



**Ion Lungu**  
coordonator

**Adela Bâra**

**Constanța Bodea**

**Iuliana Botha**

**Vlad Diaconita**

**Alexandra Florea**

**Anda Velicanu**



# **TRATAT DE BAZE DE DATE**

**BAZE DE DATE**  
**Organizare**  
**Proiectare**  
**Implementare**

**Volumeul 1**

EDITURA  
**ASE**

datele  
limbi  
nepro  
acces

SQL  
PL/SQL  
T-SQL  
MySQL  
Oracle

ER

## 4.1 Modelul relațional

Înțelegerea modelului relațional nu necesită cunoștințe teoretice deosebite, putând fi prezentat cu ajutorul unor concepe extrem de simple și intuitive. De exemplu relația, conceptul central în modelul relațional, poate fi prezentată sub forma unui tabel bidimensional de date. Reprezentarea relației în acest mod este comodă, ușor de înțeles și de utilizat, în special în cadrul operațiilor asupra datelor.

Modelul are nu numai adepti, ci și critici, care îi contestă relevanța și utilitatea în viitoarele infrastructuri informaționale [ELNA10]. În acest sens, se consideră că modelul este sărac din punctul de vedere semantic, iar structurile de date utilizate sunt simpliste pentru a servi unei bune modelări a lumii reale. Operațiile care se pot realiza asupra datelor sunt considerate simple, comparativ cu prelucrările complexe reclamate de numeroase aplicații.

Modelul relațional este unul extensibil, iar sistemele de gestiune a bazelor de date relaționale sunt suficient de flexibile pentru a se putea adapta la noile cerințe de prelucrare. În fața acestor opinii critice, au existat numeroase încercări de extindere și îmbunătățire a modelului relațional, precum: modelul relațional-obiectual, modelul relațional cu valori structurate, modelul relațional fuzzy etc.

Modelul relațional este constituit din următoarele componente: structura relațională, operatorii relaționali și restricțiile de integritate.

a) *Structura relațională*. Datele sunt organizate sub forma unor tabele bidimensionale de date, denumite *relații*. Asocierile dintre relații se reprezintă explicit prin atribute de legătură. Aceste atribute figurează într-o relație implicată în asociere (în cazul legăturilor de tip *unu la mulți*) sau sunt plasate într-o relație distinctă, construită special pentru exprimarea legăturilor (în cazul legăturilor de tip *mulți la mulți*).

b) *Operatorii relaționali*. Aceștia definesc operațiile care se pot efectua asupra relațiilor, în scopul realizării funcțiilor de prelucrare asupra bazei de date, respectiv interogarea, adăugarea, modificarea și stergerea datelor.

c) *Restricțiile de integritate*. Acestea permit definirea stărilor coerente ale bazei de date.

În continuare, vor fi prezentate aceste componente. Adoptarea modelului relațional asigură posibilități multiple de definire și manipulare a

datelor. În primul rând, modelul relațional oferă posibilitatea utilizării unor limbaje procedurale, bazate pe algebra relațională, dar și a unor limbaje neprocedurale (declarative) având la bază calculul relațional. Prin utilizarea acestui din urmă tip de limbaje, utilizatorii nu mai sunt obligați să indice modul în care se obțin datele care îi interesează, ci este suficient să precizeze proprietățile și semnificația acestora. Limbajele neprocedurale contribuie la îmbunătățirea semnificativă a comunicării dintre sistem și utilizatorii finali. În al doilea rând, manipularea datelor se realizează la nivel de mulțime (relație), fiind posibilă utilizarea paralelismului în prelucrarea datelor. Trebuie însă menționat faptul că avantajul conferit de manipularea datelor la nivel de mulțime se pierde în cazul scrierii programelor de aplicație în limbaje de programare universale, întrucât comunicarea între acestea și limbaje precum SQL nu se realizează la nivel de mulțime, ci la nivel de tuplu.

Modelul relațional asigură independența programelor de aplicație față de organizarea datelor și permite utilizarea unor instrumente eficace de control a coerenței și redundanței datelor. În precizarea prelucrărilor asupra datelor, programele de aplicație nu fac apel la pointeri, fișiere inverse sau alte elemente ale schemei interne a bazei de date. În ceea ce privește independența logică, aceasta nu este complet rezolvată nici cu ajutorul modelului relațional. Mapările între diferitele scheme externe și schema conceptuală ridică în continuare o serie de probleme, precum: definirea relațiilor virtuale, realizarea actualizărilor pe relații virtuale etc. Prin tehnica normalizării relațiilor, modelul relațional permite definirea unei structuri conceptuale optime a datelor, prin care se minimizează riscurile de eroare la actualizare, reducându-se redundanța datelor. Coerența datelor este asigurată prin mecanisme flexibile și eficiente de specificare și de utilizare a restricțiilor de integritate și a relațiilor virtuale.

#### 4.1.1 Structura relațională a datelor

Structura relațională a datelor are la bază conceptele de domeniu, relație, atribut și schemă a unei relații.

a) **Domeniu.** Acesta reprezintă un ansamblu de valori, caracterizat printr-un nume. Un domeniu se poate defini explicit, prin enumerarea tuturor valorilor aparținând acestuia sau implicit, prin precizarea proprietăților pe care le au valorile din cadrul domeniului respectiv.

Să considerăm, de exemplu domeniile  $D_1, D_2, D_3$ , definite astfel:

$$D_1 : \{"F", "M"\}$$

$$D_2 : \{x \mid x \in N, x \in [0, 100]\}$$

$$D_3 : \{s \mid s = \text{șir de caractere}\}$$

În cazul lui  $D_1$  s-a recurs la o definire explicită, în timp ce pentru  $D_2$  și  $D_3$  s-a utilizat definirea implicită. Pentru un ansamblu de domenii  $D_1, D_2, \dots, D_n$  produsul cartezian al acestora reprezintă ansamblul tuplurilor  $\langle v_1, v_2, \dots, v_n \rangle$ , unde:  $v_1$  este o valoare aparținând domeniului  $D_1$ ,  $v_2$  este o valoare din  $D_2$  și.m.d.

*De exemplu, tuplurile:  $\langle "Maria", "F", 50 \rangle$ ,  $\langle "Vasile", "M", 15 \rangle$ ,  $\langle "Vasile", "M", 20 \rangle$ ,  $\langle "Vasile", "F", 100 \rangle$  aparțin produsului cartezian:  $D_3 \times D_1 \times D_2$ .*

b) **Relație.** Să presupunem că se acordă o anumită semnificație valorilor domeniilor  $D_1, D_2, D_3$ , definite anterior. Să considerăm, de exemplu că  $D_1$  cuprinde valorile referitoare la sexul unei persoane,  $D_2$  conține valori care exprimă vârstă unei persoane și  $D_3$  cuprinde numele unor persoane. Numai unele dintre tuplurile produsului cartezian:  $D_3 \times D_1 \times D_2$  pot avea o semnificație și anume cele care conțin numele, sexul și vârsta aceleiași persoane. Dintre cele 200 de tupluri care au valoarea  $b$  pe prima poziție, numai unul poate avea o semnificație, dacă presupunem că există o singură persoană cu acest nume. Se desprinde de aici necesitatea definirii unei submulțimi de tupluri, din cadrul produsului cartezian al domeniilor, submulțime care să cuprindă numai tuplurile cu semnificație.

Relația reprezintă un subansamblu al produsului cartezian al mai multor domenii, subansamblu caracterizat printr-un nume și care conține tupluri cu semnificație. Considerând, de exemplu că nu se cunosc decât două persoane, definim relația  $R$  prin tuplurile care descriu aceste persoane:

$$R : \{\langle a, 1, \alpha \rangle, \langle b, 2, \beta \rangle\}$$

Într-o relație, tuplurile trebuie să fie distincte (nu se admit duplicări).

O reprezentare comodă a unei relații este tabelul bidimensional (tabela de date), în care liniile reprezintă tuplurile, iar coloanele corespund domeniilor (tabelul 4.1). Reprezentarea tabelară este preferată adesea altor forme de reprezentare a relațiilor, întrucât este ușor de înțeles și de utilizat. În prezentarea conceptului de relație se recurge uneori la analogii cu alte concepte, bine cunoscute, precum cel de fișier. În acest caz se poate face o

corespondență între relație și fișier, tuplu și înregistrare și între valorile din cadrul tuplului și valorile câmpurilor de înregistrare.

**Tabelul 4.1 Relația  $R$  reprezentată sub forma unei tabele de date**

$D_3$	$D_1$	$D_2$
a	1	$\alpha$
b	2	$\beta$

În cadrul modelului relațional nu interesează decât relațiile finite, chiar dacă la construirea relațiilor se admit domenii infinite. Numărul tuplurilor dintr-o relație reprezintă *cardinalul* relației, în timp ce numărul valorilor dintr-un tuplu definește *gradul* acesteia.

c) **Atribut.** Un domeniu poate apărea de mai multe ori în produsul cartezian pe baza căruia este definită relația. Să considerăm, de exemplu, că pentru o persoană dispunem de următoarele date: nume (cu valori din  $D_3$ ), sex (cu valori din  $D_1$ ), vârstă (cu valori din  $D_2$ ) și numele soțului/soției (cu valori din  $D_3$ ). O relație, denumită PERS, care conține datele despre persoane este o submulțime a produsului cartezian:  $D_3 \times D_1 \times D_2 \times D_3$ . Semnificația valorilor din cadrul unui tuplu se stabilește, în acest caz, nu numai pe baza domeniului de care aparțin valorile, ci și în funcție de poziția ocupată în cadrul tuplului. Pentru a diferenția coloanele care conțin valori ale același domeniu se asociază fiecărei coloane un nume distinct, ceea ce duce la apariția noțiunii de atribut.

Atributul reprezintă coloana unei tabele de date, caracterizată printr-un nume. Numele coloanei (atributului) exprimă, de regulă, semnificația valorilor din cadrul coloanei respective. Atributele elimină astfel dependența de poziție în cadrul unei tabele, ceea ce permite creșterea flexibilității și eficienței organizării datelor. Într-o organizare eficientă, flexibilă, ordinea liniilor și a coloanelor din cadrul tabelei de date nu trebuie să prezinte relevanță.

d) **Schema unei relații.** Aceasta este alcătuită din numele relației, urmat de lista atributelor, pentru fiecare atribut precizându-se domeniul asociat. Astfel, pentru relația  $R$ , cu attributele  $A_1, A_2, \dots, A_n$  și domeniile:  $D_1, D_2, D_m$ , cu  $m \leq n$ , schema relației  $R$  poate fi reprezentată într-una din următoarele forme:

$$R(A_1:D_1, \dots, A_n:D_m) \text{ sau}$$

$A_1:D_1$	...	$A_n:D_m$

Schema unei relații mai este cunoscută și sub numele de *intensia* unei relații, ca expresie a proprietăților comune și invariante ale tuplurilor care compun relația. Spre deosebire de intensie, *extensia* unei relații reprezintă ansamblul tuplurilor care compun la un moment dat relația, ansamblu care este variabil în timp. De obicei, extensia unei relații este stocată fizic în spațiul asociat bazei de date, caz în care relația poartă numele de *relație de bază*. Există însă și situații în care extensia nu este memorată în baza de date. Este cazul așa-numitelor *relații virtuale*, cunoscute și sub numele de *relații derivate* sau *viziuni*. Relația virtuală nu este definită explicit ca relațiile de bază, prin ansamblul tuplurilor componente, ci implicit pe baza altor relații, prin intermediul unei expresii relaționale. Stabilirea efectivă a tuplurilor care compun relația virtuală se realizează prin evaluarea expresiei, ori de câte ori utilizatorul invocă această relație.

#### 4.1.2 Operatorii modelului relațional

Modelul relațional oferă două colecții de operatori pe relații, și anume:

- din algebra relațională (AR);
- din calculul relațional (CR).

La rândul său, calculul relațional este de două tipuri:

- calcul relațional orientat pe tuplu;
- calcul relațional orientat pe domeniu.

##### A) Algebra relațională și extensiile sale

E. F. Codd a introdus algebra relațională drept o colecție de *operații pe relații*, în care operatorii descriu tipuri de prelucrări asupra relațiilor, operanții sunt relații, iar rezultatul este, de asemenea, relație.

Algebra relațională standard este constituită din șase *operatori de bază* (reuniunea, diferența, produsul cartezian, proiecția, selecția și joncțiunea), împreună cu doi *operatori derivați* (intersecția și diviziunea). Ulterior, la operatorii din algebra relațională standard au fost adăugați noi operatori, așa-numitele extensiile ale algebrei relaționale standard, precum: complementarea unei relații, spargerea (*split*) unei relații și închiderea tranzitivă.

În general, operatorii din algebra relațională pot fi grupați în:

- operatori tradiționali (descriu operații standard din teoria mulțimilor, cum ar fi reuniunea, intersecția, diferența, produsul cartezian);

- operatori specifici, precum operatorii de selecție, proiecție, de jonctiune etc.

În continuare, vor fi prezentate principali operatori ai algebrei relaționale și modul lor de utilizare.

Exemplificările în SQL ale operatorilor prezentati se vor face pe schema bazei de date prezentată în figura 4.1.

