Capitolul 8. Generalizări, ierarhii, grafuri, echivalențe

Chiar dacă am tratat şi depedențele de incluziune, pe nedrept omise de la normalizare de majoritatea lucrărilor dedicate proiectării bazelor de date, în capitolul 7 s-a simțit o boare de post-normalizare, întrucât am pornit de la o schemă normalizată pe care am garnisit-o cu elemente ce țin de implementarea unor restricții mai sofisticate, tolerând chiar o doză de redundanță în numele restricțiilor respective. În acest capitol continuăm cu probleme ce nu sunt legate, de obicei, de teoria şi practica normalizării, dar care îşi au importanța lor în elaborarea unei schemei cât mai aproape de dezideratele aplicațiilor din lumea afacerilor sau colaterale. Mai precis, ne vom ocupa, pe rând, de problemele generalizării/specializării, ierarhiilor, grafurilor şi relațiilor de echivalență.

8.1. Generalizare şi specializare

Problema generalizării în bazele de date relaționale a fost abordată înaintea definirii dependențelor de incluziune. Astfel, în două articole publicate în 1977, John şi Diane Smith delimitează două tipuri fundamentale de abstractizare: agregarea şi generalizarea[1]. Agregarea transformă o relație dintre obiecte într-un obiect superior, cei doi autori propunând un nou tip de date care respectă anumite criterii de "bună definire", numit *agregat*[2]. Ei dau ca exemplu relația dintre o persoană, un hotel şi o dată calendaristică ce poate fi abstractizată ca un nou obiect - *rezervare*. Pe noi însă ne interesează cu deosebire conceptul de generalizare care reprezintă genul de abstractizare prin care clase de obiecte sunt transformate într-un obiect "mai general". Altfel spus, o generalizare este o abstractizare care permite unei clase de obiecte individuale să fie gândită, la modul general, ca un obiect în sine[3].

Generalizare şi specializare simplă

Vom discuta, pentru început, unul din exemplele celor doi autori, pentru că structura la care ajung aceştia este una discutabilă. Pentru o companie transnațională, agenție guvernamentală (secretă sau nu) sau minister al apărării se pune problema evidențierii tuturor vehiculelor din dotare. Datorită complexității acestora, se poate apela la o ierarhie precum cea din figura 8.1.

---

[1] [Smith & Smith 77-1], [Smith & Smith 77-2]

[2] [Smith & Smith 77-1]

[3] [Smith & Smith 77-2]

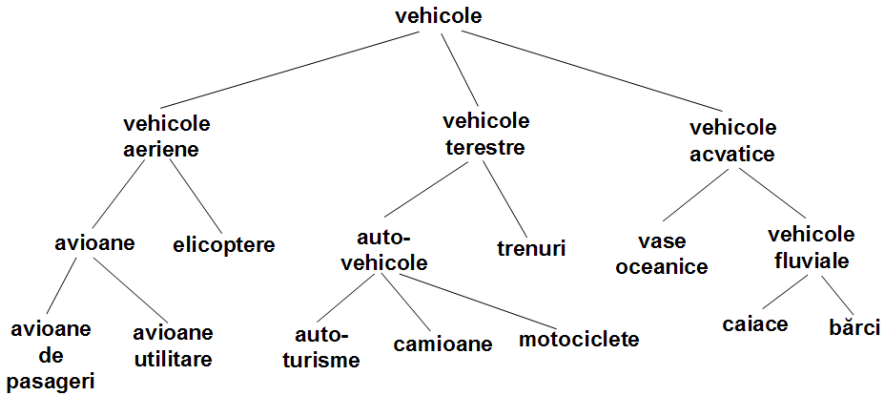

Figura 8.1. O ierarhie de vehicule (adaptare după [Smith&Smith77-2])

Avioanele de pasageri şi cele utilitare pot fi generalizate, definindu-se categoria *avioane*, iar noțiunile *autoturisme*, *camioane* şi *motociclete* pot fi generalizate în categoria *autovehicole*. O parte dintre atribute sunt comune tuturor categoriilor de vehicule: identificatorul vehicolului, denumirea, producătorul, greutatea şi prețul. Acestea sunt definite la nivelul obiectului "rădăcină" - *vehicule* - fiind "moştenite" de toate obiectele subordonate. O altă parte, însă, sunt specifice fiecărei categorii de obiecte:

- pentru vehicule aeriene: înălțime maximă de zbor; viteză maximă; nr. de ore de autonomie de zbor; capacitate rezervor, tip carburant
- pentru elicoptere: tip rotor; numărul paletelor ce alcătuiesc elicea principală;
- pentru avioane: unghi maxim de decolare; lungime minimă a pistei pentru decolare/aterizare;
- pentru avioane de pasageri: număr pasageri;
- pentru avioane utilitare: sarcină utilă;
- pentru vehicole terestre: viteză maximă, tip carburant;
- pentru autovehicole: număr roți; capacitatea cilindrică; cai putere; consum mixt la 100 Km; presiunea în pneuri;
- pentru trenuri: sarcină maximă remorcabilă; autonomie de mers etc.
- pentru autoturisme: număr portiere; număr locuri;
- pentru camioane: sarcină utilă; număr de locuri în cabină;
- pentru motociclete: tip transmisie (lanț sau cardan).

După cum notează şi autorii, acest gen de ierahie este unul de tip arbore, iar toți descendenții unui obiect de tip "nod" sunt disjuncți. Modul în care sunt construite relațiile din ieharhie este, însă, discutabil. Astfel, autorii propun construirea câte unei relații pentru fiecare nod din ierarhie - un obiect ce are subordonați; astfel, corespunzător ierarhiei din figura 8. 1 schema propusă ar avea structura:

VEHICOLE {IdVehicol, DenVehicol, Producător, Greutate, Preţ, Mediu-Navigare[4]}

VEHICOLE_AERIENE{IdVehicol, DenVehicol, Producător, Greutate, Preţ, OreAutonomie, H_Max_Zbor[5], CapacitRezervor, VitezaMax, TipCarburant, CategorieVA[6]}

AVIOANE {IdVehicol, DenVehicol, Producător, Greutate, Preţ, OreAutonomie, H_Max_Zbor, CapacitRezervor, VitezaMax, TipCarburant, UnghiMaxDecolare, LungPistă}

la care se adaugă VEHICOLE_TERESTRE{...}, AUTOVEHICOLE {...}, VEHICOLE_ACVATICE {...} şi VEHICOLE_FLUVIALE {...}.

Relaţia VEHICOLE va conţine date despre toate vehicolele din bază, VEHICOLE_AERIENE va avea, la un moment dat, un număr de linii egal cu cel al avioanelor din dotare etc. Ceea ce se poate reproşa acestui mod de construire a relaţiilor este gradul mare de redundanţă. Astfel, spre exemplu, denumirea, producătorul, greutatea şi preţul unui avion utilitar apar în trei relaţii. Avantajul ţine de faptul că fiecare relaţie, ce reprezintă o "specializare" a "clasei" VEHICOLE, are o structură şi un conţinut care-i conferă o oarecare independenţă faţă de relaţiile "superioare" (cu un grad mai mare de generalitate) şi "inferioare" (specializări ale sale).

Dacă însă am încerca să folosim graful dependenţelor, în care fiecare nivel ierarhic este reprezentat prin dependenţe de incluziune, am obţine o configuraţie precum cea din figura 8.2. Fiecare tip/sub-tip este identificat prin propriul atribut: IdVehicol $\longrightarrow$ DenVehicol, IdVehicol $\longrightarrow$ Producător, ..., IdVehicolAer $\longrightarrow$ OreAutonomie, IdVehicolAer $\longrightarrow$ CapacitRezervor... De data aceasta, în dependenţele funcţionale, fiecare identificator determină numai caracteristicile proprii obiectului (specializării) pe care îl reprezintă, iar specializarea este semnalată prin dependenţele de incluziune: IdVehicolAer $\subseteq$ IdVehicol, IdAvioane $\subseteq$ IdVehicolAer, IdAvionPasageri $\subseteq$ IdAvioane etc.

[4] MediuNavigare poate avea una dintre valorile: *terestru*, *aer*, *apă*

[5] Înălţimea maximă de zbor

[6] Domeniul atribtului CategorieVA este alcătuit din valorile *avion* şi *elicopter*

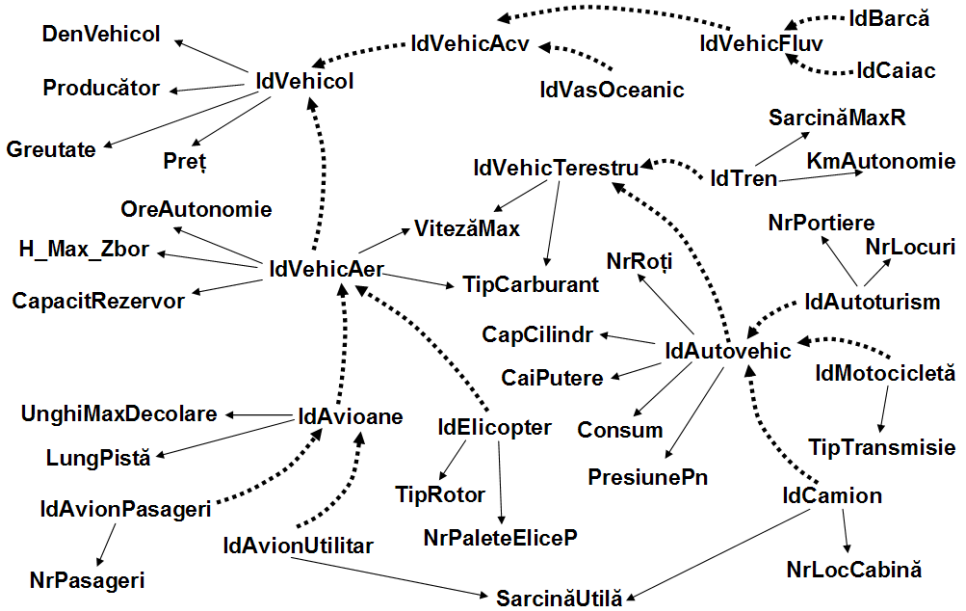

Figura 8.2. Reprezentarea generalizării/specializării prin dependenţe de incluziune

După sistemul deja curent de decupare a relațiilor, izolând fiecare sursă de dependențe, am obține relațiile:

VEHICOLE {IdVehicol, DenVehicol, Producător, Greutate, Preţ}

VEHICOLE_AERIENE {IdVehicAer, OreAutonomie, H_Max_Zbor, Capacit-Rezervor, VitezaMax, TipCarburant}

AVIOANE {IdAvion, UnghiMaxDecolare, LungPistă}

AVIOANE_PASAGERI {IdAvionPasageri, NrPasageri}

AVIOANE_UTILITARE {IdAvionUtilitar, SarcinăUtilă}

ELICOPTERE {IdElicopter, NrPaleteEliceP, TipRotor}

VEHICOLE_TERESTRE {IdVehicTerestru, VitezăMax, TipCarburant}

TRENURI {IdTren, SarcinăMaxR, KmAutonomie}

AUTOVEHICOLE {IdAutovehic, NrRoţi, CapCilindr, CaiPutere, Consum, PresiunePn}

MOTOCICLETE {IdMotocicletă, TipTransmisie}

AUTOTURISME {IdAutoturism, NrPortiere, NrLocuri}

CAMIOANE {IdCamion, NrLocCabină, SarcinăUtilă}

VEHICOLE_ACVATICE {IdVehicAcv}

VASE_OCEANICE {IdVasOceanic}

VEHICOLE_FLUVIALE {IdVehicFluv}

BĂRCI {IdBarcă}

CAIACE {IdCaiac}

Chiar dacă nu este implicată, ca sursă, în nici o dependență funcțională, pentru forice sursă de dependență de incluziune se constituie câte o relație. Aşa "răsar" relațiile VEHICULE_ACVATICE, VASE_OCEANICE, VEHICOLE_FLUVIALE, BĂRCI, CAIACE. Restricțiile referențiale se stabilesc între:

- VEHICOLE_AERIENE.IdVehicolAer (copil) şi VEHICOLE.idVehicol (părinte);
- VEHICOLE_TERESTRE.IdVehicTerestru (copil) şi VEHICOLE.IdVehicol (părinte);
- VEHICOLE_ACVATICE.IdVehicAcv (copil) şi VEHICOLE.IdVehicol (părinte);
- AVIOANE.IdAvion (copil) şi VEHICOLE_AERIENE.IdVehicolAer (părinte);
- AVIOANE_PASAGERI.IdAvionPasageri (copil) şi AVIOANE.IdAvion (părinte);
- AVIOANE_UTILITARE.IdAvionUtilitar (copil) şi AVIOANE.IdAvion (părinte);
- ELICOPTERE.IdElicopter (copil) şi VEHICOLE_AERIENE.IdVehicolAer (părinte);
- TRENURI.IdTren (copil) şi VEHICOLE_TERESTRE.IdVehicTerestru (părinte);
- AUTOVEHICOLE.IdAutovehic (copil) şi VEHICOLE_TERESTRE.IdVehicTerestru (părinte);
- AUTOTURISME.IdAutoturism (copil) şi AUTOVEHICOLE.IdAutovehic (părinte);
- MOTOCICLETE.IdMotocicletă (copil) şi AUTOVEHICOLE.IdAutovehic (părinte);
- CAMIOANE.IdCamion (copil) şi AUTOVEHICOLE.IdAutovehic (părinte);
- VASE_OCEANICE.IdVasOceanic (copil) şi VEHICOLE_ACVATICE.IdVehicAcv (părinte);

- `VASE_FLUVIALE.IdVasFluvial` (copil) şi `VEHICOLE_ACVATICE.IdVehicAcv` (părinte);
- `BĂRCI.IdBarcă` (copil) şi `VASE_FLUVIALE.IdVasFluvial` (părinte);
- `CAIACE.IdCaiac` (copil) şi `VASE_FLUVIALE.IdVasFluvial` (părinte);

Figura 8.3 este mult mai sugestivă, pentru un plus de inteligibilitate unindu-se cheile copil cu cele părinte.

