Capitolul 7. Surogate, reguli, incluziuni. Un caz practic. Implementări SQL/Oracle

Cu atuurile şi limitele ei, normalizarea reprezintă un excelent ghid pe care l-am folosit pentru a obține o schemă acceptabilă a bazei de date. Pentru ceea ce urmează încercăm să legăm normalizarea de alte elemente structurale ale bazei, cum ar fi identificarea mai simplă a entităților şi restricțiile complexe ale datelor din bază. De asemenea, acest capitol încearcă să demonstreze că, în multe cazuri, în normalizare, pe lângă dependențele funcționale, multivaloare şi de joncțiune, esențiale sunt şi cele ce fac parte dintr-o specie destul de neglijată - dependențele de incluziune.

De asemena, vom începe să prezentăm şi modalități practice de declarare/implementare a diferitot categorii de restricții discutate, iar cazul practic de la finalul capitolui încearcă să lege elementele teoretice presărate în primele şase capitole, să ilustreze câteva dintre frământări, dacă nu chiar convulsii, care însoțesc procesul de elaborare a schemei baze de date pentru o aplicație reală, chiar în condițiile în care datele problemei nu sunt complicate din cale-afară.

7.1.Chei surogat

Folosind limbajul de lemn al tehnologiei informaționale, putem spune că bazele de date stochează persistent şi gestionează varii informații privitoare la fenomene, procese, tranzacții, persoane etc. din mai toate domeniile de activitate, în funcție de aplicațiile pentru care sunt proiectate şi implementate. Una din problemele de căpătâi în lucrul cu aceste entități, procese, tranzacții etc. ține de posibilitatea identificării lor fără ambiguitate.

7.1.1. Nevoia de surogate

Închipuiți-vă ce-ar însemna ca firma noastră să vireze câteva sute de milioane de lei în contul unui client, plătind astfel o factură, iar contul respectiv să fie confundat cu un altul, al altui client. Sau ce s-ar întâmpla dacă la plata "pe card" a salariilor plătite s-ar produce niscaiva confuzii: cei mai oropsiți s-ar oripila văzând cât câştigă şefii (confirmând luminoasa teză "impozita-i-ar naiba !" a unor preşedinți retardați istoric şi economic), iar şefii ar fi cel puțin la fel de oripilați, dacă nu chiar îngroziți de mânia populară/proletară.

Aşa că vedem în jurul nostru tot soiul de "ingrediente" menite a face diferențieri sau, altfel spus, a identifica fiecare persoană, carte, factură, plată etc. Privind retrospectiv, în relația STUDENȚI_EXAMENE - figura 2.1, paragraful 2.1 - fiecare student este identificat de matricolul său, iar disciplinele au un cod (`CodDisc`) unic

prin care evităm confuzia ce s-ar putea genera prin folosirea exclusivă a denumirii disciplinei (respectiv, numelui studentului). De asemenea, valoarea atributului `NrFact` este unică pentru orice factură emisă de firma noastră şi trimisă unui client (baza de date VÎNZĂRI). `ISBN`-ul este un cod unic asociat, la nivel mondial, unei cărți tipărite, în timp ce cota identifică într-o bibliotecă un exemplar al unei cărți. Codul numeric personal (`CNP`-ul) este cel mai bun mod de a identifica o persoană, în timp ce codul fiscal este un element de încredere când ne propunem să identificăm un client.

Din contră, în paragraful 3.4, exasperați de problemele ridicate de nulitatea atributelor cheie din relația CODURI_NOI_V2 (figura 3.18), am apelat la o soluție "gordiană" introducând un atribut cu oarecare doză de artificialitate, `Id`, care ar identifica fiecare grup de adrese atribuite unui cod poştal (figura 3.20, relația CODURI_NOI_V3). Atributul `Id` este un exemplu de *cheie surogat*, adică un câmp oarecum artificial de care vorbeam. Dealtminteri, noțiunea de surogat desemnează un produs artificial sau sintetic care este utilizat drept substitut pentru un produs natural. O cheie surogat este o cheie artificială sau sintetică utilizată ca substitut al unei chei "naturale".

Pe parcursul precedentelor capitole au mai fost strecurate chei surogat precum: `IdFilm` (baza de date FILMOGRAFIE), `CodFamilie` (relația DOTARE_JUCĂRII), `CodCl`, `CodPr`, `CodÎnc` şi `CodDoc` (baza de date VÎNZĂRI), `CodProf` (baza de date EXAMENE - paragraful 5.5.2) sau `IdProf` (relația PROF_DISCIPLINE din paragraful 5.5.3).

De cele mai multe ori, cheile surogat simplifică graful dependențelor funcționale şi, implicit, sarcina demarcării relațiilor. Astfel, revenind la relația LINII_ARTICOLE_CONTABILE din figura 4.1, ştiind că o notă contabilă are mai multe operațiuni care, la rândul lor, conțin mai mai multe corespondențe `ContDebitor - ContCreditor`, introducem două atribute care să preia explicațiile/comentariile despre note, respectiv, operațiuni - `ExplicaţiiNotă` şi `ExplicaţiiOp`. Din paragraful 4.1.2 ştim că, în funcție de modul de contare, configurația dependențelor poate fi sensibil diferită. Pentru uniformizarea celor două variante de numerotare a notelor contabile, putem recurge la două atribute de tip cheie surogat. `IdNotăContabilă` va fi un număr unic asociat fiecărei note contabile, iar `IdOperaţiune` va identifica orice operațiune, indiferent de nota contabilă din care face parte. Graful ia configurația din figura 7.1.

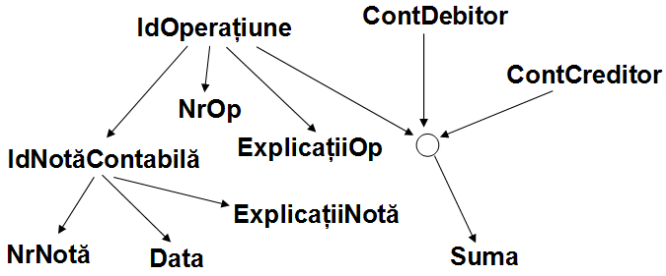

Figura 7.1. Două chei surogat pentru LINII_ARTICOLE_CONTABILE

Schema obținută prin decuparea dependențelor din graf este:

NOTE_CONTABILE {IdNotăContabilă, NrNotă, Data, ExplicaţiiNotă}

OPERAȚIUNI {IdOperaţiune, NrOp, IdNotăContabilă, ExplicaţiiOp}

DETALII_OPERAȚIUNI {IdOperaţiune, ContDebitor, ContCreditor, Suma}

Un alt caz în care folosirea cheii surogat modifică sensibil lucrurile este cel al relației DOTARE_JUCĂRII_1 reprezentată în figura 3.14 (finalul paragrafului 3.3.3), despre care ultima dată am discutat în paragraful 6.2 - vezi figura 6.13. În condițiile date la acel moment, această relație necesită trei atribute care să confere unicitate fiecărui tuplu - (CodFamilie, Copil, Jucărie). Introducem un atribut suplimentar, IdJucărie, nimic altceva decât o cheie surogat prin care diferențiem jucăriile între ele. Aceasta ar fi a doua cheie surogat din relație, după CodFamilie. Coroborând dependențele funcționale din paragraful 4.1.2 cu dependențele multi-valoare din paragraful 6.1 şi discuția din 6.2, în condițiile în care, într-o familie, orice jucărie este cumpărată în N exemplare, unde N reprezintă numărul copiilor din acea familie, grafurile sunt cele din figura 7.2, diferențierea fiind făcută de momentul cumpărării celor N "instanțe" ale jucăriei într-o familie.

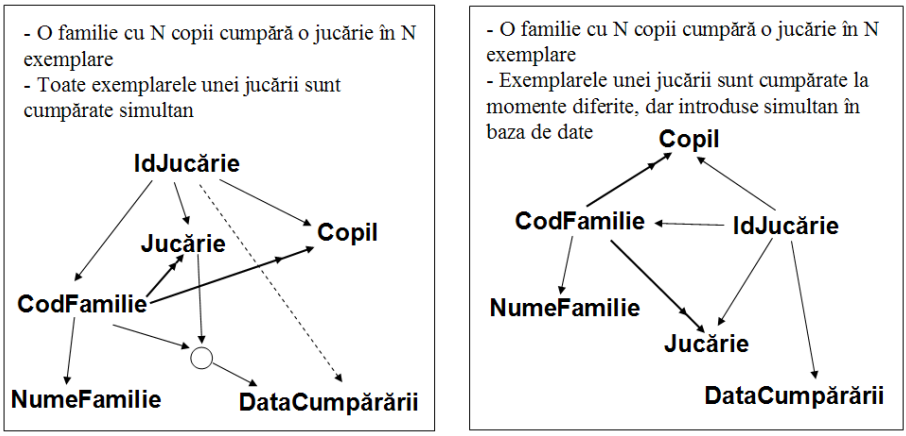

Figura 7.2. Grafurile dependenţelor pentru relaţia DOTARE_JUCĂRII "dotată" cu două chei surogat

Începem cu graful din strânga, cel care reflectă achiziționarea simultană a câte un exemplar pentru fiecare copil dintr-o familie. Dependența funcțională dintre IdJucărie şi DataCumpărării este una tranzitivă, aşa că s-a recurs la "punctarea" sa, pentru a nu fi luată în considerare. Din dependențele funcționale, am obține relațiile:

FAMILII {CodFamilie, NumeFamilie},

JUCĂRII_COPII {IdJucărie, Jucărie, CodFamilie, Copil}
şi
JUCĂRII_FAMILII {CodFamilie, Jucărie, DataCumpărării}.

Dacă ţinem cont de dependenţele multivaloare, s-ar mai adăuga acestor relaţii şi:

COPII_FAMILII {CodFamilie, Copil}
şi
JUCĂRII_FAMILII2 {CodFamilie, Jucărie }.

Ambele sunt redundante, însă în timp ce COPII_FAMILII se constituie ca un soi de nomenclator al copiilor dintr-o familie, permiţând preluarea în baza de date şi a copiilor din familiile fără nici o jucărie înregistrată (aşa cum observam la finalul paragrafului 6.2), JUCĂRII_FAMILII2 nu se justifică cu nici un preţ, toate informaţiile furnizate de aceasta găsindu-se în relaţia JUCĂRII_COPII.

Continuăm cu partea dreaptă a grafului din figura 7.2. Faptul că exemplare diferite din aceeaşi jucărie pot fi cumpărate la momente diferite, chiar dacă se preiau simultan, ne scuteşte de dependenţa funcţională cu sursa compusă vizibilă pe graful din stânga. De asemenea, dependenţa funcţională dintre IdJucărie şi DataCumpărării nu mai este tranzitivă. Decuparea DF generează relaţiile:

FAMILII {CodFamilie, NumeFamilie}
şi
JUCĂRII {IdJucărie, Jucărie, CodFamilie, Copil, DataCumpărării}.

Dacă ţinem cont de DMV, am mai obţine relaţiile COPII_FAMILII {CodFamilie, Copil} şi JUCĂRII_FAMILII2 {CodFamilie, Jucărie}. Din raţiunile expuse mai sus, ar fi bine să o păstrăm doar pe prima şi să renunţăm la cea din urmă.

Din moment ce am introdus un atribut pentru identificarea fiecărui exemplar dintr-o jucărie, pare cât se poate de normal să folosim o asemenea cheie surogat pentru identificarea fiecărui copil din baza de date. La drept vorbind, acest atribut există, şi nu e câtuşi de puţin surogat - CNPCopil. Noi, însă, fideli tematicii acestui paragraf, ne vom preface că nu avem de unde să ştim CNP-urile copiilor, aşa că vom lucra cu IdCopil.

Noul rând de grafuri ale dependenţelor capătă forma din figura 7.3. Motivul principal pentru care ne ocupăm de acest caz este apariţia în ambele grafuri a unei simetrii ciudate între o dependenţă funcţională şi una multivaloare. Astfel, în graful din partea stângă, IdCopil $\longrightarrow$ CodFamilie, dar şi CodFamilie $\longrightarrow$ $\rightarrow$ IdCopil. Simetria ce apare în graful din dreapta se manifestă între aceleaşi atribute.

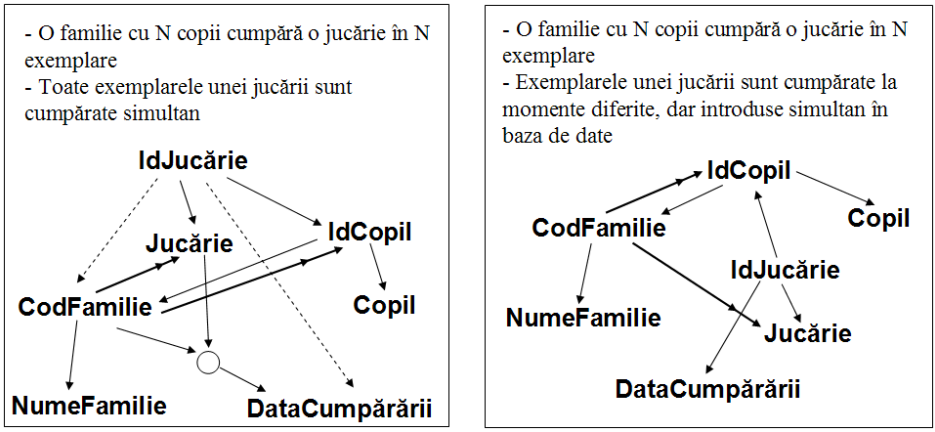

Figura 7.3. Grafurile dependenţelor pentru relaţia DOTARE_JUCĂRII "dotată" cu trei chei surogat

Nu este prima dată în această carte când ridicăm din umeri şi ne întrebăm ce-i de făcut. Primul impuls este de a elimina una dintre dependenţe. Cel mai pieziş ne uităm spre cea multivaloare, însă trebuie să ne temperăm elanul atâta vreme cât o dependenţă multivaloare se stabileşte între două atribute, dar numai în prezenţa unui al treilea: `CodFamilie` $\longrightarrow\!\!\!\!\rightarrow$ `IdCopil|Jucărie`. Ne va fi greu să justificăm prezenţa în graf a DMV `CodFamilie` $\longrightarrow\!\!\!\!\rightarrow$ `Jucărie`, atâta vreme cât `CodFamilie` $\longrightarrow\!\!\!\!\rightarrow$ `IdCopil` este eliminată din calcul. Aşa că, deocamdată nu renunţăm la nimic din simetrie...

Ocupându-ne de graful din stânga imaginii, observăm că acum două dintre dependenţe sunt punctate, pe motiv de tranzitivitate. Pe baza DF obţinem relaţiile:

FAMILII {`CodFamilie`, `NumeFamilie`}

COPII {`IdCopil`, `Copil`, `CodFamilie`}

CUMPĂRĂRI_JUCĂRII {`CodFamilie`, `Jucărie`, `DataCumpărării`}.
şi
JUCĂRII_COPII {`IdJucărie`, `Jucărie`, `IdCopil`}.

Din DMV, s-ar mai adăuga:
COPII_FAMILII {`CodFamilie`, `IdCopil`}
şi
JUCĂRII_FAMILII2 {`CodFamilie`, `Jucărie`}.

Informaţia furnizată de COPII_FAMILII, copiii unei familii, este furnizată de COPII, iar la JUCĂRII_FAMILII2 putem renunţa fără remuşcări, atâta vreme cât există CUMPĂRĂRI_JUCĂRII. Iată cum am ajuns ca DMV să nu aibă nici o

incidență directă în schema finală a bazei de date ! Graful din partea dreaptă a figurii rămâne drept temă pentru acasă.

7.1.2. Puncte de vedere privind cheile surogat

Părerile despre cheile surogat sunt, firește, împărțite. Există autori care fac distincția între chei surogat definite/controlate de utilizatori şi chei surogat gestionate exclusiv de sistem (SGBD). Pentru Codd, *cheile primare definite şi controlate de utilizator* folosite ca surogate permanente pentru entități, ar prezenta trei dificultăți la întrebuințare[1]:

- Valorile cheilor surogat trebuie uneori schimbate. De exemplu, atunci când două companii fuzionează, angajații trebuie reuniți într-o sigură tabelă, şi nu este exclus ca, datorită suprapunerilor, unele mărci să fie modificate.
- Două relații ar putea avea chei surogat utilizator definite pe domenii diferite, chiar dacă entitățile pe care le identifică sunt aceleaşi; de exemplu, o firmă poate fi deopotrivă client şi furnizor al companiei noastre. În tabela FURNIZORI firma ar putea fi identificată prin `CodFurnizor`, iar în tabela CLIENȚI prin `CodClient`. Deşi firma este aceeaşi, ea este identificată prin două chei surogat diferite.
- Uneori este necesară preluarea informației despre o entitate înaintea atribuirii unei chei surogat utilizator pentru aceasta sau, pe de altă parte, păstrarea valorii cheii surogat chiar şi după ce entitatea nu mai constituie subiect al aplicației. Cazurile tipice sunt cele ale candidaților pentru angajarea pe un post (li se va atribui o marcă doar dacă vor fi angajați, atribuirea petrecându-se după angajare), sau cele ale angajaților pensionați.

Cele trei dificultăți sunt suficiente lui Codd pentru a justifica folosirea *cheilor surogat sistem*, pe care utilizatorul nu le poate nici modifica, iar uneori nici măcar vizualiza. Astfel, fiecare entitate ar avea propria cheie surogat, unică la nivelul întregii baze de date. Două chei surogat sunt identice dacă şi numai dacă desemnează aceeaşi entitate. Codd se gândea chiar şi la o comandă specială de fuzionare (COALESCE) a două chei surogat care desemnează, în fapt, aceeaşi entitate. Mai mult, în articolul său din 1979 publicat în *ACM Transactions on Database Systems*, autorul pledează pentru un model relațional extins în care, printre multe altele, unul dintre domenii să servească drept sursă pentru toate cheile surogat din bază.

La întrebarea "Ce e mai preferabil, să folosim cheile primare "naturale" sau cele surogat ?" nu există un răspuns universal sau, chiar dacă ar exista, am evita să-l dăm. Atunci când atributul/atributele ce identifică entitatea sunt disponibile şi stabile în timp, soluția "naturală" este mai la îndemână: `CNP`-ul pentru persoane, `ISBN`-ul pentru cărți, `CodFiscal` pentru firme. Când apar probleme de

[1] [Codd79]

identificare, iar unicitatea este asigurată de un mare grup de atribute, sau când atributul/atributele potențiale chei nu prezintă stabilitate pe termen lung, atunci soluția "surogat" este cea mai indicată. La exemplele din acest paragraf, mai putem adăuga:

- Seria şi numărul de buletin/carte de identitate - deşi identifică fiecare persoană, peste un număr de ani, datorită schimbării buletinului/cărții de identitate, combinația îşi schimbă valoarea, iar dacă s-au făcut deja arhivări, vor apărea complicații.
- Matricol - în multe şcoli, licee şi universități s-a practicat un sistem simplu de atribuire a numărului matricol, astfel încât, după câțiva ani, un matricol era "reciclat". Căutarea în baza de date arhivă a liceului ar ridica probleme mari, dacă intervalul care interesează este de ordinul deceniilor. Necazuri similare ar apărea dacă unele firme ar recicla, în timp, mărcile angajaților, numerele de inventar ale mijloacelor fixe etc.

Asemenea gen de discuții este valabil însă şi la reciclarea cheilor surogat, aşadar ideea folosirii unor surogate-sistem se poate dovedi benefică.