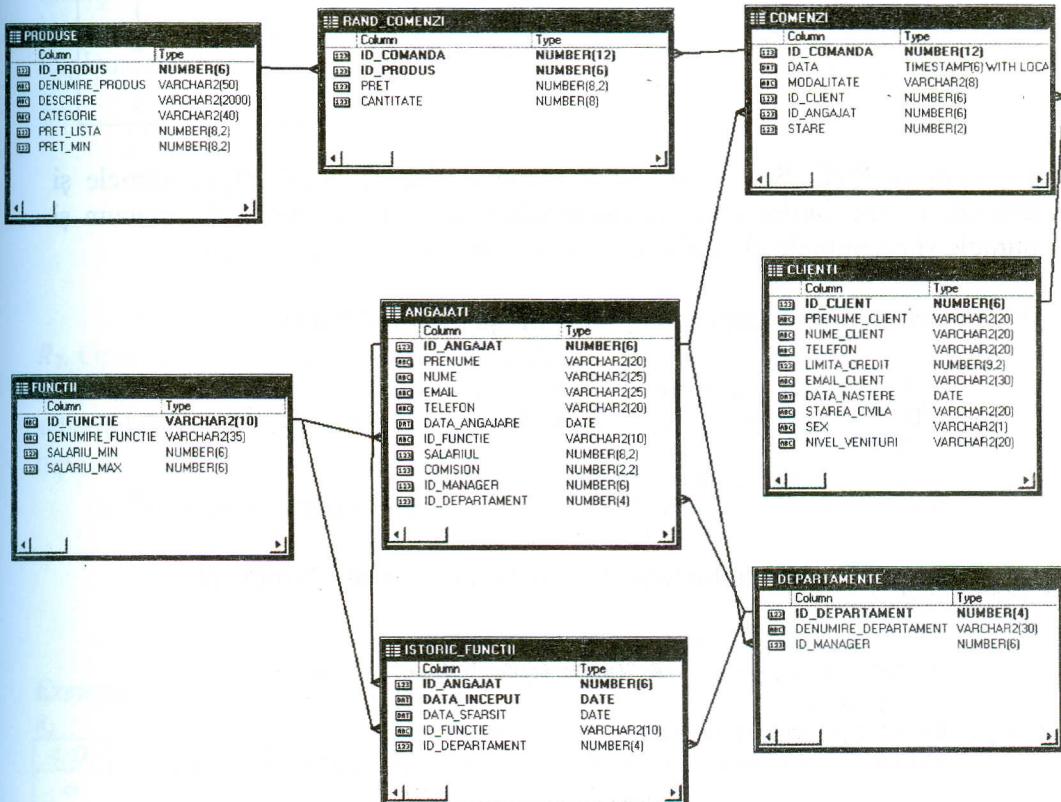


Figura 4.1 Schema bazei de date

**A.1) Reuniunea** este o operație definită pe două relații,  $R_1$  și  $R_2$ , ambele cu aceeași schemă. Operatorul de reuniune are funcția de a construi o nouă relație  $R_3$ , cu schema identică cu  $R_1$  și  $R_2$ , dar având ca extensie tuplurile din  $R_1$  și  $R_2$ , luate împreună o singură dată.

Notățile uzuale pentru reuniune sunt:

$$R_1 \cup R_2$$

$$OR(R_1, R_2)$$

$$APPEND(R_1, R_2)$$

$$UNION(R_1, R_2)$$

*Exemplu:*

$R_1$	
$A:D_1$	$B:D_2$
$\alpha$	1
$\alpha$	2
$\beta$	1

$R_2$	
$A:D_1$	$B:D_2$
$\alpha$	2
$\beta$	3

$R_1 \cup R_2$	
$A:D_1$	$B:D_2$
$\alpha$	1
$\alpha$	2
$\beta$	1
$\beta$	3

*Exemplu în SQL.* Să se afișeze folosind o singură frază SQL, numele și prenumele angajaților care au intermediat cel puțin o comandă, precum și numele și prenumele clienților care au lansat cel puțin o comandă.

- i) Pentru a elimina duplicatele folosim operatorul  $UNION$ :

```
SELECT nume, prenume FROM angajati
WHERE id_angajat IN (SELECT id_angajat FROM comenzi)
UNION
SELECT nume, prenume FROM clienti
WHERE id_client IN (SELECT id_client FROM comenzi);
```

- ii) Pentru a nu elibera duplicatele folosim operatorul  $UNION ALL$ :

```
SELECT nume, prenume FROM angajati
WHERE id_angajat IN (SELECT id_angajat FROM comenzi)
UNION ALL
SELECT nume, prenume FROM clienti
WHERE id_client IN (SELECT id_client FROM comenzi);
```

**A.2) Diferența** este o operație definită pe două relații,  $R_1$  și  $R_2$ , ambele cu aceeași schemă. Operatorul de diferență are funcția de a construi, pe baza relațiilor  $R_1$  și  $R_2$  o nouă relație,  $R_3$ , cu aceeași schemă, dar cu extensia formată din acele tupluri relației  $R_1$  care nu se regăsesc și în relația  $R_2$ .

Notățile uzuale pentru diferență sunt:

$$R_1 - R_2$$

$$REMOVE(R_1, R_2)$$

$$MINUS(R_1, R_2)$$

*Exemplu:*

$R_1$	
$A:D_1$	$B:D_2$
$\alpha$	1
$\alpha$	2
$\beta$	1

$R_2$	
$A:D_1$	$B:D_2$
$\alpha$	2
$\beta$	3

$R_1 - R_2$	
$A:D_1$	$B:D_2$
$\alpha$	1
$\beta$	1

*Exemplu în SQL.* Să se afișeze identifierul angajaților care nu au intermediat nicio comandă în anul curent. Operatorul de diferență folosit este implementat folosind *MINUS*.

```
SELECT id_client FROM clienti
MINUS
SELECT id_client FROM comenzi
WHERE extract(year from data)= extract(year from
sysdate);
```

**A.3) Produsul cartezian** este o operație definită pe două relații,  $R_1$  și  $R_2$ . Operatorul de produs cartezian are funcția de a construi, pe baza relațiilor  $R_1$  și  $R_2$  o nouă relație,  $R_3$ , a cărei schemă se obține prin concatenarea schemelor relațiilor  $R_1$  și  $R_2$  și a cărei extensie cuprinde toate combinațiile tuplurilor din  $R_1$  cu cele din  $R_2$ .

Notăriile uzuale pentru produsul cartezian sunt:

$R_1 \times R_2$

*PRODUCT* ( $R_1$ ,  $R_2$ )

*TIMES* ( $R_1$ ,  $R_2$ )

*Exemplu:*

$R_1$	
$A:D_1$	$B:D_2$
$\alpha$	1
$\beta$	2

$R_2$		
$C:D_1$	$E:D_3$	$F:D_4$
$\alpha$	10	a
$\beta$	10	a
$\beta$	20	b
$\lambda$	10	b

$R_1 \times R_2$				
$A:D_1$	$B:D_2$	$C:D_1$	$E:D_2$	$F:D_3$
$\alpha$	1	$\alpha$	10	a
$\alpha$	1	$\beta$	10	a
$\alpha$	1	$\beta$	20	b
$\alpha$	1	$\lambda$	10	b
$\beta$	2	$\alpha$	10	a
$\beta$	2	$\beta$	10	a
$\beta$	2	$\beta$	20	b
$\beta$	2	$\lambda$	10	b

*Exemplu în SQL.* Să se afișeze produsul cartezian dintre tabelele Clienți și Comenzi. O frază *SELECT* care conține două tabele și nicio condiție de joncționare implementează operatorul de produs cartezian.

```
SELECT * FROM clienti, comenzi;
```

**A.4) Proiecția** este operația definită asupra unei relații  $R$ , care constă din construirea unei noi relații  $P$ , în care se regăsesc numai acele attribute din  $R$  specificate explicit în cadrul operației. Suprimarea unor attribute din  $R$  înseamnă efectuarea unor tăieturi verticale asupra lui  $R$  și care pot avea ca efect apariția unor tupluri duplicate, care se cer să fie eliminate. Prin operația de proiecție se trece de la o relație de grad  $n$  la o relație de grad  $p$ , mai mic decât cel inițial ( $p < n$ ), adică de la un spațiu de  $n$  dimensiuni la unul cu mai puține dimensiuni, ceea ce explică și numele de proiecție.

Notățiile uzuale pentru proiecție sunt următoarele:

$$\prod A_j, \dots, A_m (R)$$

$$R[A_j, \dots, A_m]$$

$$PROJECT(R, A_j, \dots, A_m)$$

*Exemplu:*

$R$			$P$		$P$	
$A:D_1$	$B:D_2$	$C:D_2$	$A:D_1$	$C:D_2$	$A:D_1$	$B:D_2$
$\alpha$	10	1	$\alpha$	1	$\alpha$	1
$\alpha$	20	1	$\alpha$	1	$\beta$	1
$\beta$	30	1	$\beta$	1	$\beta$	2
$\beta$	40	2	$\beta$	2		

*Exemplu în SQL.* Să se afișeze perechile distincte nume, prenume ale salariaților angajați în luna iulie, anul 2010. Pentru implementarea operatorului, în fraza *SELECT* se specifică clauza *DISTINCT*, urmată de coloanele care alcătuiesc proiecția.

```
SELECT DISTINCT nume, prenume FROM angajati
WHERE to_char(data_angajare, 'yyyymm')='201007';
```

**A.5) Selectia** reprezintă operația definită asupra unei relații  $R$ , care constă din construirea unei relații  $S$ , cu aceeași schemă ca  $R$  și a cărei extensie este constituită din acele tupluri din  $R$  care satisfac condiția menționată explicit în cadrul operației. Întrucât cel mai adesea, nu toate tuplurile din  $R$  satisfac această condiție, selectia implică efectuarea de tăieturi orizontale asupra relației  $R$ , adică eliminarea unor tupluri ale relației. Condiția precizată în cadrul operației de selecție este în general de forma

„atribut operator de comparație valoare”, unde operatorul de comparație poate fi:  $<$ ,  $\leq$ ,  $\geq$ ,  $>$  sau  $\neq$ .

Notățiile uzuale pentru selecție sunt următoarele:

$$\sigma \text{ condiție } (R)$$

$$R [\text{condiție}]$$

$$\text{RESTRICT } (R, \text{condiție})$$

*Exemplu:*

*R*

<i>A:D<sub>1</sub></i>	<i>B:D<sub>1</sub></i>	<i>C:D<sub>2</sub></i>	<i>E:D<sub>2</sub></i>
$\alpha$	$\alpha$	1	7
$\alpha$	$\beta$	5	7
$\beta$	$\beta$	12	3
$\beta$	$\beta$	3	10

$$\sigma_{A = B \text{ and } E > 5} (R)$$

<i>A:D<sub>1</sub></i>	<i>B:D<sub>1</sub></i>	<i>C:D<sub>2</sub></i>	<i>E:D<sub>2</sub></i>
$\alpha$	$\alpha$	1	7
$\beta$	$\beta$	23	10

*Exemplu în SQL.* Să se afișeze produsele care au cel mai mare preț de listă al categoriei din care fac parte. Operatorul de selecție se implementează facil în SQL prin condiția din clauza *WHERE* a interogării.

```
SELECT * FROM produse t
WHERE t.pret_lista >= ALL
  (SELECT p.pret_lista FROM produse p
  WHERE t.categorie=p.categorie)
```

**A.6) Joncțiunea (join)** reprezintă operația definită pe două relații,  $R_1$  și  $R_2$ , care constă din construirea unei noi relații  $R_3$ , prin concatenarea unor tupluri din  $R_1$  cu tupluri din  $R_2$ . Se concatenează acele tupluri din  $R_1$  și  $R_2$  care satisfac o anumită condiție, specificată explicit în cadrul operației. Extensia relației  $R_3$  va conține deci combinațiile acestor tupluri care satisfac condiția de concatenare. Condiția de concatenare din cadrul operației de selecție este în general de forma „atribut din  $R_1$  operator de comparație atribut din  $R_2$ ”.

Notățiile uzuale pentru joncțiune sunt următoarele:

$$R_1 \bowtie R_2$$

$$\text{JOIN } (R_1, R_2, \text{condiție})$$

În funcție de operatorul de comparație din cadrul condiției de concatenare, joncțiunea poate fi de mai multe tipuri. Cel mai important tip de joncțiune, în sensul celei mai frecvente utilizări este joncțiunea de egalitate (*equijoin*), care reprezintă joncțiunea dirijată de o condiție de forma „atribut

din  $R_1 = \text{atribut din } R_2$ ". Operația de joncțiune se poate exprima cu ajutorul operatorilor de produs cartezian și de selecție, rezultatul unei joncțiuni fiind același cu rezultatul unei selecții operate asupra unui produs cartezian, adică:

$$\text{JOIN}(R_1, R_2, \text{condiție}) = \text{RESTRICT}(\text{PRODUCT}(R_1, R_2), \text{condiție})$$

Produsul cartezian reprezintă o operație laborioasă și foarte costisitoare, ceea ce face ca în locul produsului să fie utilizată joncțiunea ori de câte ori acest lucru este posibil. Cu toate că apare drept o operație derivată, joncțiunea este prezentată de obicei drept o operație de bază din algebra relațională, dată fiind importanța sa în cadrul sistemelor relaționale, în special la construirea relațiilor virtuale. Variantele operației de joncțiune diferă atât în funcție de tipul condițiilor de concatenare, cât și în raport cu modul de definire a schemei și, respectiv, a extensiei relației rezultate.

**A.6.1) Joncțiunea naturală.** În cazul joncțiunii de egalitate, schema relației rezultat conține toate atributele celor două relații operand, ceea ce face ca în toate tuplurile să existe cel puțin două valori egale. În scopul eliminării acestei redundanțe s-a introdus joncțiunea naturală, ca operație definită pe două relații,  $R_1$  și  $R_2$ , prin care este construită o nouă relație,  $R_3$ , a cărei schemă este obținută prin reuniunea atributelor din relațiile inițiale, iar extensia acesteia conține tuplurile obținute prin concatenarea tuplurilor din  $R_1$  cu tuplurile din  $R_2$  care au aceleași valori pentru atributele cu același nume.

