacă, din neatenție, se preiau notele din restanțe înaintea celor din sesiune, corectitudinea datelor este compromisă. Aşa încât merită să introducem în NOTE_FINALE şi atributul `DataUltimuluiExamen` care să ne scutească de orice emoție: NOTE_FINALE {Matricol, CodDisc, NotaFinală, DataUltimuluiExamen}:

```
ALTER TABLE note_finale ADD (DataUltimuluiExamen DATE)
```

Iată cum ar trebuie să arate noul declanşator de inserare (listing 9.9) :

Listing 9.9. Noul declanşator de inserare în NOTE

```
CREATE OR REPLACE TRIGGER trg_note_ins
    AFTER INSERT ON note FOR EACH ROW
BEGIN
    UPDATE note_finale
    SET notafinala = :NEW.nota, dataultimuluiexamen = :NEW.dataex
    WHERE matricol=:NEW.matricol AND coddisc=:NEW.coddisc
        AND dataultimuluiexamen < :NEW.dataex ;

    IF SQL%ROWCOUNT = 0 THEN
        -- inserarea se face numai daca nu e nici  o inregistrare pentru
        --    examinarea (matricol, coddisc) curenta în NOTE_FINALE
        INSERT INTO note_finale
            SELECT matricol, :NEW.coddisc,  :NEW.nota, :NEW.dataex
            FROM studenti WHERE matricol=:NEW.matricol AND
               NOT EXISTS
                    (SELECT 1 FROM note_finale WHERE matricol=:NEW.matricol AND
                            coddisc=:NEW.coddisc) ;
    END IF;
END ;
```

Dacă exemplul vi s-a părut cu totul exagerat, încercăm un altul: ce ziceți de o bază de date pentru gestionarea informațiilor referitoare la sezonul curent din Formula 1 ? Sezonul este alcătuit dintr-o serie de curse desfăşurate pe diferite circuite (Estoril, Monza, Silverstone etc.). O poziție în primele 8 locuri oferă un anumit punctaj: poziția 1 conferă 10 puncte, poziția 2 - 8 puncte, 3 - 6 puncte, 4-5 puncte ..., 8 - 1 punct. Graful DF este cel din figura 9.4.

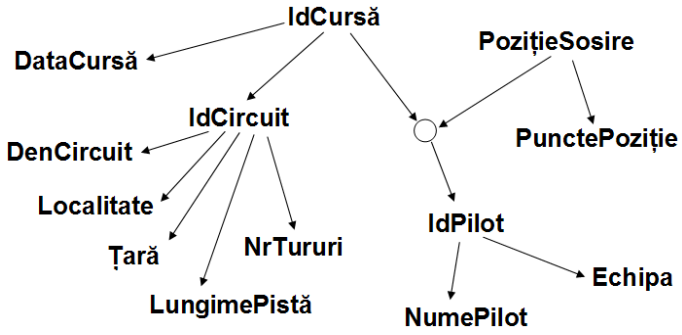

Figura 9.4. Graful DF pentru baza de date FORMULA1

Decupăm relațiile din graf (scriptul de creare este disponibil pe web sub forma listingului 9.10, iar cel de populare este conținut în listingul 9.11):

CIRCUITE {IdCircuit, DenCircuit, Localitate, Ţară, LungimePistă, NrTururi}

PILOȚI {IdPilot, NumePilot, Echipa}

LOCURI_PUNCTE {PoziţieSosire, PunctePoziţie}

CURSE {IdCursă, DataCursă}

REZULTATE_CURSE {IdCursă, PoziţieSosire, IdPilot}

Dintre informațiile cel mai frecvent solicitate, am putea lua în discuție cel puțin două. Prima este: *ce număr de puncte are pilotul P1* sau *echipa E1* ? Folosind schema de mai sus, normalizată cum e ea, trebuie operate, de fiecare dată, joncțiunile:

```
SELECT SUM(PunctePoziţie) AS NrPuncte
FROM locuri_puncte lp
INNER JOIN rezultate_curse rc
ON lp.PoziţieSosire=rc.PoziţieSosire
    INNER JOIN piloţi p ON rc.IdPilot=p.IdPilot
WHERE NumePilot = 'P1'
```

Nu este, astfel, prea îndrăzneață ideea de a introduce în relația REZULTATE_CURSE atributul NrPuncteCursă:

```
ALTER TABLE rezultate_curse ADD (NrPuncteCursa NUMBER(3)) ;
```

Acest nou atribut va fi actualizabil prin triggerele de inserare şi modificare ale tabelei REZULTATE_CURSE, după modelul din listingul 9.12:

Listing 9.12. Calculul automat al numărului de puncte obţinut pentru fiecare clasare

```
CREATE OR REPLACE TRIGGER trg_rc_ins
    BEFORE INSERT ON rezultate_curse FOR EACH ROW
DECLARE
    v_puncte NUMBER(2) := 0 ;
BEGIN
    SELECT PunctePozitie INTO v_puncte FROM locuri_puncte
    WHERE PozitieSosire = :NEW.PozitieSosire ;

    :NEW.NrPuncteCursa := v_puncte ;
END ;
```