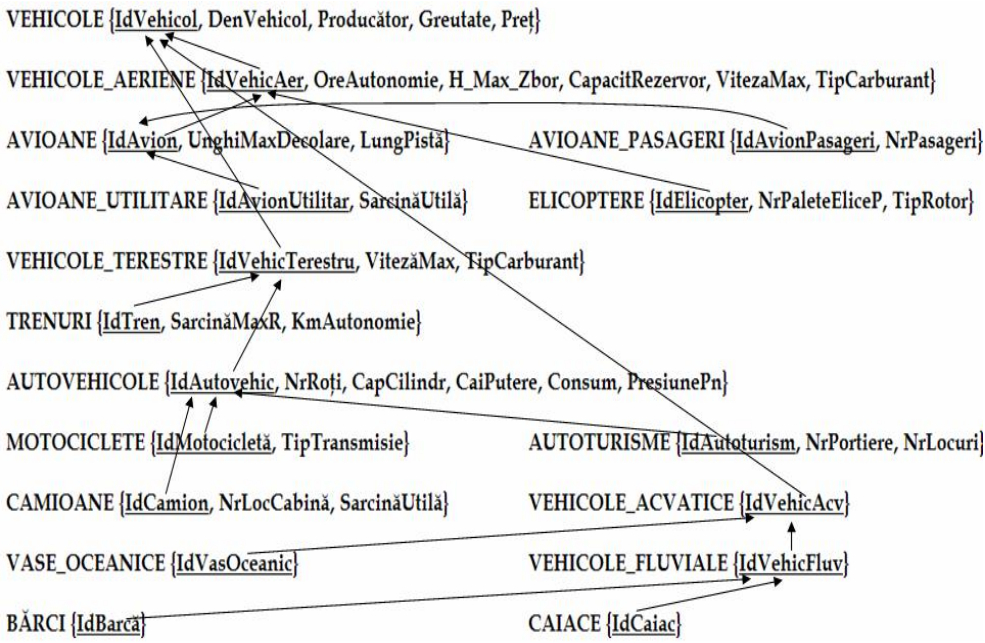

Figura 8.3. Semnalizarea restricţiilor referenţiale

Schema bazei reflectă cu acurateţe specializarea tipurilor de vehicole. Ceea ce i se poate reproşa este fragmentarea excesivă. Putem îndulci această senzaţie, dacă, pentru toţi "părinţii" care au "descendenţi" fără atribute proprii, introducem un atribut de genul *TipCopil*. Astfel, în loc de triada VEHICOLE_FLUVIALE {`IdVehicFluv`} - BĂRCI {`IdBarcă`} - CAIACE {`IdCaiac`}, am folosi doar prima relaţie, cu schema VEHICOLE_FLUVIALE {`IdVehicFluv, TipVehicFluv`}, unde `TipVehicFluv` poate lua doar valorile *Barcă* şi *Caiac*. Economia este mai degrabă nesemnificativă în cazul nostru, deşi ceva mai importantă în alte situaţii.

Un alt reproş adresabil schemei ţine de relativa dificultate în obţinerea unor informaţii. Unele interogări sunt relativ simple, altele necesită destul de multe joncţiuni. De exemplu, dacă dorim să obţinem lista producătorilor de bărci, interogarea folosită ar fi cât se poate de simplă:

```
SELECT DISTINCT Producător
FROM bărci b
       INNER JOIN vehicole v ON b.idbarcă = v.idvehicol
```

Dacă însă interesează *ce categorii de vehicule produce firma X*, lucrurile se complică întrucâtva, nu atât ca sintaxă şi dificultate, cât din punctul de vedere al lungimii interogării:

```
SELECT DISTINCT 'Avioane pasageri' AS Categorie
FROM vehicole v INNER JOIN avioane_pasageri ap
    ON v.idvehicol = ap.idavionpasageri
WHERE producător='X'
UNION
SELECT DISTINCT 'Avioane utilitare' AS Categorie
FROM vehicole v INNER JOIN avioane_utilitare au
    ON v.idvehicol = au.idavionutilitar
WHERE producător='X'
UNION
SELECT DISTINCT 'Elicoptere' AS Categorie
FROM vehicole v INNER JOIN elicoptere e
    ON v.idvehicol = e.idelicopter
WHERE producător='X'
UNION
SELECT DISTINCT 'Autoturisme' AS Categorie
FROM vehicole v INNER JOIN autoturisme a
    ON v.idvehicol = a.idautoturism
WHERE producător='X'
UNION
SELECT DISTINCT 'Camioane' AS Categorie
FROM vehicole v INNER JOIN camioane c
    ON v.idvehicol = c.idcamion
WHERE producător='X'
UNION
SELECT DISTINCT 'Motociclete' AS Categorie
FROM vehicole v INNER JOIN motociclete m
    ON v.idvehicol = m.idmotocicletă
WHERE producător='X'
UNION
SELECT DISTINCT 'Trenuri' AS Categorie
FROM vehicole v INNER JOIN trenuri t
    ON v.idvehicol = t.idtren
WHERE producător='X'
UNION
SELECT DISTINCT 'Vase oceanice' AS Categorie
FROM vehicole v INNER JOIN vase_oceanice vo
    ON v.idvehicol = vo.idvasoceanic
WHERE producător='X'
UNION
SELECT DISTINCT 'Caiace' AS Categorie
FROM vehicole v INNER JOIN caiace c
    ON v.idvehicol = c.idcaiac
WHERE producător='X'
UNION
SELECT DISTINCT 'Bărci' AS Categorie
FROM vehicole v INNER JOIN bărci b
    ON v.idvehicol = b.idbarcă
WHERE producător='X'
```

Specializare multiplă

Specific ierarhiei vehiculelor din figura 8.2 este de faptul că toate structura sa este perfect arborescentă, specializarea fiecărui nivel inferior unui nod fiind întotdeauna una simplă. De multe ori, însă, un nod în ierarhie poate împrumuta (sau moşteni) proprietăţi nu numai ale unui singur părinte, ci şi a două sau mai multe noduri. Să examinăm ierarhia din figura 8.4 ce reprezintă structura personalului la o universitate de tipul Universităţii Alexandru Ioan Cuza din Iaşi.

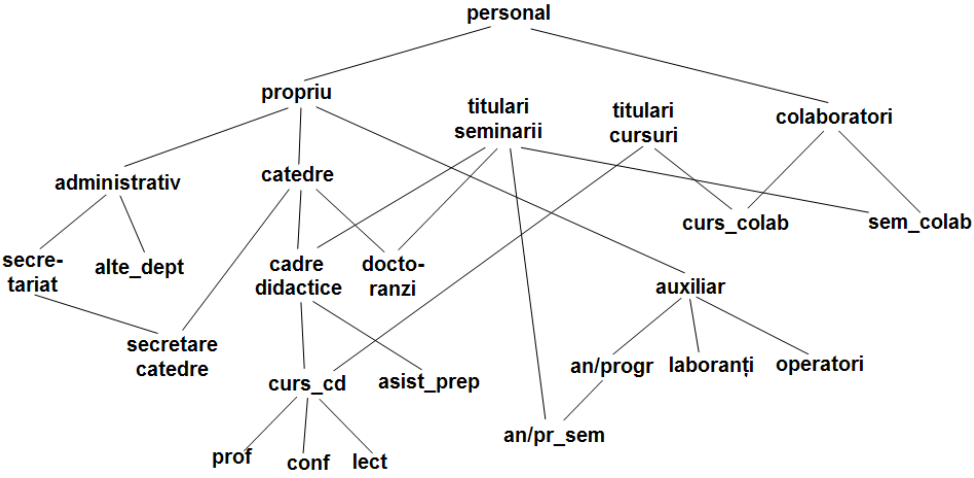

Figura 8. 4. Structura personalului unei universităţi

Activităţile din cadrul universităţii, în principal activităţile didactice, pot fi derulate atât de personalul *propriu* al universităţii, cât şi de *colaboratori* externi, cei care au carte de muncă în alte instituţii sau firme (ex. redactori la reviste de cultură pot preda cursuri la Facultatea de Litere sau Facultatea de Jurnalistică). Cursurile pot fi predate doar de *prof*esori, *conf*erenţiari şi *lect*ori, care sunt grupaţi în "nodul" *curs_cd* (de la *titulari de cursuri dintre cadrele didactice*), la care se adaugă o parte dintre colaboratori, mai precis "partea" *curs_colab* (de la *colaboratori ce pot ţine cursuri*). Seminariile pot fi ţinute de toate cadrele didactice şi doctoranzii afiliaţi catedrelor, dar şi de o subcategorie de analişti/programatori (`an/prog`) numită `an/pr_sem` (de la *analişti/programatori ce pot ţine seminarii*), plus o parte dintre colaboratori - `sem_colab`.

La fiecare catedră există o secretară. Aceasta moşteneşte atât atributele categoriei profesionale din care face parte, SECRETARE: `VitezăTehn` (viteză la tehnoredactarea documentelor, adică număr de cuvinte pe minut), `LimbăStrăină` (fiecare limbă străină pe care o cunoaşte), `NivelLS` (nivel de competenţă pentru limba respectivă), cât şi atributele catedrei de care aparţine: `DenCatedră`, `SediuCatedră`, `TelCatedră`.

Datele generale despre fiecare angajat/colaborator sunt: `CNP` (codul numeric, personal), `NumePren`, `Adresa`, `Telefon`, `EMail`. Pentru toţi colaboratorii interesează locul de muncă (`LocDeMuncă`) şi profesia (`Profesie`), în timp ce

tuturor angajaților proprii li se stochează marca (Marcă), data angajării în universitate (DataAngajării) şi data naşterii (DataNaşterii).

Despre angajații din categoria personalului administrativ (MarcăAdmin), atât secretarele, cât şi cei de la alte departamente, interesează postul pe care sunt încadrați (Post) şi compartimentul universității sau facultatea de care aparțin (Compart_Fac). Secretarele sunt o specializare a personalului administrativ cu cele trei atribute menționate mai sus.

Informațiile specifice personalului auxiliar (MarcăAux) sunt: calificarea (Calificare) şi categoria de salarizare (Categorie), însă, datorită faptului că laboranții şi operatorii nu au alte atribute specifice, celor două atribute li se mai adaugă unul tocmai pentru stabilirea tipului de personal auxiliar din care face parte un asemenea angajat (TipAux). Pentru "subspecia" analişti/programatori (MarcăAP) este important de ştiut în ce platforme software (Software) are competențe fiecare angajat, mai precis nivelul (Nivel) şi numărul de ani de experiență (AniExperiență).

Personalul didactic este structurat pe catedre (IdCatedră), pentru care interesează: denumirea (DenCatedră), camera/biroul ce reprezintă sediul catedrei (SediuCatedră) şi numărul de telefon al catedrei (TelCatedră). Tot în categoria membrilor de catedră intră şi doctoranzii "cu frecvență" (MarcăDrd) care nu sunt cadre didactice propriu-zise, dar pot ține seminarii. Domeniul de doctorat (DomeniuDoctorat) şi anul absolvirii facultății de către doctorand (AnAbsolvFacult) sunt două atribute specifice numai "subclasei" *doctoranzi*.

Pentru toate cadrele didactice dintr-o catedră (MarcăDidact), interesează gradul didactic (GradDidactic) care poate fi: *preparator*, *asistent*, *lector*, *conferențiar*, *profesor*, şi data ultimei promovări (DataUltimeiPromovări). Astfel, sunt preluați în bază toți membrii catedrei, iar calitatea automată a tuturor cadrelor didactice de a ține seminarii este reprezentată prin dependența funcțională MarcăDidact $\longrightarrow$ IdTitularSem. Deoarece numai o parte dintre cadrele didactice pot ține cursuri, se poate recurge la o subclasă a cadrelor didactice numită *curs_cd*, de aici şi identificatorul MarcăDidCurs (marca pentru cadrul didactic ce poate ține cursuri).

Datele specifice fiecărui titular de curs (IdTitularCurs) sunt numărul de discipline la care ține curs (NrCursuri) şi numărul de serii de curs (NrSeriiCurs), în timp ce numărul de grupe de seminar (NrGrupeSem) reprezintă singura informație specifică titularilor de seminarii (IdTitularSem).

Graful tuturor dependențelor funcționale şi de incluziune este cel din figura 8.5.

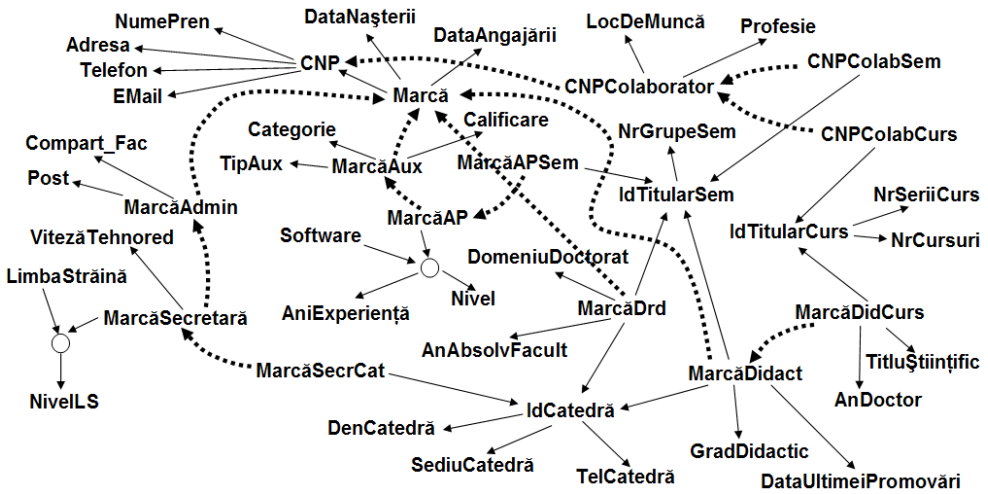

Figura 8.5. Graful dependenţelor pentru baza de date PERSONAL_UNIVERSITATE

Ceea ce ne interesează în cel mai înalt grad este reprezentarea specializărilor, unele simple, altele multiple, ale diferitelor categorii de personal. Specializările simple sunt reprezentate, în general, prin dependenţe de incluziune:

- `CNPColaborator` $\subseteq$ `CNP` înseamnă că valorile codurilor numerice personale pentru colaboratorii externi reprezintă un subset al valorilor codurilor numerice personale pentru toate persoanele avute în evidenţa universităţii;
- `MarcăAdmin` $\subseteq$ `Marcă`, `MarcăAux` $\subseteq$ `Marcă`, `MarcăDidact` $\subseteq$ `Marcă` şi `MarcăDrd` $\subseteq$ `Marcă` ilustrează categoriile principale ale personalului propriu;
- `CNPColabCurs` $\subseteq$ `CNPColaborator` şi `CNPColabSem` $\subseteq$ `CNPColaborator` sugerează specializările categoriei *colaboratori*;
- `MarcăSecretară` $\subseteq$ `MarcăAdmin` indică includerea secretarelor în rândul personalui administrativ;
- `MarcăSecrCat` $\subseteq$ `MarcăSecretară`: valorile mărcilor tuturor secretarelor de catedră se regăsesc în valorile mărcilor secretarelor;
- `MarcăAP` $\subseteq$ `MarcăAux`: analiştii/programatorii sunt personal auxiliar;
- `MarcăAPSem` $\subseteq$ `MarcăAP`: numai o parte dintre analişti/programatori pot ţine seminarii;
- `MarcăDidCurs` $\subseteq$ `MarcăDidact`: numai o parte dintre cadrele didactice dintr-o catedră pot ţine cursuri;

Faptul că un angajat propriu, identificat prin `Marcă`, se regăseşte în clasa-rădăcină *personal* este indicat prin dependenţa funcţională `Marcă` $\longrightarrow$ `CNP`.