Dintre materialele dedicate cheilor surogat, v-am recomanda pe cele scrise de Mike Lonigro[2], Ian Harrington[3], Graeme C. Simsion[4] şi Fabian Pascal. În ceea ce mă priveşte, nu împărtăşesc nici lehamitea unor autori față de cheile surogat, mergându-se, în acest sens, până la a le "demasca" drept identificatoare deghizate de obiecte (celebrele OID-uri din orientarea pe obiecte), dar nici frenezia "surogării" manifestată de mulți proiectanți de baze de date, deşi, recunosc, frenezia cu pricina nu are nimic toxic în ea, ci, mai degrabă, ține de un soi de lene (subscriu cu toată inima la ideea că lenea nu e toxică, dacă nu e însoțită de "aditivi").

O altă temere legată de cheile surogat priveşte îngreunarea accesului la informațiile din baza de date, întrucât acesta nu se mai realizează prin atribute explicite, ci prin numere destul de irelevante pentru obiectul/entitatea în cauză. Ori, utilizatorii obişnuiți ar fi obligați să rețină kilograme întregi de cifre nesuferite, doar pentru a obține datele de care au nevoie. Nici acest neajuns nu trebuie să descurajeze, deoarece trebuie să vedem o aplicație, indiferent de tipologia sa, pe cel puțin două straturi, *date* şi *interfață*. În majoritatea copleşitoare a cazurilor, utizatorii interacționează cu baza doar prin meniuri, rapoarte şi mai ales formulare, iar cheile surogat pot fi chiar ascunse.

În orice caz, dacă nu am reuşit să tranşăm discuția chei naturale-chei surogat, măcar să arătăm cu degetul pe cele mai toxice: cheile inteligente. Astfel, au existat firme în care angajații compartimentelor *personal-salarizare* audiaseră sau au citiseră în cursurile de baze de date sau analiză/proiectare despre teoria codurilor şi, la angajarea unei persoane la compartimentul *Contabilitate*, îi atribuiau o marcă de tip *CTB85CC101*, în care primele trei litere semnalizau că este încadrat la

[2] [Lonigro98]

[3] [Harrington02], pp.77-82

[4] [Simsion01], pp.282-285

compartimentul *Contabilitate*, iar cele două *C*-uri semnalizau biroul *Calculația costurilor*, în cadrul compartimentului *Contabilitate*. Partizanii acestui sistem argumentau că marca ar conține, în acest fel, informații prețioase (compartimentul şi biroul) despre fiecare angajat, cheia fiind chiar, după unii autori, inteligentă.

După cinci ani, însă, persoana respectivă se transferă la compartimentul *Financiar*. Asta înseamnă că marca, fiind inteligentă, trebuie schimbată. Costul schimbării depinde de întrebarea: unde se află cele 5 (ani) * 12 (luni) înregistrări despre calculului lunar al salariului, plus 5(ani) * 12 (luni) * 20 (zile lucrătoare) înregistrări, dacă pontajul se preia zilnic ? Răspunsul poate fi însoțit de apăsătoare dureri de cap.

7.1.3. Declararea/obținerea cheilor surogat

Modalitățile prin care un atribut poate fi declarat cheie surogat diferă de la SGBD la SGBD. Trei dintre variante sunt folosite ceva mai des. Prima este declararea unui atribut de tip *autoincrement*. Spre exemplu, în PostgreSQL există tipul de dată SERIAL:

```
CREATE TABLE note_contabile (
    idnotacontabila SERIAL PRIMARY KEY
          ...
    )
```

În momentul inserării unei linii noi în tabelă, atributul `IdNotăContabilă` va primi, pe rând, valorile: 1, 2, ... până la 2147483647[5]. Dacă se doreşte crearea unei chei surogat de tip sistem, iar valoarea maximă de mai sus nu este de ajuns, se poate folosi tipul BIGSERIAL, cu o plajă imensă.

Şi în Visual FoxPro, odată cu versiunea 8 un atribut poate fi declarat cheie surogat folosing clauza `AUTOINC`:

```
CREATE TABLE note_contabile (;
    idnotacont INTEGER AUTOINC NEXTVALUE 1001 STEP 1 PRIMARY KEY;
    )
```

Clauza adițională `NEXTVALUE` este utilă atunci când se doreşte ca valoarea inițială să nu fie 1, iar `STEP` stabileşte mărimea incrementată. În versiunile mai vechi ale VFP era necesară folosirea clauzei DEFAULT care făcea apel la o funcție stocată[6]:

```
CREATE TABLE note_contabile (;
    idnotacont NUMBER(10) DEFAULT vi_nc_idnota() PRIMARY KEY;
    )
```

funcția fiind cea din listing 7.1.

---

[5] Vezi documentația PostgreSQL de pe www.postgresql.org

[6] Vezi şi [Fotache s.a.02], pp.184-188

Listing 7.1. Funcţia stocată care furnizează valoarea implicită a cheii surogat

```
PROCEDURE vi_nc_idnota
LOCAL v_idnota (1,1)
v_idnota=0
SELECT MAX(idnotacont) FROM note_contabile ;
    INTO ARRAY v_idnota
IF v_idnota (1,1) = 0 && e prima nota contabila
    RETURN 1001
ELSE
    RETURN v_idnota(1,1) + 1
ENDIF
ENDPROC
```

O altă soluție, disponibilă în Oracle şi PostgreSQL, presupune folosirea secvențelor, care sunt obiecte ale bazei de date ce furnizează la fiecare invocare a clauzei `NextVal` o valoare unică:

```
CREATE SEQUENCE seq_idnota START WITH 1001
 MINVALUE 1001 MAXVALUE 9999999999 NOCACHE NOCYCLE ORDER
```

Secvența `seq_idnota` creată va furniza valori strict ordonate după momentul apelului, cuprinse între 1001 şi 9999999999, iar după atingerea limitei superioare secvență se blochează (clauza `NOCYCLE`). În lipsa clauzei NOCYCLE, după atingerea valorii maxime, următoarea valoare furnizată este cea minimă. Secvența va fi folosită în declanşatorul de inserare al tabelei NOTE_CONTABILE. Scriptul de crearea a celor trei tabele discutate în paragraful 7.1.1 (vezi figura 7.1) constituie subiectul listingului 7.2 care poate fi descărcat de la adresa http://www.feaa.uaic.ro/cercetare/publicatii). Iată corpul declanşatorului în listing 7.3[7].

Listing 7.3. Declanşatorul Oracle pentru valoarea implicită a cheii surogat

```
CREATE OR REPLACE TRIGGER trg_note_contabile_ins
    BEFORE INSERT ON note_contabile FOR EACH ROW
BEGIN
    SELECT seq_idnota.NextVal INTO :NEW.IdNotaContabila FROM dual ;
END ;
```

7.2. Restricții simple şi complexe. Atribute redundante ajutătoare

Nu întotdeauna atributele implicate în schema bazei pot fi identificate cu uşurință. Mai mult, deseori nu strică o doză rezonabilă de artificial, de improvizație, doză care poate salva pe termen lung o schemă sau măcar să o scutească de multe necazuri. Deşi mai puțin intuitiv decât modelul obiectual, relaționalul are certe atuuri în ceea ce priveşte mecanismul de declarare a

[7] Detalii privind declaşatoarele pentru valori implicite în Oracle sunt prezentate în paragraful 11.2 al lucrării [Fotache s.a.03], pp.386-391

restricțiilor şi mai ales în ceea ce priveşte opțiunile de extragere a informațiilor din bază (mecanismul de *manipulare a datelor*). Chiar şi aşa, rămân în discuție o mare parte din restricții care țin de integritatea bazei de date, dar sunt legate nu atât de reguli intrinseci modelului (cheie primară, restricție de entitate, integritate referențială), cât de regulile aplicației, sau, altfel spus, reguli ale afacerii. Din păcate, nici un model de date nu oferă un mecanism infailibil de declarare a tuturor restricțiilor semantice dintr-o bază de date.

7.2.1. Reguli la nivel de atribut şi înregistrare

Până în acest paragraf, cele mai pomenite restricții semantice ale bazei au fost dependențele funcționale, multivaloare şi de joncțiune. Probabil, însă, că practicienii sunt cel mai bine familiarizați cu altfel de restricții: reguli de validare la nivel de atribut (*field validation rule*) şi reguli de validate la nivel de înregistrare (*record validation rule*).

Să luăm în discuție schema bazei de date EXAMENE din paragraful 5.5.2 (caz practic nr.2), al cărei graf DF constituie subiectul figurii 5.17. Reamintim că baza de date preia informații legate de cursurile predate şi rezultatele obținute la examinările organizate la *Facultatea de Economie şi Administrarea Afacerilor*. Chiar şi la o privire sumară a celor şase relații, se pot desprinde câteva *reguli de validare la nivel de atribut*:

- în relația STUDENȚI:
  - `An` (anul de studiu al studentului) este un număr natural cuprins între 1 şi 5;
  - `Modul`-ul în care poate fi înscris un student de la FEAA este 1 sau 2;
  - `Spec`-ializarea poate avea doar una cele 10 valori: *Finanțe-Bănci*, *Contabilitate*, *Informatică economică*, *Marketing* etc.
- în relația DISCIPLINE numărul de credite al unei discipline (`NrCredDisc`) nu poate fi mai mare de 8;
- în PROFESORI, atributul `GradProf` poate avea doar una din şase valori: *colaborator*, *preparator*, *asistent*, *lector*, *conferențiar* şi *profesor*.
- examenele sunt programate doar în lunile ianuarie-februarie şi mai-iunie-iulie, corespunzător celor două sesiuni; prin urmare, componenta *lună* a atributului `DataEx` în tabelele SĂLI_EX şi NOTE trebuie să se încadreze în cele cinci valori;
- nota maxim obținută este 10, iar pentru *absent* valoarea convenită a notei este zero (discutabil !).

Acest gen de restricții poate fi legat în modelul relațional de noțiunea de domeniu de valori. După cum vom discuta spre finalul acestei lucrări, majoritatea SGBD-urilor nu se bazează pe definirea explicită a domeniului, ci prin opțiuni de declarare a acestor reguli de o manieră intuitivă, prin comenzi SQL sau interactiv, de fiecare SGBD.

În privința *regulilor la nivel de înregistrare*, principala diferență față de cele la nivel de atribut ține de numărul atributelor implicate simultan, cel puțin două. De exemplu, în relația STUDENȚI, ținând seama că primii doi ani de studiu sunt comuni, iar înscrierea la o specializare se face de fiecare student începând cu anul 3, este evident că pentru pentru firecarea linie în care `An` este 1 sau 2, valoarea atributului `Spec` nu trebuie să se regăsească în lista celor 10 descrisă mai sus; aşadar, avem nevoie şi de o valoare de genul *Trunchi comun* care se adaugă domeniului de valori pentru `Spec`; fireşte, am putea recurge şi la valoarea NULL, însă, fideli principiului *ocoliți NULL-ul ori de câte ori aveți ocazia*, vom marşa pe varianta trunchiului comun. Încercând să ne apropiem mai mult de o formalizare aproximativă, putem redacta restricția la modul: dacă valoarea atributului `An` este *1* sau *2*, atunci obligatoriu atributul `Spec` ia valoarea *Trunchi comun* !

7.2.2. Reguli, atribute redundante şi declanşatoare

Cele două tipuri de restricții acoperă o zonă destul de subțire din clasa regulilor ce pot guverna o aplicație/bază de date. Încercăm să trecem într-un registru mai dificil. Ne întoarcem la baza de date BIBLIOTECĂ, a cărei schemă a fost finalizată în paragraful 6.2 (vezi figura 6.10) prin aducerea în 4NF. Din rațiuni bugetare, conducerea facultății/bibliotecii stabileşte următoarea restricție: *O carte nu poate fi achiziționată în mai mult de 10 exemplare !* Cum se poate implementa o asemenea restricție ? Norocul nostru este că, între cele cinci relații ale bazei de date, e uşor de identificat cea împricinată - EXEMPLARE {`Cotă`, `ISBN`}. Respectarea acestei restricții presupune ca, la inserarea unei linii în această tabelă, sau la modificarea unui ISBN, să se verifice dacă numărul cotelor pentru "noul" ISBN (valoarea atributului `ISBN` de pe linia inserată sau modificată) nu depăşeşte 10, situație în care inserarea/modificarea să fie blocată.

Acest gen de restricție poate fi implementată în două moduri. Fără a schimba schema bazei de date, se recurge la declanşatoarele de inserare şi modificare ale tabelei EXEMPLARE, declanşatoare care operează după o logică simplă de genul prezentată în listingul 7.5 (listingul 7.4 poate fi descărcat de pe pagina Web amintită mai sus, conținând scriptul de creare a tabelelor):

Listing 7.5. Declanşator Oracle de inserare[8] pentru implementarea restricţiei legate de nr. de exemplare dintr-o carte

```
CREATE OR REPLACE TRIGGER trg_exemplare_ins
    BEFORE INSERT ON exemplare
    FOR EACH ROW
DECLARE
    v_nrexemplare NUMBER(5) := 0 ;
BEGIN
    SELECT COUNT(*) INTO v_nrexemplare
    FROM exemplare WHERE isbn = :NEW.isbn ;
```

[8] Declanşatorul de modificare trebuie să fie redactat astfel încât să preîntâmpine situațiile de "mutanță" a tabelei EXEMPLARE (vezi [Fotache s.a.03], pp.396-402).

```
        IF v_nrexemplare > 9 THEN
            RAISE_APPLICATION_ERROR (-20801, 'Nr. exemplarelor acestei carti creste peste 10 !')
;
        END IF ;
    END ;
```

Variantei i se poate reproşa cel puțin faptul că un mare număr de cărți poate implica timpi de execuție destul de greu de trecut cu vederea, deoarece `SELECT COUNT(*)` –ul parcurge întreaga tabelă. Aşa că ne-am putea gândi la o variantă ce pare, la prima vedere, mai laborioasă: în tabelă TITLURI introducem atributul `NrExemplare`, atribut care va fi actualizat automat prin cele trei declanşatoare ale EXEMPLARE: TITLURI {`ISBN`, `Titlu`, `Editura`, `AnApariţie`, *`NrExemplare`*}:

```
ALTER TABLE titluri ADD (NrExemplare  NUMBER(4));
```

În continuare, creăm o funcție `f_NrExemplare` căreia i se pasează un ISBN şi returnează numărul de exemplare al acelui ISBN - vezi listing 7.6.

Listing 7.6. Funcţie Oracle ce returnază nr. de exemplare ale unei cărţi

```
    CREATE OR REPLACE FUNCTION f_NrExemplare ( isbn_ titluri.isbn%TYPE)
        RETURN NUMBER
    IS
        v_nr NUMBER(4) := 0 ;
    BEGIN
        SELECT nrexemplare INTO v_nr FROM titluri WHERE isbn=isbn_ ;
        RETURN v_nr ;
    END ;
```

Declanşatoarele tabelei EXEMPLAREl, care folosesc funcția, se prezintă astfel:

- Pentru inserare - vezi listing 7.7:

Listing 7.7. Versiunea a doua de declanşatorului din listingul 7.5

```
    CREATE OR REPLACE TRIGGER trg_exemplare_ins
     BEFORE INSERT ON exemplare
     FOR EACH ROW
    BEGIN
        IF f_NrExemplare (:NEW.isbn) > 3 THEN
            RAISE_APPLICATION_ERROR (-20801,
                    'Nr. exemplarelor acestei carti creste peste 10 !') ;
        ELSE
            /* se incrementeaza numarul exemplarelor pentru ISBN-ul din linia inserata */
            UPDATE titluri SET NrExemplare = NrExemplare + 1        WHERE isbn = :NEW.isbn ;
        END IF ;
    END ;
```

- Pentru modificare - vezi listing 7.8:

Listing 7.8. Declanşatorul de modificare a valorii atributului EXEMPLARE.isbn

```
    CREATE OR REPLACE TRIGGER trg_exemplare_upd
        BEFORE UPDATE OF isbn ON exemplare
        FOR EACH ROW
```

```
BEGIN
    IF f_NrExemplare (:NEW.isbn) > 3 THEN
        RAISE_APPLICATION_ERROR (-20801,
                'Nr. exemplarelor acestei carti creste peste 10 !') ;
    ELSE
        /* se scade cu 1 numarul exemplarelor pentru
                vechiul ISBN (valoarea dinaintea modificarii) */
        UPDATE titluri SET NrExemplare = NrExemplare - 1        WHERE isbn = :OLD.isbn      ;

        /* se incrementeaza numarul exemplarelor pentru          ISBN-ul din linia inserata */
        UPDATE titluri SET NrExemplare = NrExemplare + 1        WHERE isbn = :NEW.isbn ;
    END IF ;
END ;
```

Un avantaj colateral al acestei variante de declanşator, avantaj deloc neglijabil pentru "oracliști", este că se elimină problema "mutanței" tabelei EXEMPLARE.

- Pentru ştergere - vezi listing 7.9:

Listing 7.9. Declanşatorul de ştergere a unei linii din tabela EXEMPLARE

```
CREATE OR REPLACE TRIGGER trg_exemplare_del
    BEFORE DELETE ON exemplare FOR EACH ROW
BEGIN
    /* se scade cu 1 numarul exemplarelor pentru ISBN-ul sters */
    UPDATE titluri SET NrExemplare = NrExemplare - 1 WHERE isbn = :OLD.isbn      ;
END ;
```

Puțini, probabil, vor fi înclinați să accepte că această ultimă soluție ar fi mai indicată decât prima, atâta vreme cât declanşatoarele de inserare/modificare folosesc o funcție care execută fraza `SELECT COUNT...`, apoi, ele însele, conțin comenzi `UPDATE`. Ei bine, odată pusă în aplicare, lucrurile nu sunt aşa de negre. Mai întâi, `SELECT`-ul din funcția `f_NrExemplare` are toate şanşele să se execute destul de rapid, întrucât în EXEMPLARE atributul `ISBN` este cheie străină (atributul părinte fiind `ISBN` din TITLURI), iar la declararea unei asemenea restricții referențiale, orice server de baze de date crează un index. Acest index creşte sensibil viteza atât în cazul interogării `SELECT COUNT...`, cât şi la `UPDATE`-uri. În al doilea rând, frecvența modificării unui ISBN în EXEMPLARE, ca şi ştergerii unei linii în aceeaşi tabelă, este foarte scăzută, astfel încât, statistic privind lucrurile, declanşatorul cel mai important este cel de inserare.

Marele merit al acestei ultime soluții este că o pregăteşte pe a treia, care e mult mai simplă. Folosim atributul `NrExemplare` în TITLURI, atribut modificat prin cele trei declanşatoare ale tabelei EXEMPLARE, însă restricția o implementăm în bază printr-o regulă de validare declarată pentru acest atribut, adică, după sintaxa SQL:

```
ALTER TABLE titluri ADD CONSTRAINT ck_nrexemplare
    CHECK (NrExemplare <= 10) ;
```

Acum cele trei declanşatoare ale tabelei EXEMPLARE se simplifică sensibil, după cum se observă şi în listingul 7.10.

Listing 7.10. Noile declanşatoare ale tabelei EXEMPLARE

```
CREATE OR REPLACE TRIGGER trg_exemplare_ins
 BEFORE INSERT ON exemplare FOR EACH ROW
BEGIN
        /* se incrementeaza numarul exemplarelor pentru ISBN-ul din linia inserata */
 UPDATE titluri SET NrExemplare = NrExemplare + 1
        WHERE isbn = :NEW.isbn ;
END ;
/

CREATE OR REPLACE TRIGGER trg_exemplare_upd
    BEFORE UPDATE OF isbn ON exemplare            FOR EACH ROW
BEGIN
    /* se scade cu 1 numarul exemplarelor pentru vechiul ISBN (valoarea dinaintea modificarii) */
    UPDATE titluri SET NrExemplare = NrExemplare - 1 WHERE isbn = :OLD.isbn      ;

    /* se incrementeaza numarul exemplarelor pentru    ISBN-ul din linia inserata */
    UPDATE titluri SET NrExemplare = NrExemplare + 1            WHERE isbn = :NEW.isbn ;
END ;
/

CREATE OR REPLACE TRIGGER trg_exemplare_del
    BEFORE DELETE ON exemplare FOR EACH ROW
BEGIN
    /* se scade cu 1 numarul exemplarelor pentru ISBN-ul sters */
    UPDATE titluri SET NrExemplare = NrExemplare - 1 WHERE isbn = :OLD.isbn      ;
END ;
```

Fireşte, nici de funcţie nu mai avem nevoie în noile condiţii, iar soluţia are mult mai multe şanse de a fi acceptată.