Notația uzuală pentru joncțiunea naturală este:

$$R_1 \bowtie R_2.$$

*Exemplu:*

$R$

$A:D$	$B:D$	$C:D$	$E:D$
1	2	1	3
$\alpha$	1	$\alpha$	a
$\beta$	2	$\gamma$	a
$\gamma$	4	$\beta$	b
$\alpha$	1	$\gamma$	a
$\delta$	2	$\beta$	b

$S$

$B:D$	$E:D$	$F:D$
2	3	1
1	a	$\alpha$
3	a	$\beta$
1	a	$\gamma$
2	b	$\delta$
3	b	$\varepsilon$

$R \bowtie S$

$A:D$	$B:D$	$C:D$	$E:D$	$F:D$
1	2	1	3	1
$\alpha$	1	$\alpha$	a	$\alpha$
$\alpha$	1	$\alpha$	a	$\gamma$
$\alpha$	1	$\gamma$	a	$\alpha$
$\alpha$	1	$\gamma$	a	$\gamma$
$\delta$	2	$\beta$	b	$\delta$

*Exemplu în SQL.* Să se afișeze numele și prenumele fiecărui client, precum și identificatorul comenzilor efectuate.

i) Folosind sintaxa ANSI, adică operatorii  $\text{JOIN}$  sau  $\text{NATURAL JOIN}$ :

```
SELECT nume_client, prenume_client, id_comanda
FROM clienti INNER JOIN comenzi USING (id_client);
```

sau

```
SELECT nume_client, prenume_client, id_comanda
FROM clienti NATURAL JOIN comenzi;
```

- ii) Folosind sintaxa Oracle, adică o condiție de egalitate în clauza *WHERE* pe atributele comune:

```
SELECT nume_client, prenume_client, id_comanda
FROM clienti c, comenzi o
WHERE c.id_client=o.id_client;
```

**A.6.2) Joncțiunea externă.** Atunci când relațiile care participă la joncțiune nu au proiecții identice pe atributul de joncțiune (atributul de joncțiune nu prezintă aceleași valori în relațiile care participă), operația de joncțiune conduce la pierderea de tupluri, cel puțin dintr-o relație. În scopul evitării pierderilor de informație a fost definită joncțiunea externă, ca operație prin care din două relații,  $R_1$  și  $R_2$ , se obține o nouă relație,  $R_3$ , prin joncțiunea relațiilor  $R_1$  și  $R_2$ , relație la care sunt adăugate și tuplurile din  $R_1$  și  $R_2$  care nu au participat la joncțiune. Aceste tupluri sunt completate în relația  $R_3$  cu valori *NULL* pentru atributele relației corespondente ( $R_2$ , respectiv  $R_1$ ).

Notățile uzuale pentru joncțiunea externă sunt:

$$R_1 \bowtie R_2$$

$$\text{EXT-JOIN } (R_1, R_2)$$

### Exemple în SQL

1. Să se afișeze clienții care au dat comenzi împreună cu identificatorul comenzi, precum și cei care nu au dat comenzi folosind o joncțiune externă la stânga.

- i) Folosind sintaxa ANSI, adică operatorul *LEFT JOIN*:

```
SELECT c.*, o.id_comanda
FROM clienti c LEFT JOIN comenzi o
ON (c.id_client=o.id_client);
```

- ii) Folosind sintaxa Oracle, adică o condiție de egalitate în clauza *WHERE* pe atributele comune și semnul (+) la atributul care provine din tabela deficitară (cea din care nu vor fi aduse rânduri în plus față de cele provenite din joncțiunea naturală):

```
SELECT c.*, o.id_comanda
FROM clienti c, comenzi o
WHERE c.id_client=o.id_client(+);
```

2. Să se afișeze identificatorul comenziilor, numele și prenumele angajaților care le-au intermediat, precum și identificatorul comenziilor care nu au fost intermediate de niciun angajat. Se va folosi o joncțiune externă la dreapta.

- i) Folosind sintaxa ANSI, adică operatorul *RIGHT JOIN*:

```
SELECT id_comanda, nume, prenume
FROM angajati RIGHT JOIN comenzi
USING (id_client);
```

- ii) Folosind sintaxa Oracle, adică o condiție de egalitate în clauza *WHERE* pe atributele comune și semnul (+) la atributul care provine din tabela deficitară:

```
SELECT id_comanda, nume, prenume
FROM angajati a, comenzi c
WHERE a.id_client(+) = c.id_client;
```

**A.6.3) Semijoncțiunea.** Această operație a fost introdusă în vederea optimizării cererilor de date. Semijoncțiunea conservă atributele unei singure relații participante la joncțiune, de unde și denumirea sa. Semijoncțiunea reprezintă o operație definită pe două relații,  $R_1$  și  $R_2$ , operație care constă din construirea unei noi relații  $R_3$ , a cărei extensie conține tuplurile relației  $R_1$  care participă la joncțiunea celor două relații. Semijoncțiunea produce același rezultat cu cel al operației de proiecție pe atributele din relația  $R_1$  efectuată asupra joncțiunii relațiilor  $R_1$  și  $R_2$  sau cu cel al operației de selecție asupra relației  $R_1$  realizată pe baza valorilor din  $R_2$  ale atributului de joncțiune.

Notățiile utilizate pentru semijoncțiune sunt:

$$R_1 \ltimes R_2$$

$$\text{SEMIJOIN}(R_1, R_2)$$

*Exemplu în SQL.* Să se afișeze produsele din categoria *software1* care au fost comandate. Operatorul de semijoncțiune se implementează printr-o condiție de filtrare în cadrul unei operații de joncțiune.

```
SELECT DISTINCT p.*
FROM produse p, rand_comenzi r
WHERE p.id_produs=r.id_produs AND
p.categorie='software1';
```

**A.7) Intersecția** reprezintă o operație definită pe două relații,  $R_1$  și  $R_2$  ambele cu aceeași schemă, operație care constă din construirea unei noi relații,  $R3$ , cu schema identică cu a relațiilor operand și cu extensia formată din tuplurile din  $R_1$  și  $R_2$ .

Notățiile uzuale pentru intersecție sunt:

$$R_1 \cap R_2$$

$$\text{INTERSECT } (R_1, R_2)$$

$$\text{AND } (R_1, R_2)$$

*Exemplu:*

$R_1$	
$A:D_1$	$B:D_2$
$\alpha$	1
$\alpha$	2
$\beta$	1

$R_2$	
$A:D_1$	$B:D_2$
$\alpha$	2
$\beta$	3

$R_1 \cap R_2$	
$A:D_1$	$B:D_2$
$\alpha$	2

Intersecția reprezintă o operație derivată, care poate fi exprimată prin intermediul unor operații de bază. De exemplu, operația de intersecție se poate exprima prin intermediul operației de diferență, cu ajutorul următoarelor echivalențe:

$$R_1 \cap R_2 = R_1 - (R_1 - R_2)$$

$$R_1 \cap R_2 = R_2 - (R_2 - R_1)$$

*Exemplu în SQL.* Să se afișeze identificatorul clienților având limita de credit între 2000 și 4000, care au dat cel puțin o comandă care a fost livrată ( $stare=10$ ). Se va folosi operatorul **INTERSECT**.

```
SELECT id_client FROM clienti
WHERE limita_credit BETWEEN 2000 AND 4000
INTERSECT
SELECT id_client FROM comenzi
WHERE stare=10;
```

**A.8) Diviziunea** reprezintă operația definită asupra unei relații  $R$  cu schema:  $R(A_1:D_1, \dots, A_p:D_k, A_{p+1}:D_b, \dots, A_n:D_m)$ , operație care constă din construirea, cu ajutorul unei relații  $r(A_{p+1}:D_b, \dots, A_n:D_m)$  a relației  $Q(A_1:D_1, \dots, A_p:D_k)$ . Tuplurile relației  $Q$ , concatenate cu tuplurile relației  $r$  permit obținerea tuplurilor relației  $R$ .

Notăriile uzuale pentru diviziune sunt:

$$R \div r$$

$$DIVISION(R, r)$$

*Exemplu:*

$R$	
$A:D_1$	$B:D_2$
$\alpha$	1
$\alpha$	2
$\alpha$	3
$\beta$	1
$\gamma$	1
$\delta$	1
$\delta$	3
$\delta$	4
$\epsilon$	6
$\epsilon$	1
$\beta$	2

$r$	
$B:D_2$	
1	
2	

$R \setminus r$	
$A:D_1$	
$\alpha$	
$\beta$	

Operația de diviziune reprezintă o operație derivată, care se poate exprima prin intermediul diferenței, produsului cartezian și proiecției.

*Exemplu în SQL.* Să se afișeze clienții care au comandat toate produsele. Operatorul de diviziune nu există în SQL, astfel încât va fi implementat folosind operatorii *MINUS* sau *NOT EXISTS*.

```
SELECT id_angajat FROM istoric_functii
MINUS
SELECT id_angajat FROM (
  SELECT id_angajat, id_functie
  FROM (SELECT id_angajat FROM istoric_functii), functii
  MINUS
  SELECT id_angajat, id_functie FROM istoric_functii);
```

sau

```
SELECT DISTINCT id_angajat FROM istoric_functii i
WHERE NOT EXISTS (
  SELECT null FROM functii f WHERE NOT EXISTS (
    SELECT null FROM istoric_functii
    WHERE id_functie=f.id_functie AND
    i.id_angajat=id_angajat));
```

**A.9) Complementarea** reprezintă o operație adițională din algebra relațională, care permite determinarea complementului unei relații. Aceasta este definit drept ansamblul tuplurilor din produsul cartezian al domeniilor

asociate atributelor relației, care nu figurează în extensia relației considerate. Este obligatoriu ca domeniile să fie finite. Cardinalitatea rezultatului poate fi extrem de mare, ceea ce face ca operația de complementare să fie relativ puțin utilizată.

Notății pentru desemnarea operației de complementare sunt:

$\neg R$

$NOT(R)$

$COMP(R)$

*Exemplu:*

$R:$

$A_1:D_{A1}$	$A_2:D_{A2}$
r	1
r	2
g	3

$\neg R$

$A_1:D_{A1}$	$A_2:D_{A2}$
r	3
g	1
g	2
a	1
a	2
a	3

**A.10) Spargerea unei relații (split)** reprezintă o operație adițională din algebra relațională definită asupra unei relații  $R$ , operație care în baza unei condiții definite asupra atributelor din  $R$  permite construirea a două relații:  $R_1$  și  $R_2$ , cu aceeași schemă cu  $R$ . Extensia lui  $R_1$  conține tuplurile din  $R$  care verifică condiția specificată, iar  $R_2$  conține tuplurile din  $R$  care nu verifică această condiție.

*Exemplu:* Spargerea relației  $R$  utilizând condiția:  $A_2 > 2$ .

$R:$

$A_1:D_{A1}$	$A_2:D_{A2}$
r	1
r	2
g	3

$R_1:$

$A_1:D_{A1}$	$A_2:D_{A2}$
g	3

$R_2:$

$A_1:D_{A1}$	$A_2:D_{A2}$
r	1
r	2

*Exemplu în SQL.* Să se afișeze primii trei angajați în ordinea salariilor, iar separat restul salariaților. Operatorul de spargere este implementat prin condiția din clauza *WHERE* a interogării.

```
SELECT * FROM angajati a WHERE
(SELECT COUNT(*) FROM angajati b
WHERE a.salariul < b.salariul) < 3;
```

Restul angajaților vor fi afișați prin interogarea:

```
SELECT * FROM angajati a WHERE
  (SELECT COUNT(*) FROM angajati b
   WHERE a.salariul>b.salariul)>3
```

**A.11) Închiderea tranzitivă** reprezintă o operație adițională din algebra relațională prin care se pot adăuga tupluri la o relație. Operația de închidere tranzitivă presupune executarea în mod repetat a secvenței de operații: joncțiune - proiecție - reuniune. Numărul de execuții depinde de conținutul relației, nefiind fixat dinainte. Închiderea tranzitivă este o operație definită asupra unei relații  $R$ , a cărei schemă conține două attribute,  $A_1$  și  $A_2$ , cu același domeniu asociat. Tranzitivitatea constă din adăugarea la relația  $R$  a tuplurilor care se obțin succesiv prin tranzitivitate, în sensul că, dacă există în  $R$  tuplurile  $\langle a,b \rangle$  și  $\langle b,c \rangle$  se va adăuga la  $R$  și tuplul  $\langle a,c \rangle$ .

Notăriile uzuale pentru operația de închidere tranzitivă sunt:

 $\tau(R)$ 
 $R^+$ 
 $CLOSE(R)$ 

*Exemplu:*

$R$ :

Persoana: $D_N$	Urmăs: $D_N$
"Ana"	"Maria"
"Ana"	"Ion"
"Ion"	"Vasile"
"Ion"	"Nicoleta"
"Maria"	"Oana"

 $\tau(R)$ 

Persoana: $D_N$	Urmăs: $D_N$
"Ana"	"Maria"
"Ana"	"Ion"
"Ion"	"Vasile"
"Ion"	"Nicoleta"
"Maria"	"Oana"
"Ana"	"Vasile"
"Ana"	"Nicoleta"
"Ana"	"Oana"

### Expresii din algebra relațională

O expresie a algebrei relaționale este constituită dintr-o serie de relații, legate prin operații. Să considerăm, ca exemplu, două relații:  $R_1$  și  $R_2$  cu schemele  $R_1 (A:D_A, B:D_B)$  și  $R_2 (B:D_B, C:D_C)$ . Pe baza acestor relații, putem defini o expresie  $E_1$ , astfel:

$$E_1 = \pi_c (\sigma_{A=a} (R_1 \bowtie R_2))$$

În exprimarea unei expresii se pot introduce relații intermediare. Astfel, expresia  $E_1$  se poate reprezenta cu ajutorul unor relații intermediare,  $X_1$  și  $X_2$ , astfel:

$$X_1 = R_1 \bowtie R_2$$

$$X_2 = \sigma_{A=a}(X_1)$$

$$E_1 = \pi_c(X_2)$$

Evaluarea unei expresii presupune efectuarea prelucrărilor indicate de operatorii din expresie, în ordinea apariției lor sau în ordinea fixată prin paranteze. Rezultatul evaluării unei expresii este o relație, derivată din relațiile menționate ca operanzi în cadrul expresiei.