Între specializările multiple din graf putem identifica două tipologii. Pe de o parte, faptul că o secretară este arondată unei catedre şi, totodată, face parte din categoria *secretare*, se rezolvă printr-o dependenţă funcţională, `MarcăSecrCat` $\longrightarrow$ `IdCatedră`, coroborată cu una de incluziune, `MarcăSecrCat` $\subseteq$ `MarcăSecre-`

tară. Pe de altă parte, problema fundamentală a ierahiei a fost că două dintre clase, *titulari_cursuri* şi *titulari_seminarii*, au o structură cu totul eterogenă. Astfel titulari de cursuri sunt o parte dintre cadrele didactice, identificate prin marcă, plus o parte dintre colaboratori, la care identificatorul este CNP-ul. Specializarea multiplă s-a relizat prin combinarea a două *dependențe de incluziune*, MarcăDidCurs ⊆ MarcăDidact şi CNPColabCurs ⊆ CNPColaborator, cu două *dependențe funcționale*: CNPColabCurs ⟶ IdTitularCurs şi MarcaDidCurs ⟶ IdTitularCurs. În ambele tipuri de probleme soluția a constituit-o combinația dependențe de incluziune - dependențe funcționale.

Din graf, izolând câte o relație pentru fiecare sursă de DF/DI, obținem structura din figura 8.6. Singura tabelă care nu corespunde unei surse de DF este AN_PROG (analişti-programatori) care este, însă, necesară pentru a reflecta această categorie în cadrul personalului auxiliar, deci pentru îngloba în schemă dependența de incluziune MarcăAP ⊆ MarcăAux.

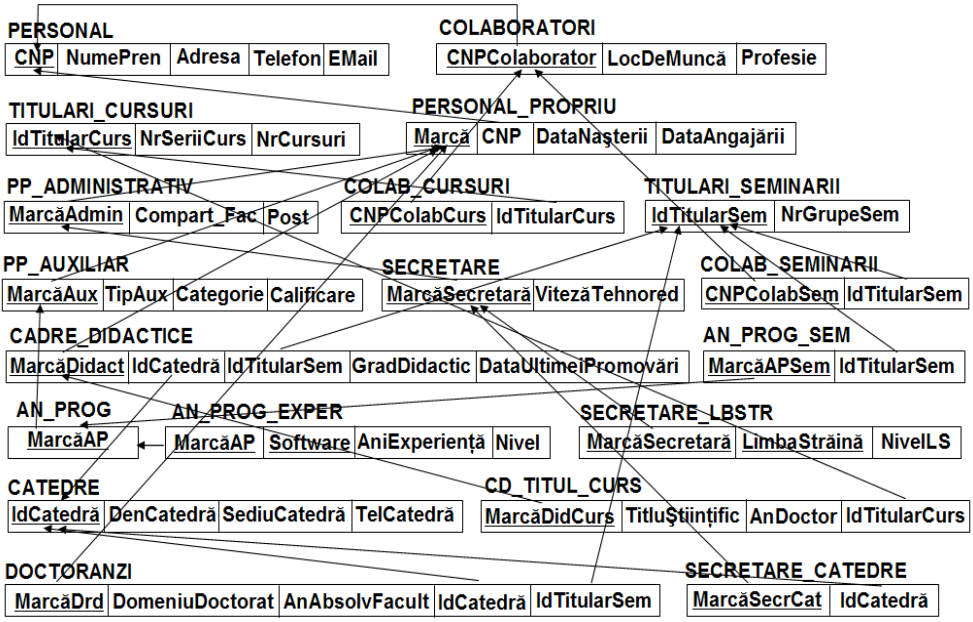

Figura 8.6. Schema bazei de date PERSONAL_UNIVERSITATE

Structura pare a fi una rezonabilă, chiar dacă numărul relațiilor este destul de mare. Este drept, unele operațiuni se traduc prin actualizarea simultană a mai multor tabele. Spre exemplu, un asistent universitar devine, în urma unui concurs, lector (la fel de universitar). Noua categorie didactică îi permite să fie titular de curs (nici nu ar putea fi declarat lector dacă nu ar avea în norma didactică un număr minim de ore de curs). Prin urmare, schimbarea gradului didactic din asistent în lector presupune:

- adăugarea unei linii în tabela TITULARI_CURSURI;

- modificarea valorii atributului `CADRE_DIDACTICE.GradDidactic`;
- inserarea unei linii în tabela CD_TITUL_CURS;

Deşi complicat, acest mod de lucru poate fi gestionat riguros, iar informaţiile furnizate de baza de date sunt conforme cu realitatea.

8.2.Ierarhii generalizate

Problema generalizării/specializării, aşa cum a fost prezentată în paragraful anterior, prezintă importanţă pentru schema bazei de date atunci când:

- numărul palierelor structurii ierarhice nu este prea mare;
- atând nodurile de pe acelaşi nivel ierarhic, cât şi nodurile subordonate au o serie de caracteristici proprii, inaplicabile altor categorii de noduri;
- ierarhia prezintă o considerabilă constanţă în timp.

Cu totul altfel stau lucrurile la structurile ierahice dinamice, în care specializarea poate să se facă pe un număr de niveluri impresionant (de mare). În aceste cazuri trebuie găsite soluţii care să asigure gradul de flexibilitate necesar. Să luăm un caz real. Firmele cu un mare număr de angajaţi şi o structură a posturilor destul de încărcată se confruntă cu una dintre problemele managementului resurselor umane, şi anume gestiunea competenţelor. În funcţie de natura activităţii, firmele pot dezvolta un nomenclator al competenţelor, organizat ierarhic ca în figura 8.7.

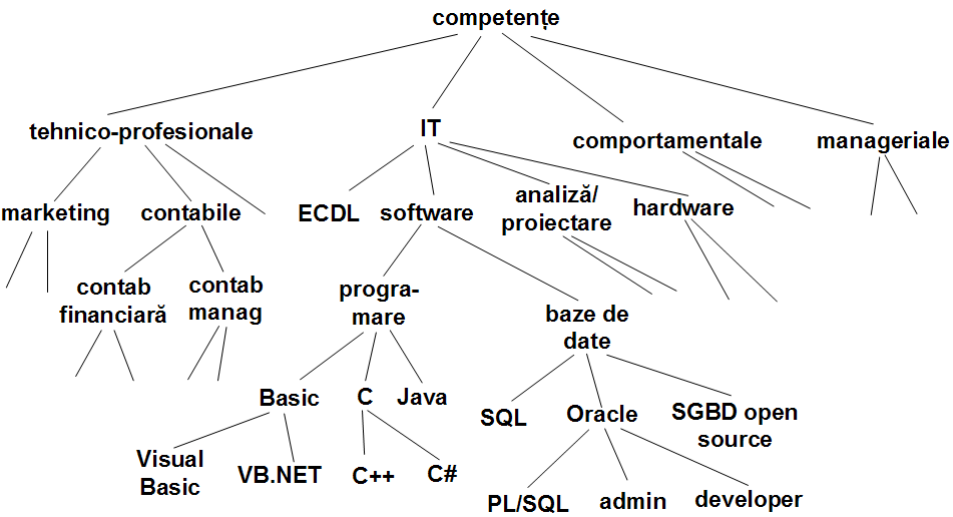

Figura 8.7. O porţiune din ierarhia competenţelor într-o organizaţie

Pentru a păstra lizibilă figura, majoritatea competenţelor nu au mai fost dezvoltate pe sub-categorii. Principalele clase de competenţe sunt:
(1) *Tehnico-profesionale*, cu sub-clasele: *proiectare*, *electrician auto*, *financiar*, *marketing*, *contabilitate* etc. Fiecare poate fi descompusă pe oricâte niveluri, aşa cum, spre

exemplu, competențele contabile se defalcă în *contabilitate financiară* şi *contabilitate managerială*. Competențele ce țin de contabilitatea financiară se pot descompune pe categoriile: *imobilizări*, *stocuri*, *trezorerie* etc.
(2) *IT* - cele mai atent tratate în figură, deşi nu sută la sută riguros. Pentru majoritatea posturilor non-informatice, nivelul ECDL (European Computer Driving Licence - permisul european de "conducere" a calculatorului) este unul suficient, deoarece presupune noțiuni elementare/medii despre: arhitectura calculatoarelor, sisteme de operare (Windows), procesare de texte (Word), foi de calcul (Excel), prezentări (Power Point), baze de date (Access) şi Internet. Posturile informatice reclamă însă o mult mai minuțioasă descompunere, aşa încât au mai fost adăugate trei categorii: hardware, software şi analiză/proiectare. La software interesează cunoştințele de programare şi baze de date, fiecare categorie fiind, la rând său, descompusă.
(3) *Comportamentale* - privesc, printre altele: răbdarea, tenacitatea, abilitățile pentru lucrul în echipă, comunicarea etc.
(4) *Manageriale* - se referă la abilități de antrenare şi motivare, rezolvarea conflictelor, organizarea echipei, managementul proiectelor etc.

Reamintim că schema de mai sus nu este una ideală (nici cu cred că v-ați făcut iluzii), ci reflectă interesele unei organizații la un moment dat. Nu este exclus, spre exemplu, ca la anumite competențe să apară sub-categorii (ex. competențele *IT* → *software* → *baze de date* → *Oracle* → *developer* să fie descompuse în categoria *Forms* şi o alta *Reports*; sau competențele *IT* → *software* → *baze de date* → *SGBD open source* să se decompună în *MySQL*, *PostgreSQL*, *Firebird* etc.)

Avantajul structurii ierahice din figură ține de faptul că fiecare competență, indiferent de tipul şi nivelul pe care se află, se caracterizează prin: denumire (`DenCompetență`), unitate de măsură (`UM`) şi modul în care se face evaluarea sa (`ModEvaluare`). La aceste atribute, mai trebuie adăugate două, unul pentru identificarea competenței (`IdCompetență`), şi un altul ce semnalizează competența superioară (`IdCompSuperioară`). Pe baza acestor atribute se poate cunoaşte în orice moment întreaga arborescență a competențelor. Ca şi la discuțiile din capitolul precedent privitoare la conturile contabile, o parte dintre competențe sunt elementare ("frunză"), în sensul că nu se descompun, în timp ce altele sunt "ramuri" (rîuri...).

În afara plăcerii discuției în sine despre competențe, există şi un motiv plauzibil pentru care am zăbovit asupra acestui subiect: firmele au o organigramă pe care sunt trecute diverse categorii de posturi, iar fiecare post (*ajutor analist-proiectant 1*, *dezvoltator de aplicații 1*, *dezvoltator de aplicații 2*, *administrator de bază de date* etc.), identificat prin atributul `IdPost`, având o denumire (`DenPost`) şi un nivel de salarizare (`Salariu`) stabilite, necesită anumite categorii de competențe, fiecare competență la un nivel minim acceptabil ce depinde chiar de natura postului.

Exemple:

- postul *dezvoltator de aplicații 1* necesită următoarele următoarele competențe:

  - VB.Net - nivel minim acceptat 9 (din 10);
  - SQL - nivel minim acceptat 8;
  - Oracle PL/SQL - nivel minim acceptat 8;
  - abilități de lucru în echipă - nivel minim acceptat 7.
- postul *dezvoltator de aplicații 2* necesită următoarele următoarele competențe:
  - SQL - nivel minim acceptat 7;
  - Oracle PL/SQL - nivel minim acceptat 9;
  - Java - nivel minim acceptat 9;
  - abilități de lucru în echipă - nivel minim acceptat 8.

Ca restricție importantă a bazei, stabilim că descrierea posturilor se realizează exclusiv prin competențe "frunză", adică elementare, altminteri ar fi destul de greu de stabilit ce ar reprezenta nivelul minim acceptat.

Prin combinarea chestiunilor legate de competențe şi posturi, cu câteva elemente sumare ce țin de angajați (marcă, nume, data angajării, compartimentul în care lucrează), compartimente (şeful acestuia) şi evaluarea competențelor angajaților (nivelul), se poate obține un graf al dependențelor precum cel din figura 8.8.

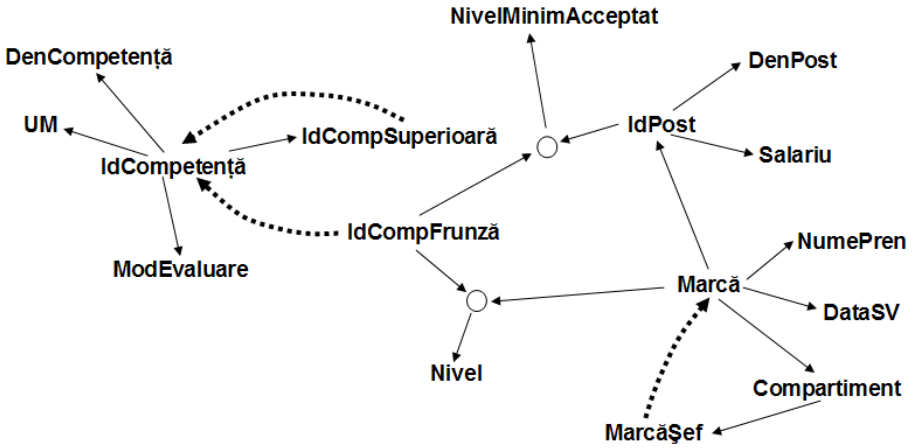

Figura 8.8. Graful DF/DI pentru gestiunea competenţelor pe posturi şi angajaţi

Există trei dependențe de incluziune: prima, pe care am întâlnit-o şi în capitolul anterior, semnifică faptul că valorile `MarcăŞef` sunt un subset ale valorilor atributului `Marcă`. La fel şi `IdCompSuperioară` versus `IdCompetenţă` şi `IdCompFrunză` versus `IdCompetenţă`. Am recurs la "specializarea" `IdCompFrunză` tocmai pentru a gestiona mai bine restricția după care în evaluarea unui post sau angajat pot fi implicate numai competențele elementare.