Atenţie, însă ! Atunci când utilizăm un asemenea atribut redundant, trebuie să fim siguri că modul său de actualizare este bine pus la punct, astfel încât să nu existe nici o modalitate prin care valoarea sa ar fi neconformă cu realitatea din baza de date. Ori, cel mai sigur mecanism este cel al declanşatoarelor. Din păcate, mulţi dezvoltatori folosesc asemenea artificii doar în interfaţă (formulare) sau logica aplicaţiei, situaţii în care actualizarea atributelor calculate poate fi ocolită, cu sau fără bună ştiinţă.

Un caz şi mai interesant

Să luăm în discuţie şi un alt exemplu. În primul paragraf din acest capitol am modificat baza de date dedicată notelor contabile introducând chei surogat, şi, pe baza grafului dependenţelor din figura 7.1, am ajuns la o schemă alcătuită din trei relaţii: NOTE_CONTABILE {IdNotăContabilă, NrNota, Data, ExplicaţiiNotă}, OPERAŢIUNI {IdOperaţiune, NrOp, IdNotăContabilă, ExplicaţiiOp} şi DETALII_OPERAŢIUNI {IdOperaţiune, ContDebitor, ContCreditor, Suma}. Conţinutul acestor relaţii, obţinut prin prelucrarea relaţiei

LINII_ARTICOLE_CONTABILE (figura 4.1) poate fi înțeles din cele câteva înregistrări din figura 7.4 (listingul de populare a celor trei tabele - 7.11 - este disponibil pe pagina web amintită).

NOTE_CONTABILE

IdNotaContabilă	NrNota	Data	ExplicaţiiNotă
1001	5	30.11.2004	...
1002	6	01.12.2004	...

OPERAȚIUNI

IdOperaţiune	NrOp	IdNotaContabilă	ExplicaţiiOp
10001	1	5	...
10002	2	5	...
10003	1	6	...

DETALII_OPERAȚIUNI

IdOperaţiune	ContDebitor	ContCreditor	Suma
10001	300	401	10000000
10001	4426	401	1900000
10002	401	5311	7000000
10002	401	5121	4800000
10003	5121	700	5250000

Figura 7.4. Câteva linii din relaţiile bazei de date CONTABILITATE

O restricție extrem de importantă în contabilite este că în orice înregistrare contabilă compusă *nu pot fi simultan două sau mai multe conturi şi pe debit şi pe credit*. Altfel spus, într-o operațiune contabilă, există fie un cont debitor şi unul sau mai multe conturi creditoare, fie un singur cont creditor şi unul sau mai multe debitoare !

Soluția de început a acestei probleme ține de valorificarea declanşatoarelor de inserare şi modificare. La fiecare inserare în DETALII_OPERAȚIUNI, declanşatorul trebuie să facă o verificare de genul celei din listingul 7.12.

Listing 7.12. Verificarea numărului de conturi pe debit şi pe credit la inserare

```
CREATE OR REPLACE TRIGGER trg_do_ins
    BEFORE INSERT ON detalii_operatiuni FOR EACH ROW
DECLARE
    v_nrcontdebit NUMBER(4) := 0 ;
    v_nrcontcredit NUMBER(4) := 0 ;
BEGIN
    SELECT COUNT(DISTINCT ContDebitor), COUNT (DISTINCT ContCreditor)
    INTO v_nrcontdebit, v_nrcontcredit
    FROM
        (SELECT ContDebitor, ContCreditor
           FROM detalii_operatiuni
           WHERE idoperatiune = :NEW.idoperatiune
           UNION
                SELECT :NEW.ContDebitor, :NEW.ContCreditor
                FROM dual
                ) T ;

    IF v_nrcontdebit > 1 AND v_nrcontcredit > 1 THEN
```

```
            RAISE_APPLICATION_ERROR (-20852,
                'Nu pot fi simultan mai multe conturi si pe debit si pe credit') ;
        END IF ;
    END ;
```

A fost nevoie de un mic artificiu pentru că SELECT-ul nu ar fi luat în calcul înregistrare ce tocmai se adaugă în tabelă (înregistrare care, în fond, determină lansarea declanşatorului): cele două conturi din noua înregistre sunt "lipite" celor deja introduse folosind reuniunea (şi tabela DUAL).

Nici la modificarea aceleaşi tabele lucrurile nu sunt mai tihnite. Pentru a evita problema mutanței, şi pornind de la premisa că printr-un UPDATE nu se modifică niciodată simultan conturile si identificatoarele mai multor operațiuni, soluția Oracle recurge la triada: variabilă publică (pachet), declanşator de actualizare la nivel de linie, declanşator de actualizare la nivel de comandă - vezi listingul 7.13.

Listing 7.13. Verificarea numărului de conturi pe debit şi pe credit la modificare

```
CREATE OR REPLACE PACKAGE pac_trg IS
    v_idoper operatiuni.idoperatiune%TYPE ;
END ;
/

CREATE OR REPLACE TRIGGER trg_do_upd1
    BEFORE UPDATE OF IdOperatiune, ContDebitor, ContCreditor
    ON detalii_operatiuni FOR EACH ROW
BEGIN
    pac_trg.v_idoper := :NEW.idoperatiune ;
END ;
/

CREATE OR REPLACE TRIGGER trg_do_upd2
    AFTER UPDATE OF IdOperatiune, ContDebitor, ContCreditor
    ON detalii_operatiuni
DECLARE
    v_nrcontdebit NUMBER(4) := 0 ;
    v_nrcontcredit NUMBER(4) := 0 ;
BEGIN
    SELECT COUNT(DISTINCT ContDebitor), COUNT (DISTINCT ContCreditor)
    INTO v_nrcontdebit, v_nrcontcredit FROM detalii_operatiuni
    WHERE idoperatiune = pac_trg.v_idoper ;

    IF v_nrcontdebit > 1 AND v_nrcontcredit > 1 THEN
        RAISE_APPLICATION_ERROR (-20852,
                'Nu pot fi simultan mai multe conturi si pe debit si pe credit') ;
    END IF ;
END ;
/
```

Nu foarte multe SGBD-uri sunt dispuse să ofere suport pentru asemenea gen de artificii, aşa că ne bate gândul să recurgem la unul sau mai multe atribute redundante care să mai limpezească din ape. Aşadar, introducem în OPERAȚIUNI două atribute, numite `NrConturiDebitoare` şi `NrConturiCreditoare`, care să furnizeze numărul conturilor din debitul şi din creditul fiecărei operațiuni

contabile. Astfel, relația ar avea structura: OPERAȚIUNI {IdOperaţiune, IdNotăContabilă, ExplicaţiiOp, NrConturiDebitoare, NrConturiCreditoare}. Curând ne dăm seama că avem o problemă, întrucât cele două atribute trebuie să ia în calcul fiecare cont o singură dată. Să fim mai expliciți. Prima operațiune din nota 101, cea care are identificatorul 10001, se scrie în "limbaj" contabil astfel:

%	401		11800000
300		10000000	
4426		1800000	

Operațiunii îi corespund primele două linii din DETALII_OPERAȚIUNI, deci stocarea se face transformând înregistrarea după următorul calapod:

Debit	Credit	Suma
300	401	10000000
4426	401	1800000

Acest mod de stocare a înregistrărilor este fundamental pentru obținerea documentelor contabile ce prelucrează notele contabile: fişe de cont, cartea-mare, balanțe de verificare. Adăugând cele două atribute în OPERAȚIUNI, pe prima linie a acestei tabele (corespunzătoare înregistrării contabile pe care o discutăm) valoarea atributului NrConturiDebitoare este 2, iar cea a NrConturiCreditoare este 1. Declanşatorul de inserare al tabelei DETALII_OPERAȚIUNI nu trebuie să incrementeze pur şi simplu cele două atribute, ci, în mod normal, numai unul, deoarece celălalt se repetă. Bine, dar cum ştie declanşatorul care cont se repetă, pe debit sau pe credit, fără a face SELECT-ul acela impresionant ?

Îngroşând gluma, mai introducem două conturi, UnContDebitor şi UnContCreditor care vor avea valoarea inițială (implicită) NULL:

```
ALTER TABLE operaţiuni ADD NrConturiDebitoare NUMBER (3) ;
ALTER TABLE operaţiuni ADD NrConturiCreditoare NUMBER (3) ;
ALTER TABLE operaţiuni ADD UnContDebitor VARCHAR(14) ;
ALTER TABLE operaţiuni ADD UnContCreditor VARCHAR(14) ;
```

Valoarea implicită a acestor patru atribute va fi NULL. La prima introducere a unei corespondențe contabile pentru o operațiune, adică la prima inserare a unei linii în DETALII_OPERAȚIUNI, NrConturiDebitoare şi NrConturiCreditoare vor primi valoarea 1, UnContDebitor va lua valoarea :NEW.ContDebitor (valoarea de pe linia inserată a atributului ContDebitor), iar UnContCreditor pe :NEW.ContCreditor. La a doua inserare a unei corespondențe *cont debitor - cont creditor* pentru aceeaşi notă contabilă, atributul NrConturiDebitoare trebuie incrementat cu 1 numai dacă valoarea :NEW.ContDebitor este diferită de cea a UnContDebitor, iar NrConturiCreditoare trebuie incrementat cu 1 doar dacă valoarea :NEW.ContCreditor este diferită de cea a atributului UnContCreditor (să nu spuneți că n-ați înțeles !).

Cu un pic de imaginație, reducem întreg declanşatorul pentru inserare în tabela DETALII_OPERAȚIUNI la o singură, dar generoasă, comandă UPDATE - vezi listing 7.14

Listing 7.14. Noul declanşator de inserare ce actualizează atributele "redundante"

```
CREATE OR REPLACE TRIGGER trg_do_ins
    AFTER INSERT ON detalii_operatiuni FOR EACH ROW
BEGIN
    UPDATE operatiuni
    SET  nrconturidebitoare =  NVL(nrconturidebitoare, 1) +
         CASE WHEN :NEW.contdebitor <> NVL(uncontdebitor,:NEW.contdebitor)
         THEN 1 ELSE 0 END,
         nrconturicreditoare =  NVL(nrconturicreditoare, 1) + CASE
         WHEN :NEW.contcreditor <> NVL(uncontcreditor,:NEW.contcreditor)
              THEN 1 ELSE 0 END,
        uncontdebitor = :NEW.contdebitor,
         uncontcreditor = :NEW.contcreditor
  WHERE idoperatiune = :NEW.idoperatiune ;
END ;
```

Acum putem declara liniştiți restricția sub forma regulii de validare:

```
ALTER TABLE operaţiuni ADD CONSTRAINT ck_nrconturi
       CHECK ( NOT (NVL(nrconturidebitoare,0) > 1 AND
                    NVL(nrconturicreditoare,0) > 1) ) ;
```

În noile condiții, şi operațiunile derulate la modificări survenite în tabela DETALII_OPERAȚIUNI se simplifică sensibil. Putem renunța la pachet şi unul dintre declanşatoare, iar noua formă trigger-ului de modificare este cea din listing 7.15.

Listing 7.15. Noul declanşator de modificare în DETALII_OPERAŢIUNI

```
DROP TRIGGER trg_do_upd1 ;
DROP TRIGGER trg_do_upd2 ;
DROP PACKAGE pac_trg ;
/

CREATE OR REPLACE TRIGGER trg_do_upd
    AFTER UPDATE OF IdOperatiune, ContDebitor, ContCreditor
    ON detalii_operatiuni FOR EACH ROW
BEGIN
    -- mai intii, se fac modificarile pentru vechile valori
    UPDATE operatiuni
    SET  nrconturidebitoare = nrconturidebitoare -
         CASE WHEN nrconturidebitoare > 1 THEN 1      ELSE 0 END,
         nrconturicreditoare =  nrconturicreditoare -
         CASE WHEN nrconturicreditoare > 1 THEN 1 ELSE 0 END
    WHERE idoperatiune = :OLD.idoperatiune ;

    -- apoi, se fac modificarile pentru noile valori
    UPDATE operatiuni
    SET  nrconturidebitoare =  nrconturidebitoare +
         CASE WHEN :NEW.contdebitor <> uncontdebitor THEN 1              ELSE 0 END,
              nrconturicreditoare =  nrconturicreditoare +
         CASE WHEN :NEW.contcreditor <> uncontcreditor THEN 1 ELSE 0 END,
```

```
            uncontdebitor = :NEW.contdebitor,
            uncontcreditor = :NEW.contcreditor
        WHERE idoperatiune = :NEW.idoperatiune ;
    END ;
```

Listingul 7.16 conţine declanşatorul de ştergere care asigură decrementarea numărului de conturi debitoare şi a celui de conturi creditoare şi, în caz că se şterge ultima linie dintr-o operaţiune notabilă, NULL-izarea atributelor `UnContDebitor` şi `UnContCreditor`.

Listing 7.16. Declanşatorul de ştergere în DETALII_OPERAŢIUNI

```
CREATE OR REPLACE TRIGGER trg_do_del
        BEFORE DELETE ON detalii_operatiuni FOR EACH ROW
BEGIN
    UPDATE operatiuni
        SET  nrconturidebitoare = nrconturidebitoare -   CASE WHEN nrconturidebitoare > 1
                    OR nrconturidebitoare+nrconturicreditoare=2 THEN 1 ELSE 0 END,
             nrconturicreditoare =  nrconturicreditoare - CASE WHEN nrconturicreditoare > 1
                    OR nrconturidebitoare+nrconturicreditoare=2 THEN 1 ELSE 0 END,
             uncontdebitor = CASE WHEN nrconturidebitoare+nrconturicreditoare=2
                            THEN NULL ELSE uncontdebitor END,
             uncontcreditor = CASE WHEN nrconturidebitoare+nrconturicreditoare=2
                            THEN NULL ELSE uncontcreditor END
        WHERE idoperatiune = :OLD.idoperatiune ;
END ;
```

7.2.3. Restricţii şi relaţii noi

În paragraful 7.2.1 am enumerat câteva posibile reguli de validare la nivel de atribut, printre care şi una pentru relaţia STUDENŢI prin care, în primii doi ani, studenţii urmează un trunchi comun de discipline, iar începând cu anul 3 de studiu îşi aleg specializarea, astfel încât atributul `Spec` trebuie să ia o valoarea pe domeniul : (*Contabilitate şi informatică de gestiune*, *Economie generală*, *Finanţe-Bănci* etc.). Chiar dacă priveşte atributul `Spec`, declararea acestei restricţii printr-o clauză CHECK este destul de anevoioasă:

```
CREATE TABLE studenţi (
   matricol ...
   ...
   Spec VARCHAR(50) NOT NULL CHECK
   (Spec IN 'Contabilitate şi informatică de gestiune', '',
   'Informatică economică', ...)
       ...
   );
```

Lista celor 10 specializări, plus *Truchi comun*, este una destul de mare, iar dacă se ia decizia implementării aplicaţiei la nivelul unei universităţi precum Al. I. Cuza din Iaşi, ale cărei facultăţi patronează zeci de specializări, scrierea clauzei CHECK devine o sarcină pe care o s-o execute doar cel care nu va avea cui să i-o paseze. Ceea ce vă propunem este o soluţie cunoscută multor practicieni: constituirea unei

relaţii dedicate specializărilor (SPECIALIZĂRI), urmând ca regula să fie ulterior implementată printr-o restricţie referenţială STUDENŢI-SPECIALIZĂRI.

Dacă dorim să păstrăm acelaşi număr de atribute, creăm o tabelă cu un singur atribut:

```
CREATE TABLE specializări (
   spec VARCHAR(50) NOT NULL PRIMARY KEY
   );
```

şi apoi

```
   CREATE TABLE studenţi (
       matricol ...
       ...
       ,spec VARCHAR(50) NOT NULL
              REFERENCES specializări (spec)
       ...
       );
```

A doua variantă este mai nimerită, chiar dacă avem un atribut suplimentar de tip cheie surogat prin care identificăm fiecare specializare:

```
CREATE TABLE specializări (
   idspec NUMBER(4) NOT NULL PRIMARY KEY
   ,spec VARCHAR(50)
   );

CREATE TABLE studenţi (
   matricol ...
   ...
   ,idspec NUMBER(4) NOT NULL
       REFERENCES specializări (idspec)
   ...
   );
```

Preţul ultimelor două soluţii ţine de complexitatea presupusă de gestionarea unei chei străine, precum şi timpul mai mare necesar interogărilor şi verificărilor ce decurg din restricţia referenţială. Marele avantaj ţine de flexibilitate, orice specializare nouă sau modificare a denumirii unei specializării neatrăgând modificarea schemei bazei, fiind suficientă doar o comandă DML.

Întrucât folosirea clauzei CHECK prezintă avantajul net al vitezei şi simplităţii (să nu uităm că la restricţiile referenţiale apar proleme de genul DELETE RESTRICT/CASCADE şi mai ales UPDATE CASCADE/RESTRICT), recomandabil este ca, atunci când lista valorilor acceptate nu e prea lungă, iar acestea (valorile) sunt constante, să recurgem la o regulă de validare la nivel de atribut, în timp ce atunci când valorile sunt mai numeroase şi/sau mai susceptibile a fi modificate, să creăm o nouă relaţie (şi, implicit, o nouă restricţie referenţială).

Păstrând discuţia în baza de date EXAMENE, să vedem ce implicaţii ar avea o restricţie de genul: *fiecare profesor poate preda oricâte discipline, dar numai la specializările care îi sunt repartizate*. Există, prin urmare, profesori ce predau la o

singură specializare, profesori ce predau la două specializări etc. Relația DISC_SPEC_PROFI {CodDisc, An, Modul, IdSpec, CodProf} este cea "împricinată", însă aici implementarea acestei restricții printr-o regulă de validare la nivel de înregistrare este cu totul improbabilă, aşa că mai realist ar fi să creăm o relație de genul: PROFI_SPECIALIZĂRI {An, Modul, IdSpec, CodProf}. Interesant că, spre deosebire de exemplul precedent, cheia străină este compusă rău de tot:

```
CREATE TABLE profi_specializări (
    an NUMBER(1) NOT NULL
   ,modul VARCHAR(20) NOT NULL
   ,idspec NUMBER(5) NOT NULL REFERENCES specializari(idspec)
   ,codprof NUMBER(5) NOT NULL REFERENCES profesori (codprof)
   ,PRIMARY KEY (an, modul, spec, codprof)
   );

CREATE TABLE disc_spec_profi (
    coddisc CHAR(6) NOT NULL REFERENCES discipline (coddisc)
   ,an NUMBER(1) NOT NULL
   ,modul VARCHAR(20) NOT NULL
   ,idspec NUMBER(5) NOT NULL REFERENCES specializari(idspec)
   ,codprof NUMBER(5) NOT NULL REFERENCES profesori (codprof)
   ,PRIMARY KEY (coddisc, an, modul, spec)
   ,FOREIGN KEY (an, modul, spec, codprof)
       REFERENCES profi_specializări(an, modul, idspec, codprof)
   );
```

Oricărei alocări a unei specializări unui profesor îi va corespunde o linie în tabela PROFI_SPECIALIZĂRI. Forma finală al script-ului de creare a tabelelor poate de descărcat de pe pagina web cunoscută (listing 7.17).