Două expresii se numesc echivalente, dacă în urma evaluării lor se obține ca rezultat o aceeași relație.

## B) Calculul relațional și variantele sale

Colecția de operatori din cadrul modelului relațional cunoscută sub numele de calcul relațional (CR) a fost introdusă, ca și algebra relațională, de către E.F. Codd. Calculul relațional reprezintă o adaptare a calculului cu predicate la domeniul bazelor de date relaționale. Ideea de bază a calculului relațional este de a identifica o relație cu un predicat. Pe baza unor predicate (relații) inițiale, prin aplicarea unor operatori ai calculului cu predicate se pot defini noi predicate (relații). Spre deosebire de derivarea procedurală a relațiilor din cadrul algebrei relaționale, calculul relațional permite definirea neprocedurală (declarativă) a relațiilor, în sensul precizării lor prin intermediul proprietăților tuplurilor și nu prin maniera de derivare efectivă a acestor tupluri.

În varianta inițială, calculul relațional utiliza variabile definite asupra relațiilor, variabile ale căror valori reprezintă tupluri de relație. Din această cauză, variabilele au fost denumite *variabile tuplu*, iar calculul relațional introdus de Codd a primit numele de *calcul relațional orientat pe tuplu*. Ulterior, a mai fost introdusă o variantă a calculului relațional, în care variabilele sunt definite asupra domeniilor, așa-numitele *variabile domeniu*, variantă cunoscută sub numele de *calcul relațional orientat pe domeniu*.

Operând proiecțiile în cascadă obținem  $E_2 = \pi_{1,3}(S \times E) \cup \pi_{3,1}(S \times E)$ . Conform cazului 2, expresia pentru  $E^2 \cap \{wx \mid \neg R(wx)\}$  se prezintă sub forma  $E^2 - R$ .

De asemenea, pe baza cazului 1 al inducției efectuate, expresia algebrică pentru  $\{wx \mid \neg R(wx) \vee (\exists y)(S(wy) \vee S(xy))\}$  este  $(E^2 - R) \cup E^2$ .

Prin urmare, expresia algebrică  $E_A$  echivalentă cu expresia  $E_c$  este  $E_A = E^2 - ((E^2 - R) \cup E_2) = R - E_2$  (întrucât  $R$  și  $E_2$  sunt submulțimi ale lui  $E^2$ ). Explicitându-l pe  $E_2$ , se obține  $E_A = R - (\pi_{1,3}(S \times E) \cup \pi_{3,1}(S \times E))$ .

Demonstrarea reducerii calculului relațional orientat pe domeniu la algebra relațională reprezintă ultima etapă în demonstrarea echivalenței între algebra relațională și calculul relațional. Acest lucru înseamnă ca orice relație posibil de definit în algebra relațională, prin intermediul unei expresii algebrice poate fi definită și în cadrul calculului relațional, printr-o expresie bine formată și reciproc. Algebra relațională și calculul relațional au, prin urmare, aceeași putere de expresie.

#### 4.1.3 Restricțiile de integritate

Restricțiile de integritate, denumite și reguli de integritate, definesc cerințele pe care trebuie să le satisfacă datele din cadrul bazei de date pentru a putea fi considerate corecte și coerente în raport cu domeniul pe care îl reflectă. Se afirmă adesea că modelul relațional este prea simplu pentru a putea exprima semantica datelor. Relațiile bazei de date sunt toate la același nivel, iar setul de operatori este relativ sărac, neputând exprima semantica entităților complexe, care rămâne a fi definită, exclusiv, la nivelul programelor de aplicație.

Cu toate acestea, sistemele relaționale dețin o serie de mecanisme pentru tratarea aspectelor de ordin semantic. Restricțiile de integritate reprezintă principalul mod de integrare a semanticii datelor în cadrul modelului relațional al datelor, mecanismele de definire și de verificare a acestor restricții reprezentând principalele instrumente pentru controlul semantic al datelor. Avantajele încorporării semanticii datelor în cadrul bazelor de date constau, în principal, în posibilitatea întreținerii mai ușoare a aplicațiilor și posibilitatea implementării unor mecanisme fizice mai eficiente.

În teoria sistemelor relaționale, restricțiile de integritate sunt studiate în principal sub aspectul puterii lor de modelare și al posibilităților de verificare eficace a respectării lor. Un exemplu semnificativ îl reprezintă dependențele între date și, în primul rând, dependențele funcționale. Dependențele între date, ca restricții de integritate constituie un suport teoretic solid pentru problema de modelare informatică. În acest sens, dependențele funcționale au permis definirea conceptului de *structură relațională optimă*, stând la baza teoriei optimizării structurii relaționale a datelor, respectiv teoria normalizării relațiilor.

Restricțiile de integritate ale modelului relațional sunt de două tipuri, și anume: structurale și de comportament.

a) *restricții de integritate structurale*, inerente modelului relațional, care se definesc prin egalitatea sau inegalitatea unor valori din cadrul relațiilor. Din categoria restricțiilor de integritate structurală fac parte: restricția de unicitate a cheii, restricția referențială, restricția entității și dependența între date;

b) *restricții de integritate de comportament*, proprii unei anumite baze de date relaționale (BDR), restricții care țin cont de semnificația valorilor din cadrul bazei de date. Putem considera, de exemplu restricția încadrării vârstei, înălțimii sau greutății unei persoane între anumite limite acceptabile.

În cadrul unei baze de date relaționale se pot defini restricții de integritate de comportament foarte diverse: temporale, de domeniu etc. Utilizarea modelului relațional nu impune definirea și verificarea tuturor acestor tipuri de restricții de integritate, structurale și de comportament. Din acest punct de vedere, putem considera că există:

a) *restricții de integritate minimale*, obligatoriu de definit și de respectat atunci când se lucrează cu modelul relațional. Din această categorie fac parte: restricția de unicitate a cheii, restricția referențială și restricția entității.

b) *alte restricții de integritate*, categorie din care fac parte dependențele între date și restricțiile de comportament.

a) *Restricțiile de integritate minimale*. Modelul relațional prezintă următoarele restricții de integritate minimale:

i. *Restricția de unicitate a cheii*. Reprezintă restricția de integritate care impune ca într-o relație  $R$ , care are cheia  $K$ , oricare ar fi tuplurile  $t_1$  și  $t_2$ , să fie satisfăcută inegalitatea:  $t_1(K) \neq t_2(K)$ . Această inegalitate semnifică

faptul că într-o relație nu pot exista două tupluri cu o aceeași valoare pentru attributele cheie.

*ii. Restricția referențială (integritatea referirii).* Reprezintă restricția de integritate care impune ca într-o relație  $R_1$  care referă o relație  $R_2$ , valorile cheii externe să figureze printre valorile restricției de unicitate din relația  $R_2$  sau să fie valori *NULL* (nedefinite).  $R_1$  și  $R_2$  nu trebuie să fie neapărat distincte. Semnificația restricției de integritate a referirii este următoarea: o asociere nu poate exista decât între parteneri cunoscuți, adică parteneri deja definiți. Atunci când, într-o anumită situație, asocierea nu este aplicabilă, unul dintre parteneri va fi desemnat prin valoarea *NULL*, cu semnificația de „partener inexistent”.

*iii. Restricția entității (integritatea entității).* Restricția entității reprezintă restricția de integritate care impune ca într-o relație attributele cheii primare să fie nenule. Unicitatea cheii impune ca la adăugarea unui tuplu, valoarea cheii să fie cunoscută, pentru a se putea verifica faptul că această valoare nu există deja încărcată (tuplul nu figurează deja în baza de date). Cu valori *NULL*, cheia își pierde rolul de identificator de tuplu. Restricția de integritate a entității nu se aplică cheilor externe dintr-o relație, dacă acestea nu aparțin cheii primare, ceea ce conferă o anumită suplete modelului relațional.

Uneori, se consideră drept restricții de integritate minimale numai restricțiile de integritate a referirii și a entității, deoarece restricția de unicitate a cheii este considerată implicit, ca fiind inclusă în definirea cheii relației.

#### b) Alte restricții de integritate

Restricțiile referitoare la dependența datelor vor fi prezentate în paragraful 4.2. În ceea ce privește *restricțiile de comportament*, în funcție de domeniul modelat, se pot defini mai multe tipuri de restricții de integritate de comportament, și anume restricții de domeniu, restricții temporale etc.

De exemplu, o restricție de domeniu se poate defini în cadrul unei relații ANGAJATI pentru atributul *salariu*. Conform acestei restricții, valorile atributului trebuie să se încadreze între anumite limite. O restricție temporală se poate referi la valorile atributului *stoc* din cadrul unei relații PRODUSE, prin care să se impună ca valorile rezultate în urma actualizării să depășească valorile anterioare actualizării.



## 4.2 Baze de date relaționale

O **bază de date relațională** (BDR) reprezintă un ansamblu de relații (tabele) de date împreună cu legăturile dintre ele.

**Relația** este definită, la rândul său, drept o mulțime de tupluri (seturi) de date. Întrucât o mulțime nu poate conține elemente componente duplicate, relația nu poate prezenta tupluri identice. Întrucât tuplurile sunt unice trebuie să existe posibilitatea identificării lor în mod unic în cadrul unei relații. Identificarea unui tuplu în cadrul unei relații fără a se face apel la toate valorile din tuplu a dus la definirea noțiunii de cheie.

*Cheia unei relații* reprezintă ansamblul minimal de atribute prin care se poate identifica în mod unic orice tuplu din cadrul relației. Orice relație posedă cel puțin o cheie. Cheia poate să fie constituită fie dintr-un singur atribut, fie din totalitatea atributelor din schema relației respective. Atunci când cheia este constituită dintr-un singur atribut poartă numele de *cheie simplă*, iar atunci când este formată din mai multe atribute este denumită *cheie compusă*.

*Exemplu:*

<u>R<sub>1</sub></u>	
<u>A:D<sub>A</sub></u>	<u>B:D<sub>B</sub></u>
a <sub>1</sub>	b <sub>1</sub>
a <sub>2</sub>	b <sub>3</sub>
a <sub>3</sub>	b <sub>2</sub>

<u>R<sub>2</sub></u>	
<u>A:D<sub>A</sub></u>	<u>B:D<sub>B</sub></u>
a <sub>1</sub>	b <sub>1</sub>
a <sub>1</sub>	b <sub>2</sub>
a <sub>2</sub>	b <sub>3</sub>
a <sub>3</sub>	b <sub>2</sub>

Cheia relației  $R_1$  este o cheie simplă, în timp ce cheia relației  $R_2$  este o cheie compusă. Atributele cheie sunt subliniate.

Determinarea cheii unei relații necesită cunoașterea semantică relației respective și deci a tuturor extensiilor posibile, nu numai a celei din momentul în care se stabilește cheia. Să presupunem că  $R_1$  și  $R_2$  sunt versiuni ale aceleiași relații  $R$ , la momente de timp diferite  $t_1$  și, respectiv  $t_2$ . Alegerea la momentul  $t_1$  drept cheie a relației  $R$  a atributului  $A$  numai pe baza analizei extensiei din acel moment se dovedește greșită, întrucât atributul  $A$  nu face posibilă identificarea unică a tuplurilor și la momentul  $t_2$ . Cheia relației trebuie să fie definită prin perechea de atribute ( $A, B$ ).

Într-o relație pot exista mai multe combinații de atribute cu proprietatea de identificare unică a tuplurilor. Se spune în acest caz că relația

posedă mai multe *chei candidate*. În această situație, administratorul bazei de date va alege dintre cheile candidate una care să servească în mod efectiv la identificarea tuplurilor și care va primi numele de *cheie primară*. Restul cheilor candidate se vor numi și *chei unice*.

Cheia unei relații trebuie să fie minimală, în sensul că nicio parte a sa nu trebuie să aibă proprietatea de identificare unică a tuplurilor relației (nu trebuie să fie la rândul ei cheie). Un grup de atrbute din cadrul unei relații care conține o cheie a relației poartă numele de *supercheie*.

Modelul relațional servește la reprezentarea entităților din lumea reală și a asocierilor dintre acestea. Modelarea asocierilor dintre entități impune recurgerea la conceptul de cheie externă.

O *cheie externă* reprezintă un atrbut sau un grup de atrbute dintr-o relație  $R_1$  ale cărui/căror valori sunt definite pe același/aceleași domeniu/domenii ca și cheia primară a unei alte relații,  $R_2$  și care are rolul de a modela asocierea între entitățile reprezentate cu ajutorul relaților  $R_1$  și  $R_2$ . În acest context,  $R_1$  este denumită *relație care referă*, în timp ce  $R_2$  poartă numele de *relație referită*. Să considerăm, ca exemplu, relațile DEPARTAMENTE și ANGAJAȚI. Într-un departament lucrează mai mulți angajați, în timp ce un angajat lucrează într-un singur departament. Atributul *id\_departament* din cadrul relației ANGAJAȚI precizează departamentul în care lucrează un angajat reprezentând o cheie externă, servind la încorporarea în cadrul bazei de date a asocierii dintre angajați și departamente.

Dependențele între date semnifică modul în care datele depind unele de altele și pot fi de mai multe tipuri: dependențe funcționale, dependențe multivaloare și dependențe jonctiune.

a) *Dependența funcțională* reprezintă dependența între date prin care un atrbut sau grup de atrbute poate fi determinat prin intermediul altui atrbut sau grup de atrbute. Fiind dată o relație  $R$ , un atrbut  $Y$  din  $R$  este dependent funcțional de un alt atrbut  $X$  din  $R$ , dacă și numai dacă fiecare valoare a lui  $X$  are asociată o valoare precisă a lui  $Y$ . Notația utilizată pentru desemnarea dependențelor funcționale este  $X \rightarrow Y$ . Partea stângă a dependenței funcționale poartă numele *determinant*, iar partea dreaptă *determinat*. Astfel, în cadrul dependenței  $X \rightarrow Y$ ,  $X$  este determinantul, iar  $Y$  este determinatul.