Mergând mai departe, n-ar fi o idee tocmai rea ca în tabela PILOŢI să centralizăm punctele obţinute pe întreg sezonul printr-un atribut special dedicat - `PuncteSezon`. Gestiunea sa este oarecum banală, realizându-se prin intermediul declanşatoarelor tabelei REZULTATE_CURSE, mai precis a atributului `NrPuncteCursă`. În acest fel am fi în măsură să obţinem în orice moment clasamentul piloţilor. Pentru a avea un clasament *on-line* pe echipe, soluţia cea mai la îndemână ar fi o tabelă virtuală:

```
CREATE VIEW puncte_echipe
   SELECT echipa, SUM(PuncteSezon) AS PuncteEchipă
   FROM piloţi
   GROUP BY echipe
```

Amatorii de senzaţii tari pot folosi, însă, şi o relaţie specială:

PUNCTE_ECHIPE {Echipa, Puncte}

ale cărei înregistrări sunt gestionate exclusiv prin intermediul declanşatoarelor relaţiei PILOŢI. Această relaţie acoperă, într-o măsură mai mare sau mai mică, atât categoria celor dedicate obţinerii rapide a unor informaţii frecvent solicitate, cât şi categoria relaţiilor-raport. Rămâne să vă încordaţi forţele pe cont propriu, deoarece noi sărim cu discuţia la alte cazuri.

Pentru exemplificarea tabelelor redundante/sintetice destinate accelerării calculelor în operaţiuni complexe, revenim la paragraful 8.3, care s-ar fi vrut dedicat grafurilor, dar a sfârşit doar prin a contura o schemă de baze de date utilă unei firme de transport intern de călători dotată cu celebrele microbuze cu minunaţii lor şoferi. Structura tabelelor este ilustrată în figura 8.23, iar listingul 8.14, vecin figurii, propunea o funcţie (F_COST_BILET) ce primeşte trei parametri: localitatea de urcare a călătorului, cea de coborâre şi identificatorul rutei urmate de mijlocul de transport, şi furniza, după cum îi spune şi numele, contravaloarea

biletului (presupunând că şoferul n-ar face un mic ghişeft, în dulcele spirit balcanic).

Deşi poate interesantă, funcţia mănâncă ceva timp şi resurse, deoarece la fiecare bilet eliberat se parcurge cursorul în care se află încărcate toate localităţile de pe ruta specificată aflate între localitatea de urcare şi de coborâre, adunându-se numărul de kilometri şi, la final, determinându-se costul călătoriei. Având în vedere că recalcularea (scumpirea, de fapt) biletelor este din ce în ce mai rară, o idee nu tocmai rea ar fi să se recurgă la o relaţie suplimentară, altminteri destul de banală:

TARIFE { IdRută, Loc1, Loc2, Preţ }

Înregistrările acestei tabele pot fi generate printr-o procedură care va apela şi funcţia din listingul 8.14, iar, după generare, întregul calcul va fi înlocuit cu accesul, incomparabil mai rapid, la una din înregistrările acestei noi tabele. Unde mai punem că această nouă tabelă nici nu (prea) trebuie modificată până la următoarea scumpire a biletelor sau adăugarea de noi rute.

Cei care aţi avut răbdare să citiţi până spre capăt capitolul 8, aţi sesizat că nici în privinţa rezervărilor de locuri la cursele firmei nu ne simţeam prea bine. Declanşatorul relaţiei REZERVĂRI făcea un adevărat tur de forţă încercând să determine, pentru cursa respectivă, vreun segment de rută pentru care numărul biletelor rezervate depăşeşte numărul locurilor microbuzului. Şi în această situaţie putem îndulci rezolvarea creând o relaţie suplimentară/redundantă/sintetică s.a.m.d:

CURSE_REZERVĂRI {IdCursă, Loc1, Loc2, NrLocuriRez}

Actualizată prin declanşatoarele tabelei REZERVĂRI, această nouă relaţie poate simplifica şi accelera semnificativ urmărirea locurilor disponibile pentru fiecare cursă.

O categorie mai exotică de relaţii sintetice, negăsită (de mine) în materialele dedicate denormalizării, este cea a tabelelor-semnal, apropiată, ca finalitate, de tabelele jurnal pe care le vom prezenta pe scurt în ultimul capitol. Dacă tot n-am apucat să ne despărţim de aplicaţia pentru firma de transport călători, am putea imagina o tabelă în care să consemnăm momentul în care, pentru o cursă, numărul de locuri rezervate depăşeşte capacitatea microbuzului, astfel încât, fie se sistează rezervările (cazul descris în declanşatorul din listingul 8.15), fie se apelează la un alt microbuz, mai mare (adică *macrobuz*), fie se primesc oricâte rezervări şi, în funcţie de acestea, se suplimentează cursa cu încă unul sau mai multe mijloace de transport:

DEPĂŞIRI_LOCURI {IdDepăşire, IdCursa, NrLocuriDepăşire}

Exemplul nu este chiar mişcător, aşa că am putea încerca un altul. În privinţa unei aplicaţii pentru gestiunea stocurilor, există firme, nu prea numeroase, care operează după următorul scenariu: unele materii prime/materiale/mărfuri sunt aprovizionate de la furnizori plasaţi la oarecare distanţă. Deoarece durata procesului de aprovizionare poate pune firma în situaţii de blocaje, se ia decizia ca fiecărui sortiment să-i fie stabilit un stoc de siguranţă, iar aplicaţia dedicată stocurilor să semnalizeze momentul în care stocul curent al materialului respectiv scade sub nivelul celui de siguranţă.