7.3. Dependențe de incluziune

Este greu de explicat motivl pentru care toate cărțile majore care tratează problema normalizării şi, în general, proiectarea bazelor de date, sunt atât de discrete în materie de dependențe de incluziune. Nu-i vorbă că noțiunea ar fi din cale-afară de dificilă, ci mai deagrabă că în aplicațiile economice, chiar de calibru mediu, este aproape imposibil să nu se manifeste un asemenea gen de dependență.

Într-o exprimare lejeră, între două atribute X şi Y există o dependență de incluziune (DI) dacă şi numai dacă orice valoare a lui X este obligatoriu şi valoare a lui Y şi se notează simplu $X \subseteq Y$. Mai general, o DI semnifică faptul că proiecția pe *m* atribute date ale relației R este un subset al proiecției pe *m* atribute date din relația S. De aici şi caracterizarea DI ca dependențe interrelații[9]. Definiția însă nu obligă ca R şi S să fie neapărat distincte, deci DI se poate institui între atribute şi grupe de atribute ale aceleaşi relații. După Casanova s.a., DI semnalizează relațiile

[9] [Fagin 81]

de tip *este-un/o*[10], iar Sciore este şi mai limpede când scrie că folosim DI pentru a arăta că două atribute din relaţii diferite se referă la acelaşi lucru[11]. Am putea amenda afirmaţia lui Sciore prin precizarea faptului că atributele nu trebuie neapărat să fie din relaţii diferite.

Cel mai frecvent exemplu de DI este `Manager` ⊆ `Angajat`[12], care înseamnă că orice manager este un angajat al companiei. Graful din figura 7.5 ilustrează o situaţie comună în firmele de pretutindeni: fiecare angajat al unei întreprinderi este arondat unui compartiment (*Producţie*, *Marketing*, *Personal*, *Financiar*, *Contabilitate* etc.); fiecare compartiment are un singur şef; fiecare şef este, şi el, angajat, chiar dacă mai uită uneori de lucrul acesta.

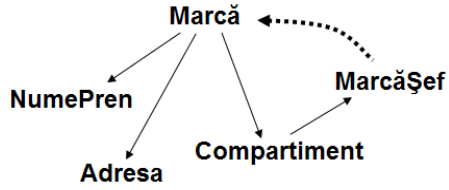

Figura 7.5. O dependenţă de incluziune

Dependenţa de joncţiune reprezentată prin săgeata punctată este, deci, `MarcăŞef` ⊆ `Marcă`. Decuparea relaţiilor din graf nu ridică mari probleme. Avem două surse de DF, prin urmare cele două relaţii vor fi: PERSONAL {`Marcă`, `NumePren`, `Adresă`, `Compartiment`} şi COMPARTIMENTE {`Compartiment`, `MarcăŞef`}. Dependenţa de incluziune va fi încorporată în schemă sub forma restricţiei referenţiale dintre `COMPARTIMENTE.MarcăŞef` şi `PERSONAL.Marcă`.

Frunze şi conturi

Ne îndepărtăm de exemplele consacrate, apelând la baza de date CONTABILITATE, pe care, după discuţia din paragraful 7.2.2, am adus-o la schema: NOTE_CONTABILE {`IdNotăContabilă`, `Data`, `ExplicaţiiNotă`}, OPERAŢIUNI {`IdOperaţiune`, `IdNotăContabilă`, `ExplicaţiiOp`, `NrConturiDebitoare`, `NrConturiCreditoare`, `UnContDebitor`, `UnContCreditor`} şi DETALII_OPERAŢIUNI {`IdOperaţiune`, `ContDebitor`, `ContCreditor`, `Suma`}. Ei bine, a sosit momentul pentru a introduce una din cele mai importante restricţii ale unei aplicaţii de contabilitate generală: conturile care apar în operaţiunile contabile nu trebuie să aibă conturi sintetice/analitice subordonate !

Să începem explicaţiile mai din "amonte". Pentru reflectarea elementelor patrimoniale, a capitalului, datoriilor şi obligaţiilor, contabilitatea foloseşte conturi

[10] [Casanova s.a.82]

[11] [Sciore83]

[12] [Fagin81], [Casanova s.a.82], [Johnson & Klug82], [Sciore83]

organizate în serii, clase şi grupe[13]. De exemplu, în seria 1 sunt incluse conturile organice de bilanţ, în seria 2 conturile de procese, în seria 3 conturile de rezultate, în seria 4 conturile în afara bilanţului (extrapatrimoniale), iar în seria 5 conturile privind circuitul contabilităţii manageriale. Seria 1 conţine 7 clase, pentru: capitaluri, active imobilizate, stocuri, terţi, trezorerie, regularizare şi conturi rectificative. Fiecare clasă are una sau mai multe serii de conturi. Astfel, clasa *Stocuri* are şapte grupe: materii şi materiale, obiecte de inventar; producţie în curs; produse; stocuri la terţi; animale; mărfuri; ambalaje. În fiecare grupă sunt conturile propriuzise care pot fi sintetice de ordinul 1 (alcătuite din trei cifre) sau de ordinul 2 (patru cifre). Fiecare sintetic de ordinul 2 aparţine unui sintetic de ordinul 1. Tabelul 7.1 indică o porţiune din planul de conturi al firmei X. Orice tip cont sintetic poate fi descompus pe conturi analitice, în funcţie de specificul şi interesele întreprinderii.

Tabel 7.1. Extras din planul de conturi al unei firme

Simbol cont	TipCont	Denumire cont
Clasa: Conturi de CAPITALURI		
...		
101	pasiv	Capital social
1011	pasiv	Capital social subscris ne-vărsat
1012	pasiv	Capital social subscris vărsat
...		
211	activ	Terenuri şi amenajări
2111	activ	Terenuri
2112	activ	Amenajări
....		
301	activ	Materii prime
301.01	activ	Nisip
301.02	activ	Ciment
301.03	activ	Var
301.09	activ	Alte materii prime
...		
308	bifuncţional	Diferenţe de preţ la materii prime
...		
401	pasiv	Furnizori
401.01	pasiv	Furnizor A
401.02	pasiv	Furnizor B
401.99	pasiv	Alţi furnizori
...		
428	bifuncţional	Alte creanţe şi datorii faţă de salariaţi
4281	activ	Alte creanţe faţă de salariaţi
4282	pasiv	Alte datorii faţă de salariaţi
...		
411	activ	Clienţi
...		
4426	activ	TVA deductibilă
4427	pasiv	TVA colectată
...		
601	activ	Cheltuieli cu materii prime

[13] Vezi, spre exemplu, Horomnea, E. - *Bazele contabilităţii. Concepte, aplicaţii, lexicon*, Editura Sedcom Libris, Iaşi, 2004, pp. 178-180

601.01	activ	Cheltuieli cu nisipul
601.02	activ	Cheltuieli cu cimentul
601.03	activ	Cheltuieli cu varul
601.09	activ	Cheltuieli cu alte materii prime
...		

Astfel, contul 301 - *Materii prime* este decompus pe patru analitice. Primele trei, 301.01, 301.02 şi 301.03, reprezintă cele mai importante materii prime pentru firmă, iar ultimul, 301.09 le grupează pe toate celelalte. Mai sunt descompuse pe analitice conturile de *furnizori* (401) şi *cheltuieli cu materiile prime* (600). Interesant că un cont sintetic de gradul I bifuncțional (428) poate avea un "subordonat" de activ (4281), şi un altul de pasiv (4282) !

Din moment ce am devenit mai riguroşi, este evident că, practic, putem introduce în schemă atributele `SimbolCont`, `TipCont`, `DenumireCont`, iar între `ContDebitor` şi `SimbolCont`, pe de o parte, şi `ContCreditor` şi `SimbolCont`, pe de altă parte, avem de-a face cu o dependență de incluziune. Graful dependențelor ia forma din figura 7.6.

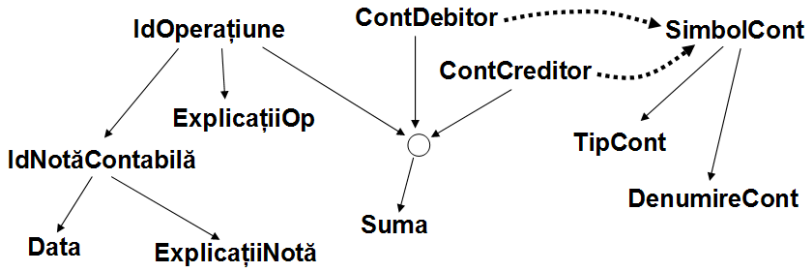

Figura 7.6. Dependenţele funcţionale şi de incluziune pentru BD CONTABILITATE

Relația nouă ce rezultă din graf este evidentă: PLAN_CONTURI {`SimbolCont`, `TipCont`, `DenumireCont`}, iar cele două dependențe de incluziune se vor materializa la implementarea bazei în două restricții referențiale (vezi listingul 7.18 de pe pagina web).

Revenind la restricția pe care am definit-o şi caracterizat-o drept fundamentală într-o aplicație destinată contabilității generale, dacă avem în vedere structura arborescentă a planului de conturi, putem reformula cerința astfel: orice cont debitor şi creditor ce apare într-o înregistrare contabilă trebuie să fie "frunză", adică nu poate avea conturi (sintetice de ordinul 2 sau analitice) subordonate. Aşadar, declanşatoarele de inserare şi modificare ale tabelei DETALII_OPERAȚIUNI trebuie să verifice calitatea de frunză pentru `ContDebitor` şi `ContCreditor`.

Continuăm nu doar declanşatorul de inserare pe care l-am conturat inițial în paragraful 7.2.2 - vezi listing 7.19.

Listing 7.19. Un declanşator de inserare nou-nouţ pentru tabela DETALII_OPERAŢIUNI

```
CREATE OR REPLACE TRIGGER trg_do_ins
   AFTER INSERT ON detalii_operatiuni FOR EACH ROW
DECLARE
```

```
   v_nr NUMBER(3) := 0 ;
BEGIN
   -- se numara câte conturi încep cu simbolurile contului
   -- debitor, pentru a vedea daca acesta se descompune
   SELECT COUNT(*) INTO v_nr FROM plan_conturi
   WHERE SUBSTR(SimbolCont,1,LENGTH(:NEW.ContDebitor))
     = :NEW.ContDebitor ;
   IF v_nr > 1 THEN
      RAISE_APPLICATION_ERROR(-20194,
        'EROARE ! ContDebitor nu este FRUNZA !!!') ;
   END IF ;

   -- se numara câte conturi încep cu simbolurile contului
   -- creditor, pentru a vedea daca acesta se descompune
   SELECT COUNT(*) INTO v_nr FROM plan_conturi
   WHERE SUBSTR(SimbolCont,1,LENGTH(:NEW.ContCreditor))
     = :NEW.ContCreditor ;
   IF v_nr > 1 THEN
      RAISE_APPLICATION_ERROR(-20195,
        'EROARE ! ContCreditor nu este FRUNZA !!!') ;
   END IF ;

   UPDATE operatiuni
   SET  nrconturidebitoare =  NVL(nrconturidebitoare, 1) +
         CASE WHEN :NEW.contdebitor <>
                    NVL(uncontdebitor,:NEW.contdebitor)
         THEN 1 ELSE 0 END,
         nrconturicreditoare =  NVL(nrconturicreditoare, 1) +
        CASE WHEN :NEW.contcreditor <>
                   NVL(uncontcreditor,:NEW.contcreditor)
          THEN 1 ELSE 0 END,
        uncontdebitor = :NEW.contdebitor,
        uncontcreditor = :NEW.contcreditor
     WHERE idoperatiune = :NEW.idoperatiune ;
END ;
```

Rezolvarea poate fi ameliorată, dacă de gândim că fiecare inserare a unei linii în DETALII_OPERAȚIUNI atrage după sine două rânduri de scanare (pentru numărare) a întregului plan de conturi. Avem deja o oarecare experiență în lucrul cu atribute redundate, aşa încât cel mai la îndemână ar fi să adăugăm în PLAN_CONTURI un câmp de tip BOOLEAN (logic) pe care să-l numim chiar `EsteFrunză`, astfel încât noua structură a tabelei va fi PLAN_CONTURI {`SimbolCont`, `TipCont`, `DenumireCont`, `EsteFrunză`}. Soluția Oracle este însă:

```
ALTER TABLE plan_conturi ADD (EsteFrunza CHAR(1)
    DEFAULT 'D' CHECK (EsteFrunza IN ('D', 'N'))) ;
```

Deoarece, din păcate, în Oracle un atribut nu poate avea tipul BOOLEAN, aşa ca am recurs la tipul CHAR(1), limitând valorile la 'D' (da) şi 'N' (nu).

Declanşatoarele tabelei vor "veghea" la corectitudinea acestui atribut. Astfel, când se inserează o linie în PLAN_CONTURI se verifică dacă noul cont *nu* este este sintetic de ordinul II sau analitic. Dacă da (ex. 301.01), se verifică dacă superiorul

său (301 în exemplul nostru) era "frunză" înainte de inserare. În caz că există măcar o înregistrare contabilă în care "superiorul" apare pe debit sau pe credit, inserarea în planul de conturi este interzisă, întrucât este imposibil de revenit asupra notelor contabile deja existente şi a le "re-conta" (vezi listing 7.20).

Listing 7.20. Declanşatorul de inserare în PLAN_CONTURI

```
CREATE OR REPLACE TRIGGER trg_pc_ins
    BEFORE INSERT ON plan_conturi FOR EACH ROW
DECLARE
   v_parinte VARCHAR(15) ;
   v_gata BOOLEAN := FALSE ;
   v_nr NUMBER(3) := 0 ;
BEGIN
   IF LENGTH(:NEW.SimbolCont) > 3 THEN
      --contul este sintetic de ordin 2 sau analitic

      -- se încearca gasirea parintelui, taind câte un caracter
      v_parinte := :NEW.SimbolCont ;
      WHILE v_gata=FALSE AND LENGTH(v_parinte)>3 LOOP
         v_parinte := SUBSTR(v_parinte,1, LENGTH(v_parinte) - 1);
         SELECT COUNT(*) INTO v_nr FROM plan_conturi
         WHERE SimbolCont=v_parinte ;
         IF NVL(v_nr,0) = 1 THEN
               v_gata := TRUE ;
               EXIT ;
         END IF ;
      END LOOP ;
      IF v_gata THEN
         -- exista un parinte pentru noul cont
         -- testam daca parintele este frunza
         SELECT COUNT(ContDebitor)+ COUNT(ContCreditor)
         INTO v_nr FROM detalii_operatiuni
         WHERE ContDebitor=v_parinte OR ContCreditor=v_parinte ;
         IF NVL(v_nr,0) >= 1 THEN
               RAISE_APPLICATION_ERROR (-20196,
               'EROARE ! Contul parinte apare deja in operatiuni,'||
               ' iar noul cont nu mai poate fi adaugat !!!') ;
         END IF ;
         UPDATE plan_conturi SET EsteFrunza = 'N'
         WHERE SimbolCont = v_parinte ;
      END IF;
   END IF ;
   :NEW.EsteFrunza := 'D' ;
END ;
```

Acum ştim că, atunci când adăugăm un cont, se actualizează automat atributul `EsteFrunză` pentru contul părinte. Pentru asigurarea corectitudinii valorilor acestui atribut, trebuie realizate şi declanşatoarele modificare şi ştergere.

Trecem acum la declanşatoarele tabelei DETALII_OPERAŢIUNI. Mai întâi, creăm o funcţie (listing 7.21) care primeşte drept parametru simbolul unui cont şi returnează valoarea atributului `EsteFrunză` pentru contul respectiv.

Listing 7.21. Funcţia PL/SQL F_ESTE_FRUNZA

```
CREATE OR REPLACE FUNCTION f_este_frunza (simbol_cont VARCHAR2)
        RETURN BOOLEAN
AS
        v_frunza CHAR(1)  ;
BEGIN
    SELECT EsteFrunza INTO v_frunza FROM plan_conturi
     WHERE SimbolCont = simbol_cont ;
    IF v_frunza = 'D' THEN
        RETURN TRUE ;
    ELSE
        RETURN FALSE ;
    END IF ;
END ;
```

Sarcina declanşatoarelor se simplifică sensibil. Iată-l pe cel de inserare (listing 7.22).

Listing 7.22. Noul declanşator de inserare în DETALII_OPERAŢIUNI

```
CREATE OR REPLACE TRIGGER trg_do_ins
    AFTER INSERT ON detalii_operatiuni FOR EACH ROW
DECLARE
   v_nr NUMBER(3) := 0 ;
BEGIN
   IF f_este_frunza (:NEW.ContDebitor)=FALSE OR
              f_este_frunza (:NEW.ContCreditor)=FALSE THEN
      -- EROARE! Cel putin unul dintre conturile
      -- de pe debit sau credit nu este FRUNZA !!!
     RAISE_APPLICATION_ERROR(-20194,
       'EROARE ! Contul de pe debit sau credit nu este FRUNZA !!!') ;
   END IF ;
   UPDATE operatiuni
   SET  nrconturidebitoare =  NVL(nrconturidebitoare, 1) +
      CASE WHEN :NEW.contdebitor <>
               NVL(uncontdebitor,:NEW.contdebitor)
          THEN 1      ELSE 0 END,
          nrconturicreditoare =  NVL(nrconturicreditoare, 1) + CASE
          WHEN :NEW.contcreditor <>
               NVL(uncontcreditor,:NEW.contcreditor)
           THEN 1 ELSE 0 END,
        uncontdebitor = :NEW.contdebitor,
        uncontcreditor = :NEW.contcreditor
     WHERE idoperatiune = :NEW.idoperatiune ;
END ;
```

Proiecte vechi şi noi

Am recurs la un exemplu privind proiectele derulate într-o firmă cu ocazia paragrafului 5.4, pentru a ilustra forma normalizată Boyce-Codd. Acum, însă, ne apropiem de realitatea dintr-o companie, luând în calcul următoarele elemente:

- un angajat, identificat prin `Marcă` este inclus în schema unui compartiment funcțional (*Contabilitate*, *Marketing* etc.);
- orice compartiment are un singur şef (`MarcăŞef`);

- firma, datorită specificului activităţii sale, lucrează pe bază de proiecte; fiecare proiect are un identificator (`IdProiect`), demarează la o anumită dată (`DatăProiect`) şi are un termen de realizare (`Termen`);
- un proiect este alcătuit din una sau mai multe activităţi (`IdActivitate`), o activitate având o titulatură (`DenActivit`) şi o scurtă descriere (`DescActivit`);
- orice angajat este capabil să desfăşoare una sau mai multe activităţi;
- un angajat poate lucra la mai multe proiecte;
- la un proiect sunt angajate una sau mai multe persoane;
- o activitate poate fi inclusă în unul sau mai multe proiecte;
- în cadrul unui proiect, fiecare activitate începe la o anumită dată (`Data-Început`), se finalizează la o altă dată (`DataFinal`) şi constă într-o serie de operaţii, responsabilităţi şi rezultate (`Detalii`);
- o aceeaşi persoană poate desfăşura, în acelaşi proiect, una, două sau mai multe activităţi;
- o activitate într-un proiect poate fi desfăşurată de mai multe persoane.

Dependenţele corepunzătoare acestor cerinţe pot fi reprezentate sub formă de graf ca în figura 7.7.