În general, considerând o relație,  $R$  cu schema:  $R(A1:D_{A1}, A2:D_{A2}, \dots, An:D_{An})$ , se spune că pentru două grupuri de atrbute din  $R$ ,  $X$  și  $Y$ ,  $X$  determină funcțional pe  $Y$ , dacă pentru oricare două tupluri  $t$  și  $r$  din  $R$ , egalitatea  $r[X] = t[X]$  implică egalitatea  $r[Y] = t[Y]$ .

Dependența funcțională  $X \rightarrow Y$  reprezintă o restricție de integritate aplicată tuplurilor relației  $R$ , în sensul că oricare două tupluri din  $R$  care prezintă o aceeași valoare pentru  $X$  trebuie să prezinte o aceeași valoare și pentru  $Y$ .

Dependența funcțională  $X \rightarrow Y$  se numește *dependență funcțională trivială* dacă  $Y \subseteq X$ .

O dependență funcțională  $X \rightarrow Y$  se numește *dependență funcțională parțială* dacă și numai dacă  $Y$  este dependent funcțional atât de  $X$ , cât și de o parte a lui  $X$ .

Se numește *dependență funcțională completă* o dependență funcțională de forma  $X \rightarrow Y$  în care  $Y$  este dependent funcțional de  $X$ , fără să fie dependent funcțional de niciuna din componentele lui  $X$ . O categorie importantă de dependențe funcționale o reprezintă cele tranzitive.

O dependență  $X \rightarrow Y$  reprezintă o *dependență funcțională tranzitivă* atunci când se manifestă concomitent cu alte două dependențe funcționale și anume  $C \rightarrow X$  și  $C \rightarrow Y$ . Coexistența acestor dependențe funcționale face ca  $Y$  să fie dependent funcțional de  $C$  în două moduri:

- direct, prin dependența  $C \rightarrow Y$ ;
- indirect, prin lanțul de dependențe  $C \rightarrow X \rightarrow Y$ .

b) *Dependența multivaloare* reprezintă acel tip de dependență între date în care un atribut/grup de atribute poate prezenta mai multe valori pentru o singură valoare a unui alt atribut/grup de atribute. Fie o relație  $R$ , în care apar attributele/grupurile de atribute:  $X$ ,  $Y$  și  $Z$ . În cadrul relației  $R$  există o dependență multivaloare dacă și numai dacă mulțimea valorilor lui  $Y$  corespunzătoare unei perechi (*valoare X*, *valoare Z*) depinde numai de valoarea lui  $X$ , nu și de valoarea lui  $Z$ . Dependența multivaloare se notează cu  $X \rightarrow\rightarrow Y$ . Pentru a avea  $X \rightarrow\rightarrow Y$  este necesar să avem și  $X \rightarrow\rightarrow Z$ . Din acest motiv se obișnuiește ca dependența multivaloare să se noteze  $X \rightarrow\rightarrow Y/Z$ . În general, având o relație  $R$ , cu schema  $R(X:D_X, Y:D_Y, Z:D_Z)$  și notând cu  $val_x$ ,  $val_y$  și  $val_z$  valorile atributelor  $X$ ,  $Y$  și  $Z$  se definește mulțimea:  $Y_{XZ} = \{val_y | <val_x, val_y, val_z> \subset R\}$ . Există o dependență multivaloare  $X \rightarrow\rightarrow Y$  dacă și numai dacă  $Y_{XZ}$  depinde numai de valorile lui  $X$ , adică  $Y_{XZ1} = Y_{XZ2}$ ,  $Z_1 \neq Z_2$ . Dependența funcțională reprezintă un tip particular de dependență multivaloare, pentru care mulțimea valorilor dependente este constituită dintr-o singură valoare, adică  $Y_{xz}$  reprezintă o mulțime cu un singur element.

Pentru o relație  $R$ , cu schema:  $R(X:D_X, Y:D_Y)$  se consideră *dependență multivaloare trivială* dependența:  $X \rightarrow\rightarrow Y$ .

c) *Dependență joncțiune*. Se consideră o relație  $R$  cu schema  $R(X:D_X, Y:D_Y, Z:D_Z)$  pentru care nu se manifestă dependențe funcționale sau dependențe multivaloare, adică relația  $R$  se poate asimila unei chei compuse. Asupra acestei relații se formulează următoarea restricție: dacă în relația  $R$  figurează tuplurile  $\langle X_1, Y_1, Z_2 \rangle$ ,  $\langle X_2, Y_1, Z_1 \rangle$  și  $\langle X_1, Y_2, Z_1 \rangle$ , atunci în  $R$  trebuie să figureze și tuplul  $\langle X_1, Y_1, Z_1 \rangle$ . Această restricție exprimă o dependență între date similară dependenței funcționale sau dependenței multivaloare. Descompunerea fără pierderi de date a relației  $R$  se poate realiza sub forma următoarelor proiecții:  $P_1(X:D_X, Y:D_Y)$ ,  $P_2(Y:D_Y, Z:D_Z)$ ,  $P_3(Z:D_Z, X:D_X)$ . Reconstituirea relației  $R$  se realizează prin joncțiunea proiecțiilor  $P_1$ ,  $P_2$  și  $P_3$ . Din această cauză, dependența din cadrul relației  $R$  a primit numele de dependență joncțiune. Dependența joncțiune exprimă o restricție mai generală decât cea exprimată de dependența multivaloare sau dependența funcțională.

### 4.3 Standardul SQL

SQL (*Structured Query Language*) este un limbaj de descriere și manipulare acceptat, practic, de toate sistemele de gestiune a bazelor de date relaționale. Atât ANSI (*American National Standards Institute*), cât și ISO (*International Standards Organization*) îl consideră drept un standard pentru limbajele de interogare a bazelor de date relaționale.

Limbajul SQL a fost dezvoltat la începutul anilor 1970 de către o echipă de la IBM, în cadrul laboratorului de Cercetare de la San Jose și a fost prezentat, pentru prima dată, la o conferință ACM, în anul 1974, cu denumirea SEQUEL - Structured Query Language [CHRA74]. Deși autorul limbajului SQL este IBM, primul producător de sisteme de gestiune a bazelor de date relaționale care îl utilizează este Oracle Corporation. Primele implementări comerciale ale SQL sunt pentru SGBD relaționale de dimensiuni medii, bazate pe UNIX, precum Oracle, Ingres și Informix.

În 1981, IBM lansează SQL/DS, precursorul DB2. ANSI a publicat primul standard SQL în 1986, sub denumirea SQL-86. O versiune a standardului internațional emis de către ISO a apărut în 1987. O actualizare semnificativă a SQL-86 a fost lansată în 1989 (SQL-89). Practic, în prezent toate SGBD relaționale au la bază standardul din 1989. În 1992, standardul a fost din nou revizuit (SQL-92), adăugându-se o serie de facilități, ceea ce face ca SQL-92 să fie un SQL-89 extins [MOWI08]. Din acest motiv,

aplicațiile au migrat cu multă ușurință spre noul standard. SQL-92 standard a fost înlocuit de SQL:1999, care a fost, din nou, o extensie a standardului precedent. Această extensie se referă la adevararea cu modelul obiectual-relațional și la posibilitatea ca metodele, funcțiile, procedurile să poată fi scrise în limbaje procedurale bazate pe SQL (precum PL/SQL) sau să fie scrise într-un alt limbaj de programare, cum ar fi C++ sau Java și apoi să fie invocate din cadrul altor comenzi SQL. Ca rezultat, SQL devine mai puțin relațional, o tendință criticată de adeptii orientării relaționale pure. Nici chiar standardul SQL:1999 nu transformă SQL într-un limbaj propriu-zis de programare, lipsindu-i declarații de intrare/ieșire de sine stătătoare (*stand alone*). Cu toate acestea, SQL:1999 include structuri repetitive și alternative, pentru implementarea structurilor de control a prelucrărilor, apropiindu-se astfel de limbajele de programare universale. Aceste structuri sunt utilizate la scrierea de proceduri stocate și declanșatori.

Standardul SQL a fost actualizat de trei ori, în versiunile denumite SQL:2003, SQL:2006, și SQL:2008 [ELNA10]. Aceste revizuiri au adăugat suport pentru XML (*eXtended Markup Language*), platformă independentă de reprezentare a datelor folosind fișiere text.

Există două moduri de lucru cu baze de date, și anume:

- *SQL interactiv*, în care un utilizator tastează o singură comandă, care este trimisă imediat bazei de date. Rezultatul unei interogări interactive este o tabelă de date în memoria principală;
- *SQL incorporat (embedded SQL)*, în care declarațiile SQL sunt plasate într-un program de aplicație. Interfața poate fi pe bază de videoformate sau linie de comandă. SQL poate fi static (în cazul în care întreaga comandă este specificată la momentul scrierii programului) sau dinamic (pe baza datelor introduse de utilizator se formează comanda pentru baza de date).

În plus față de cele două metode de scriere a comenziilor SQL, există și o serie de constructori de interogare grafică. Aceștia furnizează o modalitate pentru un utilizator care nu cunoaște limbajul SQL pentru a realiza interogări ale bazei de date. Multe dintre aceste programe sunt editoare de rapoarte, nefiind destinate modificării datelor sau structurii unei baze de date.

În perioada interfețelor grafice (*Graphical User Interface – GUI*), mediile bazate pe linie de comandă, aşa cum sunt mediile SQL interactive, par a fi anacronice. Cu toate acestea, liniile de comandă SQL continuă să ofere acces de bază la bazele de date relaționale și sunt utilizate pe

scără largă. Un mediu linie de comandă oferă, de asemenea, suport pentru interogări ad-hoc, care nu au o anumită regularitate. Utilizatorii SQL experimentați pot lucra mai rapid utilizând mediile bazate pe linie de comandă.

Celălalt mod de lucru (*embedded SQL*) poate să ofere un constructor de interogări, mai precis un mediu în care utilizatorul este ghidat să formuleze interogări ale bazei de date. În acest fel, este mult mai ușor pentru utilizatori să construiască declarații SQL corecte. Încorporarea SQL într-un limbaj general de programare prezintă o provocare interesantă. Limbajele gazdă (de exemplu, Java, C++ sau COBOL) au compilatoare care nu recunosc SQL. Soluția este de a oferi sprijin SQL printr-o bibliotecă de aplicație, care se poate asocia unui program. Programul în limbaj sursă este prelucrat de un pre-compilator care schimbă comenziile SQL în rutine de bibliotecă. Codul sursă modificat va fi prelucrat de compilatorul limbajului gazdă.

În afara dificultăților de compilare a programelor care încorporează comenzi SQL, există o nepotrivire fundamentală între SQL și un limbaj de programare cu uz general. Limbajele de programare sunt concepute pentru a procesa, la un moment dat, un tuplu de valori. Limbajul SQL este proiectat să proceseze mai multe tupluri de date concomitent. Folosind limbajul PL/SQL se poate procesa o interogare, tuplu cu tuplu, folosind mecanismul de cursor.

Deși există mai multe moduri de a crea o comandă SQL, aceste comenzi sunt alcătuite din aceleasi elemente [PRLA08], [RISC09]:

- *cuvinte-cheie*. Fiecare comandă SQL începe cu un cuvânt-cheie, cum ar fi SELECT, INSERT, UPDATE, care spune procesorului tipul de operație care urmează să fie executată. Restul de cuvinte-cheie preced tabelele din care urmează să fie luate datele, indică operațiunile specifice care urmează să fie efectuate pe date etc.;
- *relații*. O comandă SQL include numele relațiilor asupra cărora acționează comanda SQL;
- *attribute*. O comandă SQL include numele atributelor vizate de comandă;
- *funcții*. O funcție este o prelucrare disponibilă pentru a fi executată utilizând limbajul SQL. De exemplu, funcția AVG calculează media valorilor numerice ale unor date.

Structura generală a unei comenzi SQL este următoarea:

```
SELECT [domeniu] lista_selectie
      FROM nume_tabel1, nume_tabela2, ...
      [WHERE criteriu_de_selectie]
      [GROUP BY câmp_de_grupare
      [HAVING criteriu_de_grupare]]
[ORDER BY câmpuri_criteriu [ASC|DESC]];
```

## 4.4 Sisteme de gestiune a bazelor de date relaționale

Teoria relațională a dat o fundamentare solidă realizării de sisteme de gestiune a bazelor de date performante. La sfârșitul anilor '80 și apoi în anii '90 au apărut, în special odată cu pătrunderea în masă a microcalculatoarelor, numeroase sisteme de gestiune a bazelor de date relaționale. Aceasta a însemnat o evoluție de la sistemele de gestiune a bazelor de date de generația întâi (arborescente și rețea) spre cele de generația a doua (relaționale). Această evoluție s-a materializat în principal în: oferirea de limbaje de interogare neprocedurale, îmbunătățirea integrității și securității datelor, optimizarea și simplificarea accesului la date.

Un **sistem de gestiune a bazelor de date relaționale** (SGBDR) reprezintă un SGBD care utilizează drept concepție de organizare a datelor modelul relațional. Altfel spus, SGBDR reprezintă un SGBD care suportă modelul relațional. Definiția de mai sus este mult prea generală pentru a putea fi operațională, deoarece modul de implementare a modelului relațional diferă, de regulă atât între diferențele SGBDR, cât și în raport de modelul teoretic, cel definit în cadrul teoriei relaționale, datorită eforturilor producătorilor de realizare a unor sisteme performante, care să satisfacă exigențele utilizatorilor. Codd a formulat 13 reguli care exprimă cerințele pe care trebuie să le satisfacă un SGBDR.

Diversitatea modelelor relaționale operaționale au determinat, în mod natural, existența unei varietăți de SGBDR. Au apărut o serie de sintagme, precum: sisteme cu interfață relațională, sisteme pseudorelaționale, sisteme complet relaționale etc. În general, conceptele utilizate la prezentarea SGBDR și a modelelor relaționale operaționale diferă de cele din cadrul teoriei relaționale.