MATERIALE {IdSortiment, DenSortiment, UM, ContClasa3, Grupa, StocSiguranţă}
...
INTRĂRI_MATERIALE {IdIntrare, IdSortiment, CantIntrată, PreţIntrare}
...
IEŞIRI_MATERIALE {IdIeşire, IdSortiment, CantIeşită, CostDescărcare}

SEMNALE {IdSemnal, DataOraSemnal, IdSortiment, CantitateSubStocSiguranţă}

Cei care se ocupă cu aprovizionarea nu ar avea decât să interogheze periodic această tabelă, sau să fie atenţionaţi într-un fel sau altul la fiecare inserare de înregistrări în tabela SEMNALE.

O altă categorie enumerată la începutul acestui paragraf făcea referinţă la relaţiile sintetice de tip raport. O asemenea tabelă poate fi considerată PUNCTE_ECHIPE {Echipa, Puncte} din exemplul curselor de formula I, deşi prezentarea sa a fost făcută la categoria relaţiilor redundante/sintetice pentru mărirea vitezei de furnizare a informaţiilor frecvent solicitate.

Şi aici putem vorbi de două situaţii, una în care rapoartele pot fi obţinute destul de uşor creând o tabelă virtuală (view) pe scheletul machetei listei, în timp ce a doua chiar necesită tabele în toată regula, cu tot cu redundanţa lor. După exemplul tabelei virtuale PUNCTE_ECHIPE de aproximativ două pagini mai sus, ne reîntoarcem la figura 9.1 şi discuţiile din preajma ei. Ţinând seama de frecvenţa accesării soldurilor şi rulajelor, am apelat, pentru tabela CONTURI_ELEMENTARE, la două atribute noi, RulajDB şi RulajCR, actualizabile exclusiv prin declanşatoarele tabelei DETALII_OPERAŢIUNI (vezi şi paragraful 7.3). Un raport solicitat frecvent este balanţa de verificare, din care o porţiune este prezentată în figura 9.5[26].

[26] Din raţiuni de lizibilitate, în figură nu au fost incluse ultimele două coloane ale balanţei, care conţin soldurile finale debitoare şi creditoare.

CONT	DENUMIRECONT	T	SOLDINITDB	SOLDINITCR	RULAJDB	RULAJCR	TOTALSUMEDB	TOTALSUMECR
101	Capital social	P	0	0	0	0	0	0
1011	Capital social subscris ne-varsat	P	0	0	0	0	0	0
1012	Capital social subscris varsat	P	0	0	0	0	0	0
121	Profit si pierdere	B	0	0	375000	0	375000	0
211	Terenuri si amenajari	A	0	0	11900000	1900000	11900000	1900000
2111	Terenuri	A	0	0	11900000	0	11900000	0
2112	Amenajari	A	0	0	0	1900000	0	1900000
301	Materii prime	A	0	0	4000000	375000	4000000	375000
308	Diferente de pret la materii prime	B	0	0	0	0	0	0
401	Furnizori	P	0	0	0	16660000	0	16660000
411	Clienti	A	0	0	0	0	0	0
428	Alte creante si datorii fata de salariati	B	0	0	0	0	0	0
4281	Alte creante fata de salariati	A	0	0	0	0	0	0
4282	Alte datorii fata de salariati	P	0	0	0	0	0	0
4426	TVA deductibila	A	0	0	2660000	0	2660000	0
4427	TVA colectata	P	0	0	0	0	0	0
601	Cheltuieli cu materii prime	A	0	0	375000	375000	375000	375000
=====	=== T O T A L	=	0	0	19310000	19310000	19310000	19310000