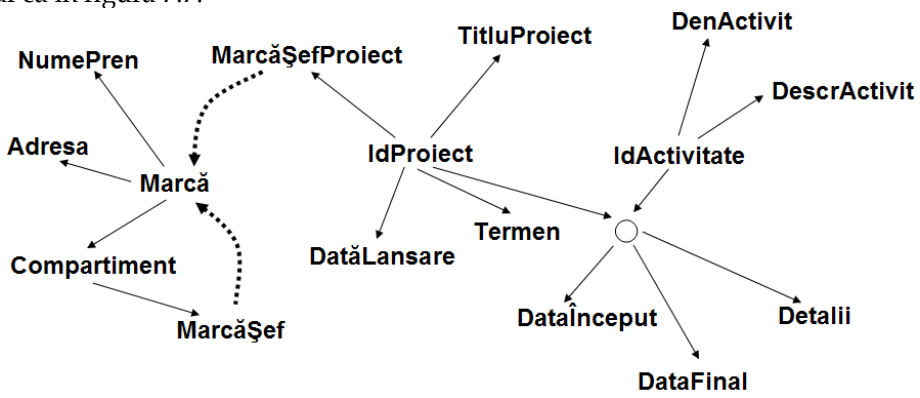

Figura 7.7. Dependenţele bazei de date PROIECTE

Paragraful 5.5.3 ne-a introdus într-una dintre cele mai delicate probleme ale normalizării, şi anume imposibilitatea de a prelua sub formă de dependenţe anumite relaţii semantice. Şi graful de mai sus are câteva asemenea dureri de cap. Mai precis, există o serie de informaţii care nu pot fi furnizate de schema construită pe baza decupării relaţiilor:

- care sunt activităţile pe care le poate desfăşura o persoană ?
- în ce proiecte a fost implicat un angajat ?
- care sunt persoanele implicate într-un anumit proiect ?
- care este activitatea, sau activităţile, desfăşurate de o anumită persoană într-un proiect dat ?
- care sunt persoanele care au desfăşurat o anumită activitate într-un anumit proiect ?

Sugestia din paragraful 5.5.3 era de a introduce în schema bazei şi alte atribute care să "fixeze" toate legăturile neprinse în dependenţe. Astfel, un angajat poate fi "legat" de o anumită activitate pe care o poate desfăşura printr-un atribut nou, `Competenţe`, care să sugereze abilităţile şi experienţa angajatului în privinţa activităţii cu pricina: (`Marcă`, `IdActivitate`) ⟶ `Competenţe`. Pe de altă parte, participarea unui angajat într-o activitate din cadrul unui proiect poate fi descrisă cu ajutorul a trei atribute - `Atribuţii`, `Probleme` şi `Rezultate` - care indică ce sarcini a avut angajatul legat de acea activitate din cadrul proiectului, ce probleme a întâmpinat şi care au fost rezultatele muncii sale. Dependenţele sunt evidente: (`Marcă`, `IdProiect`, `IdActivitate`) ⟶ `Atribuţii`, (`Marcă`, `IdProiect`, `IdActivitate`) ⟶ `Probleme` şi (`Marcă`, `IdProiect`, `IdActivitate`) ⟶ `Rezultate`. Noul graf s-ar prezenta ca în figura 7.8, noile atribute fiind scrise "aplecat" (italic).

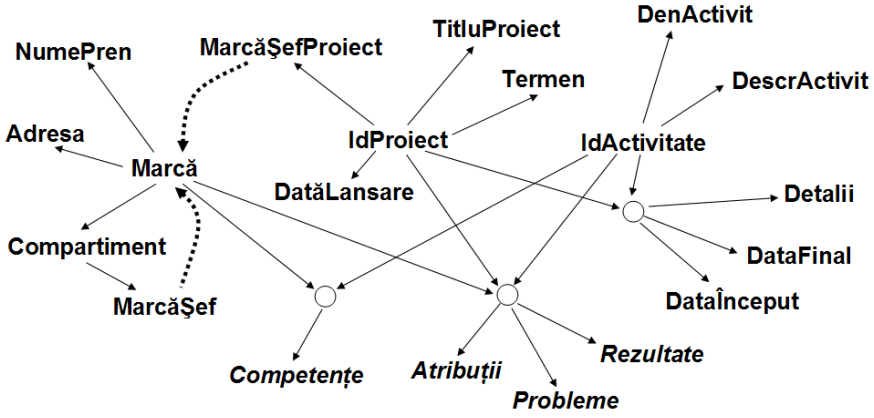

Figura 7.8. Noul graf al dependenţelor BD PROIECTE

Relaţiile decupate din graf sunt următoarele:

PERSONAL {`Marcă`, `Numepren`, `Adresă`, `Compartiment`}

COMPARTIMENTE {`Compartiment`, `MarcăŞef`}

ACTIVITĂŢI {`IdActivitate`, `DenActivit`, `DescrActivit`}

COMPETENŢE {`Marcă`, `IdActivitate`, `Competenţe`}

PROIECTE {`IdProiect`, `TitluProiect`, `MarcăŞefProiect`, `DatăLansare`, `Termen`}

ACTIVITĂŢI_PROIECTE {`IdProiect`, `IdActivitate`, `DataÎnceput`, `DataFinal`, `Detalii`}

PERSOANE_ACTIVITĂŢI_PROIECTE {`IdProiect`, `IdActivitate`, `Marcă`, `Atribuţii`, `Probleme`, `Rezultate`}

Alte frunze, dar aceleaşi conturi

Soluției la care am ajuns în acest paragraf pentru schema bazei de date CONTABILITATE i se poate reproşa, printre altele, faptul că, uneori, la inserarea unei linii în PLAN_CONTURI trebuie modificate alte linii din aceeaşi tabelă, ceea ce nu este posibil în toate SGBD-urile. Iar dacă la inserare lucrurile mai pot funcționa, declanşatorul de modificarea are şanse considerabile să se împotmolească.

Pentru a încerca să ajungem la următoarea soluție, pornim de la dependențele conținute în graful din figura 7.6. Dependențele de incluziune dintre `ContDebitor`, pe de o parte, şi `ContCreditor`, pe de altă parte, şi `SimbolCont` sunt discutabile. De fapt, restricția este ca toate conturile debitoare şi creditoare din orice înregistrare contabilă să fie elementare (frunze), adică să nu fie descompuse pe analitice (sau sintetice de ordinul 2). Putem defini, deci, un atribut denumit `ContElementar` care identifică (printr-o valoarea booleană *TRUE* sau şir de caractere "*Da*") un cont "frunză". Putem vorbi de o "specializare" a conturilor, cele două dependențe de incluziune inițiale fiind acum `ContDebitor` ⊆ `ContElementar`, respectiv `ContCreditor` ⊆ `ContElementar`, la care se adaugă `ContElementar` ⊆ `SimbolCont`. Spre deosebire de celelate două, ultima DI indică o specializare. Noul graf al dependențelor este cel din figura 7.9.

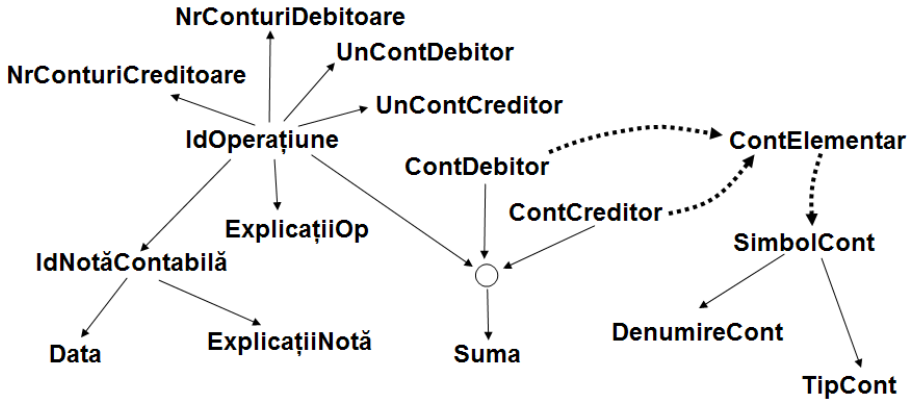

Figura 7.9. Trei dependenţe de incluziune, dintre care una semnifică o specializare

Decupând relațiile din graf, obținem:

PLAN_CONTURI {`SimbolCont`, `TipCont`, `DenumireCont`}

CONTURI_ELEMENTARE {`ContElementar` }

NOTE_CONTABILE {`IdNotăContabilă`, `Data`, `ExplicațiiNotă`}

OPERAȚIUNI {IdOperaţiune, IdNotăContabilă, ExplicaţiiOp, NrConturiDebitoare, NrConturiCreditoare, UnContDebitor, UnContCreditor}

DETALII_OPERAȚIUNI {IdOperaţiune, ContDebitor, ContCreditor, Suma}.

Lucrurile sunt chiar interesante, deoarece acum restricțiile referențiale se stabilesc astfel:

- între DETALII_OPERAŢIUNI.ContDebitor (copil) şi CONTURI_ELEMENTARE.ContElementar (părinte);
- între DETALII_OPERAŢIUNI.ContCreditor (copil) şi CONTURI_ELEMENTARE.ContElementar (părinte);
- între CONTURI_ELEMENTARE.ContElementar (copil) şi PLAN_CONTURI.SimbolCont.

Prin urmare, specializarea se poate traduce, în acest caz, printr-o nouă relație cu un singur atribut. Listingul 7.23 pentru re-crearea bazei de date CONTABILITATE poate fi descărcat de la adresa web menționată cu alte prilejuri.

Şi declanşatoarele tabelei PLAN_CONTURI se simplifică în mare măsură. Pentru comparație, îl vom discuta doar pe cel de inserare. Dar, mai înainte, să creăm o funcție (listing 7.24) care să primească drept parametru un cont şi care să returneze TRUE dacă acest cont apare măcar într-o operațiune contabilă (pe debit sau credit) şi FALSE în caz contrar:

Listing 7.24. Funcţia F_APARE

```
CREATE OR REPLACE FUNCTION f_apare (
   simbol_cont plan_conturi.SimbolCont%TYPE)
   RETURN BOOLEAN
AS
   v_unu NUMBER(1)  ;
BEGIN
   SELECT 1 INTO v_unu FROM dual WHERE EXISTS (
     SELECT 1 FROM detalii_operatiuni
     WHERE ContDebitor=simbol_cont OR ContCreditor=simbol_cont);
   RETURN TRUE ;
   EXCEPTION
      WHEN NO_DATA_FOUND THEN
         RETURN FALSE ;
   END ;
```

Funcția ne este de folos în declanşatorul de inserare în PLAN_CONTURI (listing 7.25) pentru a verifica dacă părintele contului inserat a fost debitat sau creditat în operațiuni contabile:

Listing 7.25. Noul declanşator de inserare al PLAN_CONTURI

```
CREATE OR REPLACE TRIGGER trg_pc_ins
    AFTER INSERT ON plan_conturi FOR EACH ROW
DECLARE
   v_parinte VARCHAR(15) ;
   v_gata BOOLEAN := FALSE ;
```

```
   v_nr NUMBER(3) := 0 ;
BEGIN
   v_parinte  :=  SUBSTR(:NEW.SimbolCont,1,  LENGTH(:NEW.SimbolCont)-
1);

   WHILE LENGTH(v_parinte) > 3 LOOP
      DELETE FROM conturi_elementare
      WHERE ContElementar = v_parinte ;

      IF SQL%ROWCOUNT > 0 THEN -- exista un parinte-frunza
         IF f_apare (v_parinte) THEN
              RAISE_APPLICATION_ERROR(-20193,
              'EROARE ! Nu se poate adauga contul dorit !!!') ;
         END IF ;
      END IF;
      v_parinte := SUBSTR(v_parinte,1,LENGTH(v_parinte)-1);
   END LOOP ;
   -- noul cont este frunza !!!
   INSERT INTO conturi_elementare VALUES (:NEW.SimbolCont) ;
END ;
```

Cu noua structură se schimbă şi funcţia `f_este_frunză`: - vezi listing 7.26.

Listing 7.26. Noul conţinut al funcţiei F_ESTE_FRUNZĂ

```
CREATE OR REPLACE FUNCTION f_este_frunza (simbol_cont VARCHAR2)
   RETURN BOOLEAN
AS
   v_unu NUMBER(1)  ;
BEGIN
   SELECT 1 INTO v_unu FROM DUAL WHERE EXISTS
    (SELECT 1 FROM conturi_elementare
      WHERE ContElementar = simbol_cont) ;
   RETURN TRUE ;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      RETURN FALSE ;
END ;
```

În schimb, declanşatorul de inserare în DETALII_OPERAȚIUNI poate rămâne neschimbat.

7.4. Caz practic - centru de închiriere casete video

Deşi nu întotdeauna la zi cu legislația pentru protecția drepturilor de autor, centrele de închiriere de casete video au constituit o afacere tentantă, mai ales în anii '90 şi, am putea spune, îşi are încă clientela sa, deşi HBO-ul, în general, televiziunea prin cablu constituie un concurent foarte puternic. Pentru a-l putea diferenția de concurență, vrem ca baza de date a centrului pentru care lucrăm să ofere şi date care să ajute la fidelizarea clienților:

- informații despre filme: genul de film, distribuție, realizare (scenariu, regie) etc.;

- premiile importante luate de filme: premiul, categoria, anul decernării;
- date despre împrumuturi, restituiri şi întârzieri;
- informaţii statistice, de genul: filmele cele mai cerute, clienţii cei mai fideli, clienţii cu cele mai mari întârzieri la returnarea casetelor;
- date despre clienţi: zi de naştere/onomastică, genuri preferate şi, implicit, structura pe vârste şi ocupaţii ale clienţilor;
- un sistem de evaluare (*rating*, în termeni elevaţi) a filmelor de către clienţi;

Ceea ce am discutat despre baza de date FILMOGRAFIE în capitolele anterioare ne este deci de mare folos pentru cele ce urmează. În privinţa atributelor surogat, de la început este evident că am avea nevoie de cel puţin trei:

- `IdFilm` - pentru identificarea fără ambiguitate a unui film;
- `IdCasetă` - un număr unic pentru diferenţierea între ele a casetelor;
- `IdÎmprumut` - număr atribuit fiecărui împrumut de una sau mai multe casete, la un moment oarecare.

Pentru identificarea clienţilor am putea să considerăm codul numeric personal (CNP-ul) ca fiind mulţumitor.

Din paragraful 6.2 ne-au rămas câteva dependenţe memorabile, unele funcţionale:

(1) `IdFilm` $\longrightarrow$ `TitluOriginal`
(2) `IdFilm` $\longrightarrow$ `TitluRO`
(3) `IdFilm` $\longrightarrow$ `AnLans`
(4) `DenPremiu` $\longrightarrow$ `LocDecernare`
(5) (`IdFilm, Rol`) $\longrightarrow$ `Actor`
(6) (`DenPremiu`, `Categorie`, `IdFilm`) $\longrightarrow$ `AnPremiu`.
(7) (`DenPremiu`, `Categorie`, `IdFilm`, `Actor`) $\longrightarrow$ `AnPremiu`

iar altele multi-valorice:

`IdFilm` $\longrightarrow\!\!\!\!\rightarrow$ `Producător | Regizor`
`IdFilm` $\longrightarrow\!\!\!\!\rightarrow$ `Producător | Gen` sau `IdFilm` $\longrightarrow\!\!\!\!\rightarrow$ `Regizor | Gen`

Adăugăm şi scenaristul, aşa că:

`IdFilm` $\longrightarrow\!\!\!\!\rightarrow$ `Producător | Scenarist` sau `IdFilm` $\longrightarrow\!\!\!\!\rightarrow$ `Regizor | Gen`

Caseta este identificată de atributul `IdCasetă`, astfel că putem scrie:

(8) `IdCasetă` $\longrightarrow$ `DataCumpărării`
(9) `IdCasetă` $\longrightarrow$ `ProducătorCasetă`
(10) `IdCasetă` $\longrightarrow$ `AnProdCasetă`
(11) `IdCasetă` $\longrightarrow$ `PreţCumpărare`

Anticipând o cerere mai mare, un film poate fi compărat în mai multe exemplare, deci pe mai multe casete: `IdFilm` $\not\longrightarrow$ `IdCasetă`. Nici reciproca nu e prea valabilă, întrucât pe o casetă pot fi adunate filme de mai mică dimensiune, scurt

metraje (doar trei exemple: filmuleţele cu Stan şi Bran, Chaplin şi Tom şi Jerry): IdCasetă —/→ IdFilm.

Pentru a putea gestiona coerent toate fimele (scurt metraje sau nu) de pe o casetă, apelăm la un atribut de genul FilmNr, adică numărul de ordine al filmului de pe o casetă (vă amintiţi de atributul Linie pentru facturi ?). Aşa că:

(12) (IdCasetă, FilmNr) ⟶ IdFilm

Am zice că problema raportului filme-casete este tranşată. Da' de unde ! Persoanele mai simţitoare poate-şi amintesc de febra pre-telenovelistă declanşată de un film epopee - *Pasărea Spin*. Ei bine, ca şi alte filme-maraton, acesta se întinde pe mai multe casete. Astfel încât, spre exemplu, pe o casetă poate fi partea a patra a filmului XYZ. Din fericire, dependenţa de mai sus nu este compromisă, însă am avea nevoie de un atribut adiţional - ParteFilm, iar dependenţa s-ar putea scrie:

(13) (IdCasetă, FilmNr) ⟶ ParteFilm

Ba chiar putem fi siguri că, pe o casetă, casa producătoare poate să introducă ultima parte a unui film plus un scurt-metraj de acelaşi regizor sau un medalion actoricesc etc.

Despre clienţi, numai de bine:

(14) CNPClient ⟶ NumeClient
(15) CNPClient ⟶ AdresaClient
(16) CNPClient ⟶ TelefonClient
(17) CNPClient ⟶ DataNaştereClient
(18) CNPClient ⟶ NivelStudiiClient

Un client poate împrumuta una sau mai multe casete. Fiecare împrumut este identificat printr-o cheie surogat - IdÎmprumut, aşa că:

(19) IdÎmprumut ⟶ DataOraImprumut
(20) IdÎmprumut ⟶ CNPClient

Deoarece simultan se pot împrumuta mai multe casete, IdÎmprumut —/→ IdCasetă, ceea ce e destul de neliniştitor, atâta vreme cât obiectivul central al aplicaţiei este de a gestiona împrumuturile de casete. Fireşte, prima tentanţie este de a recurge la un atribut suplimentar, ca în cazul liniilor din facturi, atribut căreia îi putem spune chiar NrCrtÎmpr (adică un soi de număr curent al împrumutului, număr care indică a câta casetă din împrumut este cea curentă), aşa că, cu un pic de cârpeală, problema s-ar rezolva:

(IdÎmprumut, NrCrtÎmpr) ⟶ IdCasetă

Dacă adăugăm că restituirea casetelor nu este întotdeauna simultană, adică un client poate împrumuta trei casete într-o sâmbătă dimineaţă şi să returneze una sâmbătă seara şi celelalte două duminică, se poate scrie, fie (IdÎmprumut,

NrCrtÎmpr) ⟶ DataOraRestituirii, fie (IdÎmprumut, IdCasetă) ⟶ DataOraRestituirii. Eu, unul, aş alege varianta din urmă.

Ei bine, dacă privim mai atent ultima dependență, observăm că, la o adică, aceasta ar face de prisos atributul NrCrtÎmpr. Diferența dintre varianta folosirii atributului NrCrtÎmpr (varianta 1) şi cea fără (a doua) este ilustrată în figura 7.10.