Prezentăm în continuare cele mai importante SGBDR în prezent: Oracle, SQL Server, DB2 și MySQL. Acestea implementează caracteristicile

de bază ale SGBDR, și anume: atomicitatea, consistența, izolarea și durabilitatea tranzacțiilor, restricțiile referențiale și standardul de reprezentare și manipulare a datelor UNICODE. Se pot folosi fără restricții operatorii relaționali, inclusiv juncțiunile externe. MySQL nu permite scrierea de expresii folosind INTERSECT sau EXCEPT. Aceste SGBD implementează limbaje procedurale în care se pot folosi construcții, cum ar fi: cursori, funcții, proceduri, declanșatori.

**ORACLE Database** reprezintă un SGBD relațional-obiectual produs de către Oracle Corporation și ajuns în prezent la versiunea 11g optimizat pentru lucru cu baze de date de mari dimensiuni. Prima versiune, apărută în 1979, a fost denumită versiunea 2 [GRST08] din considerente de marketing, fiind primul SGBD comercial care a utilizat limbajul SQL. De-a lungul vremii produsul *Oracle Database* a fost deschizător de drum în domeniul bazelor de date relaționale, implementând în premieră soluții preluate în timp și de ceilalți producători de SGBDR, cum ar fi: tranzacțiile, modelul client-server, limbajul procedural (PL/SQL), modelul distribuit, stocarea și gestiunea de date XML, suportul pentru 64 biți, tehnologia *Grid* etc. Oracle pune la dispoziție cinci versiuni *Enterprise Edition*, *Standard Edition*, *Standard Edition One*, *Express Edition*, *Oracle Database Lite* care diferă în funcție de facilitățile oferite, acestea fiind proporționale cu prețul.

Oracle Database rulează pe o multitudine de platforme și sisteme de operare, versiunea 10g fiind cea mai generoasă din acest punct de vedere. Acesta rulează pe: Apple Mac OS X Server, HP HP-UX, HP Tru64 UNIX, HP OpenVMS, IBM AIX5L, IBM z/OS, Linux (x86, x86-64 etc), Microsoft Windows (x86, x86-64, Itanium), Sun Solaris (SPARC, x86, x86-64). Oracle oferă suport pentru majoritatea tipurilor de date fără limitări cu privire la dimensiunea maximă a bazei de date sau a unei valori stocate într-o coloană de tip *Character large object* (CLOB). Oracle oferă suport pentru tabele temporare, tabele virtuale materializate, îndeși de tipul *R-/R+ tree*, *hash*, la nivel de expresie, parțiali, inversi, *bitmap*, pentru căutări în text și spațiali. Oracle oferă nativ partaționarea *hash*, *range*, listă și mixtă precum și modalități complexe, certificate de nivel 4 *Evaluation Assurance Level – EAL4+*, de control al accesului [KNGA09] (criptarea comunicațiilor, *Lightweight Directory Access Protocol* - LDAP, protecție la tentative de spargere a parolei, setarea regulilor de complexitate a parolei, auditare, stabilirea de limitări a resurselor etc.).

**Microsoft SQL SERVER** este un SGBD relațional produs de către Microsoft. Prima versiune a apărut în 1989, cu suport pentru 16 biți și rula

doar pe OS/2, sistemul de operare creat de *Microsoft* și IBM și dezvoltat ulterior doar de către IBM. Prima versiune pentru WinNT a fost 4.21 apărută în 1993. Versiunea 2005 a introdus suport pentru stocarea și manipularea datelor XML, facilități pentru tratarea erorilor, partaționarea tabelelor și a indecșilor, baze de date în oglindă. Cea mai recentă versiune apărută este SQL Server 2008 R2 care oferă facilități pentru lucrul cu date nestructurate, semi-structurate, spațiale, precum și o multitudine de utilitare. Există nu mai puțin de 11 ediții ale acestui SGBD adresate diverselor clase de utilizatori, *SQL Server Enterprise Edition* fiind cea mai completă variantă. Acest SGBDR rulează doar pe sisteme de operare Windows. SQL Server 2008 oferă suport pentru majoritatea tipurilor de date cu limitări privind dimensiunea maximă a bazei de date (16TB) sau a unei valori stocate într-o coloană de tip CLOB (2 GB) [RABE10]. Se oferă suport pentru tabele temporare, tabele virtuale materializate, indecși de tip *hash*, la nivel de expresie, parțiali, inversi, *bitmap*, pentru căutări performante în text și spațiali. SQL Server oferă suport nativ doar pentru partaționarea de tip *hash*. Sunt implementate modalități complexe, certificate EAL4+, de control al accesului.

**IBM DB2** este un SGBDR produs de către IBM. Prima versiune a apărut în 1983 pentru platforma MVS. Înainte de DB2, IBM a dezvoltat în anii '70 primul SGBDR System R, la nivel de prototip, implementând un precursor al limbajului SQL, acest produs nefiind comercializat. SQL/DS a fost prima implementare de SGBDR pentru calculatoare *mainframe* a IBM [GRAN08]. Cea mai recentă versiune, 9.7 a fost lansată în anul 2009 și poate rula pe Windows, Mac OS X, Linux, Unix și z/OS. Există trei ediții comerciale ale acestui produs: *Express Edition*, *Workgroup Server Edition* și *Enterprise Server Edition* adresate diverselor clase de utilizatori, precum și una oferită gratuit DB2 Express-C. DB2 oferă suport pentru majoritatea tipurilor de date cu limitări privind dimensiunea maximă a bazei de date (512TiB) sau a unei valori stocate într-o coloană de tip CLOB (2GB). Se oferă suport pentru tabele temporare, tabele virtuale materializate, indecși la nivel de expresie, inversi, *bitmap* și pentru căutări în text. DB2 oferă nativ partaționarea *hash*, *range*, listă și mixtă precum și modalități complexe, certificate EAL4+, de control al accesului (criptarea comunicațiilor, LDAP, setarea regulilor de complexitate a parolei, auditare, stabilirea de limitări a resurselor etc).

**MySQL** este un SGBDR dezvoltat în prezent de către MySQL AB, o subsidiară a Oracle Corporation. MySQL este în general oferit gratuit

existând și posibilități de licențiere comercială în anumite situații. MySQL este folosit de multe produse WWW, cele mai cunoscute fiind Facebook, Wikipedia și Google. Prima versiune a fost lansată în anul 1995, fiind dezvoltată de către Michael Widenius și David Axmark. Versiunea 5.0 a oferit suport pentru proceduri și funcții stocate, cursori și declanșatori. În 2008 MySQL AB este achiziționat de Sun Microsystems fiind lansată ulterior versiunea 5.1 cu suport pentru partiționare și replicare, dar conținând inițial numeroase erori. În 2010 Oracle achiziționează Sun, ultima versiune de MySQL lansată fiind 5.5. MySQL, care rulează practic pe toate platformele: Windows, Linux, Unix, Solaris, BSD, Symbian, Amiga, z/OS etc. MySQL oferă suport pentru majoritatea tipurilor de date fără limitări privind dimensiunea maximă a bazei de date și putând stoca valori într-o coloană de tip CLOB de maxim 4 GB. Se oferă suport pentru tabele temporare, dar nu și pentru tabele virtuale materializate. Suportul nativ pentru lucrul cu indecsi este mai limitat: *R-/R+ tree, hash*, pentru căutări performante în text [SCZA08]. MySQL oferă nativ partiționarea *hash, range, listă și mixtă*, precum și anumite modalități, fără certificare EAL de control al accesului.

## Rezumat

Capitolul tratează patru părți importante, analizate într-o ordine logică. Pentru început este prezentat modelul de date relațional ca fiind constituit din structura relațională, operatorii relationali și restricțiile de integritate. Modelul relațional oferă două colecții de operatori pe relații, și anume algebra relațională și calculul relațional. La rândul său, calculul relațional este de două tipuri: calcul relațional orientat pe tuplu și calcul relațional orientat pe domeniu.

În continuare se definește o bază de date relațională (BDR) ca fiind un ansamblu de relații (tabele) de date, împreună cu legăturile dintre ele.

Se tratează apoi sistemele de gestiune a bazelor de date relaționale (SGBDR). Codd a formulat 13 reguli care exprimă cerințele pe care trebuie să le satisfacă un SGBD pentru a fi complet relațional. Există SGBDR care combină modelul relațional cu cel orientat mixt, numindu-se sisteme de gestiune a bazelor de date relațional-obiectuale.

Cele mai importante SGBDR, din punctul de vedere al performanțelor dar și a cotei de piață, în prezent sunt Oracle, SQL Server, DB2 și MySQL. Acestea implementează caracteristicile de bază ale unui SGBDR, și anume: atomicitatea, consistența, izolarea și durabilitatea tranzacțiilor, restricțiile referențiale și standardul de reprezentare și manipulare a datelor UNICODE, se pot folosi fără restricții operatorii relationali, inclusiv joncțiunile externe. Aceste sisteme implementează limbaje procedurale în care se pot folosi construcții cum ar fi: cursori, funcții, proceduri, declanșatori.

În finalul capitolului este prezentat limbajul de bază al tuturor SGBDR – limbajul SQL. Acesta este un limbaj de descriere și manipulare acceptat de către ANSI și ISO drept un standard pentru bazele de date relaționale.

## Note bibliografice

Noțiuni legate de bazele de date relaționale sunt prezentate pe larg în lucrările autorilor [LUSA03], [LUNG05a]. La realizarea capitolului au fost valorificate și surse bibliografice externe, precum cercetările publicate în [CODD70], [BLAS90] și [ULLM89].



## Capitolul 5

# REALIZAREA BAZELOR DE DATE RELAȚIONALE

Realizarea unei baze de date relaționale este o activitate incrementală care pornește de la analiza de sistem, pentru domeniul în care se realizează aceasta și de la cerințele informaționale asociate, se continuă cu proiectarea structurii (schema conceptuală, externă și internă) și cu încărcarea datelor. După realizarea acestor etape se poate trece la exploatarea și întreținerea unei baze de date relaționale.

Cerințele informaționale inițiale pot suferi în timp modificări, de aceea se vor parcurge din nou una sau mai multe etape ale realizării, actualizându-se conținutul, dar și structura bazei de date.

**Cuvinte-cheie:** analiză de sistem, analiză structurală, normalizare, asocieri, diagrama entitate-asociere, schemă conceptuală, schemă internă, schemă externă.

*The development of a relational database is an incremental activity that starts from the system analysis for which the associated information and requirements are concern, continuing with the design structure (conceptual schema, external and internal schemas) and loading data into the database. After performing these steps, one can proceed to operation and maintenance of relational databases.*

*Initial information requirements may suffer some changes in time, so it is recommended to revise some stages of implementation and also for updating the content and structure of the database.*

**Keywords:** systems analyses, structural analyses, normalization, relationships, entity-relationships diagram, conceptual schema, internal schema, external schema.

## 5.1 Etapele de realizare a unei baze de date relaționale

Realizarea unei baze de date relaționale presupune parcurgerea unei succesiuni de etape. Acestea sunt:

- analiza de sistem, pentru domeniul în care se realizează baza de date relațională și a cerințelor informaționale asociate;
- proiectarea structurii bazei de date relaționale (schema conceptuală, externă și internă);
- implementarea bazei de date relaționale;
- exploatarea și întreținerea bazei de date relaționale.

Conținutul acestor etape, mai precis activitățile implicate și modul lor de desfășurare (figura 5.1) depind, în general, de domeniul pentru care se construiește baza de date relațională. Se pornește de la un set inițial de cerințe informaționale, pe baza cărora se parcurg mai multe dintre etapele prezentate în figură. Ulterior, plecând de la niște cerințe informaționale actualizate și de la rezultatele proiectării se pot parcurge una sau mai multe etape ale realizării, actualizându-se conținutul, dar și structura bazei de date. Anumite cerințe actualizate pot determina modificări în proiectarea bazei de date și implicit a structurii acesteia. Există însă aspecte cu caracter general, precum metodele și tehniciile utilizate, care nu sunt influențate de specificul unui anumit domeniu economico-social. Cele mai importante tehnici de analiză și proiectare, respectiv tehnica diagramelor entitate-asociere și tehnica normalizării relațiilor vor fi prezentate în cadrul acestui capitol.

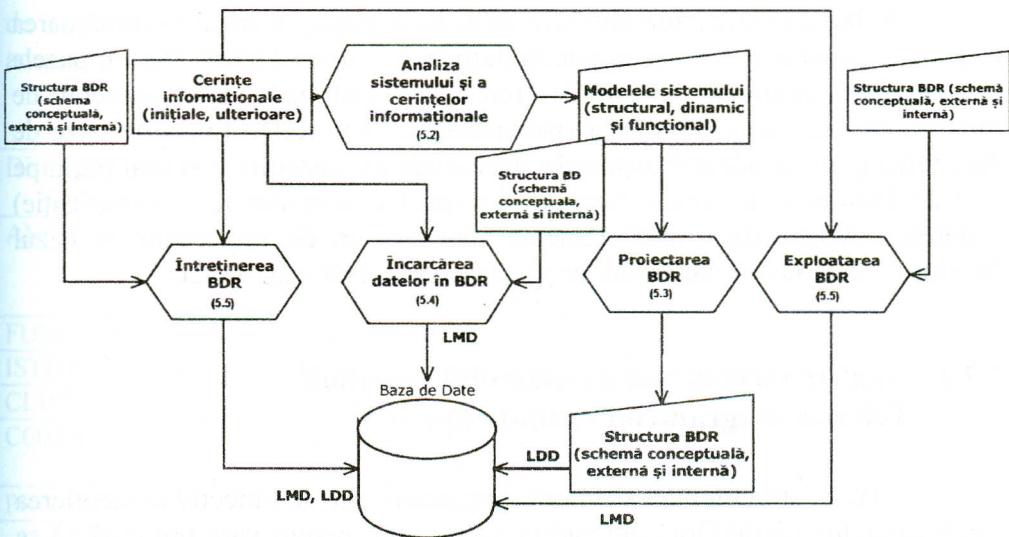


Figura 5.1 Etapele de realizare a unei baze de date relaționale

## 5.2 Analiza de sistem

Realizarea unei analize de sistem riguroase presupune realizarea unei succesiuni de activități. Acestea sunt:

- analiza componentelor sistemului și a legăturilor (asocierilor) dintre acestea, activitate cunoscută sub numele de *analiză structurală* sau *statică*, în urma căreia se obține *modelul structural (static)* al sistemului;
- analiza stăriilor sistemului și a tranzițiilor posibile între aceste stări, în raport de anumite evenimente. Este așa numita *analiză temporală (de comportament)*, prin care se obține modelul dinamic (*temporal*) al sistemului;
- analiza cerințelor informaționale, respectiv a transformărilor de date (a tranzacțiilor) din cadrul sistemului prin care sunt satisfăcute cerințele informaționale asociate domeniului economic. În urma acestei activități se obține *modelul funcțional (transformatiunal)* al sistemului economic;
- integrarea modelelor sistemului economic (structural, dinamic și funcțional) în scopul corelării și completării lor.