Figura 9.5. O porţiune dintr-o balanţă de verificare

Dificultatea ține de faptul că soldurile inițiale şi operațiunile contabile sunt introduse la nivel de conturi elementare, în timp ce balanța se obține doar la nivelul conturile sintetice de gradul 1 şi 2. Se poate concepe o tabelă specială în acest scop – BALANȚĂ {SimbolCont, DenumireCont, TipCont, SoldInitDB, SoldInitCR, RulajDB, RulajCR, TotalSumeDB, TotalSumeCR, SoldFinalDB, SoldFinalCR} -, în care actualizările să se facă automat, din declanşatoarele tabelei CONTURI_ELEMENTARE. Efortul de scriere a pachetelor şi declanşatoarelor n-ar fi prea mare, însă soluția este cam penibilă, întrucât, ori de câte ori s-ar dori listat conținutul balanței curente, ar fi suficientă o comandă SELECT de genul:

```
SELECT SimbolCont AS Cont, DenumireCont, TipCont,
       (SELECT NVL(SUM(SoldInitDB),0) FROM conturi_elementare
        WHERE SUBSTR(ContElementar, 1, LENGTH(pc.SimbolCont))
              = pc.SimbolCont) SoldInitDB,
       (SELECT NVL(SUM(SoldInitCR),0) FROM conturi_elementare
        WHERE SUBSTR(ContElementar, 1, LENGTH(pc.SimbolCont))
              = pc.SimbolCont) SoldInitCR,
       (SELECT NVL(SUM(RulajDB),0) FROM conturi_elementare
        WHERE SUBSTR(ContElementar, 1, LENGTH(pc.SimbolCont))
              = pc.SimbolCont) RulajDB,
       (SELECT NVL(SUM(RulajCR),0) FROM conturi_elementare
        WHERE SUBSTR(ContElementar, 1, LENGTH(pc.SimbolCont))
              = pc.SimbolCont) RulajCR,
       (SELECT NVL(SUM(SoldInitDB),0)+ NVL(SUM(RulajDB),0)
        FROM conturi_elementare
        WHERE SUBSTR(ContElementar, 1, LENGTH(pc.SimbolCont))
              = pc.SimbolCont) TotalSumeDB,
       (SELECT NVL(SUM(SoldInitCR),0)+ NVL(SUM(RulajCR),0)
        FROM conturi_elementare
        WHERE SUBSTR(ContElementar, 1, LENGTH(pc.SimbolCont))
              = pc.SimbolCont) TotalSumeCR,
       (SELECT NVL(SUM(
              CASE TipCont
              WHEN 'A' THEN NVL(SoldInitDB,0) + NVL(RulajDB,0)
                    - NVL(RulajCR ,0)
              WHEN 'P' THEN 0
              ELSE
```

```
                    CASE WHEN NVL(SoldInitDB,0) + NVL(RulajDB,0)
                           - NVL(RulajCR ,0) < 0
                    THEN 0 ELSE NVL(SoldInitDB,0) + NVL(RulajDB,0)
                           - NVL(RulajCR,0)
                    END
             END), 0)
      FROM conturi_elementare
      WHERE SUBSTR(ContElementar, 1, LENGTH(pc.SimbolCont))
             = pc.SimbolCont) SoldFinalDB,
      (SELECT NVL(SUM(
             CASE TipCont
             WHEN 'P' THEN NVL(SoldInitCR,0) + NVL(RulajCR,0)
                    - NVL(RulajDB ,0)
             WHEN 'A' THEN 0
             ELSE
                    CASE WHEN NVL(SoldInitCR,0) + NVL(RulajCR,0)
                           - NVL(RulajDB ,0) < 0
                    THEN 0 ELSE NVL(SoldInitCR,0) + NVL(RulajCR,0)
                           - NVL(RulajDB,0)
                    END
             END), 0)
      FROM conturi_elementare
      WHERE SUBSTR(ContElementar, 1, LENGTH(pc.SimbolCont))
             = pc.SimbolCont) SoldFinalCR
FROM plan_conturi pc
WHERE LENGTH(SimbolCont)=4 OR LENGTH(SimbolCont)=3
UNION
SELECT '=====', '=== T O T A L ', '=',
      (SELECT NVL(SUM(SoldInitDB),0) FROM conturi_elementare),
      (SELECT NVL(SUM(SoldInitCR),0) FROM conturi_elementare),
      (SELECT SUM(RulajDB) FROM conturi_elementare),
      (SELECT SUM(RulajCR) FROM conturi_elementare),
      (SELECT NVL(SUM(SoldInitDB),0)+ NVL(SUM(RulajDB),0)
              FROM conturi_elementare),
       (SELECT NVL(SUM(SoldInitCR),0)+ NVL(SUM(RulajCR),0)
       FROM conturi_elementare),
      (SELECT NVL(SUM(
             CASE TipCont
             WHEN 'A' THEN NVL(SoldInitDB,0) + NVL(RulajDB,0)
                    - NVL(RulajCR ,0)
             WHEN 'P' THEN 0
             ELSE
                    CASE WHEN NVL(SoldInitDB,0) + NVL(RulajDB,0)
                           - NVL(RulajCR ,0) < 0
                    THEN 0 ELSE NVL(SoldInitDB,0) + NVL(RulajDB,0)
                           - NVL(RulajCR,0)
                    END
             END), 0)
             FROM conturi_elementare ce INNER JOIN plan_conturi pc2
                    ON contelementar=pc2.simbolcont),
             (SELECT NVL(SUM(
                    CASE TipCont
                    WHEN 'P' THEN NVL(SoldInitCR,0) + NVL(RulajCR,0)
                           - NVL(RulajDB ,0)
                    WHEN 'A' THEN 0
                    ELSE
                           CASE WHEN NVL(SoldInitCR,0) +
```

```
                          NVL(RulajCR,0)- NVL(RulajDB ,0) < 0
                         THEN 0
                         ELSE NVL(SoldInitCR,0) + NVL(RulajCR,0)
                          - NVL(RulajDB,0)
                         END
                  END), 0)
          FROM conturi_elementare ce INNER JOIN plan_conturi pc2
                ON contelementar=pc2.simbolcont)
FROM dual
```