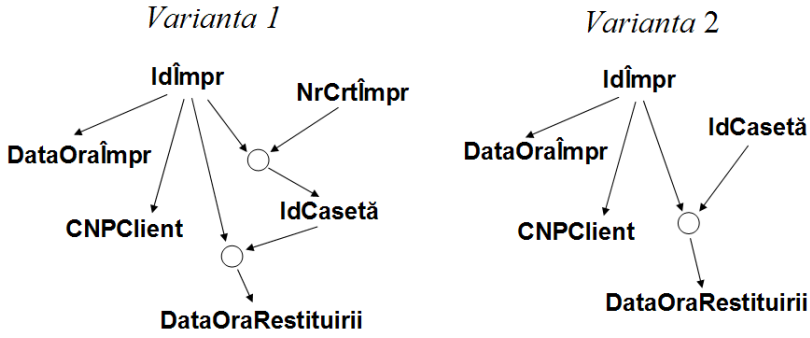

Figura 7.10. Două variante de dependenţe pentru împrumuturi şi restituiri

Fiecare îşi are avantajele şi dezavantajele sale. Prima pare mai complicată şi, totodată, are un uşor aer de artificialitate indus de folosirea NrCrtÎmpr. Cel mai important avantaj al său ține de faptul că preia foarte bine succesiunea temporală împrumut (prima DF cu sursa compusă) - restituire (a doua). Consecința esențială este că în schemă nu apar valori nule. Graful din dreapta figurii este mult mai simplu. Din momentul împrumutului până în cel al restituirii valoarea atributului DataOraRestuirii ar fi NULLă. Astfel, pentru a afla în orice moment casetele aflate la clienți, condiția în SQL ar fi DataOraRestuirii IS NULL. Tocmai adversarii folosirii valorilor nule ar dezaproba această a doua soluție. Noi, însă, nefiind duşmani (dar nici prieteni) ai nulității în bazele de date relaționale, vom opta pentru această a doua soluție, chiar dacă lenea a cântărit mult în alegere. Aşadar:

(21) (IdÎmprumut, IdCasetă) ⟶ DataOraRestituirii

Să adunăm tot ce-am discutat până acum într-un graf. Figura 7.11 este reprezentare destul de bună a ansamblului de dependențe funcționale (1)-(21), plus cele multivaloare. Este drept, aglomerația este cam mare, dar arta cere sacrificii.

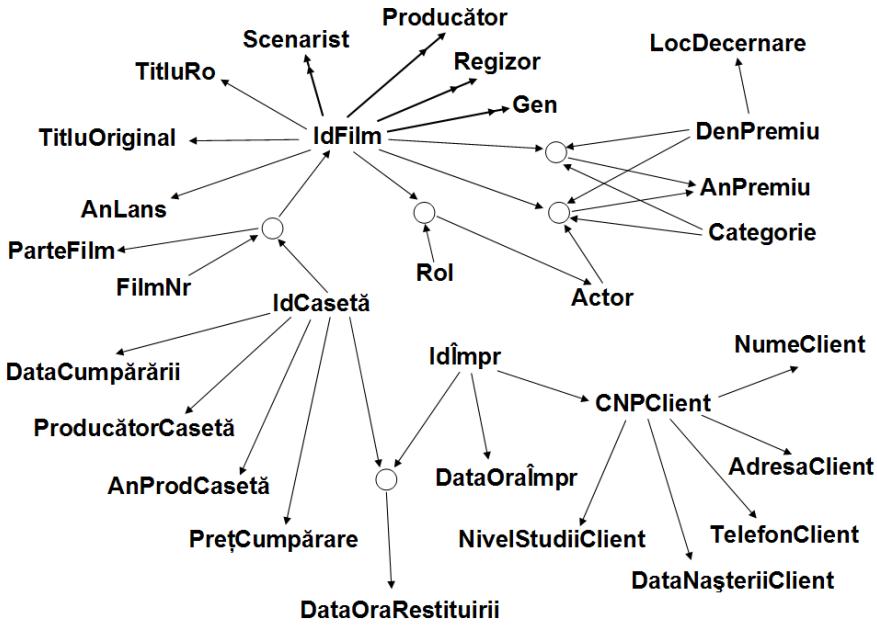

Figura 7.11. Graful DF pentru BD Centru de închiriere - versiunea 1.0

Decupăm relaţiile din graf:

PREMII_DENUMIRI {DenPremiu, LocDecernare}

FILME {IdFilm, TitluOriginal, TitluRO, AnLans}

DISTRIBUŢIE {IdFilm, Rol, Actor}

PREMII_FILME {IdFilm, DenPremiu, Categorie, AnPremiu}

PREMII_INTERPRETARE {IdFilm, Actor , DenPremiu, Categorie, AnPremiu}

PRODUCĂTORI {IdFilm, Producător}

REGIZORI {IdFilm, Regizor}

SCENARIŞTI {IdFilm, Scenarist}

FILME_GENURI {IdFilm, Gen}

CASETE {IdCasetă, DataCumpărării, ProducătorCasetă, AnProdCasetă, PreţCumpărare}

CASETE_FILME {IdCasetă, FilmNr, IdFilm, ParteFilm}

CLIENȚI {CNPClient, NumeClient, AdresaClient, TelefonClient, DataNaşteriiClient, NivelStudiiClient}

ÎMPRUMUTURI {IdÎmpr, DataOraÎmpr, CNPClient}

CASETE_ÎMPRUMUTATE {IdÎmpr, IdCasetă, DataOraRestituirii}

Schema pare a răspunde necesarului informațional pe care ni-l propusesem inițial. Astfel, putem afla: numărul de casete împrumutate de un client pe o anumită perioadă; ponderea comediilor între filmele închiriate de un client, sau pe tot centrul; ce genuri de filme preferă tinerii între 30 şi 35 de ani cu studii medii etc.

Să ne ocupăm însă de regulile scrise sau nescrise ale centrului:

- un client nu poate împrumuta mai mult de patru casete simultan !
- la fiecare 10 casete împrumutate, un client are dreptul la o casetă împrumutată gratuit;
- împrumutul este pentru 48 de ore; la remiterea casetei se calculează o penalizare de 75% din prețul de închiriere al casetei pentru fiecare zi de întârziere; iată şi tarifele zilnice la închiriere:
  - 50 000 lei pentru casetele produse în anul calendaristic curent şi cel precedent;
  - 40 000 lei pentru casetele produse în urmă cu doi şi trei ani;
  - 30 000 lei pentru restul.
- pierderea sau distrugerea unei casete antrenează o amendă care reprezintă dublul prețului casetei.

Prima regulă s-ar implementa în schema actuală a bazei de maniera următoare: atunci când în tabela CASETE_ÎMPRUMUTATE se introduce o linie nouă, pentru care valorile ar fi notate cu *(:NEW.IdImpr, :NEW.IdCaseta, NULL)*, după un calapod similar Oracle, se numără, pentru clientul aferent împrumutului curent, câte dintre casetele împrumutate au valoarea atributului `DataOraRestituirii` nulă:

```
SELECT COUNT(*)
INTO v_CaseteNerestituite
FROM casete_imprumutate c INNER JOIN imprumuturi i
      ON c.idimpr=i.idimpr
WHERE DataOraRestituirii IS NULL AND CNPClient IN
      (SELECT CNPClient
       FROM  imprumuturi
       WHERE idimpr = :NEW.idimpr)
```

Ulterior, valoarea variabilei se testează dacă este mai mică decât patru, caz în care inserarea este permisă; altminteri, inserarea se respinge. Fraza SELECT de mai sus poate fi folosită în formularul de lucru (care rulează pe platforma client), într-o

secvența de cod de pe serverul de aplicații (dacă arhitectura este de tip *Web*, adică pe trei sau mai multe straturi), dar cel mai sigur ar fi să se recurgă la un declanşator pe tip INSERT pentru această tabelă. Bineînțeles, numai ca SGBD-ul folosit să "suporte" declanşatoare (nu este cazul MySQL-ului, din păcate, decât în câteva versiuni comerciale).

Ei bine, recomandarea noastră este nu numai să se folosească un SGBD ce permite lucrul cu declanşatoare, dar să se introducă un atribut redundant numit `NrCaseteNerestituite`, atribut actualizabil la declanşatoarele de inserare, modificare şi ştergere în CASETE_ÎMPRUMUTATE. Acest atribut ar depinde de `CNPClient`: `CNPClient` $\longrightarrow$ `NrCaseteNerestituite.`

Ba chiar am putea exagera şi mai mult, folosind două atribute dependente de `CNPClient`, unul `NrCaseteÎmprumutate` şi altul `NrCaseteRestituite`. Paradoxal, deşi ideea pare deplasată, există argumente în favoarea sa. Iar, întrucât a doua regulă spune că *la fiecare 10 casete împrumutate, un client are dreptul la o casetă împrumutată gratuit* chiar ne convin ambele atribute, aşa că, în dezaprobarea publicului, ne precipităm şi scriem:

(22) `CNPClient` $\longrightarrow$ `NrCaseteÎmprumutate`

(23) `CNPClient` $\longrightarrow$ `NrCaseteRestituite`

Mecanismul de actualizare este unul comun pentru utilizatorii de declanşatoare (triggere): `NrCaseteÎmprumutate` se incrementează la inserarea unei linii în CASETE_ÎMPRUMUTATE, iar `NrCaseteRestituite` la modificarea unei linii în CASETE_ÎMPRUMUTATE, atunci când condiția îndeplinită este `:OLD.DataOraRestituirii IS NULL AND :NEW.DataOraRestituirii IS NOT NULL`. Explicația condiției este una simplă: restituirea unei casete împrumutate se consemnează în baza noastră de date prin, să-i zicem, "de-nulizarea" atributului `CASETE_ÎMPRUMUTATE.DataOraRestituirii`.

A doua regulă e centrului reprezintă o tentativă de a fideliza clienții, mai ales pe cei dependenți: *la 10 casete închiriate, a 11-a este împrumutată gratuit*. Noroc de atributul `NrCaseteÎmprumutate` de mai sus, pentru că, în condițiile în care acesta este actualizat corect, putem şti în orice moment dacă se cuvine să acordăm gratuitatea sau nu. Pentru a cunoaşte, totuşi, în timp, câte casete au fost împrumutate cu titlu gratuit, am putea recurge la un atribut nou, `Gratuită`, care să ia valorile „D" (da) sau „N" şi să depindă funcțional astfel:

(24) `(IdÎmprumut, IdCasetă)` $\longrightarrow$ `Gratuită`

A treia regulă e ceva mai dură: *împrumutul este pentru 48 de ore; la remiterea casetei se calculează o penalizare de 75% din prețul casetei pentru fiecare zi de întârziere*. Implementarea sa ar presupune că, la "de-nulizarea" valorii unui atribut `DataOraRestituirii` să verificăm dacă diferența `CASETE_ÎMPRUMUTATE.DataOraRestituirii - ÎMPRUMUTURI.DataÎmpr` este mai mare de două zile (întrucât cele două atribute sunt de tip DATETIME, adică dată şi oră, diferența lor furnizează, de obicei, numărul de zile dintre cele două momente). Dacă da, atunci calculăm penalizarea după relația: `Penalizare := 0.75 * (CASETE_ÎMPRU-`

MUTATE.DataOraRestituirii - ÎMPRUMUTURI.DataÎmpr -2). Este evidentă nevoia de atributul Penalizare care va depinde funcțional astfel:
(25) (IdÎmprumut, IdCasetă) ⟶ Penalizare

Cu această ocazie, ne putem propune să rezolvăm o problemă pe care trebuia s-o avem în vedere mai demult, şi anume *Cât trebuie să achite clientul la fiecare împrumut (închirierea a una, două, trei sau patra casete) ?* Pentru acest scop apelăm la atributul ValoareÎnchir (de la valoare închiriere):
(26) IdÎmprumut ⟶ ValoareÎnchir

În fine, ultima regulă, potrivit căreia *pierderea sau distrugerea unei casete antrenează o amendă care reprezintă dublul prețului casetei*, are o dublă implicație: pe de o parte, regretabilul eveniment trebuie consemnat la (un soi de) restituire, iar atributul Penalizare va conține acum dublul contravalorii casetei; pe de altă parte, trebuie să existe cumva în baza de date o indicație precum că respectiva caseta nu mai poate fi închiriată, adică este pierdută sau distrusă. Soluția ştergerii înregistrării corespunzătoare din tabela CASETE nu este una prea ingenioasă, întrucât ar trebui să pierdem toți "copiii" acestei înregistrări, deci inclusiv de câte ori şi cui a fost împrumutată, şi, astfel, informațiile statistice privind filmele, clienții şi împrumuturile vor fi serios afectate. Cel mai nimerit pare să recurgem la un atribut numit StareCasetă care să aibă o valoare implicită *OK* (sau nulă, deşi nu suntem fani nullişti) şi, dacă este cazul, *Pierdută*, *Distrusă*, ba chiar, dacă tot îl avem, putem consemna şi situațiile în care caseta a fost scoasă din uz "de moarte bună" (*Casată*) sau se apropie de această stare (*Uzată*):
(27) IdCasetă ⟶ StareCasetă

Nu este însă suficient să cunoaştem starea casetei, ci şi de la ce împrumut i se trage pierderea sa distrugerea. Aşă că apelăm la un atribut special, StareLaRestituire, care indică halul în care un client ne-a restituit caseta (să fim înțeleşi: e vorba de halul casetei, nu al clientului !) şi care depinde funcțional astfel:
(28) (IdÎmprumut, IdCasetă) ⟶ StareLaRestituire

Ca un alt artificiu propus, ne imaginăm că, în ton cu vremurile, clienții vor putea să-şi rezerve/comande casete pe web, iar centrul să aibă un serviciu de furnizare a casetelor la domiciliu (de tipul *pizza delivery*) sau, în cel mai rău caz, datorită dimensiunii impresionante a centrului (dă, Doamne !), vor fi amplasate trei-patru terminale într-un colț prin care clientul poate să caute filmele după actori, regizori, subiect etc. Tocmai pentru a evita execuția unei interogări care să implice tabelele CASETE, ÎMPRUMUTURI şi CASETE_ÎMPRUMUTATE prin care să se afle dacă o casetă este disponibilă sau împrumutată la un moment dat, se poate recurge la un alt atribut redundant numit Disponibilitate, actualizabil automat prin declanşatoarele tabelei CASETE_ÎMPRUMUTATE:
(29) IdCasetă ⟶ Disponibilitate

Apropo, ne propusesem şi un mic mecanism de rating al filmelor de către clienţi, deşi, din câte am văzut pe site-urile româneşti dedicate vânzărilor (legale) de carte şi CD-uri, sistemul nu prea a prins la noi. La modul cel mai simplu, putem alege o scală de punctare de la 0 la 5, de genul celei practicate de Laurenţiu Brătan în revista *22*, atributul `Punctaj` fiind "implicat" în dependenţa:

(30) $(\texttt{IdFilm}, \texttt{CNPClient}) \longrightarrow \texttt{Punctaj}$.

Tot frământându-ne mintea cu mecanismul de căutare a informaţiilor despre filme, realizăm că, la un moment dat, este posibil ca unii clienţi să dorească a viziona filme în care joacă actori spanioli, sau filme regizate de cineaşti născuţi în perioada 1930-1940 sau site-ul/pagina Web dedicată unei anumite personalităţi cinematografice. Putem vorbi de generalizare: actorii, regizorii şi scenariştii sunt toţi cineaşti, ca să nu amintim numeroşi actori care sunt şi regizori, sau regizori ce sunt şi producători şi scenarişti. Aşa că am putea introduce o serie de atribute pentru identificarea şi caracterizarea oricărui actor/scenarist/producător/regizor: `IdCineast`, `NumeCineast`, `DataNaşterii`, `DataMorţii`, `Naţionalitate`, `PaginăWeb`. Fără a schimba numele celor patru atribute (ar fi fost mai nimerite titulaturile `IdScenarist`, `IdProducător`, `IdRegizor` şi `IdActor`), reprezentăm cele patru dependenţe de incluziune $\texttt{Scenarist} \subseteq \texttt{IdCineast}$, $\texttt{Producător} \subseteq \texttt{IdCineast}$, $\texttt{Regizor} \subseteq \texttt{IdCineast}$, $\texttt{Actor} \subseteq \texttt{IdCineast}$.

Punând cap la cap tot ceea ca am discutat despre legăturile semantice dintre atributele bazei de date, obţinem reprezentarea sub formă de graf din figura 7.12. Pentru un plus de lizibilitate, am grupat toate destinaţiile surselor `IdFilm`, `IdCineast`, `IdCasetă` şi `CNPClient`.

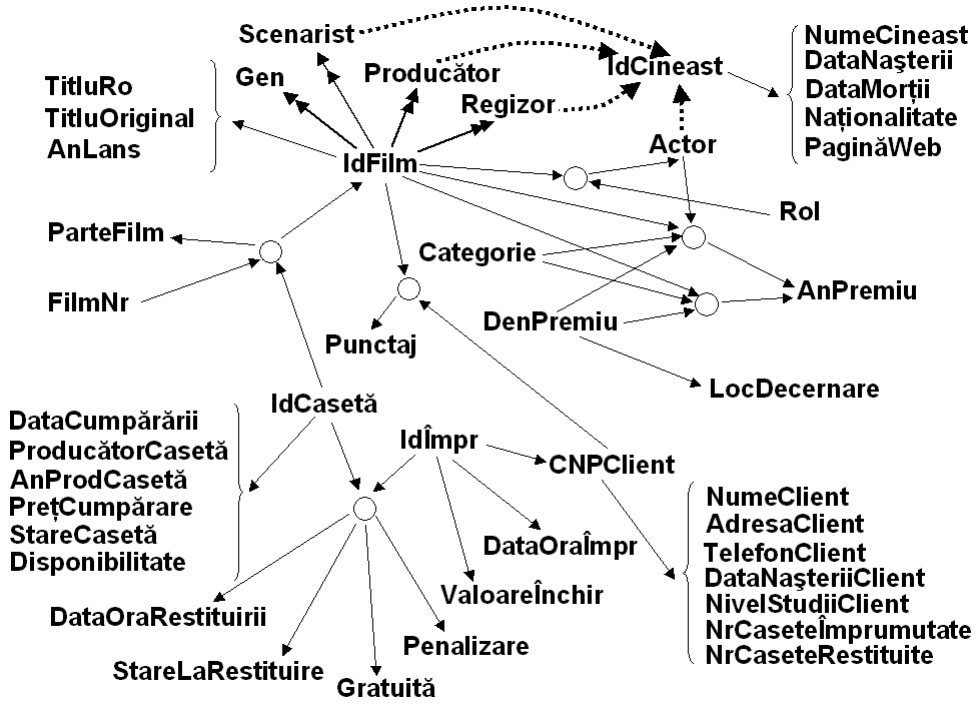

Figura 7.12. Graful DF pentru BD Centru de închiriere - versiunea 2.0

Pe baza grafului, schema finală a bazei de date este următoarea:

PREMII_DENUMIRI {DenPremiu, LocDecernare}

FILME {IdFilm, TitluOriginal, TitluRO, AnLans}

CINEAŞTI {IdCineast, NumeCineast, DataNaşterii, DataMorţii, Naţionalitate, PaginăWeb}

DISTRIBUŢIE {IdFilm, Rol, Actor*}

PREMII_FILME {IdFilm, DenPremiu, Categorie, AnPremiu}

PREMII_INTERPRETARE {IdFilm, Actor* , DenPremiu, Categorie, AnPremiu}

PRODUCĂTORI {IdFilm, Producător*}

REGIZORI {IdFilm, Regizor*}

SCENARIŞTI {IdFilm, Scenarist*}

FILME_GENURI {IdFilm, Gen}

CASETE {IdCasetă, DataCumpărării, ProducătorCasetă, AnProdCasetă, PreţCumpărare, StareCasetă, Disponibilitate}

CASETE_FILME {IdCasetă, FilmNr, IdFilm, ParteFilm}

CLIENŢI {CNPClient, NumeClient, AdresaClient, TelefonClient, DataNaşteriiClient, NivelStudiiClient, NrCaseteÎmprumutate, NrCaseteRestituite}

ÎMPRUMUTURI {IdÎmpr, DataOraÎmpr, CNPClient, ValoareÎnchir}

CASETE_ÎMPRUMUTATE {IdÎmpr, IdCasetă, DataOraRestituirii, Gratuită, Penalizare, StareLaRestituire }

APRECIERI_FILME {IdFilm, CNPClient, Punctaj}

Atributele marcate cu asterisc sunt chei străine, părintele comun fiind câmpul IdCineast din tabela CINEAŞTI.