Pe baza rezultatelor obținute în această etapă se trece la următoarea etapă de realizare a BDR și anume la definirea structurii BDR. Pentru bazele de date pre-relaționale (ierarhice, rețea) și relaționale este deosebit de importantă analiza structurală a sistemului întrucât aceste tipuri de baze de date reflectă preponderent aspectele structurale ale sistemului și mai puțin pe cele de dinamică (de comportament) și respectiv de semantică (semnificație) a datelor. Aceste din urmă elemente sunt extrem de importante în cazul bazelor de date post-relaționale, în special a celor orientate-obiect.

### 5.2.1 Analiza structurală a sistemului economic Tehnica diagramelor entitate-asociere

Analiza structurală a sistemului economic are ca obiectiv evidențierea componentelor (entităților) din cadrul sistemului, pentru care urmează să se colecteze și să se memoreze date în cadrul bazei de date, precum și evidențierea legăturilor dintre aceste componente. Se cunosc mai multe tehnici de analiză structurală și anume:

- tehnică entitate-asociere (E-R: *Entity-Relationship*), introdusă de P.P.S. Chen, în 1976 [CHEN76];
- tehnică SDM (*Semantic DataModel*), introdusă de Hammer în 1981 [HAHA92];
- tehnică IFO introdusă de Serge Abiteboul și Richard Hull [ABHU87];
- tehnică RM/T (*Relational Model/Tasmania*) introdusă de Codd [CODD79] etc.

Dintre acestea, tehnică entitate-asociere este cea mai utilizată. Tehnica entitate-asociere permite constituirea modelului structural sub forma unei diagrame entitate-asociere prin parcurgerea următorilor pași:

- identificarea componentelor (entităților) din cadrul sistemului economic;
- identificarea asocierilor dintre entități și calificarea acestora;
- identificarea atributelor aferente entităților și a asocierilor dintre entități;
- stabilirea atributelor de identificare a entităților.

#### A) Identificarea componentelor sistemului economic

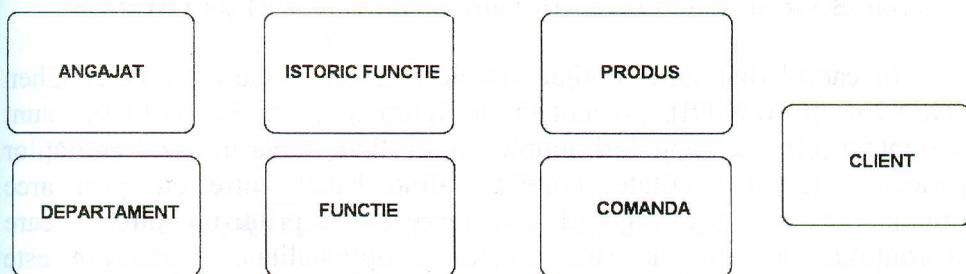
Componentele sistemului economic sunt denumite *entități* și sunt reprezentate în cadrul diagramei prin blocuri dreptunghulare. Să considerăm,

de exemplu activitatea comercială dintr-o organizație economică. Pentru acest domeniu pot fi puse în evidență entitățile din tabelul 5.1.

**Tabelul 5.1 Descrierea entităților folosite**

DEPARTAMENT	Departamentele din interiorul firmei. Fiecare departament are un <b>șef</b> .
ANGAJAT	Angajații firmei. Fiecare angajat poate face parte dintr-un departament, are o funcție prezentă, un istoric al funcțiilor și poate avea un manager.
FUNCTIE	Funcțiile pe care le poate avea angajații.
ISTORIC_FUNCTIE	Funcțiile pe care le-au avut angajații.
CLIENT	Clienții firmei.
COMANDA	Comenzile date de către clienți, <i>online</i> , telefonic sau intermediate de către un angajat.
PRODUS	Produsele comercializate de către firmă.

Fiecare entitate prezintă mai multe instanțe (realizări). De exemplu, organizația are mai mulți clienți, mai mulți furnizori etc. În urma acestei etape se elaborează o primă formă a diagramei entitate-asociere. Figura 5.2 prezintă prima formă a diagramei entitate-asociere realizată pentru exemplul considerat.



**Figura 5.2 Diagrama entitate-asociere pentru domeniul comercial (prima formă)**

### B) Identificarea asocierilor dintre entități și calificarea acestora

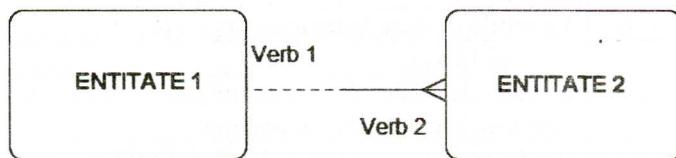
În cadrul sistemului analizat, entitățile nu sunt decât arareori izolate. Cel mai adesea, între acestea se stabilesc legături, asocieri. Există mai multe modele de reprezentare a asocierilor dintre entități, în figurile 5.3, 5.4, 5.5, 5.6 prezentăm notațiile a trei dintre ele: Chen, Oracle și Bachman. Luând, ca exemplu entitățile COMANDA și CLIENT din figura 5.2, între aceste entități există o asociere, clienții realizează comenzi, altfel spus comenzile sunt realizate de clienți. Întrucât asocierile dintre entități pot fi de mai multe tipuri

și pot avea diferite semnificații, se impune ca odată cu identificarea să se realizeze și calificarea lor.

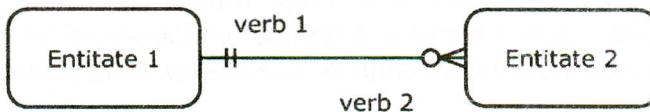
a) Diagrama E-R Chen



b) Diagrama E-R Oracle



c) Diagrama E-R Bachman



**Figura 5.3 Reprezentarea asocierii dintre entitățile CLIENT și COMANDA**

În cadrul diagramei entitate-asociere (E-R) propuse de Peter Chen ([CHEN76], [CHEN99]), prezentată în figura 5.3a. și 5.4, entitățile sunt reprezentate prin dreptunghiuri simple sau cu linie dublă în cazul entităților dependente de altă entitate, entitățile fiind legate între ele prin arce neorientate. Semnificația legăturii este reprezentată printr-un romb în care este conținut un verb, iar cardinalitatea și optionalitatea legăturilor este reprezentată printr-o pereche de valori ( $X,Y$ ) la fiecare capăt al acesteia. Atributele sunt reprezentate prin cercuri legate prin linii de dreptunghiul care reprezintă entitatea, identificatorul unic fiind subliniat. Cardinalitatea asocierilor exprimă numărul minim și maxim de realizări (instanțe) de entitate care pot fi asociate cu o realizare a partenerului de asociere. Perechea (0,1) este atașată ENTITĂȚII 1 și are următoarea semnificație: numărul minim de realizări ale ENTITĂȚII 1 care se pot asocia cu o realizare a ENTITĂȚII 2 este zero, în timp ce numărul maxim este unu. Perechea (1,N) este asociată ENTITĂȚII 2 și se interpretează astfel: numărul minim de realizări ale ENTITĂȚII 2 care se pot asocia cu o realizare a ENTITĂȚII 1 este unu, în timp ce numărul maxim este nelimitat.

Notatii ERD Peter Chen			
Cardinalitate		Optionalitate	
1	Unu	0	Optional
N	Multi	1	Obligatoriu

Figura 5.4 Cardinalitate și optionalitate în diagrama Chen

În cadrul diagramei E-R utilizată de Oracle (în pachete *software* precum Designer, SQL Developer Data Modeler), prezentată în figura 5.3b și 5.5, entitățile sunt reprezentate prin dreptunghiuri cu marginile rotunjite legate între ele prin arce neorientate. Optionalitatea legăturii este reprezentată prin linie punctată, obligativitatea prin linie continuă, cardinalitatea prin linie dreaptă și „picior de cioară”. Atributele sunt reprezentate în interiorul entității, identificatorul unic fiind marcat prin #.

Notatii ERD Oracle	
— verb 1/verb 2 —	Relatia
—————	Obligatoriu, Unu
————→	Obligatoriu, Multi
-----	Optional, Unu
-----→	Optional, Multi

Figura 5.5 Cardinalitate și optionalitate în diagrama Oracle

În cadrul diagramei E-R cu notații Bachman (utilizată în utilitare precum Visible Analyst și Visio), prezentată în figura 5.3c. și 5.6, entitățile și legăturile sunt asemănătoare diagramei E-R Oracle cu excepția faptului că participarea optională este marcată printr-un cerc, iar cea obligatorie prin două linii verticale paralele (există și variante cu o singură linie verticală).

Notatii ERD Bachman	
— verb 1/verb 2 —	Relatia
—————	Cardinalitate (Unu)
—————<	Cardinalitate (Multi)
—————	Obligatoriu, Unu
————— <	Obligatoriu, Multi
—————○	Optional, Unu
—————○<	Optional, Multi

Figura 5.6 Cardinalitate și optionalitate în diagrama Bachman

Determinarea cardinalității unei asocieri presupune analiza comportamentului realizărilor de entitate în cadrul asocierii (figura 5.7).

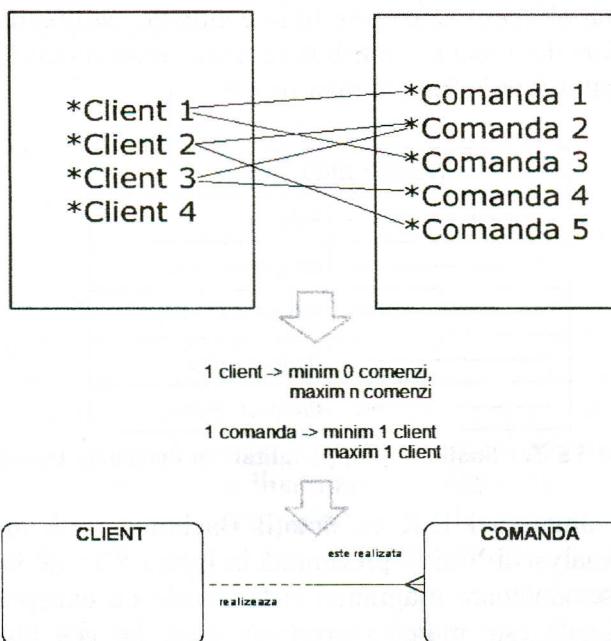


Figura 5.7 Determinarea cardinalității dintre entitățile CLIENT și COMANDA

Asocierile dintre entități pot fi clasificate după mai multe criterii, și anume:

#### A1. Cardinalitatea asocierii:

- gradul asocierii (maximele cardinalității):
  - asocieri de tip *unu la unu*;
  - asocieri de tip *unu la mulți*;
  - asocieri de tip *mulți la mulți*.
- obligativitatea participării entităților la asociere (minimele cardinalității):
  - asocieri parțiale;
  - asocieri totale (complete).
- concomitent după gradul asocierii și după obligativitatea participării la asociere:
  - asocieri parțiale de tip *unu la unu*;
  - asocieri totale de tip *unu la unu*;
  - asocieri parțiale de tip *unu la mulți*;
  - asocieri totale de tip *unu la mulți*;
  - asocieri parțiale de tip *mulți la mulți*;
  - asocieri totale de tip *mulți la mulți*.

#### A2. Numărul de entități distincte care participă la asociere:

- asocieri binare (între două entități distincte);
- asocieri recursive (asocieri ale entităților cu ele însese);
- asocieri complexe (între mai mult de două entități distincte).

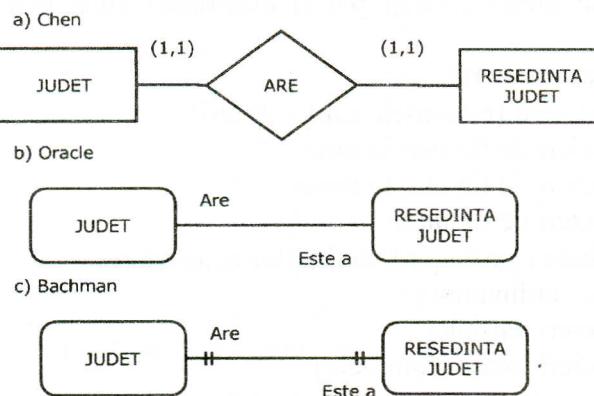
#### A3. Semnificația asocierii:

- este un (generalizare);
- este conținut în (aggregare).

În continuare, vor fi prezentate pe rând aceste tipuri de asocieri între entități. Întrucât la prezentarea asocierilor în funcție de cardinalitate exemplificările sunt realizate cu ajutorul asocierilor binare, la caracterizarea asocierilor în funcție numărul de participanți se vor avea în vedere numai asocierile recursive și cele complexe.