Pentru cei mai puţin răbdători, fraza supraponderală a fost salvată ca listing 9.13 (pe web), iar eliminarea neplăcerii de a-i admira „rotunjimile" la lansare, se poate rezolva prin crearea unei tabele virtuale care va fi consultată ulterior cu un SELECT obişnuit:

```
CREATE VIEW view_balanta AS
SELECT.....
```

Pentru unele rapoarte mai sofisticate, cum sunt fişele de cont, fişele-şah (tot pentru conturi) etc. argumentele creării unor tabele speciale sintetice sunt ceva mai uşor de luat în seamă. De exemplu, pentru obţinerea fişelor de cont, se poate apela la o tabelă redundantă de genul:

FIŞE_CONT {SimbolCont, DenumireCont, Debit_Credit, SoldIniţial, Cont_Corespondent, Rulaj}

Actualizarea unei asemenea tabele nu ar costa prea mult, iar avantajele ar fi ceva mai vizibile decât în cazul balanţei de verificare.

Mergând către finalul (semi) clasificării noastre din acest paragraf, vom zăbovi preţ de câteva fraze la relaţiile suplimentare necesare regrupărilor de date, analizelor multidimensionale etc. Un lua un prim exemplu ar viza analiza vânzărilor. În relaţia produse PRODUSE {CodPr, DenPr, UM, ProcTVA, Grupa} există un atribut – Grupa - prin care specificăm dacă produsul face parte din categoria mărfurilor *alimentare*, sau *ţigări*, *tutun*, *dulciuri* etc. La un moment dat, ne-ar putea interesa ponderea vânzărilor de mărfuri alimentare din carne de porc faţă de cele de vită sau de pui. Sau care ar fi tipul de cafea decofeinizată cel mai bine vândut. Pentru a răspunde la aceast gen de probleme, putem defini un soi de ierarhie a mărfurilor, după ideea aplicată competenţelor necesare posturilor dintr-o organizaţie (vezi capitolul 8).

Astfel, în categoria *băuturi*, am include *răcoritoarele*, *berea*, *vinul*, *apa minerală* şi *băuturile spirtoase*. *Berea*, ca să luăm una dintre subcategorii, am împărţi-o în *neagră/brună* şi *blondă*, iar tipurile de *bere blondă* le-am descompune pe *mărci* (Carlsberg, Beck's, Timişoreana – asta-i reclamă nemascată, nu-i aşa ?), apoi pe *forme de prezentare* (sticle, cutii, peturi), apoi pe volumul (număr de centilitri) de bere conţinut. Am putea avea o relaţie:

CATEGORII {IdCategorie, Dencategorie, Caracteristică, MărimeaCaracteristicii, IdCategorieSuperioară}

iar în PRODUSE, în loc de Grupa, am folosi atributul-cheie străină IdCategorie. Ei bine, ce facem dacă vrem să comparăm vânzările de bere la *cutie*, faţă de cea la *sticlă* şi cele la *pet*, indiferent de producător ? Mai general spus, ne interesează categorii de sortimente care sunt plasate pe mai multe ramuri ale clasificaţiei arborescente. Soluţia care ne poate rezolva problema este de a defini o tabelă de criterii şi alta care să conţină toate categoriile incluse în criteriul respectiv:

CRITERII {IdCriteriu, DenCriteriu, Observaţii}
şi
CRITERII_CATEGORII {IdCriteriu, IdCategorie}

Astfel, în CRITERII vom insera o linie pentru criteriul *Bere la cutie*, pentru care în CRITERII_CATEGORII va exista câte o linie pentru fiecare sortiment de bere la cutie existent: *bere Timişoareana Gold – cutie 500 ml*; *bere Timişoareana Pils – cutie 500 ml*; *bere Guiness – cutie 500 ml etc.*; apoi, criteriul *Bere la sticlă* s.a.m.d.

Un al doilea exemplu vine din aplicaţia de CONTABILITATE. În PLAN_CONTURI nomenclatorul general al conturilor a fost completat cu analitice definite de organizaţie. Astfel, putem defini în clasa 3 analitice pentru fiecare tip de materie primă/material: 301.001 pentru nisip, 301.002 pentru var, 301.003 pentru ciment etc. Conturile de cheltuieli din clasa 6 le-am putea grupa după obiectivele în lucru şi natura operaţiilor derulate, o serie de operaţii fiind comune mai multor obiective:

- 601.001.001 – Cheltuieli la obiectivul nr.1, operaţiunea 1;
- 601.001.003 – Cheltuieli la obiectivul nr.1, operaţiunea 3;
- 601.003.001 – Cheltuieli la obiectivul nr.3, operaţiunea 1;