Nu ne mai rămâne decât să decupăm relațiile şi să le "botezăm", schema fiind cea din figura 8.9. Scriptul Oracle de creare a tabelelor poate fi descărat de pe web sub forma listingului 8.1. Deşi nu este cea mai simplă soluție, ne încăpățânăm să folosim o tabelă specială pentru competențele ce pot fi evaluate şi incluse în

descrierea unui post (COMPETENȚE_ELEMENTARE). Considerațiile sunt similare celor din paragraful 7.4 (vezi figura 7.9).

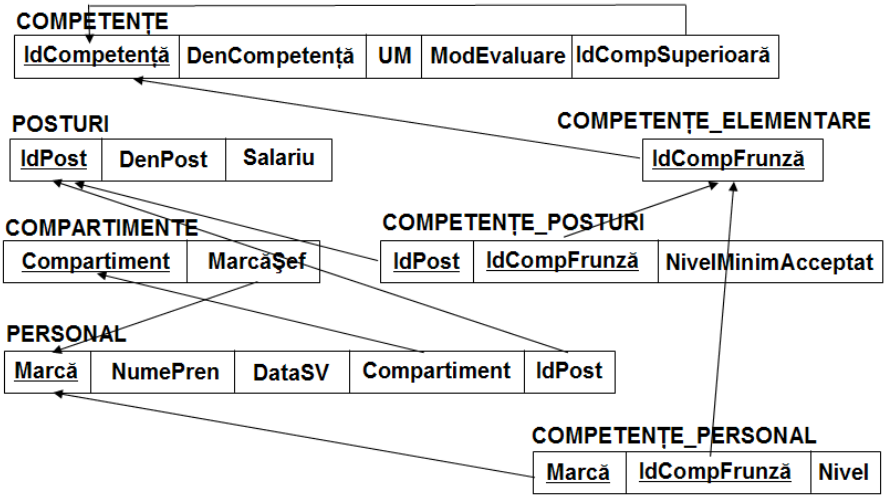

Figura 8.9. Schema obţinută pe baza grafului din figura 8.8

Spre deosebire de planul de conturi, ierarhia competențelor reiese nu din identificator, ci din folosirea unui atribut special dedicat - `IdCompSuperioară`. Ceea ce ne interesează de faptul că, neavând un regim atât de draconic precum înregistrările contabile, am putea folosi o serie de artificii "liber alese".

Să luăm un caz la întâmplare. În ierarhia din figura 8.7 competențele legate de SQL sunt elementare (frunză). Dar, după ceva timp, firma realizează că în unele proiecte are nevoie de SQL-işti care să lucreze cu obiecte, funcții OLAP şi alte opțiuni care sunt prezente începând cu stardardul SQL:1999. Soluția necesită descompunerea competențelor *SQL* în două subcategorii: *SQL-92* şi *SQL:1999*. Bun, dar ce facem cu posturile care au deja descrise deja, de ani buni, în cerințele lor competențe *SQL* ? Plus angajații care au fost evaluați până acum la categoria "frunză" *SQL* ? Ei bine, o idee nu foarte respingătoare este ca, odată cu introducerea primei subcategorii SQL, să inserăm automat categoria *SQL_Veche*, prin care să sugerăm că este o competență elementară artificială, necesară păstrării datelor deja existente în tabelele COMPETENȚE_POSTURI şi COMPETENȚE_PERSONAL. Noua ramificație ar fi cea din figura 8.10.

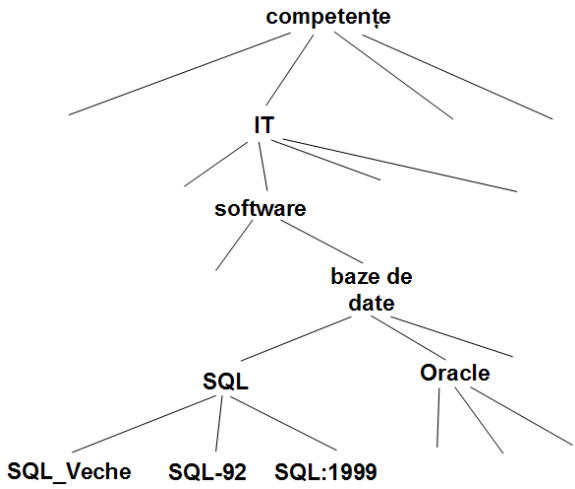

Figura 8.10. Descompunerea "competenţei" SQL

Atributul `IdCompetenţă` a fost gândit ca o cheie surogat, deci scenariul adăugării primei competențe dintre *SQL-92* sau *SQL:1999* ar fi următorul:

- se inserează o înregistrare în COMPETENȚE corespunzătoare *SQL_Veche*, înregistrare care va avea o valoare a `IdCompetenţă` pe care o vom nota `IdCompetenţă(SQL_Veche)` şi, de asemenea:
  - `DenCompetenţă(SQL_Veche) := DenCompetenţă(SQL) || '_Veche'`
  - `UM(SQL_Veche) := UM(SQL)`
  - `ModEvaluare(SQL_Veche) := ModEvaluare(SQL),`
  - `IdCompSuperioară(SQL_Veche) := ICompetenţă(SQL);`
- în tabela COMPETENȚE_ELEMENTARE valoarea `IdCompFrunză(SQL)` se înlocuieşte cu `IdCompetenţă(SQL_Veche)`. Datorită restricțiilor referențiale (sau declanşatorului de modificare a tabelei) se vor modifica în "cascadă", în tabelele COMPETENȚE_POSTURI şi COMPETENȚE_PERSONAL, toate valorile `IdCompetenţă(SQL)` cu `IdCompetenţă(SQL_Veche);`

Implementarea acestui mecanism reclamă folosirea unor funcții pe care le vom creea după modelul Oracle PL/SQL, că şi în alte dăți. Puțin ciudat, poate, nu ne vom apuca de treabă până nu vom fi implementat declanşatoarele de actualizare în cascadă a cheilor primare din tabelele părinte (vezi listing 8.2 descărcabil de pe web). Continuăm cu discutarea primei funcții, *f_este_compfrunză* - vezi listing 8.3 - care primeşte ca argument un identificator de competență şi returnează TRUE în cazul în care competența respectivă este una elementară, şi FALSE în caz contrar.

Listing 8.3. Funcţia PL/SQL F_ESTE_COMPFRUNZĂ

```
CREATE OR REPLACE FUNCTION f_este_compfrunza (
    id_competenta NUMBER) RETURN BOOLEAN
AS
    v_unu NUMBER(1) ;
```

```
BEGIN
    SELECT 1 INTO v_unu FROM dual WHERE EXISTS
        (SELECT 1 FROM competente_elementare
         WHERE IdCompFrunza = id_competenta );
    RETURN TRUE ;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE ;
END ;
```

A doua funcție verifică dacă o competență a fost deja folosită (măcar o dată) la descrierea competențelor unui post sau a unui angajat - F_DEJA_FOLOSITĂ - vezi listing 8.4.

Listing 8.4. Funcţie ce verifică folosirea unei competenţe

```
CREATE OR REPLACE FUNCTION f_deja_folosita (
     id_competenta NUMBER) RETURN BOOLEAN
AS
    v_unu NUMBER(1)  ;
BEGIN
    SELECT 1 INTO v_unu FROM dual WHERE EXISTS
        (SELECT 1 FROM competente_posturi      WHERE IdCompFrunza = id_competenta
      UNION
      SELECT 1 FROM competente_personal WHERE IdCompFrunza = id_competenta
       );
    RETURN TRUE ;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE ;
END ;
```

Cele două funcții ne vor fi de folos în declanşatoarele de inserare în COMPETENȚE (sunt două, unul de tip *before* şi altul *after*) în care se va verifica dacă "părintele" competenței adăugate este, în momentul inserării, competență elementară. În caz că da, se obține identificatorul competenței "artificiu" care se adaugă în tabelă preluând valorile atributele din cele ale "părintelui", apoi se modifică identificatorul în COMPETENȚE_ELEMENTARE.

Listing 8.5. Un pachet şi două declanşatoare pentru inserare în COMPETENŢE

```
CREATE OR REPLACE PACKAGE pac_competente
AS
 v_intra BOOLEAN := TRUE ;
 v_competente competente%ROWTYPE ;
END pac_competente ;
/

---------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_competente_ins_before
 BEFORE INSERT ON competente FOR EACH ROW
BEGIN
  IF pac_competente.v_intra = TRUE THEN
    SELECT seq_idcompetente.NextVal INTO :NEW.IdCompetenta FROM dual ;
    pac_competente.v_competente.IdCompetenta := :NEW.IdCompetenta ;
    pac_competente.v_competente.IdCompSuperioara := :NEW.IdCompSuperioara ;
```

```
   END IF ;
END ;
/

--------------------------------------------------------------------------
CREATE OR REPLACE TRIGGER trg_competente_ins_after
 AFTER INSERT ON competente
DECLARE
  v_id competente.IdCompetenta%TYPE ;
BEGIN
   IF pac_competente.v_intra = TRUE THEN
     IF f_este_compfrunza (pac_competente.v_competente.IdCompSuperioara) THEN
              -- parintele era "frunza" înaintea inserarii
             IF f_deja_folosita (pac_competente.v_competente.IdCompSuperioara) = FALSE THEN
                -- competenta parinte nu apare in evaluarea niciunui post/angajat
                        UPDATE competente_elementare
                        SET IdCompFrunza = pac_competente.v_competente.IdCompetenta
                        WHERE IdCompFrunza = pac_competente.v_competente.IdCompSuperioara ;
          ELSE
         -- competenta parinte este deja folosita in COMPETENTE_POSTURI
              --          sau COMPETENTE_ANGAJATI
             INSERT INTO competente_elementare
                        VALUES (pac_competente.v_competente.IdCompetenta) ;

             SELECT * INTO pac_competente.v_competente FROM competente
             WHERE IdCompetenta = pac_competente.v_competente.IdCompSuperioara ;

             SELECT seq_idcompetente.NextVal INTO v_id FROM dual ;

             pac_competente.v_intra := FALSE ;
             INSERT INTO competente VALUES (
                 v_id, pac_competente.v_competente.DenCompetenta || '_VECHE_',
                pac_competente.v_competente.UM, pac_competente.v_competente.ModEvaluare,
                 pac_competente.v_competente.IdCompetenta) ;
              pac_competente.v_intra := FALSE ;

             UPDATE competente_elementare SET IdCompFrunza = v_id
               WHERE IdCompFrunza = pac_competente.v_competente.IdCompetenta ;
         END IF ;
     ELSE
     -- parintele NU era "frunza" înaintea inserarii
     INSERT INTO competente_elementare
              VALUES (pac_competente.v_competente.IdCompetenta) ;
     END IF ;
  END IF ;
END ;
/
```

Soluția nu poate funcționa dacă nu este implementat mecanismul de actualizare în cascadă a modificării `COMPETENȚE_ELEMENTARE.IdCompFrunză` în tabelele COMPETENȚE_POSTURI şi COMPETENȚE_PERSONAL (prin comanda `CREATE TABLE.... ON UPDATE CASCADE`, acolo unde este posibil, sau prin declanşatoare). Pentru verificarea funcționării declanşatoarelor vă invit să lansați scriptul de populare a tabelelor din bază, descărcabil de pe web de la adresa cunoscută, sub forma listingului 8.6.

În continuare ne propunem să obținem câteva informații interesante din tabelele bazei. Să începem cu răspunsul la întrebarea: Ce competențe necesită postul *Dezvoltator aplicații - 2* ? Soluția este relativ simplă:

```
SELECT DenCompetenta, UM, ModEvaluare, NivelMinimAcceptat
FROM competente_posturi cp
   INNER JOIN posturi p ON cp.IdPost = p.IdPost
   INNER JOIN competente c ON cp.IdCompFrunza = c.IdCompetenta
WHERE DenPost = 'Dezvoltator aplicatii - 2'
```

iar rezultatul poate fi cel din figura 8.11.

DENCOMPETENTA	UM	MODEVALUARE	NIVELMINIMACCEPTAT
Java	puncte	test scris	9
SQL	puncte	test scris	7
PL/SQL	puncte	test scris	9
lucru in echipa	puncte	superior	8

Figura 8.11. Competenţele necesare postului *Dezvoltator aplicaţii - 2*

Să trecem la o interogare ceva mai interesantă: Care sunt cei mai indicați angajați pentru postul *Dezvoltator aplicații - 2* ? Mai întâi, ne permitem o scurtă digresiune pentru a afişa toate competențele necesare acestui post şi punctajele fiecărui angajat evaluat la aceleaşi competențe ca ale postului:

```
SELECT DenPost, DenCompetenta, UM,
      NivelMinimAcceptat AS Punctaj_Minim,
      NumePren, Nivel AS NivelAngajat
FROM competente_posturi cpo
      INNER JOIN posturi p ON cpo.IdPost = p.IdPost
      INNER JOIN competente c
            ON cpo.IdCompFrunza = c.IdCompetenta
      LEFT OUTER JOIN competente_personal cpe
            ON cpo.IdCompFrunza = cpe.IdCompFrunza
      INNER JOIN personal9 p9 ON cpe.marca=p9.marca
WHERE DenPost = 'Dezvoltator aplicatii - 2'
ORDER BY DenPost, DenCompetenta
```

Rezultatul e destul de stufos (vezi figura 8.12), dar ne ajută să înțelegem mai bine ce avem de făcut în continuare.