Fiind vorba de un caz practic pe care dorim să-l ducem cât mai aproape de „pânzele albe", vom urma promisiunea din deschiderea capitolul, discutând şi câteva chestiuni de implementare a schemei propuse. Începem cu crearea tabelelor şi definirea restricţiilor de non nulitate, cheilor primare, restricţiilor de integritate referenţială, precum şi regulilor de validare la nivel de atribut şi înregistrare – vezi listing 7.27. Atributele pentru care au fost definite restricţii sunt:

- AnLans din FILME (valori mai mari decât 1900);
- AnPremiu din PREMII_FILME şi PREMII_INTERPRETARE (valori mai mari decât 1920);
- DataCumpărării din CASETE (valori mai mari de 1 ianuarie 1998);
- AnProdCaseta din CASETE (valori mai mari decât 1980);
- StareCaseta din CASETE – valori limitate la lista: *Ok*, *Pierdută*, *Distrusă*, *Casată* şi *Uzată*;
- Disponibilitate din CASETE – numai valorile *D* (da) şi *N* (nu);
- FilmNr din CASETE_FILME: deoarece pe o casetă nu pot încăpea mai mult de 30 (clipuri/scurt metraje), valorile acestui atribut trebuie să se încadreze în plaja 1-30;
- ParteFilm din CASETE_FILME: deoarece un film poate fi „rupt" în maximum 30 de părţi (nu credem ca vreun centru să achiziţioneze pe casete *Tânăr şi neliniştit* !), valorile acestui atribut trebuie să se încadreze în plaja 1-30;

- `NrCaseteImprumutate` şi `NrCaseteRestituite` din CLIENȚI – valorile trebuie să fie întregi pozitive;
- `StareLaRestituire` din CASETE_ÎMPRUMUTATE – valori limitate la lista: *Ok, Pierdută, Distrusă;*
- `Gratuită` din CASETE_ÎMPRUMUTATE – numai valorile *D* (da) şi *N* (nu);
- `Punctaj` din APRECIERI_FILME trebuie să se încadreze între 0 şi 10.

Listing 7.27. Scriptul Oracle de creare a tabelelor

```
-- creare BD centru de inchirieri VIDEO
--
DROP TABLE aprecieri_filme ;
DROP TABLE casete_imprumutate ;
DROP TABLE imprumuturi ;
DROP TABLE clienti ;
DROP TABLE casete_filme ;
DROP TABLE casete ;
DROP TABLE premii_interpretare ;
DROP TABLE premii_filme ;
DROP TABLE premii_denumiri ;
DROP TABLE scenaristi ;
DROP TABLE regizori ;
DROP TABLE producatori ;
DROP TABLE distributie ;
DROP TABLE filme_genuri ;
DROP TABLE cineasti ;
DROP TABLE filme ;

CREATE TABLE cineasti (
   IdCineast NUMBER(6) NOT NULL PRIMARY KEY,
   NumeCineast VARCHAR2(40) NOT NULL,
   DataNasterii DATE,
   DataMortii DATE,
   Naţionalitate VARCHAR2(30),
   PaginaWeb VARCHAR2(200),
   CHECK (NVL(DataNasterii, DATE'1970-01-01') <
      NVL2(DataMortii, DataNasterii, DATE'1970-01-02'))
   );

CREATE TABLE filme (
   IdFilm NUMBER(10) NOT NULL PRIMARY KEY,
   TitluOriginal VARCHAR2(100),
   TitluRO VARCHAR2(100) NOT NULL,
   AnLans NUMBER(4) CHECK (AnLans > 1900)
   ) ;

CREATE TABLE filme_genuri (
   IdFilm NUMBER(10) NOT NULL REFERENCES filme (IdFilm),
   Gen VARCHAR2(25) NOT NULL,
   PRIMARY KEY (IdFilm, Gen)
   ) ;

CREATE TABLE distributie (
```

```
   IdFilm NUMBER(10) NOT NULL REFERENCES filme (IdFilm),
   Rol VARCHAR2(30) NOT NULL,
   Actor NUMBER(6) REFERENCES cineasti (IdCineast),
   PRIMARY KEY (IdFilm, Rol)
   );

CREATE TABLE producatori (
   IdFilm NUMBER(10) NOT NULL REFERENCES filme (IdFilm),
   Producator NUMBER(6) NOT NULL REFERENCES cineasti (IdCineast),
   PRIMARY KEY (IdFilm, Producator)
   );

CREATE TABLE regizori (
   IdFilm NUMBER(10) NOT NULL REFERENCES filme (IdFilm),
   Regizor NUMBER(6) NOT NULL REFERENCES cineasti (IdCineast),
   PRIMARY KEY (IdFilm, Regizor)
   ) ;

CREATE TABLE scenaristi (
   IdFilm NUMBER(10) NOT NULL REFERENCES filme (IdFilm),
   Scenarist NUMBER(6) NOT NULL REFERENCES cineasti (IdCineast),
   PRIMARY KEY (IdFilm, Scenarist)
   );

CREATE TABLE premii_denumiri (
   DenPremiu VARCHAR2(40) NOT NULL PRIMARY KEY,
   LocDecernare VARCHAR2(50)
   );

CREATE TABLE premii_filme (
   IdFilm NUMBER(10) NOT NULL REFERENCES filme (IdFilm),
   DenPremiu  VARCHAR2(40)  NOT  NULL  REFERENCES  premii_denumiri
(DenPremiu),
   Categorie VARCHAR2(30) NOT NULL,
   AnPremiu NUMBER(4) CHECK (Anpremiu > 1920),
   PRIMARY KEY (IdFilm, DenPremiu, Categorie)
   );

CREATE TABLE premii_interpretare (
   IdFilm NUMBER(10) NOT NULL REFERENCES filme (IdFilm),
   Actor NUMBER(6) REFERENCES cineasti (IdCineast),
   DenPremiu  VARCHAR2(40)  NOT  NULL  REFERENCES  premii_denumiri
(DenPremiu),
   Categorie VARCHAR2(30) NOT NULL,
   AnPremiu NUMBER(4) CHECK (AnPremiu > 1920),
   PRIMARY KEY (IdFilm, Actor, DenPremiu, Categorie)
   ) ;

CREATE TABLE casete (
   IdCaseta NUMBER(12) NOT NULL PRIMARY KEY,
   DataCumpararii DATE CHECK (DataCumpararii > DATE'1998-01-01'),
   ProducătorCaseta VARCHAR2(30),
   AnProdCaseta NUMBER(4) DEFAULT EXTRACT (YEAR FROM CURRENT_DATE)
      CHECK (AnProdCaseta > 1980),
   PreţCumparare NUMBER(8),
   StareCaseta VARCHAR2(20) DEFAULT 'Ok' NOT NULL
```

```
      CHECK (StareCaseta IN ('Ok', 'Pierduta', 'Distrusa', 'Casata',
'Uzata')),
   Disponibilitate CHAR(1) DEFAULT 'D' NOT NULL
      CHECK (Disponibilitate IN ('D','N'))
   );

CREATE TABLE casete_filme (
   IdCaseta NUMBER(12) NOT NULL REFERENCES casete (Idcaseta),
   FilmNr NUMBER(2) DEFAULT 1 NOT NULL
      CHECK (FilmNr BETWEEN 1 AND 30),
   IdFilm NUMBER(10) NOT NULL REFERENCES filme (IdFilm),
   ParteFilm NUMBER(2) DEFAULT 1 NOT NULL
      CHECK (ParteFilm BETWEEN 1 AND 30),
   PRIMARY KEY (IdCaseta, FilmNr)
   ) ;

CREATE TABLE clienti (
   CNPClient NUMBER(13) NOT NULL PRIMARY KEY,
   NumeClient VARCHAR2(45) NOT NULL,
   AdresaClient VARCHAR2(100),
   TelefonClient VARCHAR2(25),
   DataNasteriiClient DATE,
   NivelStudiiClient VARCHAR2(15) DEFAULT 'Medii',
   NrCaseteImprumutate NUMBER(6) DEFAULT 0
      CHECK (NrCaseteImprumutate >= 0),
   NrCaseteRestituite NUMBER(6) DEFAULT 0
      CHECK (NrCaseteRestituite >= 0),
   CHECK (NrCaseteImprumutate - NrCaseteRestituite BETWEEN 0 AND 4)
   );

CREATE TABLE imprumuturi (
   IdImpr NUMBER(15) NOT NULL PRIMARY KEY,
   DataOraImpr DATE DEFAULT CURRENT_DATE,
   CNPClient NUMBER(13) NOT NULL REFERENCES clienti(CNPClient),
   ValoareInchir NUMBER(7) DEFAULT 0 NOT NULL ;
   );

CREATE TABLE casete_imprumutate (
   IdImpr NUMBER(15) NOT NULL REFERENCES imprumuturi (IdImpr),
   IdCaseta  NUMBER(12) NOT NULL REFERENCES casete (Idcaseta),
   DataOraRestituirii DATE,
   StareLaRestituire VARCHAR2(20) DEFAULT 'Ok'
      CHECK (StareLaRestituire IN ('Ok', 'Pierduta', 'Distrusa')),
   Gratuita CHAR(1) DEFAULT 'N' NOT NULL
      CHECK (Gratuita IN ('D','N')),
   Penalizare NUMBER(9) DEFAULT 0,
   PRIMARY KEY (IdImpr, IdCaseta)
   );

CREATE TABLE aprecieri_filme (
   IdFilm NUMBER(10) NOT NULL REFERENCES filme (IdFilm),
   CNPClient  NUMBER(13) NOT NULL REFERENCES clienti(CNPClient),
   Punctaj NUMBER(2) DEFAULT 0 NOT NULL
      CHECK (Punctaj BETWEEN 0 AND 10),
   PRIMARY KEY (IdFilm, CNPClient)
   );
```

Regulile de validare la nivel de înregistrare prezente în script privesc următoarele restricții:

- în tabela CINEAŞTI: eventuala dată a decesului să succeadă datei de naştere; întrucât, pe de o parte, pentru cineaştii în viață, `DataMorţii` este nulă, iar, pe de altă parte, pentru o serie de cineaşti nu se cunoaşte, încă, una sau ambele date, trebuie avute în vedere problemele ce apot apărea din „manevrarea" valorilor nule; cea mai nimerită este, în acest context, funcția NVL2[14];
- în CLIENȚI, având în vedere că nici un client nu poate închiria, la orice moment dat, mai mult de patru casete, regula este una simplă: `NrCaseteImprumutate - NrCaseteRestituite BETWEEN 0 AND 4`; mai dificil va fi mecanismul de asigurare a corectitudinii celor două atribute;

Continuăm implementarea în Oracle a regulilor stabilite, rezolvând problema cheilor surogat, care prespune folosirea secvenţelor şi declanşatoarelor de inserare, după modelul din listing 7.28. După cum spuneam şi cu alte ocazii, majoritatea SGBD-urilor disponibile permit declararea de atribute auto-incrementate.

Listing 7.28. Secvenţe şi declanşatoare de inserare PL/SQL pentru generarea cheilor surogat

```
-- autoincrementarea atributului CINEASTI.IdCineast
DROP SEQUENCE seq_IdCineast ;
CREATE SEQUENCE seq_IdCineast START WITH 1
   MINVALUE 1 MAXVALUE 999999
   NOCYCLE NOCACHE ORDER ;

CREATE OR REPLACE TRIGGER trg_cineasti_ins
   BEFORE INSERT ON cineasti FOR EACH ROW
BEGIN
   SELECT seq_IdCineast.NextVal INTO :NEW.IdCineast FROM dual ;
END;
/

-- autoincrementarea atributului FILME.IdFilm
DROP SEQUENCE seq_IdFilm ;
CREATE SEQUENCE seq_IdFilm START WITH 1
   MINVALUE 1 MAXVALUE 9999999999
   NOCYCLE NOCACHE ORDER ;

CREATE OR REPLACE TRIGGER trg_filme_ins
   BEFORE INSERT ON filme FOR EACH ROW
BEGIN
   SELECT seq_IdFilm.NextVal INTO :NEW.IdFilm FROM dual ;
END;
/

-- autoincrementarea atributului CASETE.IdCaseta
```

[14] Pentru câteva detalii despre funcția NVL2, vezi şi [Fotache s.a.03], p.190

```
DROP SEQUENCE seq_IdCaseta ;
CREATE SEQUENCE seq_IdCaseta START WITH 10000000001
   MINVALUE 1 MAXVALUE 999999999999
   NOCYCLE NOCACHE ORDER ;

CREATE OR REPLACE TRIGGER trg_casete_ins
   BEFORE INSERT ON casete FOR EACH ROW
BEGIN
   SELECT seq_IdCaseta.NextVal INTO :NEW.IdCaseta FROM dual ;
END;
/

-- autoincrementarea atributului IMPRUMUTURI.IdImpr
DROP SEQUENCE seq_IdImpr ;
CREATE SEQUENCE seq_IdImpr START WITH 1000000000001
   MINVALUE 1 MAXVALUE 99999999999999
   NOCYCLE NOCACHE ORDER ;

CREATE OR REPLACE TRIGGER trg_imprumuturi_ins
   BEFORE INSERT ON imprumuturi FOR EACH ROW
BEGIN
   SELECT seq_IdImpr.NextVal INTO :NEW.IdImpr FROM dual ;
END;
/
```

Tabela CASETE_FILME conține un caz special de atribut a cărui valoare trebuie gestionată automat. Astfel, atributul `FilmNr` trebuie inițializat cu 1 la primul film de pe fiecare casetă, apoi trebuie incrementat cu 1 pentru orice alt film declarat ulterior ca fiind înregistrat pe caseta respectivă. Necazul este că, la ştergerea unui film de pe o casetă, trebuie re-verificată ordinea filmelor pe caseta respectivă. De exemplu, dacă pe o casetă sunt 5 filme (scurt metraje) şi se şterge linia care se referă la al treilea film de pe casetă, atunci filmul al patrulea devine al treilea, iar al cincilea devine al patrulea.

Iar ca delectarea să fie totală, trebuie să se interzică modificarea interactivă (prin UPDATE), atât a atributului `CASETE_FILME.FilmNr`, cât şi atributelor `CASETE_FILME.IdCasetă` şi `CASETE_FILME.IdFilm`. Aceste ultime două atribute pot fi, eventual, modificate doar prin declanşatoarele de tip UPDATE CASCADE la actualizarea atributelor `CASETE.IdCasetă` şi `FILME.IdFilm`.

Iată în listing 6.29 cele trei declanşatoare ale tabelei CASETE_FILME, împreună cu pachetul necesar declarării variabilelor globale.

Listing 7.29. Un pachet şi patru declanşatoare PL/SQL pentru gestionarea valorilor atributului FilmNr din CASETE_FILME

```
-----------------------------------------------------------------
-- declansatorul de inserare
CREATE OR REPLACE TRIGGER trg_casete_filme_ins
   BEFORE INSERT ON casete_filme FOR EACH ROW
DECLARE
   v_filmnr casete_filme.FilmNr%TYPE ;
BEGIN
   SELECT MAX(FilmNr) INTO v_filmnr FROM casete_filme
   WHERE IdCaseta = :NEW.IdCaseta ;
```

```
   :NEW.FilmNr := NVL(v_filmnr,0) + 1 ;
END ;
/

-----------------------------------------------------------------------
-- pachetul cu variabile publice
CREATE OR REPLACE PACKAGE pac_centru AS
   -- variabila pentru semnalizarea locului de actualizare
   --    in CASETE_FILME
   vp_trg_casete_filme_del BOOLEAN := FALSE ;
   vp_idcaseta casete.IdCaseta%TYPE ;
   vp_filmnr casete_filme.FilmNr%TYPE ;

   -- variabile pentru semnalizarea modificarilor in cascada
   vp_trg_casete_upd BOOLEAN := FALSE ;
   vp_trg_filme_upd BOOLEAN := FALSE ;

END pac_centru ;
/

-----------------------------------------------------------------------
-- declansator de stergere LA NIVEL DE LINIE
CREATE OR REPLACE TRIGGER trg_casete_filme_del_after_row
   AFTER DELETE ON casete_filme FOR EACH ROW
DECLARE
   v_filmnrcrt casete_filme.FilmNr%TYPE ;
BEGIN
   pac_centru.vp_trg_casete_filme_del := TRUE ;
   pac_centru.vp_idcaseta := :OLD.IdCaseta ;
   pac_centru.vp_filmnr := :OLD.FilmNr ;
   pac_centru.vp_trg_casete_filme_del := TRUE ;
END;
/

-- declansatorul de stergere LA NIVEL DE COMANDA
CREATE OR REPLACE TRIGGER trg_casete_filme_del_stat
   AFTER DELETE ON casete_filme
DECLARE
   v_filmcrt casete_filme.FilmNr%TYPE ;

   CURSOR c_casete_filme (idcaseta_ casete.IdCaseta%TYPE,
             filmcrt_ casete_filme.FilmNr%TYPE)
      IS SELECT * FROM casete_filme
       WHERE IdCaseta = idcaseta_ AND FilmNr >= filmcrt_
         ORDER BY FilmNr ;
BEGIN
   pac_centru.vp_trg_casete_filme_del := TRUE ;
   v_filmcrt := pac_centru.vp_filmnr ;

   FOR rec_casete_filme IN c_casete_filme (pac_centru.vp_idcaseta,
         pac_centru.vp_filmnr) LOOP
      IF rec_casete_filme.FilmNr = v_filmcrt THEN
         UPDATE casete_filme
         SET FilmNr = 29
         WHERE IdCaseta=rec_casete_filme.IdCaseta AND
               FilmNr=rec_casete_filme.FilmNr ;
```

```
      ELSE
         UPDATE casete_filme
         SET FilmNr = v_filmcrt
         WHERE IdCaseta = rec_casete_filme.IdCaseta AND
               FilmNr=rec_casete_filme.FilmNr ;
         v_filmcrt := v_filmcrt + 1 ;
      END IF ;
   END LOOP;

   pac_centru.vp_trg_casete_filme_del := FALSE ;
END;
/

-----------------------------------------------------------------
-- declansator de modificare
CREATE OR REPLACE TRIGGER trg_casete_filme_upd
   BEFORE UPDATE ON casete_filme FOR EACH ROW
DECLARE
   v_filmnrcrt casete_filme.FilmNr%TYPE ;
BEGIN
   IF :NEW.IdCaseta <> :OLD.IdCaseta AND
         pac_centru.vp_trg_casete_upd = FALSE THEN
      RAISE_APPLICATION_ERROR(-20789,
         'Nu puteti modifica interactiv IdCaseta !');
   END IF;
   IF :NEW.IdFilm <> :OLD.IdFilm AND
         pac_centru.vp_trg_filme_upd = FALSE THEN
      RAISE_APPLICATION_ERROR(-20788,
         'Nu puteti modifica interactiv IdFilm !');
   END IF;
   IF :OLD.FilmNr <> :NEW.FilmNr AND
         pac_centru.vp_trg_casete_filme_del = FALSE THEN
      RAISE_APPLICATION_ERROR(-20787,
         'Nu puteti modifica interactiv FilmNr !');
   END IF;
END;
/
```

Nici n-am apucat bine să ne despărţim de tratarea cheilor surogat, că ne şi întoarcem la ele, de data aceasta punându-ne problema actualizării lor. În mod obişnuit, într-o restricţie referenţială modificarea valorii cheii primare (alternative) atrage după sine necesitatea modificării automate a valorilor tuturor cheilor străine ale acesteia. În stardardele SQL şi în unele SGBD-uri există, la creare, opţiunea ON UPDATE CASCADE. Oracle, însă, este mai zgârcit în privinţa asta, aşa că operaţiunea de modificare în cascadă trebuie întreprinsă prin declanşatoarele de modificare ale tabelei părinte.