La orice consum de materie primă/material se debitează contul din clasa 6 ce indică obiectivul şi operaţia destinaţie şi se creditează analiticul ce indică natura materialui consumat. Putem, astfel, afla toate cheltuiele cu materiale pentru fiecare obiectiv în parte, ca şi modul în care au fost consumate nisipul, varul, cimentul pe obiective. Însă consumul de materiale pe operaţiuni (pentru toate obiectivele) este ceva mai este greu de aflat, iar dacă aceeaşi operaţiune are analitic diferit, de la obiectiv la obiectiv, greul devine imposibil. Soluţia ar fi crearea unei relaţii noi:

CHELTUIELI_OBIECTIVE_OPERAŢIUNI {ContAnalitic, Obiectiv, Operaţiune}

Atenţie, însă ! Nu cred că ar fi nimerit să asimilăm aceste artificii denormalizării propriu-zise, întrucât relaţii introduse aduc informaţii noi, greu, dacă nu chiar imposibil, de obţinut printr-un alt artificiu. Aşa că, în loc de denormalizare, mai bine vorbim de post-normalizare, că sună mai bine !

Pentru ultima categorie de relație redundant-sintetică dintre cele enumerate ne rezervăm pentru ultimul capitol.

9.6. Atribute introduse pe baza grupurilor repetitive

În capitolul 1 am discutat diverse soluții de eliminare a grupurilor repetitive şi atomizării valorilor tuturor atributelor din bază, chiar dacă teoreticienii modelului relațional au eliminat dintre poruncile relaționalului atomicitatea atributelor, aşa cum a fost înțeleasă de majoritatea autorilor/practicienilor, drept "scalaritatea" valorilor dintr-un domeniu. Ca idee generală, recomandam ca eliminarea grupurilor repetitive să fie făcută în funcție de specificul aplicației sau chiar să nu fie făcută deloc.

Exemplu

În urma normalizării, ținând seama că o firmă client poate avea mai multe telefoane de contact sau adrese e-mail, iar, pe de altă parte, nu există două numere de telefon sau adrese de e-mail absolut identice, s-ar putea ajunge la o structură de genul:

CLIENȚI {CodCl, DenCl, CodFiscal, StradaCl, NrStadaCl, BlocScApCl, CodPost}

TELEFOANE_CLIENȚI { Telefon, CodCl}

E_MAIL_CLIENȚI {eMail, CodCl}

Fără nici un dubiu, schema este scrupulos normalizată, neexistând atribute compuse. Cu toate acestea, nici Telefon, nici eMail nu sunt atribute care să fie supuse operațiunilor de agregare (numărare, însumare), şi, mai important, nu participă nici în joncțiuni cu alte tabele ale bazei de date. Este motivul pentru care, în acest gen de situații se poate lua, fără prea multe remuşcări, decizia acceptării valorilor compuse pentru atributele Telefon şi eMail:

CLIENȚI3 {CodCl, DenCl, CodFiscal, StradaCl, NrStadaCl, BlocScApCl, CodPost, Telefoane, eMailuri}.

Putem, astfel, vorbi de un soi de denormalizare, de revenire la o formă pre-1FN. Vestea bună e că tipurile de date compozite pot fi gestionate în multe SGBD-uri fără eforturi deosebite. Spre exemplu, Oracle prezintă două tipuri de colecții ce pot fi stocate într-o bază de date, *tabele incluse* (imbricate), în original *NESTED TABLES*, şi *vectori de mărime variabilă* - *VARRAYS*. Tocmai pentru ca exemplul de mai sus să prindă viață, în listingul 9.14 se crează două tipuri de tabele incluse, unul pentru numere de telefon şi altul pentru adresele de e-mail. Aceste tipuri sunt folosite în tabela CLIENȚI3.

Listing 9.14. Valori neatomice folosind tabele incluse (nested tables) în Oracle

```
DROP TABLE clienti3;
DROP TYPE nt_adrese_email ;
DROP TYPE nt_telefoane ;

CREATE TYPE nt_telefoane AS TABLE OF CHAR(11)
/

CREATE TYPE nt_adrese_email AS TABLE OF VARCHAR2(30)
/

CREATE TABLE CLIENTI3 (
    CodCl NUMBER(6) PRIMARY KEY,
    DenCl VARCHAR2(40),
    CodFiscal CHAR(8),
    StradaCl VARCHAR2(30),
    NrStadaCl VARCHAR2(7),
    BlocScApCl VARCHAR2(25),
    CodPost NUMBER(6),
    Telefoane nt_telefoane,
    eMailuri nt_adrese_email
    )
NESTED TABLE telefoane STORE AS telefoane_tab,
NESTED TABLE eMailuri STORE AS eMailuri_tab
/

INSERT INTO clienti3 VALUES (1001, 'Client 1', 'R133303', 'Sapientei',
    '22bis', NULL, 710710,
    nt_telefoane('0232344444', '0788881919', '0744444444'),
    nt_adrese_email('client1@k.ro', 'client.1@yahoo.com')
    ) ;
INSERT INTO clienti3 VALUES (1002, 'Client 2', 'R1344533', 'Pacientei',
    '13bis', 'Bl.H, Sc.B, Ap.5', 710705,
    nt_telefoane('0239344554', '0723881919', '0722444444', '0744456789'),
    nt_adrese_email('client2@hotmail.ro', 'client2@yahoo.com')
    ) ;
```