DENPOST	DENCOMPETENTA	UM	PUNCTAJ_MINIM	NUMEPREN	NIVELANGAJAT
Dezvoltator aplicatii - 2	Java	puncte	9	Stroe Ionut	9
Dezvoltator aplicatii - 2	Java	puncte	9	Babias Vasile	6
Dezvoltator aplicatii - 2	Java	puncte	9	Lemnaru Mihai	6
Dezvoltator aplicatii - 2	Java	puncte	9	Butuca George	5
Dezvoltator aplicatii - 2	Java	puncte	9	Slavici Minodora	5
Dezvoltator aplicatii - 2	Java	puncte	9	Fodor Iuliana	6
Dezvoltator aplicatii - 2	Java	puncte	9	Slabu Florin	7
Dezvoltator aplicatii - 2	PL/SQL	puncte	9	Stroe Ionut	7
Dezvoltator aplicatii - 2	PL/SQL	puncte	9	Babias Vasile	9
Dezvoltator aplicatii - 2	PL/SQL	puncte	9	Lemnaru Mihai	9
Dezvoltator aplicatii - 2	PL/SQL	puncte	9	Butuca George	5
Dezvoltator aplicatii - 2	PL/SQL	puncte	9	Fodor Iuliana	7
Dezvoltator aplicatii - 2	PL/SQL	puncte	9	Slabu Florin	9
Dezvoltator aplicatii - 2	SQL	puncte	7	Stroe Ionut	7
Dezvoltator aplicatii - 2	SQL	puncte	7	Babias Vasile	5
Dezvoltator aplicatii - 2	SQL	puncte	7	Butuca George	6
Dezvoltator aplicatii - 2	SQL	puncte	7	Dediu Cristina	10
Dezvoltator aplicatii - 2	SQL	puncte	7	Fodor Iuliana	6
Dezvoltator aplicatii - 2	SQL	puncte	7	Slabu Florin	4
Dezvoltator aplicatii - 2	SQL	puncte	7	Slavici Minodora	4
Dezvoltator aplicatii - 2	SQL	puncte	7	Lemnaru Mihai	10
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Vasilescu Georgeta	7
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Strat Costel	7
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Fodor Iuliana	5
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Slabu Florin	3
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Butuca George	10
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Lemnaru Mihai	3
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Babias Vasile	5
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Mihuleac Iulian	9
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Stroe Ionut	10
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Irimia Magdalena	9
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Baboi Cosmina	10
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Ailenei Jenica	3
Dezvoltator aplicatii - 2	lucru in echipa	puncte	8	Cosmovici Ionel	2

Figura 8.12. Evaluările angajaţilor la competenţele necesare postului *Dezvoltator aplicaţii - 2*

Pentru a obține rezultatul corect, trebuie să ținem cont de restricția: dacă un angajat nu are nivelul competenței peste cel minim acceptat al postului respectiv, atunci adună zero puncte pentru acea competență:

```
SELECT NumePren, SUM(CASE WHEN Nivel >= NivelMinimAcceptat
  THEN Nivel ELSE 0 END) AS Punctaj
FROM competente_posturi cpo
  INNER JOIN posturi p ON cpo.IdPost = p.IdPost
  LEFT OUTER JOIN competente_personal cpe
      ON cpo.IdCompFrunza = cpe.IdCompFrunza
  INNER JOIN personal9 p9 ON cpe.marca=p9.marca
WHERE DenPost = 'Dezvoltator aplicatii - 2'
GROUP BY NumePren
ORDER BY Punctaj DESC
```

Rezultatul este un top al celor mai "apropiați", din punctul de vedere al competențelor solicitate, de postul *Dezvoltator aplicații - 2* - vezi figura 8.13.

NUMEPREN	PUNCTAJ
Stroe Ionut	26
Lemnaru Mihai	19
Baboi Cosmina	10
Butuca George	10
Dediu Cristina	10
Babias Vasile	9
Mihuleac Iulian	9
Slabu Florin	9
Irimia Magdalena	9
Ailenei Jenica	0
Slavici Minodora	0
Strat Costel	0
Vasilescu Georgeta	0
Cosmovici Ionel	0
Fodor Iuliana	0

Figura 8.13. Topul celor mai nimeriţi angajaţi pentru postul *Dezvoltator aplicaţii - 2*

Iar ca să ne continuăm crescendo-ul, încercăm să răspundem la întrebarea: Care sunt cel mai competenți oameni ai firmei pe locurile pe care sunt angajați ? Interogarea este cel puțin interesantă (vezi listing 8.7 pe web):

```
SELECT NumePren, Punctaj_pe_post, T1.IdPost
FROM
   (SELECT  NumePren, IdPost, SUM(Nivel) AS Punctaj_pe_post
    FROM
       (SELECT NumePren, marca, p9.idpost, IdCompFrunza,
              NivelMinimAcceptat, Nivel
        FROM personal9 p9 INNER JOIN competente_posturi cpo
              ON p9.IdPost=cpo.IdPost
              INNER JOIN competente_personal cpe
                     ON p9.marca=cpe.marca
                     AND cpo.IdCompFrunza=cpe.IdCompFrunza
                     AND NivelMinimAcceptat <= Nivel
       )
    GROUP BY numepren, IdPost) T1 INNER JOIN
       (SELECT IdPost, MAX(Puncte) AS Pct_MAX
        FROM
              (SELECT  IdPost, NumePren, SUM(Nivel) AS Puncte
               FROM
                     (SELECT NumePren, marca, p9.idpost,
                            IdCompFrunza, NivelMinimAcceptat, Nivel
                      FROM personal9 p9 INNER JOIN
                            competente_posturi cpo
                            ON p9.IdPost=cpo.IdPost
                            INNER JOIN competente_personal cpe
                            ON p9.marca=cpe.marca
                            AND cpo.IdCompFrunza=cpe.IdCompFrunza AND
                            NivelMinimAcceptat <= Nivel
                     )
               GROUP BY IdPost, NumePren)
       GROUP BY IdPost
       ) T2 ON T1.IdPost=T2.IdPost AND Punctaj_pe_post =  Pct_MAX
```

Interpretarea rezultatului din figura 8.14 este că, în cazul a două posturi (1021 şi 1022), există doi angajaţi cu acelaşi punctaj (maxim) pe post.

NUMEPREN	PUNCTAJ_PE_POST	IDPOST
Cosmovici Ionel	28	1001
Baboi Cosmina	37	1002
Babias Vasile	19	1021
Lemnaru Mihai	19	1021
Butuca George	10	1022
Dediu Cristina	10	1022
Fodor Iuliana	19	1023

Figura 8.14. Cei mai buni pe postul lor

Dacă în interogare înlocuim funcţia MAX cu MIN (vezi literele îngroşate) obţinem o informaţie cu mult mai interesantă - angajaţii cei mai incompetenţi !

De fapt, în cazul ierarhiilor aşa-zis "generalizate", adică nelimitate ca număr de niveluri, problema stocării este una rezonabilă. Dificultatea cea mai mare ţine de navigarea între niveluri, parcurgerea tuturor descendenţilor unui nod, dată fiind recursivitatea operaţiei. Opţiunile puse la dispoziţie de SGBD-urile actuale sunt relativ izolate, chiar dacă standardul SQL:1999 conţine clauze dedicate tocmai lucrului cu ierarhii (`WITH RECURSIVE... SELECT...`).

Oracle este unul din primele produse care au implementat opţiuni pentru obţinerea de informaţii din ierarhii. Cele două clauze principale sunt `START WITH` prin care se stabileşte linia (linile) "rădăcină" ale rezultatului, adică linia/liniile de la care începe căutarea descendenţilor, şi `CONNECT BY [PRIOR]` pentru specificarea modului în care se construieşte ierarhia. Astfel, pentru a afla competenţele incluse în categoria *baze de date* este suficientă interogarea:

```
SELECT *
FROM competente
START WITH DenCompetenta='baze de date'
CONNECT BY PRIOR IdCompetenta=IdCompSuperioara
```

rezultatul fiind de genul celui din figura 8.15.

IDCOMPETENTA	DENCOMPETENTA	UM	MODEVALUARE	IDCOMPSUPERIOARA
100017	baze de date			100008
100018	SQL	puncte	test scris	100017
100036	SQL-92	puncte	test scris	100018
100037	SQL_VECHE_	puncte	test scris	100018
100019	Oracle	puncte	test scris	100017
100020	PL/SQL	puncte	test scris	100019
100021	administrare	puncte	test Oracle	100019
100022	developer	puncte	test scris	100019
100023	SGBD open source	puncte	test scris	100017

Figura 8.15. Competenţele de tip *baze de date*

Cum modul de afişare al ierarhiei nu este dintre cele mai clare, tot în Oracle putem folosi opțiunea LEVEL (însoțită de funcția LPAD) ce indică nivelul "de adâncime" al descendentului:

```
SELECT IdCompetenta, LPAD('*', 7 * LEVEL, '-')
   || DenCompetenta AS Denumire, UM, ModEvaluare
FROM competente
START WITH DenCompetenta='baze de date'
CONNECT BY PRIOR IdCompetenta=IdCompSuperioara
```

Ceea ce se obține (vezi figura 8.16) este mult mai uşor de înțeles.

IDCOMPETENTA	DENUMIRE	UM	MODEVALUARE
100017	------*baze de date		
100018	----------*SQL	puncte	test scris
100036	--------------------*SQL-92	puncte	test scris
100037	--------------------*SQL_VECHE_	puncte	test scris
100019	----------*Oracle	puncte	test scris
100020	--------------------*PL/SQL	puncte	test scris
100021	--------------------*administrare	puncte	test Oracle
100022	--------------------*developer	puncte	test scris
100023	----------*SGBD open source	puncte	test scris

Figura 8.16. Afişarea arborescentă a competenţelor de tip *baze de date*

La întrebarea: care sunt competențele elementare de tip *baze de date* ?, soluția Oracle este banală:

```
SELECT *
FROM competente
WHERE IdCompetenta IN
   (SELECT IdCompFrunza FROM competente_elementare)
START WITH DenCompetenta='baze de date'
CONNECT BY PRIOR IdCompetenta=IdCompSuperioara
```

O altă categorie importantă de informații priveşte nivelul unei anume competențe sau familii de competențe pe ansamblul organizației. În funcție de datele furnizate de aceste interogări, firma poate lua decizia organizării unor cursuri de "actualizare" sau trimiterea personalului la programe gen master/postuniversitare, cursuri de specializare (Microsoft, Oracle, Cisco etc.). Astfel, interesează la care dintre competențele de tip *baze de date* angajații au cele mai slabe punctaje. Iată şi interogarea ce furnizează în Oracle răspunsul:

```
SELECT DenCompetenta, TRUNC(AVG(Nivel),3) Punctaj_Mediu
FROM
   (SELECT *
    FROM competente
    WHERE IdCompetenta IN
          (SELECT IdCompFrunza FROM competente_elementare)
    START WITH DenCompetenta='baze de date'
```

```
    CONNECT BY PRIOR IdCompetenta=IdCompSuperioara)
       COMP_ELEM_BD INNER JOIN competente_personal cpe
             ON COMP_ELEM_BD.IdCompetenta=cpe.IdCompFrunza
GROUP BY DenCompetenta
ORDER BY Punctaj_Mediu ASC
```

Rezultatul din figura 8.17 indică un oarecare deficit de SQL, ceea ce înseamnă că unii autori (cel puțin unul) nu şi-au dozat prea bine efortul...

DENCOMPETENTA	PUNCTAJ_MEDIU
SQL	6.5
administrare	7.333
PL/SQL	7.666

Figura 8.17. Competenţele de tip *baze de date* cele mai deficitare

8.3. Grafuri

Problema grafurilor este una extrem de generoasă şi complexă, cu puternice ingrediente matematice şi aplicații dintre cele mai diverse. Tratarea noastră va fi însă una simplistă, chiar penibilă pentru cei riguroşi. Singurul avantaj ține doar de gradul (îndoielnic) de aplicabilitate al exemplelor discutate.

8.3.1. Distanțe între localități. Trasee

Începem prin a imagina problema unei firme de transport intern de persoane care acoperă o zonă geografică oarecare din țara noastră, după cum credem că v-au devenit familiare zecile de „maxi-taxi”-uri care cutreieră şoselele noastre în ultimii ani. Figura 8.18 ilustrează localitățile "deservite" de firma cu pricina. La urma urmei, este de presuspus că acest gen de graf este folosit şi de aplicațiile "rutiere" care ne indică variantele de drum între două localități europene sau de aiurea. Prin comparație cu aplicațiile serioase, graful nostru nu indică şi categoria drumului (şi, implicit, viteza maximă cu care se poate circula), ci numai numărul de kilometri dintre două localități. Astfel, variantele de drum pentru traseul Iaşi-Bacău sunt:

- Bacău-Vaslui-Iaşi care are 85 + 71 = 156 de kilometri;
- Bacău-Roman-Vaslui-Iaşi: 41 + 80 + 71 = 192 km;
- Bacău-Roman-Tg.Frumos-Iaşi: 41 + 40 + 53 = 134 km.

Întâmplarea face ca ruta cea mai scurtă să fie şi cea mai rapidă (cu drumul cel mai bun), însă lucrul acesta este valabil doar uneori.

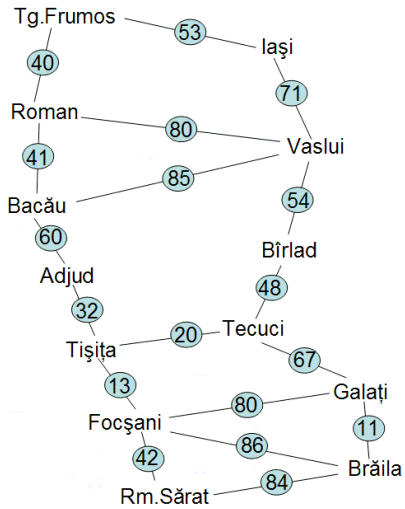

Figura 8.18. Graful a câteva localităţi

Punerea sub formă relaţională a acestor distanţe nu ridică nici un fel de probleme. Listingul 8.8 conţine scriptul de creare şi populare Oracle a tabelei DISTANŢE.

Listing 8.8. Crearea şi popularea tabelei DISTANŢE

```
DROP TABLE distante ;
CREATE TABLE distante
       (Loc1 VARCHAR2(20),
        Loc2 VARCHAR2(20),
        Distanta NUMBER(5),
        PRIMARY KEY (Loc1, Loc2)
        ) ;
INSERT INTO distante VALUES ('Iasi', 'Vaslui', 71) ;
INSERT INTO distante VALUES ('Iasi', 'Tg.Frumos', 53) ;
INSERT INTO distante VALUES ('Tg.Frumos', 'Roman', 40) ;
INSERT INTO distante VALUES ('Roman', 'Bacau', 41) ;
INSERT INTO distante VALUES ('Roman', 'Vaslui', 80) ;
INSERT INTO distante VALUES ('Vaslui', 'Birlad', 54) ;
INSERT INTO distante VALUES ('Birlad', 'Tecuci', 48) ;
INSERT INTO distante VALUES ('Tecuci', 'Tisita', 20) ;
INSERT INTO distante VALUES ('Tisita', 'Focsani', 13) ;
INSERT INTO distante VALUES ('Bacau', 'Adjud', 60) ;
INSERT INTO distante VALUES ('Bacau', 'Vaslui', 85) ;
INSERT INTO distante VALUES ('Adjud', 'Tisita', 32) ;
INSERT INTO distante VALUES ('Focsani', 'Galati', 80) ;
INSERT INTO distante VALUES ('Focsani', 'Braila', 86) ;
INSERT INTO distante VALUES ('Focsani', 'Rm.Sarat', 42) ;
INSERT INTO distante VALUES ('Rm.Sarat', 'Braila', 84) ;
INSERT INTO distante VALUES ('Braila', 'Galati', 11) ;
INSERT INTO distante VALUES ('Tecuci', 'Galati', 67) ;
COMMIT ;
```

Date fiind două localităţi, interesează, mai întâi, ce rute directe şi ocolitoare le unesc. Dacă aveţi cumva impresia că e floare la ureche, iată în figura 8.19 cele mai

scurte 17 variante de drum ar fi pentru a ajunge de la Iaşi la Focşani. Cea mai scurtă rută are 206 km, iar cea mai lungă (dintre cele 17) 468.