Dacă în cazul atributui `CLIENTI.CNPClient` nu se întrezăreşte nici o problemă, pentru tabelele ce folosesc chei surogate gestionate prin secvenţă apare un necaz. Să presupunem că în tabela CINEAŞTI, ultima linie introdusă are valoarea `IdCineast` egată cu 344, iar secvenţa `SEQ_IDCINEAST` s-a oprit pe această „poziţie”. În acest moment se lansează o comandă UPDATE prin care Id-ul 256 este făcut 345:

```
UPDATE cineasti SET IdCineast = 345 WHERE IdCineast=256;
```

Necazul apare la următoarea comandă INSERT în CINEAŞTI, deoarece valoarea „secretată" de secvență se suprapune pentru valoarea stabilită prin UPDATE, iar restricția de cheia primară va fi încălcată. De aceea, în declanşatoarele de modificare în cascadă a atributelor cheilor surogat se va testa în prealabil dacă noua valoare depăşeşte limita actuală a secvenței – vezi listing 7.30.

Listing 7.30. Declanşatoarele pentru propagarea modificărilor cheilor primare în tabelele copil

```
-----------------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_cineasti_upd
   AFTER UPDATE OF IdCineast ON cineasti FOR EACH ROW
DECLARE
   v_urmatorulid cineasti.IdCineast%TYPE ;
BEGIN
   SELECT last_number INTO v_urmatorulid
   FROM user_sequences
   WHERE sequence_name='SEQ_IDCINEAST' ;
   IF :NEW.IdCineast >= v_urmatorulid THEN
      RAISE_APPLICATION_ERROR(-20779,
      'Valoarea IdCineast depaseste valoarea curenta a secventei !');
   ELSE
      UPDATE distributie SET Actor=:NEW.IdCineast
         WHERE Actor=:OLD.IdCineast ;
      UPDATE premii_interpretare SET Actor=:NEW.IdCineast
         WHERE Actor=:OLD.IdCineast ;
      UPDATE producatori SET Producator=:NEW.IdCineast
         WHERE Producator=:OLD.IdCineast ;
      UPDATE regizori SET Regizor=:NEW.IdCineast
         WHERE Regizor=:OLD.IdCineast ;
      UPDATE scenaristi SET Scenarist=:NEW.IdCineast
         WHERE Scenarist=:OLD.IdCineast ;
   END IF ;
END ;
/

-----------------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_filme_upd
   AFTER UPDATE OF IdFilm ON filme FOR EACH ROW
DECLARE
   v_urmatorulid filme.IdFilm%TYPE ;
BEGIN
   SELECT last_number INTO v_urmatorulid
   FROM user_sequences
   WHERE sequence_name='SEQ_IDFILM' ;
   IF :NEW.IdFilm >= v_urmatorulid THEN
      RAISE_APPLICATION_ERROR(-20778,
         'Valoarea IdFilm depaseste valoarea curenta a secventei !');
   ELSE
      pac_centru.vp_trg_filme_upd := TRUE ;
      UPDATE distributie SET IdFilm=:NEW.IdFilm
         WHERE IdFilm=:OLD.IdFilm ;
      UPDATE distributie SET IdFilm=:NEW.IdFilm
```

```
        WHERE IdFilm=:OLD.IdFilm ;
      UPDATE premii_filme SET IdFilm=:NEW.IdFilm
        WHERE IdFilm=:OLD.IdFilm ;
      UPDATE premii_interpretare SET IdFilm=:NEW.IdFilm
        WHERE IdFilm=:OLD.IdFilm ;
      UPDATE producatori SET IdFilm=:NEW.IdFilm
        WHERE IdFilm=:OLD.IdFilm ;
      UPDATE regizori SET IdFilm=:NEW.IdFilm
        WHERE IdFilm=:OLD.IdFilm ;
      UPDATE scenaristi SET IdFilm=:NEW.IdFilm
        WHERE IdFilm=:OLD.IdFilm ;
      UPDATE filme_genuri SET IdFilm=:NEW.IdFilm
        WHERE IdFilm=:OLD.IdFilm ;
      UPDATE casete_filme SET IdFilm=:NEW.IdFilm
        WHERE IdFilm=:OLD.IdFilm ;
      UPDATE aprecieri_filme SET IdFilm=:NEW.IdFilm
        WHERE IdFilm=:OLD.IdFilm ;
      pac_centru.vp_trg_filme_upd := FALSE ;
   END IF ;
END ;
/

-----------------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_casete_upd
   AFTER UPDATE OF IdCaseta ON casete FOR EACH ROW
DECLARE
   v_urmatorulid casete.IdCaseta%TYPE ;
BEGIN
   SELECT last_number INTO v_urmatorulid
   FROM user_sequences
   WHERE sequence_name='SEQ_IDCASETA' ;
   IF :NEW.Idcaseta >= v_urmatorulid THEN
      RAISE_APPLICATION_ERROR(-20778,
      'Valoarea IdCaseta depaseste valoarea curenta a secventei !');
   ELSE
      pac_centru.vp_trg_casete_upd := TRUE ;
      UPDATE casete_filme SET IdCaseta=:NEW.IdCaseta
        WHERE Idcaseta=:OLD.Idcaseta ;
      UPDATE casete_imprumutate SET IdCaseta=:NEW.IdCaseta
        WHERE Idcaseta=:OLD.Idcaseta ;
      pac_centru.vp_trg_casete_upd := FALSE ;
   END IF ;
END ;
/

-----------------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_clienti_upd
   AFTER UPDATE OF CNPClient ON clienti FOR EACH ROW
BEGIN
   UPDATE imprumuturi SET CNPClient=:NEW.CNPClient
      WHERE CNPClient=:OLD.CNPClient ;
   UPDATE aprecieri_filme SET CNPClient=:NEW.CNPClient
      WHERE CNPClient=:OLD.CNPClient ;
END ;
/

-----------------------------------------------------------------------
```

```
CREATE OR REPLACE TRIGGER trg_imprumuturi_upd
   AFTER UPDATE OF IdImpr ON imprumuturi FOR EACH ROW
DECLARE
   v_urmatorulid imprumuturi.IdImpr%TYPE ;
BEGIN
   SELECT last_number INTO v_urmatorulid
   FROM user_sequences
   WHERE sequence_name='SEQ_IDIMPR' ;
   IF :NEW.IdImpr >= v_urmatorulid THEN
      RAISE_APPLICATION_ERROR(-20778,
         'Valoarea IdImpr depaseste valoarea curenta a secventei !');
   ELSE
      UPDATE casete_imprumutate SET IdImpr=:NEW.IdImpr
         WHERE IdImpr=:OLD.IdImpr ;
   END IF ;
END ;
/
```

Putem bifa liniştiţi acum problemele ce ţin de restricţiile referenţiale din bază. Ne apropiem uşor de miezul problemei, anume de mecanismul de actualizare a atributelor legate de casetele împrumutate (închiriate) şi restituirea acestora, inclusiv regulile formulate cu privire la gratuităţi şi penalizări. Mecanismul de care vorbim se „învârte" în jurul declanşatoarelor tabelei CASETE_ÎMPRMUTATE. Din raţiuni de spaţiu, ne vom opri doar la declanşatorul de inserare în această tabelă – vezi listing 7.31 – prefaţat de antetul şi specificaţiile pachetului PAC_CENTRU. După cum se poate observa, în pachet au fost introduse o serie de funcţii:

- F_NRCASETEÎNCHIRIATE1 - furnizează numărul de casete închiriate de un client dat;
- F_NRCASETEÎNCHIRIATE2 - furnizează numărul de casete închiriate de un client, cunoscând însă doar identificatorul împrumutui;
- F_CNPCLIENT – returnează codul numeric personal al unui client, cunoscând identificatorul împrumutului;
- F_CHIRIEZI – calculează tariful zilnic de închiriere a unei casete (pe baza anului în care a fost produsă);
- F_PREŢCASETA – returnează preţul la care a fost cumpărată caseta;
- F_DATAÎMPR – returnează data (şi ora) la care a avut loc un împrumut;
- F_CASETADISPONIBILA – returnează TRUE în cazul în care caseta nu este împrumutată, şi FALSE în caz contrar.

Declararea acestor funcţii are ca scop atât micşorarea dimensiunii declanşatoarelor tabelei, şi, implicit, o mai buna lizibilitate şi viteză de execuţie, cât şi modularizarea aplicaţiei, întrucât acestea vor fi necesare şi în (cel puţin) celalte două declanşatoare – de modificare şi ştergere - pe care nu le vom mai discuta.

Listing 7.31. Declanşatoarul de inserare în CASETE_ÎMPRUMUTATE, precedat de pachetul PAC_CENTRU

```
-------------------------------------------------------------------
-- pachetul cu variabile publice se intoarce
CREATE OR REPLACE PACKAGE pac_centru AS
```

```
   -- variabila pentru semnalizarea locului de actualizare
   --    in CASETE_FILME
   vp_trg_casete_filme_del BOOLEAN := FALSE ;
   vp_idcaseta casete.IdCaseta%TYPE ;
   vp_filmnr casete_filme.FilmNr%TYPE ;

   -- variabile pentru semnalizarea modificarilor in cascada
   vp_trg_casete_upd BOOLEAN := FALSE ;
   vp_trg_filme_upd BOOLEAN := FALSE ;

   -- funcţii
   FUNCTION f_nrcaseteinchiriate1
      (cnpclient_ clienti.CNPClient%TYPE)
      RETURN clienti.NrCaseteImprumutate%TYPE;
   FUNCTION f_nrcaseteinchiriate2 (idimpr_ imprumuturi.IdImpr%TYPE)
      RETURN clienti.NrCaseteImprumutate%TYPE;
   FUNCTION f_cnpclient (idimpr_ imprumuturi.IdImpr%TYPE)
      RETURN clienti.CNPClient%TYPE;
   FUNCTION f_chiriezi (idcaseta_ casete.Idcaseta%TYPE)
      RETURN NUMBER ;
   FUNCTION f_pretcaseta (idcaseta_ casete.IdCaseta%TYPE)
      RETURN casete.PretCumparare%TYPE ;
   FUNCTION f_dataimpr (idimpr_ casete_imprumutate.IdImpr%TYPE)
      RETURN imprumuturi.DataOraImpr%TYPE ;
   FUNCTION f_casetadisponibila (idcaseta_ casete.IdCaseta%TYPE)
      RETURN BOOLEAN ;

END pac_centru ;
/
-------------------------------------------------
-- corpul pachetului
CREATE OR REPLACE PACKAGE BODY pac_centru AS
   FUNCTION f_nrcaseteinchiriate1 (
      cnpclient_ clienti.CNPClient%TYPE)
      RETURN clienti.NrCaseteImprumutate%TYPE
   IS
      V_nrci clienti.NrCaseteImprumutate%TYPE ;
   BEGIN
      SELECT NrCaseteImprumutate INTO v_nrci
      FROM clienti
      WHERE CNPClient = cnpclient_ ;
      RETURN v_nrci ;
   END f_nrcaseteinchiriate1 ;
   --------------------------
   FUNCTION f_nrcaseteinchiriate2 (idimpr_ imprumuturi.IdImpr%TYPE)
      RETURN clienti.NrCaseteImprumutate%TYPE
   IS
      V_nrci clienti.NrCaseteImprumutate%TYPE ;
   BEGIN
      SELECT NrCaseteImprumutate INTO v_nrci
      FROM clienti
      WHERE CNPClient = f_cnpclient(idimpr_) ;
      RETURN v_nrci ;
   END f_nrcaseteinchiriate2  ;
   --------------------------
   FUNCTION f_cnpclient (idimpr_ imprumuturi.IdImpr%TYPE)
      RETURN clienti.CNPClient%TYPE
```

```
IS
   v_cnpclient clienti.CNPClient%TYPE ;
BEGIN
   SELECT CNPClient INTO v_cnpclient
   FROM imprumuturi
   WHERE IdImpr = idimpr_ ;
   RETURN v_cnpclient ;
END f_cnpclient ;
----------------------------
FUNCTION f_chiriezi (idcaseta_ casete.Idcaseta%TYPE)
   RETURN NUMBER
IS
   v_anprodcaseta casete.AnProdCaseta%TYPE ;
BEGIN
   SELECT NVL(AnProdCaseta, EXTRACT (YEAR FROM SYSDATE))
   INTO v_anprodcaseta
   FROM casete WHERE Idcaseta=idcaseta_ ;
   CASE
   WHEN EXTRACT (YEAR FROM SYSDATE) - v_anprodcaseta
           BETWEEN 0 AND 1 THEN
      RETURN 50000 ;
   WHEN EXTRACT (YEAR FROM SYSDATE) - v_anprodcaseta
           BETWEEN 2 AND 3 THEN
      RETURN 40000  ;
   ELSE
      RETURN 30000 ;
   END CASE ;
END f_chiriezi ;
----------------------------
FUNCTION f_pretcaseta (idcaseta_ casete.IdCaseta%TYPE)
   RETURN casete.PretCumparare%TYPE
IS
   v_pret casete.PretCumparare%TYPE ;
BEGIN
   SELECT PretCumparare INTO v_pret FROM casete
   WHERE Idcaseta=idcaseta_ ;
   RETURN v_pret ;
END f_pretcaseta ;
----------------------------
FUNCTION f_dataimpr (idimpr_ casete_imprumutate.IdImpr%TYPE)
   RETURN imprumuturi.DataOraImpr%TYPE
IS
   v_data imprumuturi.DataOraImpr%TYPE ;
BEGIN
   SELECT DataOraImpr INTO v_data
   FROM imprumuturi WHERE IdImpr = idimpr_ ;
   RETURN v_data ;
END f_dataimpr ;
----------------------------
FUNCTION f_casetadisponibila (idcaseta_ casete.IdCaseta%TYPE)
   RETURN BOOLEAN
IS
   v_disponibilitate CHAR(1);
BEGIN
   SELECT Disponibilitate INTO v_disponibilitate
   FROM casete WHERE IdCaseta=idcaseta_ ;
   IF v_disponibilitate='D' THEN
```

```
        RETURN TRUE;
      ELSE
        RETURN FALSE ;
      END IF ;
   END f_casetadisponibila ;

END pac_centru ;
/

--------------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_casete_imprumutate_ins
   BEFORE INSERT ON casete_imprumutate FOR EACH ROW

DECLARE
   v_rest NUMBER(2) ; -- restul impartirii la 11 (pt. caseta
                      --          inchiriata gratuit)
BEGIN
   -- se inregistreaza imprumutul unei casete

   -- se creste numarul casetelor imprumutate clientului respectiv
   UPDATE clienti
   SET NrCaseteImprumutate = NrCaseteImprumutate + 1
   WHERE CNPClient = pac_centru.f_cnpclient(:NEW.IdImpr) ;

   v_rest := MOD(pac_centru.f_nrcaseteinchiriate2(:NEW.IdImpr),11) ;

   IF v_rest = 0 THEN
      :NEW.Gratuita := 'D' ;
   ELSE
      :NEW.Gratuita := 'N' ;
      UPDATE imprumuturi SET ValoareInchir = ValoareInchir +
         pac_centru.f_chiriezi(:NEW.IdCaseta)
      WHERE IdImpr = :NEW.IdImpr ;
   END IF ;

   IF :NEW.DataOraRestituirii IS NULL THEN
      -- se verifica disponibilitatea casetei
      IF pac_centru.f_casetadisponibila(:NEW.IdCaseta) = FALSE THEN
         RAISE_APPLICATION_ERROR(-20678, 'Caseta indisponibila') ;
      END IF ;

      -- caseta se declara indisponibila (pina la returnare)
      UPDATE casete SET Disponibilitate='N'
         WHERE IdCaseta = :NEW.IdCaseta ;
      :NEW.penalizare := 0 ;
   ELSE
      -- caseta e deja returnata (se opereaza simultan
      --      inchirierea si returnarea)
      UPDATE casete SET Disponibilitate='D'
         WHERE IdCaseta = :NEW.IdCaseta ;

      -- se incrementeaza numarul casetelor retrnate
      --      de clientul respectiv
      UPDATE clienti
      SET NrCaseteRestituite = NrCaseteRestituite + 1
      WHERE CNPClient IN
         (SELECT CNPClient FROM imprumuturi
```

```
        WHERE IdImpr=:NEW.IdImpr) ;

     -- se verifica intirzierile la returnare (mai mult de 2 zile)
     IF :NEW.DataOraRestituirii -
           pac_centru.f_dataimpr (:NEW.IdImpr) > 2 THEN
        :NEW.penalizare := 0.75 *
           pac_centru.f_chiriezi(:NEW.IdCaseta) *
        (:NEW.DataOraRestituirii -
           pac_centru.f_dataimpr (:NEW.IdImpr) - 2)  ;
     ELSE
        :NEW.penalizare := 0 ;
     END IF ;
   END IF ;

   IF :NEW.StareLaRestituire IN ('Pierduta', 'Distrusa') THEN
     UPDATE casete
     SET StareCaseta = :NEW.StareLaRestituire
     WHERE IdCaseta=:NEW.Idcaseta ;
     :NEW.Penalizare := :NEW.Penalizare +
        2 * pac_centru.f_pretcaseta(:NEW.IdCaseta) ;
   END IF ;

END ;
/
```

În privința declanşatorului propriu-zis, pornim de la premisa că inserarea se poate referi atât la operarea „on-line" a închirierii unei casete, dar şi la operarea unui împrumut mai vechi, pentru care se înregistrează atât închirierea, cât şi restituirea. Logica declanşatorului este următoarea:

- se incrementează numărul casetelor împrumutate de clientul respectiv;
- se calculează dacă numărul respectivei casete împrumutate clientului este multiplu de 11, caz în care închirierea sa este gratuită;
- dacă nu avem cazul de gratuitate de mai sus, se adaugă chiria casetei curente la valoarea pe care trebuie să o plătească clientul pentru împrumutul din care face parte şi caseta curentă;
- dacă se operează împrumutul unei casete ce nu a fost restituită până în momentul inserării (acesta este, de fapt, cazul cel mai frecvent), se verifică dacă se poate închiria caseta (dacă este disponibilă); după operarea împrumutului, caseta se declară indisponibilă (până la restituirea sa);
- dacă inserarea priveşte o casetă împrumutată şi restituită, atunci se incrementează numărul de casete restituite de client, şi apoi se trece la calcularea eventualelor penalități de întârziere sau eventualei penalități de pierdere sau distrugere a casetei.

Din păcate, mai este destul de mult de lucru. Mai trebuie create:

- declanşatoarele de ştergere şi modificare pentru CASETE_ÎMPRUMUTATE;
- declanşatorul de modificare pentru CASETE, întrucât o casetă poate fi pierdută sau distrusă şi de personalul propriu sau datorită unui incendiu,

inundații etc.; pe de altă parte, pentru atributele calculate nu trebuie permisă modificarea interactivă (altfel decât prin declanşatoare);

- declanşatorul de modificare pentru CLIENȚI şi ÎMPRUMUTURI: pentru atributele calculate nu trebuie permisă modificarea interactivă (altfel decât prin declanşatoare);

Pentru detalii privitoare la aceste operațiuni, puteți consulta şi lucrarea [Fotache s.a.03].