Tabela CLIENŢI3 este populată cu două înregistrări, primul client având trei numere de telefon şi două adrese e-mail, iar al doilea patru numere de telefon şi tot două adrese. Conţinutul tabelei este afişat după cum arată figura 9.6. "Relaţionalizarea" conţinutului tabelei incluse presupune folosirea expresiei `TABLE` plasată în clauza `FROM` a unei fraze `SELECT`. Astfel,

```
SELECT *
FROM TABLE (SELECT c.telefoane FROM clienti3 c
            WHERE c.codcl=1002) x ;
```

afişează telefoanele celui de-al doilea client (vezi figura 9.6), în timp ce:

```
SELECT c.dencl, t.*
FROM clienti3 c, TABLE(c.telefoane) t WHERE codcl=1002 ;
```

le afişează precedându-le de numele clientului.

```
SQL> SELECT * FROM clienti3 c ;

     CODCL DENCL           CODFISCA STRADACL        NRSTADA BLOCSCAPCL            CODPOST
---------- --------------- -------- --------------- ------- ------------------ ----------
TELEFOANE
--------------------------------------------------------------------------------------------
EMAILURI
--------------------------------------------------------------------------------------------
      1001 Client 1        R133303  Sapientei       22bis                          710710
NT_TELEFOANE('0232344444 ', '0788881919 ', '0744444444 ')
NT_ADRESE_EMAIL('client1@k.ro', 'client.1@yahoo.com')

      1002 Client 2        R1344533 Pacientei       13bis   Bl.H, Sc.B, Ap.5       710705
NT_TELEFOANE('0239344554 ', '0723881919 ', '0722444444 ', '0744456789 ')
NT_ADRESE_EMAIL('client2@hotmail.ro', 'client2@yahoo.com')


SQL> SELECT *
  2  FROM TABLE (SELECT c.telefoane FROM clienti3 c
  3    WHERE c.codcl=1002) x ;

COLUMN_VALU
-----------
0239344554
0723881919
0722444444
0744456789
```

Figura 9.6. Gestionarea tabelelor incluse

Pentru a de-denormaliza, adică a re-normaliza telefoanele şi adresele e-mail, soluţia este relativ simplă:

```
SELECT c.codcl, t.*
FROM clienti3 c, TABLE(c.telefoane) t  ;
```

respectiv:

```
SELECT c.codcl, t.*
FROM clienti3 c, TABLE(c.emailuri) t  ;
```

Întrucât aflarea numerelor de telefon ale unui client nu ridică probleme, să vedem cum se soluţionează problema: *care dintre clienţi are numărul 0722444444 ?* Dacă ţinem cont că numele automat al atributului obţinut prin aplicarea funcţiei `TABLE` este COLUMN_VALUE, soluţia este mai mult decât banală:

```
SELECT c.dencl, t.*
FROM clienti3 c, TABLE(c.telefoane) t
WHERE t.column_value='0722444444'
```

În acest mod, stocăm atributele multi-valoare de o manieră ceva mai „naturală”, în timp ce pentru a obţine informaţii continuăm să apelăm la frazele SELECT în maniera obişnuită de interogare în SQL.

Contra-exemplu

Pentru baza de date dedicată gestionării cărților dintr-o bibliotecă, avem şi un de contraexemplu de folosire a grupurilor repetitive. Astfel, fiecare carte are unul, doi sau mai mulți autori, aşa încât ar fi fost tentantă o structură de genul:

CĂRȚI {ISBN, Titlu, Autori, ...}.

Spre deosebire de cazul telefoanelor şi adreselor e-mail, nu este recomandabil ca atributul Autori să fie declarat de tip colecție, şi aceasta din trei motive. Pe de o parte, căutările după un autor, sau doi autori etc. sunt relativ numeroase, iar asemenea gen de căutare folosind opțiuni SQL ar fi îngreunată în condițiile folosirii colecțiilor. În al doilea rând, este foarte posibil ca o aplicație „profesională" să folosească o tabelă-nomenclator AUTORI, astfel încât postura atributului Autori ar fi de cheie străină care va apărea, inevitabil, în joncțiuni. De asemenea, un al treilea motiv ține de faptul că *ordinea* autorilor este foarte importantă; la afişarea datelor despre o carte este neapărat ca ordinea autorilor să fie cea de pe copertă. Aşa încât cea mai indicată variantă foloseşte un atribut special pentru a indica ordinea fiecărui autor pe coperta unei cărți.

Iar dacă, pentru un raport, se doreşte afişarea bibliografică „clasică": autori (în ordinea de pe copertă), titlu, editură/publicație, anul apariției, se poate crea o tabelă virtuală, iar valoarea atributului Autori să fie „calculată" cu o funcție specială care să returneze un şir de caractere de lungime variabilă, în funcție de câți autori au participat la scrierea respectivului material.