RUTA	DISTANTA
*Iasi**Vaslui**Birlad**Tecuci**Tisita**Focsani*	206
*Iasi**Tg.Frumos**Roman**Bacau**Adjud**Tisita**Focsani*	239
*Iasi**Vaslui**Bacau**Adjud**Tisita**Focsani*	261
*Iasi**Vaslui**Roman**Bacau**Adjud**Tisita**Focsani*	297
*Iasi**Tg.Frumos**Roman**Vaslui**Birlad**Tecuci**Tisita**Focsani*	308
*Iasi**Vaslui**Birlad**Tecuci**Galati**Focsani*	320
*Iasi**Vaslui**Birlad**Tecuci**Galati**Braila**Focsani*	337
*Iasi**Tg.Frumos**Roman**Bacau**Vaslui**Birlad**Tecuci**Tisita**Focsani*	354
*Iasi**Tg.Frumos**Roman**Vaslui**Bacau**Adjud**Tisita**Focsani*	363
*Iasi**Vaslui**Birlad**Tecuci**Galati**Braila**Rm.Sarat**Focsani*	377
*Iasi**Tg.Frumos**Roman**Bacau**Adjud**Tisita**Tecuci**Galati**Focsani*	393
*Iasi**Vaslui**Bacau**Adjud**Tisita**Tecuci**Galati**Focsani*	415
*Iasi**Tg.Frumos**Roman**Vaslui**Birlad**Tecuci**Galati**Focsani*	422
*Iasi**Vaslui**Bacau**Adjud**Tisita**Tecuci**Galati**Braila**Focsani*	432
*Iasi**Tg.Frumos**Roman**Vaslui**Birlad**Tecuci**Galati**Braila**Focsani*	439
*Iasi**Vaslui**Roman**Bacau**Adjud**Tisita**Tecuci**Galati**Focsani*	451
*Iasi**Tg.Frumos**Roman**Bacau**Vaslui**Birlad**Tecuci**Galati**Focsani*	468

Figura 8.19. Câteva variante de ajuns de la Iaşi la Focşani

La drept vorbind, mai putem adăuga acestora şi alte rute, mai ocolitoare, însă exemplul este suficient de relevant. Dificultatea problemei ține, şi în acest caz, nu de structura de stocare (banală, dealtminteri), ci de modul de parcurgere a grafului. Soluția pe care o discutăm este una "semi-SQL", în sensul că pentru filtrarea înregistrărilor din tabelele virtuale, corespunzătoare localităților de plecare şi sosire, folosim două variabile publice, `loc_init` şi `loc_dest`. Cum Oracle este mediul în care ne-am apucat să implementăm exemplele noastre, iar în Oracle într-o frază SQL nu pot fi folosite direct variabile, ci funcții (ce returnează tocmai valorile variabilelor publice), am recurs la pachetul PAC_LOC - vezi listing 8.9.

Listing 8.9. Pachetul şi tabelele virtuale pentru trasee

```
CREATE OR REPLACE PACKAGE pac_loc AS
 loc_init distante.loc1%TYPE := 'Iasi';
 loc_dest distante.loc1%TYPE := 'Focsani' ;
 FUNCTION f_loc_init RETURN loc_init%TYPE ;
 FUNCTION f_loc_dest RETURN loc_init%TYPE ;
END  ;
/
-----------------------------------------------
CREATE OR REPLACE PACKAGE BODY pac_loc AS
FUNCTION f_loc_init RETURN loc_init%TYPE IS
BEGIN
 RETURN pac_loc.loc_init ;
END ;

FUNCTION f_loc_dest RETURN loc_init%TYPE IS
BEGIN
 RETURN pac_loc.loc_dest ;
END ;
END ;
/
 ----------------------
DROP VIEW v_ordin1 ;
```

```
CREATE  VIEW v_ordin1 AS
 SELECT loc1, loc2, distanta, '*'|| loc1 || '**' || loc2 ||'*' AS sir
 FROM distante
 WHERE loc1=pac_loc.f_loc_init()
 UNION
 SELECT loc2, loc1, distanta, '*'|| loc2 || '**' || loc1 ||'*' AS sir
 FROM distante
 WHERE loc2=pac_loc.f_loc_init()
WITH READ ONLY ;
---------------------------
DROP VIEW v_ordin2 ;
CREATE  VIEW v_ordin2 AS
SELECT v.loc1, v.loc2, d.loc2 AS loc3, v.distanta + d.distanta AS distanta,
 sir ||'*'|| d.loc2 ||'*' AS sir
FROM v_ordin1 v INNER JOIN distante d ON v.loc2=d.loc1
WHERE INSTR(sir, '*'||d.loc2||'*') = 0 AND v.loc2 <> pac_loc.f_loc_dest()
UNION
SELECT v.loc1, v.loc2, d.loc1, v.distanta + d.distanta, sir ||'*'|| d.loc1 ||'*' AS sir
FROM v_ordin1 v INNER JOIN distante d ON v.loc2=d.loc2
WHERE INSTR(sir, '*'||d.loc1||'*') = 0 AND v.loc2 <> pac_loc.f_loc_dest()
WITH READ ONLY ;
----------------------

--...

----------------------
DROP VIEW v_ordin9 ;
CREATE  VIEW v_ordin9 AS SELECT v.loc1, v.loc2, v.loc3, v.loc4, v.loc5, v.loc6, v.loc7, v.loc8,
 v.loc9, d.loc2 AS loc10, v.distanta + d.distanta AS distanta, sir ||'*'|| d.loc2 ||'*' AS sir
FROM v_ordin8 v INNER JOIN distante d ON v.loc9=d.loc1
WHERE INSTR(sir, '*'||d.loc2||'*') = 0 AND v.loc9 <> pac_loc.f_loc_dest()
UNION
SELECT v.loc1, v.loc2, v.loc3, v.loc4, v.loc5, v.loc6, v.loc7, v.loc8, v.loc9, d.loc1,
     v.distanta + d.distanta, sir ||'*'|| d.loc1 ||'*' AS sir
FROM v_ordin8 v INNER JOIN distante d ON v.loc9=d.loc2
WHERE INSTR(sir, '*'||d.loc1||'*') = 0 AND v.loc9 <> pac_loc.f_loc_dest()
WITH READ ONLY ;

DROP VIEW trasee ;
CREATE VIEW trasee AS
 SELECT sir, distanta FROM v_ordin1
     WHERE loc1= pac_loc.f_loc_init() AND loc2=pac_loc.f_loc_dest()
 UNION
     ...
 SELECT sir, distanta FROM v_ordin9
     WHERE loc1= pac_loc.f_loc_init() AND loc10=pac_loc.f_loc_dest()
WITH READ ONLY ;
```

Soluția conținută în listing nu este din cale-afara de elegantă, bazându-se pe tabele virtuale stratificate prin care se construiesc rutele. Astfel, V_ORDIN1 conține ruta directă dintre cele două localități (dacă există) sau localitățile cu care localitatea de plecare se învecinează. Cum de la Iaşi la Focşani nu este nici un drum direct, cele două linii din exemplul nostru se refereră la distanțele dintre Iaşi şi cele două localități învecinate, Tg.Frumos şi Vaslui - vezi figura 8.20.

```
SQL> SELECT * FROM v_ordin1;

LOC1         LOC2           DISTANTA SIR
------------ ------------ ---------- --------------------------------------------
Iasi         Tg.Frumos            53 *Iasi**Tg.Frumos*
Iasi         Vaslui               71 *Iasi**Vaslui*

SQL>  SELECT * FROM v_ordin2;

LOC1         LOC2         LOC3           DISTANTA SIR
------------ ------------ ------------ ---------- ----------------------------------------------
Iasi         Tg.Frumos    Roman                93 *Iasi**Tg.Frumos**Roman*
Iasi         Vaslui       Bacau               156 *Iasi**Vaslui**Bacau*
Iasi         Vaslui       Birlad              125 *Iasi**Vaslui**Birlad*
Iasi         Vaslui       Roman               151 *Iasi**Vaslui**Roman*

SQL> SELECT * FROM v_ordin3;

LOC1         LOC2         LOC3         LOC4           DISTANTA SIR
------------ ------------ ------------ ------------ ---------- --------------------------------
Iasi         Tg.Frumos    Roman        Bacau               134 *Iasi**Tg.Frumos**Roman**Bacau*
Iasi         Tg.Frumos    Roman        Vaslui              173 *Iasi**Tg.Frumos**Roman**Vaslui*
Iasi         Vaslui       Bacau        Adjud               216 *Iasi**Vaslui**Bacau**Adjud*
Iasi         Vaslui       Bacau        Roman               197 *Iasi**Vaslui**Bacau**Roman*
Iasi         Vaslui       Birlad       Tecuci              173 *Iasi**Vaslui**Birlad**Tecuci*
Iasi         Vaslui       Roman        Bacau               192 *Iasi**Vaslui**Roman**Bacau*
Iasi         Vaslui       Roman        Tg.Frumos           191 *Iasi**Vaslui**Roman**Tg.Frumos*
```

Figura 8.20. Conţinutul primelor trei tabele virtuale în cazul rutei Iaşi-Focşani

A doua tabelă virtuală - V_ORDIN2 - uneşte vecinii Iaşului cu vecinii vecinilor Iaşului ş.a.m.d., iar tabela virtuală TRASEE obţine primele *n* variante, directe şi mai ales ocolitoare, dintre cele două localităţi.

8.3.2. Generarea traseelor

Dacă pentru turişti discuţia de mai sus poate prezenta interes în alegerea unei rute optime dintre două localităţi (deşi un element fundamental în luarea deciziei, şi care ne lipseşte, este categoria de drum şi calitatea sa), pentru firma de transport intern de călători interesul este de a obţine traseele pe care se vor deplasa vehiculele proprii. De aceea, vom salva rutele care interesează sub sub forma a două tabele, RUTE, ce conţine informaţii generale despre punctele terminus ale unui traseu şi distanţa totală dintre ele, şi LOC_RUTE, adică prin (sau pe lângă) ce localităţi trece traseul respectiv. Fiecare rută va fi identificată printr-un atribut de tip cheie surogat, drept care folosim şi o secvenţă - `seq_idruta`. Prin clauza ON DELETE CASCADE ne asigurăm că, la ştergerea unei rute (care e prea ocolitoare sau, oricum, inacceptabilă), se vor şterge toate localităţile de pe traseul respectiv. Listingul 8.10 mai conţine şi declanşatoarele de inserare.

Listing 8.10. Secvenţa, două tabele şi declanşatoare pentru generarea rutelor

```
DROP SEQUENCE seq_idruta ;
CREATE SEQUENCE seq_idruta START WITH 1 MINVALUE 1
    MAXVALUE 555555 ORDER NOCYCLE NOCACHE INCREMENT BY 1;

DROP TABLE loc_rute ;
DROP TABLE rute ;
CREATE TABLE rute (
    idruta NUMBER(6) NOT NULL PRIMARY KEY,
    data_generarii DATE DEFAULT CURRENT_DATE NOT NULL,
```

```
    loc_init VARCHAR(20),
    loc_dest VARCHAR(20),
    sir VARCHAR2(1000)  );

CREATE TABLE loc_rute (
    idruta NUMBER(6) NOT NULL REFERENCES rute (idruta) ON DELETE CASCADE,
    loc_nr NUMBER(4) NOT NULL,
    loc VARCHAR(20) NOT NULL );

CREATE OR REPLACE TRIGGER trg_rute_ins BEFORE INSERT ON rute FOR EACH ROW
BEGIN
 SELECT seq_idruta.NextVal INTO :NEW.idruta FROM dual ;
END ;
/
CREATE OR REPLACE TRIGGER trg_loc_rute_ins BEFORE INSERT ON loc_rute FOR EACH ROW
DECLARE
    v_ultim_nrloc NUMBER(5) := 0 ;
BEGIN
    SELECT MAX(loc_nr) INTO v_ultim_nrloc FROM loc_rute WHERE idruta = :NEW.idruta ;
    IF NVL(v_ultim_nrloc,0) = 0 THEN
        :NEW.loc_nr := 1 ;
    ELSE
        :NEW.loc_nr := v_ultim_nrloc + 1 ;
    END IF ;
END ;
```

Cel al tabelei RUTE este unul banal, vizând atribuirea următoarei valori din secvență atributului cheie surogat - `IdRuta`. Nici cel al tabelei LOC_RUTE nu este din cale-afară de complicat, vizând doar gestionarea corectă a atributului care indică a câta localitate de pe un traseu este localitatea de pe linia curentă.

Listingul 8.11 este dedicat procedurii PL/SQL prin care se generează cele mai importante rute posibile dintre două localități la alegere, localități ce sunt tocmai parametrii procedurii. Pe baza valorilor primite drept parametri de intrare, procedura modifică valoarea celor două variabile publice, apoi parcurge fiecare dintre tabelele virtuale "stratificate" - V_ORDIN1, ..., V_ORDIN*n*, generând câte o rută (inserând o linie în RUTE) pentru fiecare linie a unei tabele virtuale în care localitățile de plecare şi sosire sunt cele indicate de către parametrii de intrare, iar, pentru fiecare rută, se introduce câte o linie în LOC_RUTE pentru orice localitate parcursă.