9.7.Ruperea relațiilor pentru creşterea vitezei de acces

Una dintre poruncile multor autori importanți în domeniul proiectării bazelor de date este: *nu rupeți (descompuneți) relațiile deplin normalizate în relații mai mici* ![27] Ca buni români, trebuie să privim orice lege cu rezervele de rigoare, căutând porțiunile mai laxe. Revenim la relația STUDENȚI. Fără a complica inutil discuția, în capitolele şi paragrafele precedente am invocat doar câteva informații care interesează secretariatul unei facultăți. Pentru o aplicație dedicată gestiunii datelor despre studenți şi activități didactice, discuția trebuie extinsă serios, deoarece documentele ce trebuie întocmite sunt dintre cele mai diverse:

- pentru cazările în căminele universității este necesară o listă a studenților, pe sexe (băieți/fete), ani de studii şi specializări, listă în care să nu fie incluşi cei din oraşul Iaşi (e de presupus că aceştia stau cu părinții);
- centrele militare județene solicită în fiecare an o listă cu băieții care sunt studenți ai facultății (pentru a nu întocmi pentru ei ordine de încorporare);
- dosarul fiecărui student trebuie să cuprindă şi date legate de liceul absolvit, anul absolvirii, media de la bacalaureat şi media pe cei patru ani de liceu;

[27] Vezi, spre exemplu, [Fleming & vonHalle89], p.195

- în foile matricole şi alte documente legate de şcolaritate apar: numele dinaintea căsătoriei (adică numele din certificatul de naştere, pentru persoanele care în urma căsătoriei şi-au schimbat numele de familie), numele mamei şi al tatălui, data şi locul naşterii etc.

Chiar dacă ne oprim aici cu discuţia, ar însemna că tabela STUDENŢI ar avea o structură de genul: STUDENŢI {Matricol, NumePren, Sex, An, Modul, IdSpec, Grupa, Adresa_Str, Adresa_Nr, Adresa_Bloc, Adresa_Scară, Adresa_Etaj, Adresa_Apart, Adresa_CodPoştal, NumeCertificatNaştere, DataNaşterii, LoculNaşterii, JudeţulNaşterii, Mama, Tata, Liceu_Absolvit, CodPoştal_Liceu, Medie_Bac, Medie_Ani_Liceu, StagiuMilitar}. Iar dacă ne mai concentram puţin, mai găseam nişte informaţii suplimentare. Toate atributele sunt în depedenţă funcţională faţă de Matricol, aşa că includerea lor în aceeaşi relaţie este întru totul legitimă.

Cu toate acestea, din momentul preluării acestor date, în anul I, până după momentul absovirii (şi, implicit, arhivării datelor), dintre toate aceste atribute doar câteva sunt folosite frecvent, cea mai mare parte fiind utile doar de câteva ori. De aceea, atunci când numărul studenţilor este de ordinul miilor sau zecilor de mii (cazul unei universităţi), numărul disciplinelor poate fi de câteva sute, iar numărul de note preluate se poate ridica la sute de mii, chiar milioane, n-ar fi rău deloc ca relaţia STUDENŢI să se spargă astfel:

STUDENŢI {Matricol, NumePren, An, Modul, IdSpec, Grupa}

STUD_DATEGENERALE {Matricol, Sex, Adresa_Str, Adresa_Nr, Adresa_Bloc, Adresa_Scară, Adresa_Etaj, Adresa_Apart, Adresa_CodPoştal, NumeCertificatNaştere, DataNaşterii, LoculNaşterii, JudeţulNaşterii, Mama, Tata, StagiuMilitar}

STUD_LICEU {Matricol, Liceu_Absolvit, CodPoştal_Liceu, Medie_Bac, Medie_Ani_Liceu, StagiuMilitar}

Prin comparaţie cu "relaţia-mamut", economia de timp este semnificativă la obţinerea datelor privitoare la examene, prin joncţionarea cu relaţia NOTE. Risipa de timp şi efort se face simţită la obţinerea informaţiilor extrase din STUD_DATEGENERALE şi STUD_LICEU, întrucât sunt necesare joncţiuni suplimentare cu tabela STUDENŢI. Plus timpul necesar gestionării restricţiei referenţiale.

Inclusă de mulţi autori la „capitolul" denormalizare, ruperea sau partiţionarea pe verticală adaugă redundanţă (apar restricţii referenţiale suplimentare), dar nu economiseşte din joncţiuni, ci dimpotrivă. Şi aici s-ar potrivi observaţia lui Chris Date precum că mulţi autori cad în capcana confuziei dintre denormalizare şi redundanţă. Decizia segmentării pe verticală a bazei de date trebuie asumată numai dacă numărul de atribute din tabelă şi lungimea lor sunt foarte mari, volumul datelor poate creşte de o manieră impresionantă în timp şi, în plus, gradul

de accesare al diferitelor atribute sau categorii de atribute prezintă diferențe semnificative.