Listing 8.11. Procedura de generare a rutelor între două localităţi

```
CREATE OR REPLACE PROCEDURE p_generare_rute (loc_init VARCHAR2, loc_dest VARCHAR2)
IS
BEGIN
    pac_loc.loc_init := loc_init ;
    pac_loc.loc_dest := loc_dest ;
    FOR rec_view IN (SELECT * FROM v_ordin1
                        WHERE loc1= pac_loc.f_loc_init() AND loc2=pac_loc.f_loc_dest() ) LOOP
        INSERT INTO rute (loc_init, loc_dest, sir, distanta)
            VALUES (rec_view.loc1, rec_view.loc2, rec_view.sir, rec_view.distanta) ;
        INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc1) ;
        INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc2) ;
    END LOOP ;
```

```
    FOR rec_view IN (SELECT * FROM v_ordin2
                        WHERE loc1= pac_loc.f_loc_init() AND loc3=pac_loc.f_loc_dest() ) LOOP
        INSERT INTO rute (loc_init, loc_dest, sir, distanta)
            VALUES (rec_view.loc1, rec_view.loc3, rec_view.sir, rec_view.distanta) ;
        INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc1) ;
        INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc2) ;
        INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc3) ;
    END LOOP ;

     --...

    FOR rec_view IN (SELECT * FROM v_ordin9
                        WHERE loc1= pac_loc.f_loc_init() AND loc10=pac_loc.f_loc_dest() ) LOOP
        INSERT INTO rute (loc_init, loc_dest, sir, distanta)
            VALUES (rec_view.loc1, rec_view.loc10, rec_view.sir, rec_view.distanta) ;
        INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc1) ;
        INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc2) ;
        INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc3) ;
        INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc4) ;
        INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc5) ;
        INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc6) ;
        INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc7) ;
        INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc8) ;
        INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc9) ;
        INSERT INTO loc_rute (idruta, loc) VALUES  (seq_idruta.CurrVal, rec_view.loc10) ;
    END LOOP ;
END ;
```

Pentru ilustrare, figura 8.21 prezintă modul de lansare a procedurii, localitatea de plecare fiind Iaşi, iar cea de sosire Focşani. Pentru simplificarea discuției, sunt apoi eliminate toate trasele mai lungi de 250 Km, ramânând (după cum se vede în frazele SELECT ce urmează comenzii DELETE) doar două rute, una prin Vaslui-Bîrlad-Tecuci-Tişiţa şi cealaltă prin Tg.Frumos-Roman-Bacău-Adjud. Întrucât la declararea restricției referențiale a fost folosită clauza ON DELETE CASCADE, ştergerea liniilor din RUTE va fi însoțită de ştergerea tuturor liniilor copil. Ultimul SELECT indică tocmai modul de parcurgere a localităților pe cele două rute.

```
SQL>  EXECUTE p_generare_rute ('Iasi', 'Focsani')

PL/SQL procedure successfully completed.

SQL>  DELETE FROM rute WHERE distanta > 250 ;

21 rows deleted.

SQL> SELECT idruta, sir, distanta FROM rute ;

 IDRUTA SIR                                                          DISTANTA
------- ------------------------------------------------------------ ----------
     49 *Iasi**Vaslui**Birlad**Tecuci**Tisita**Focsani*                      206
     50 *Iasi**Tg.Frumos**Roman**Bacau**Adjud**Tisita**Focsani*              239

SQL> SELECT * FROM loc_rute ;

 IDRUTA     LOC_NR LOC
------- ---------- --------------------
     49          1 Iasi
     49          2 Vaslui
     49          3 Birlad
     49          4 Tecuci
     49          5 Tisita
     49          6 Focsani
     50          1 Iasi
     50          2 Tg.Frumos
     50          3 Roman
     50          4 Bacau
     50          5 Adjud
     50          6 Tisita
     50          7 Focsani
```

Figura 8.21. Conţinutul tabelelor RUTE şi LOC_RUTE pentru cele două trasee alese între Iaşi şi Focşani

8.3.3. Rezervări şi gestiunea călătoriilor

Pe drept cuvânt, veți spune că lucrurile pe care le-am discutat în ultimele două paragrafe au prea puțin în comun cu proiectarea bazelor de date. Nu vă contrazicem (prea vehement). Odată ce am prezentat o soluție de "cutreierat" grafurile (deşi nu ne-a cerut-o nimeni), trecem la fondul problemei. Firma şi-a stabilit traseele pe care le va acoperi. Fiecare traseu, numit mai sugestiv *rută* va fi identificat prin celebrul atribut `IdRută`, iar dependențele pot fi scrise astfel:

- `IdRută` ⟶ `DenRută`, `IdRută` ⟶ `LocPlecare`, `IdRută` ⟶ `Destinaţie`;
- (`IdRută`, `LocalitNr`) ⟶ `Loc`;
- (`Loc1`, `Loc2`) ⟶ `Distanţă`;
- `Localitate` ⟶ `Judeţ`;
- `Localitate` ⟶ `Obs` (aici putem indica dacă se merge pe centură sau se intră în oraş, sau alte detalii despre localitate).

Pentru fiecare rută, se stabileşte orarul plecărilor: momentul plecării (`OraPlecare`), dacă circulă sâmbăta sau duminica (`CirculăSD`) şi tipul de vehicul alocat, fiecare tip de vehicul având un anumit număr de locuri disponibile:

- IdPlecare ⟶ IdRuta;
- IdPlecare ⟶ OrăPlecare;
- IdPlecare ⟶ TipAuto;
- TipAuto ⟶ NrLocuri;

Cursele efective care sunt desemnate în limbaj curent în genul *cursa de la 9:00 din 10 ianuarie 2005*; în baza de date, însă, ele sunt identificate printr-o altă cheie surogat, IdCursă:

- IdCursa ⟶ IdPlecare;
- IdCursa ⟶ DataCursă;
- IdCursa ⟶ NrAuto (ex. IS-39-SQL);
- IdCursa ⟶ Şofer;
- NrAuto ⟶ TipAuto;

În timp, din cauza condiţiilor meteo nefavorabile sau lipsei de clienţi, o serie de plecări pot fi suspendate pe o anumită perioadă:

- IdSuspendare ⟶ IdPlecare;
- IdSuspendare ⟶ DataSuspendare;
- IdSuspendare ⟶ DataIniţială;
- IdSuspendare ⟶ DataFinală;

Ei, şi acum vine partea şi mai interesantă: clienţii îşi pot rezerva (telefonic sau pe web) locuri la anumite curse. Fiecare rezervare este identificată printr-o cheie surogat (IdRezervare), iar informaţiile care interesează sunt: data şi ora la care s-a făcut rezervarea (DataOraRez), pe ce nume a fost făcută rezervarea (NumeRez), telefonul la care se poate cere confirmarea (TelRez), numărul de bilete (locuri) solicitate (NrBilete) şi cele două localităţi, de urcare şi coborâre (DeLa, PânăLa):

- IdRezervare ⟶ DataOraRez;
- IdRezervare ⟶ NumeRez;
- IdRezervare ⟶ TelRez;
- IdRezervare ⟶ NrBilete;
- IdRezervare ⟶ DeLa;
- IdRezervare ⟶ PânăLa;

Cu sau fără rezervare, călătorii primesc la urcare (îmbarcare) un bilet, a cărui contravaloare depinde de numărul de kilometri dintre localităţile între care călătoresc, precum şi de tariful pe kilometru corespunzător.

- IdBilet ⟶ DataOraBilet;
- IdBilet ⟶ SerieNrBilet;
- IdBilet ⟶ IdRezervare;
- IdBilet ⟶ DeLa;
- IdBilet ⟶ PânăLa;
- IdBilet ⟶ ValBilet;

Costul pe kilometru se stabileşte pe intervale: de la 0 la 20 Km tariful pe kilometru este de 1500 lei, de la 20 la 40 - 1450, de la 40 la 70 - 1425 etc.:

- (`KmLimitaInf`, `KmLimitaSup`) ⟶ `CostPeKm`;

Dacă mai putem la socoteală şi pe cele de incluziune, obținem un graf al dependențelor precum cel din figura 8.22.

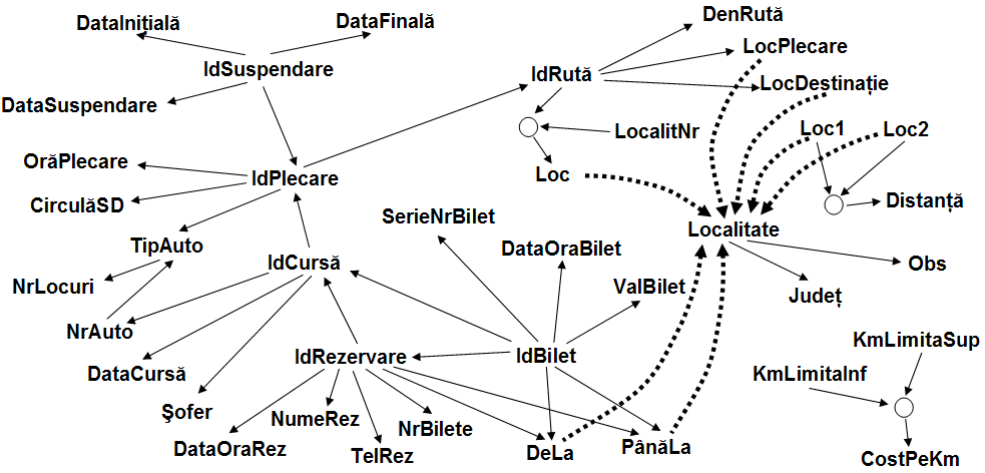

Figura 8.22. Graful dependenţelor pentru baza de date TRANSPORT_INTERN_CĂLĂTORI

Decupând relațiile din graf obținem tabelele din figura 8.23. Scriptul de creare a tabelor şi de definire a restricțiilor (8.12) poate fi descărcat de pe pagina Web indicată în alte dăți.

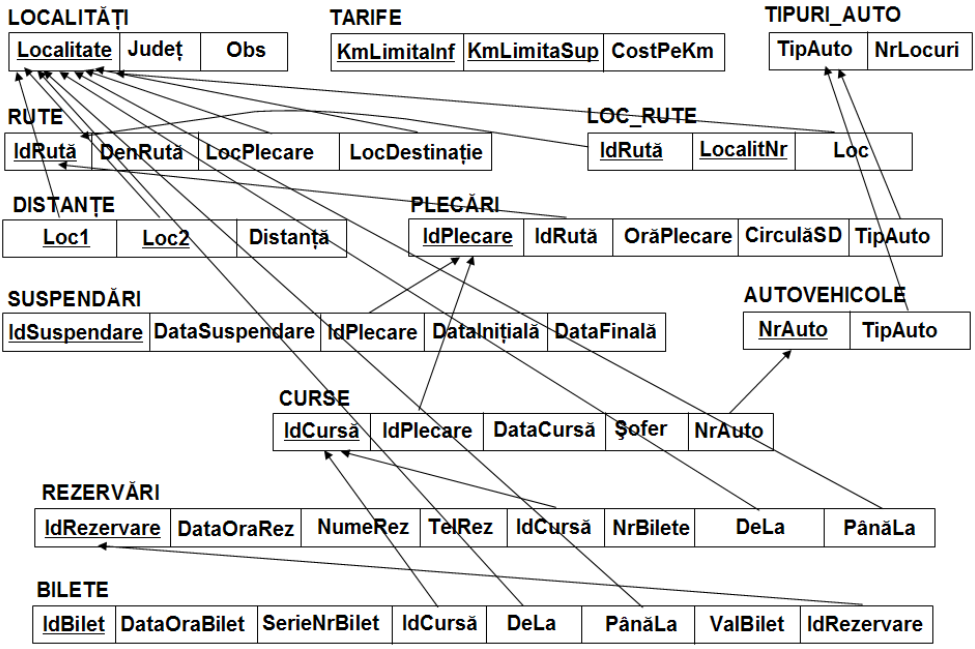

Figura 8.23. Tabelele bazei de date TRANSPORT_INTERN_CĂLĂTORI

Datorită dimensiunilor apreciabile, scriptul de populare a celor mai multe dintre tabele nu a fost inclus în carte, fiind disponibil pe web pe aceeaşi adresă sub forma listingului 8.13. În schimb, putem cheltui câteva cuvinte pe funcția din listingul 8.14, menită a calcula valorea unui bilet. Funcției i se transmit trei parametri: `vLocUrcare` indică stația de urcare a pasagerului, `vLocCoborîre` stația de coborâre, iar `vIdRuta` - ruta pe care o urmează autovehicolul cu care călătoreşte. Pe baza înregistrărilor din tabela LOC_RUTE, se determină a câta localitate de pe rută este cea de urcare (`i_urcare`) şi a câta este cea de coborâre (`i_coborâre`). Cu ajutorul cursorului definit prin structura FOR... END LOOP se parcurg toate localitățile dintre urcare şi coborâre, adunându-se, de fiecare dată, numărul de kilometri în variabila `v_NrKm`.

Listing 8.14. Funcţia pentru calculul preţului unei călătorii

```
CREATE OR REPLACE FUNCTION f_cost_bilet (
    vLocUrcare localitati.Localitate%TYPE,
    vLocCoborire localitati.Localitate%TYPE,
    vIdRuta rute.IdRuta%TYPE ) RETURN NUMBER
IS
    i_urcare NUMBER(3) := 0 ;
    i_coborire NUMBER(3) := 0 ;
    v_nrkm NUMBER(4) := 0 ;
    v_distanta NUMBER(4) := 0 ;
    v_cost tarife.CostPeKm%TYPE := 0;
    v_loc1 localitati.Localitate%TYPE ;
    v_loc2 localitati.Localitate%TYPE ;
BEGIN
```

```
    SELECT LocalitNr INTO i_urcare FROM loc_rute WHERE IdRuta=vIdRuta AND loc=vLocUrcare ;
    SELECT LocalitNr INTO i_coborire FROM loc_rute WHERE IdRuta=vIdRuta AND
loc=vLocCoborire ;

    IF i_urcare >= i_coborire THEN
        RAISE_APPLICATION_ERROR(-20177, 'Ruta/localitati gresite !');
    END IF ;

    FOR rec_loc IN (SELECT * FROM loc_rute WHERE IdRuta=vIdRuta
        AND LocalitNr BETWEEN i_urcare AND i_coborire ORDER BY LocalitNr ) LOOP
        IF rec_loc.LocalitNr = i_urcare THEN
            v_loc1 := rec_loc.Loc ;
        ELSE
            v_loc2 := rec_Loc.Loc ;
            SELECT distanta INTO v_distanta FROM distante
            WHERE loc1 = LEAST(v_loc1, v_loc2) AND
            loc2 = GREATEST(v_loc1, v_loc2) ;
            v_NrKm := v_NrKm + v_distanta ;
            v_loc1 := v_loc2 ;
        END IF ;
    END LOOP ;

    SELECT ROUND(CostPeKm * v_NrKm,-3) INTO v_cost FROM tarife
    WHERE v_NrKm >= KmLimitaInf AND v_NrKm < KmLimitaSup ;

    RETURN v_cost ;
END ;
```

În final, pe baza numărului de kilometri, se calculează contravaloarea biletului prin înmulţirea numărului cu tariful pe kilometru din tranşa corepunzătoare a tabelei TARIFE. Figura 8.24 indică modul de apel în Oracle SQL*Plus a funcţiei pentru a determina costul biletului dintre Iaşi şi Focşani pe varianta Tg.Frumos-Roman-Adjud (IdRută=4) şi varianta Vaslui-Tecuci (IdRută=3).

```
SQL>  begin
  2   dbms_output.put_line(f_cost_bilet('Iasi', 'Focsani', 4));
  3   end ;
  4  /
311000

PL/SQL procedure successfully completed.

SQL> begin
  2  dbms_output.put_line(f_cost_bilet('Iasi', 'Focsani', 3));
  3  end ;
  4  /
268000

PL/SQL procedure successfully completed.

SQL>
```

Figura 8.24. Două apeluri ale funcţiei F_COST_BILET

Schema de stocare la care am ajuns permite rezolvarea şi unor probleme ceva mai delicate. Să ne oprim un pic la rezervările *on* sau *off* line. Un posibil călător sună la dispecerat şi declară că vrea două bilete de la Iaşi la Tecuci pentru cursa de

la ora 6:00 din data de 2 decembrie 2004. Microbuzul alocat acestei curse are 16 locuri. Problema este că, înainte de a confirma rezervarea, aplicaţia trebuie să verifice câte locuri au fost rezervate nu numai între Iaşi şi Tecuci, ci şi în orice interval care cuprinde cele două localităţi, cum ar fi: Vaslui-Bîrlad, Vaslui-Tecuci, Vaslui-Focşani, Bîrlad-Tecuci etc. O soluţie nu cale-afară de rapidă ţine de folosirea unei funcţii care să returneze poziţia unei localităţi pe un traseu (rută), funcţie folosită de declanşatorul de inserare al tabelei REZERVĂRI - vezi listing 8.15.

Listing 8.15. O secvenţă, o funcţie pentru determinarea poziţiei pe o rută a unei localităţi şi declanşatorul de inserare pentru rezervări

```
DROP SEQUENCE seq_IdRezervare ;
CREATE SEQUENCE seq_IdRezervare START WITH 1 INCREMENT BY 1 MINVALUE 1
 MAXVALUE 99999999 NOCYCLE NOCACHE ORDER ;

CREATE OR REPLACE FUNCTION f_ordine_loc (
    Loc_ loc_rute.Loc%TYPE, IdRuta_ loc_rute.IdRuta%TYPE)
    RETURN loc_rute.LocalitNr%TYPE
IS
    v_nr loc_rute.LocalitNr%TYPE ;
BEGIN
    SELECT LocalitNr INTO v_nr FROM loc_rute WHERE IdRuta=IdRuta_ AND loc=Loc_ ;
    RETURN v_nr ;
END ;
/

CREATE OR REPLACE TRIGGER trg_rezervari_ins
    BEFORE INSERT ON rezervari FOR EACH ROW
DECLARE
    v_NrRezervari rezervari.NrBilete%TYPE := 0 ;
    v_NrLocuriAuto rezervari.NrBilete%TYPE := 0 ;
    i_urcare NUMBER(3) := 0 ;
    i_coborire NUMBER(3) := 0 ;
    i NUMBER(3) := 0 ;
    v_IdRuta rute.IdRuta%TYPE ;
BEGIN
    -- In variabile v_NrLocuriAuto se stocheaza capacitatea auto/micro-buzului
    SELECT NrLocuri INTO v_NrLocuriAuto FROM tipuri_auto WHERE TipAuto=
        (SELECT TipAuto FROM autovehicole WHERE NrAuto=
            (SELECT NrAuto FROM curse2 WHERE IdCursa = :NEW.IdCursa)) ;

    -- se determina Id-ul rutei pentru a vedea ce localitati se parcurg
    SELECT IdRuta INTO v_IdRuta
    FROM curse2 c INNER JOIN plecari p ON c.IdPlecare=p.IdPlecare
    WHERE IdCursa=:NEW.IdCursa ;

    -- se obtine numarul localitatii de urcare si al celei de coborire de pe traseu
    i_urcare := f_ordine_loc (:NEW.DeLa, v_IdRuta) ;
    i_coborire :=  f_ordine_loc (:NEW.PinaLa, v_IdRuta) ;
    IF i_urcare >= i_coborire THEN
        RAISE_APPLICATION_ERROR(-20177, 'Ruta/localitati gresite !');
    END IF ;

    -- se parcurge fiecare portiune dintre localitatile de urcare si coborire
    -- si se verifica daca nr. rezervarilor depaseste numarul locurilor din autovehicol
    FOR i IN i_urcare..i_coborire-1 LOOP
        SELECT SUM(NrBilete) INTO v_NrRezervari
```

```
        FROM (SELECT * FROM rezervari WHERE IdCursa=:NEW.IdCursa)
        WHERE f_ordine_loc(DeLa, v_IdRuta) <= i AND f_ordine_loc(PinaLa, v_IdRuta) >= i + 1 ;

        IF NVL(v_NrRezervari,0) + :NEW.NrBilete >= v_NrLocuriAuto  THEN
            RAISE_APPLICATION_ERROR(-20178, 'Nu mai sunt locuri ! ');
        END IF ;
    END LOOP ;

    SELECT seq_IdRezervare.NextVal INTO :NEW.IdRezervare FROM dual ;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR (-20179,
            'Ruta/cursa pentru care se doreste rezervarea nu este introdusa !') ;
END ;
```

În mare, ideea declanşatorului este următoarea: traseul dintre cele două localităţi se rupe pe porţiuni delimitate de localităţile definite în tabela LOC_RUTE, iar pentru fiecare porţiune a traseului se verifică dacă rezervările deja efectuate pentru această cursă, plus numărul locurile solicitate la rezervarea curentă, nu depăşesc numărul locurilor din autovehicol. Soluţia nu pare, însă, prea rapidă, dacă volumul datelor este imens (după ani şi ani de folosire a aplicaţiei).

8.4. Incluziuni & echivalenţe

Continuăm discuţiile din acest capitol cu o altă problemă marginală. Figura 8.4 conţine structura personalului unei instituţii academice, iar interesul nostruu a fost de a ilustra modul în care se poate gestiona o structură ierarhică atunci când ierarhiile simple se combină cu cele multiple. Extindem un pic problema, gândindu-ne la un mini-mecanism de căutare a cărţilor într-o biliotecă, librărie etc., căutare derulată după anumite cuvinte cheie sau, mai elevat spus, *sintagme*. Partea interesantă, dar şi dificilă a discuţiei ţine de faptul că relaţiile în care se află sintagmele unele cu altele sunt, în afară de *independenţă*, nu numai de *incluziune*, simplă şi multiplă, dar şi de *echivalenţă*. Să luăm o porţiune dintr-o "încrengătură" de sintagme - vezi figura 8.25.

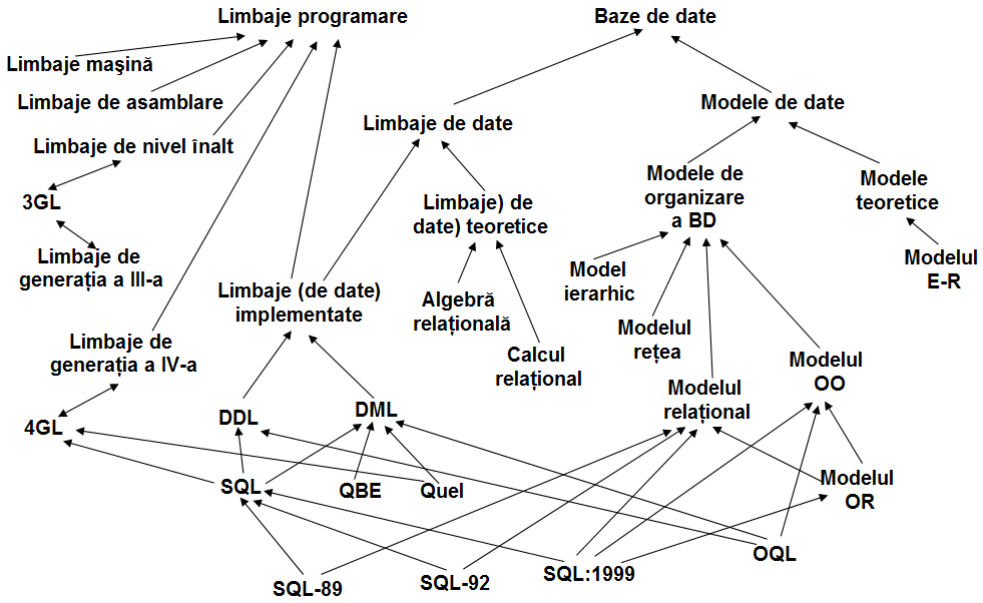

Figura 8.25. Rapoarte de incluziune şi echivalenţă între cuvinte cheie (sintagme)

Astfel, sintagmele: *limbaje maşină*, *limbaje de asamblare*, *limbaje de nivel înalt*, *limbaje de generația a IV-a* şi *limbajele (de date) implementate* sunt, toate, incluse în categoria *limbaje de programare*. Sintagma *SQL-89* ține de două noțiuni, *SQL* şi *modelul relațional*. În plus, sintagma *4GL* este echivalentă cu *limbaje de generația a IV-a*, iar *3GL* este echivalentă atât cu *limbajele de nivel înalt*, cât şi cu *limbajele de generația a III-a*. Scopul acestei organizări alambicate este ca, în funcție de modul în care s-au definit cuvintele cheie pentru cărțile din bibliotecă, la căutarea după o anumită sintagmă să fie indicate nu numai cărțile cu sintagma respectivă, ci şi cele la care au fost definite sintagme echivalente şi sintagme subordonate.

O soluție de rezolvare a acestei probleme ar putea face apel la un "nomenclator" de cuvinte-cheie (sintagme), apoi fiecare incluziune, adică relație sintagmă copil-sintagmă părinte să fie identificată printr-un atribut de tip cheie surogat (`IdIncluziune`) iar, de asemeni, fiecare echivalență dintre două sintagme să fie consemnată prin alocarea unui identificator de acelaşi tip surogat (`IdEchivalență`). Astfel, graful dependențelor funcționale şi de incluziune ar putea arăta ca în figura 8.26. Atributul `RelevanțăISBN` sugerează cât de bine (mult) este tratat subiectul desemnat printr-o sintagmă în cartea curentă, valorile acestuia fiind acordate după sistemul de notare de la 1 la 10, sau de la 1 la 5.

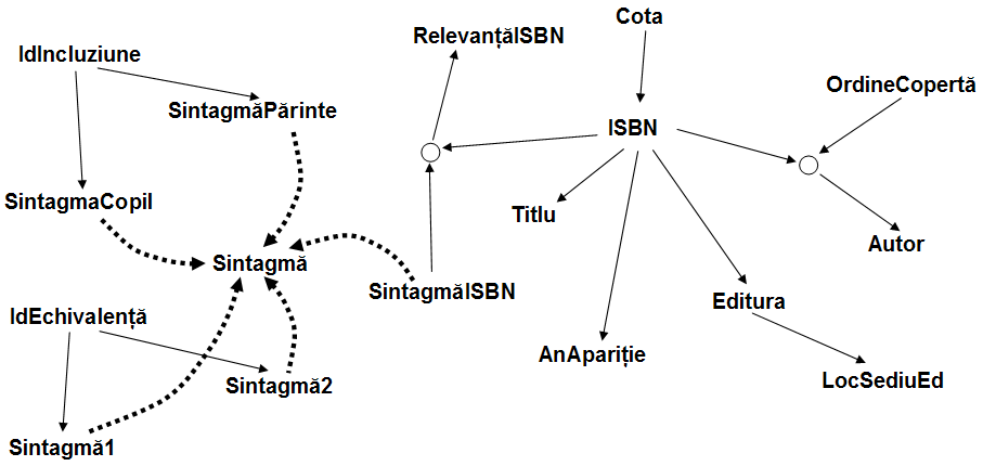

Figura 8.26. Noul graf al dependenţelor pentru BD BIBLIOTECĂ

Decuparea relațiilor din graf este banală, după toată experiența pe care am acumulat-o - vezi figura 8.27.

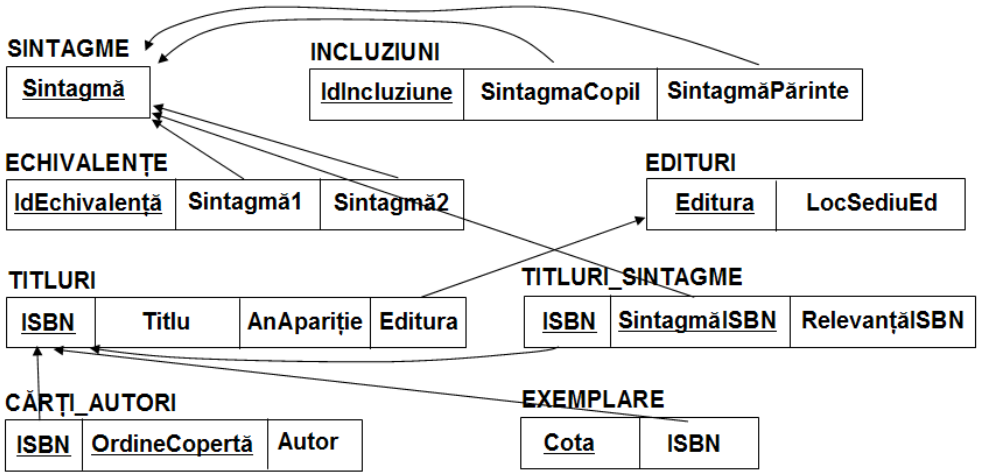

Figura 8.27. Noua schemă a bazei de date BIBLIOTECĂ

Odată pusă la punct problematica stocării relațiilor de echivalență, ca şi în cazul grafurilor dificultatea este pasată mecanismului de „parcurgere” a țesăturii de incluziuni şi echivalențe. Se poate imagina un mecanism la fel de primitiv precum cel folosit la grafuri, de construire a unor tabele virtuale „concentrice” care să furnizeze, pe rând, vecinii sintagmei de plecare, apoi pe vecinii vecinilor acesteia s.a.m.d. Noi însă ne oprim aici cu acest capitol, chiar dacă veți spune că v-am lăsat în mijlocul drumului.