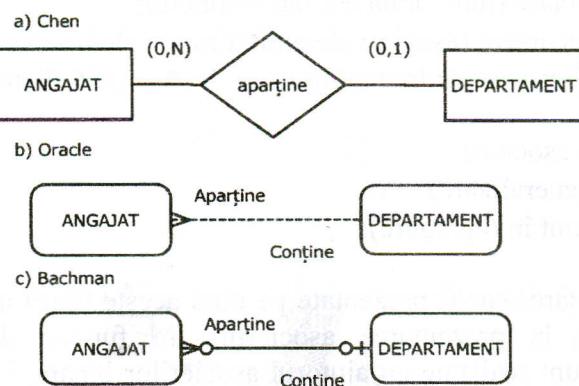
#### **Asocieri de tip *unu la unu***

Sunt asocieri în care maximele cardinalității au valoare unu, spre exemplu legătura dintre județ și reședința de județ (figura 5.8).

Figura 5.8 Asociere de tip *unu la unu*

### Asocieri de tip *unu la mulți*

Sunt asocieri în care una dintre maximele cardinalității are valoare unu, iar cealaltă are valoarea *mulți*. Figura 5.9 prezintă asocieri de tip *unu la mulți* între entitățile CLIENT și COMANDA. În situația prezentată în figură, un angajat poate aparține minim niciunui departament și maxim unui departament, iar un departament poate să nu conțină niciun angajat, iar maxim  $N$  angajați.

Figura 5.9 Asociere de tip *unu la mulți*

### Asocieri de tip *mulți la mulți*

Sunt asocierile în care maximele cardinalității au valoarea *mulți* (figura 5.10). Această figură prezintă asocierea de tip *mulți la mulți* dintre entitățile COMANDA și PRODUS. O comandă poate să nu conțină niciun

produs atunci când este creată inițial, iar maxim poate conține  $N$  produse. Un produs poate să nu fie comandat niciodată de către vreun client, iar maxim poate fi comandat de  $M$  ori.

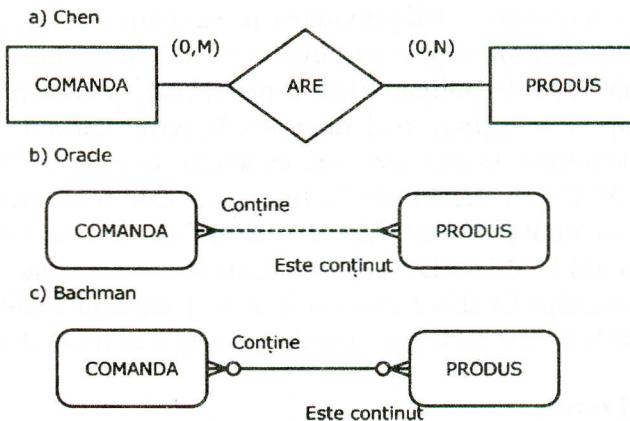


Figura 5.10 Asociere de tip **mulți la mulți**

Se obișnuiește ca asocierile de tip *mulți la mulți* să se transforme în două asocieri de tip *unu la mulți*. Această transformare, care se realizează prin introducerea unei entități intermediare (figura 5.11), permite ca în locul asocierilor de tip *mulți la mulți*, să se lucreze cu asocieri de tip *unu la mulți*, de regulă mai ușor de implementat și de utilizat. Gradul asocierii (*unu la unu*, *unu la mulți*, *mulți la mulți*) reprezintă principala caracteristică a unei asocieri. În acest caz, se introduce entitatea RAND\_COMANDA și două asocieri parțiale de tipul *unu la mulți*.

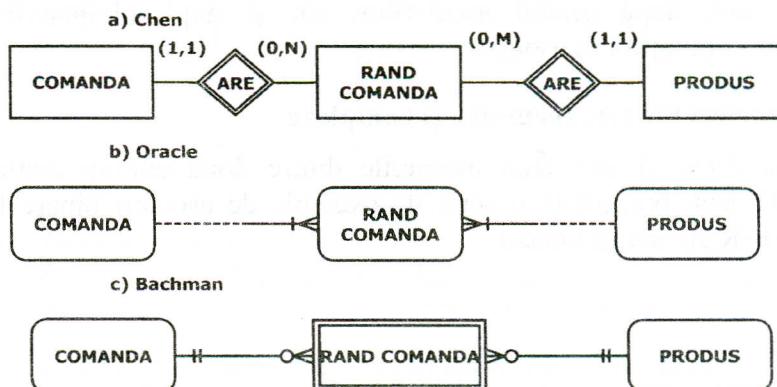


Figura 5.11 Transformarea unei legături mulți la mulți în două legături unu la mulți

### Asocieri parțiale

O asociere este parțială atunci când nu există obligativitatea participării la această asociere a tuturor entităților vizate, ci numai a uneia dintre ele sau a niciuneia. Obligativitatea participării entităților la asociere este reflectată de către minimele cardinalității. Valoarea zero a uneia dintre minime exprimă lipsa de obligativitate a participării partenerului la această asociere, în timp ce o valoare mai mare decât zero exprimă obligativitatea participării partenerului la asociere. De exemplu, asocierea dintre entitățile CLIENT și COMANDA prezentată în figura 5.7 este o asociere parțială, în care participarea entității COMANDA nu este obligatorie (numărul minim de realizări ale entității COMANDA care participă la asociere este zero), în timp ce participarea entității CLIENT este obligatorie (numărul minim de realizări ale entității CLIENT care participă la realizare este mai mare decât zero).

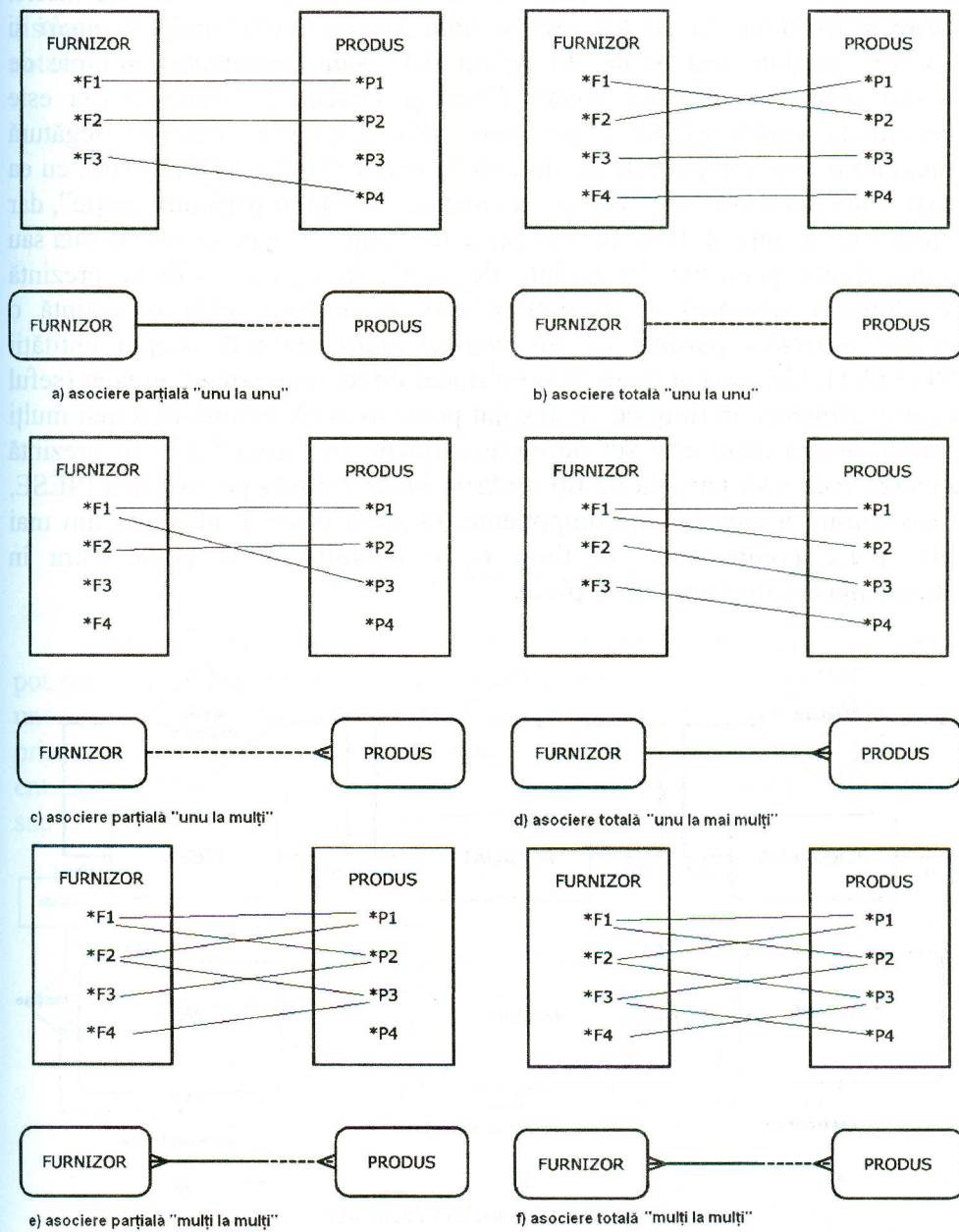
### Asocieri totale

Sunt asocierile la care este obligatorie participarea tuturor entităților la asociere, altfel spus sunt asocierile pentru care minimele cardinalității au valori mai mari decât zero. Figura 5.8 prezintă o asociere totală între entitățile JUDET și RESEDINTA\_JUDET.

Atunci când se califică o anumită asociere, de cele mai multe ori este necesară caracterizarea acesteia atât în funcție de gradul său, cât și de obligativitatea participării entităților la asociere. Din această perspectivă, o asociere se caracterizează ca fiind: asociere parțială de tip *unu la unu*, asociere totală de tip *unu la unu*, asociere parțială de tip *unu la mulți*, asociere totală de tip *unu la mulți*, asociere parțială de tip *mulți la mulți* sau asocierea totală de tip *mulți la mulți*. Si în figura 5.11 sunt prezentate exemple de calificări, atât după gradul asocierilor, cât și după obligativitatea de participare a entităților la asociere.

### Asocieri binare, recursive și complexe

*Asocierile binare* sunt asocierile dintre două entități distincte. În figura 5.12 sunt prezentate o serie de exemple de asocieri binare folosind diagrame E-R cu notății Oracle.



**Figura 5.12 Asocieri binare, calificate concomitent după gradul lor și după obligativitatea participării entităților la asociere**

*Asocierile recursive* reprezintă asocieri ale entităților cu ele însese. Aceste asocieri pot fi de tip *unu la unu*, *unu la mulți*, *mulți la mulți* și respectiv, parțiale sau totale. În figura 5.13 sunt prezentate exemple de asocieri recursive folosind notații Chen și Oracle. În figura 5.13a este reprezentată asocierea între persoane prin căsătorie. Această legătură reprezintă o asociere parțială de tip *unu la unu* a entității PERSOANE cu ea însăși. Unei persoane „soț” i se poate asocia, eventual o persoană „soție”, dar nu mai mult de una, în timp ce unei persoane „soție” i se poate asocia una sau niciuna dintre persoane, în calitate de „soț”. În figura 5.13b se prezintă subordonarea ierarhică a angajaților unei organizații, care reprezintă o asociere recursivă parțială de tip *unu la mulți*, definită asupra entității ANGAJAȚI. Un angajat poate fi subordonat direct unui singur angajat (șeful de compartiment), în timp ce un angajat poate avea ca subordonați mai mulți angajați, atunci când este șef de compartiment. În figura 5.13c se prezintă asocierea recursivă parțială de tip *mulți la mulți* definită pe entitatea PIESE, de structurare a pieselor în componente. O piesă poate fi alcătuită din mai multe piese componente, în timp ce o anumită piesă poate intra în componență (în structura) altor piese.

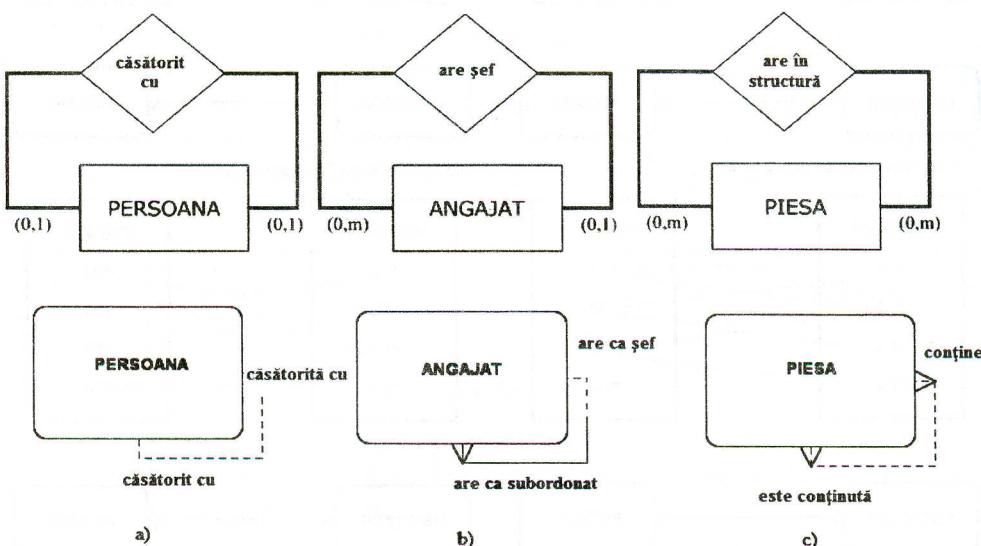


Figura 5.13 Asocieri recursive

*Asocierile complexe* reprezintă asocieri între mai mult de două entități distincte. Figura 5.14 prezintă un exemplu de asociere complexă, folosind notații Chen, între entitățile: ANGAJAT, CLIENT și PRODUS.

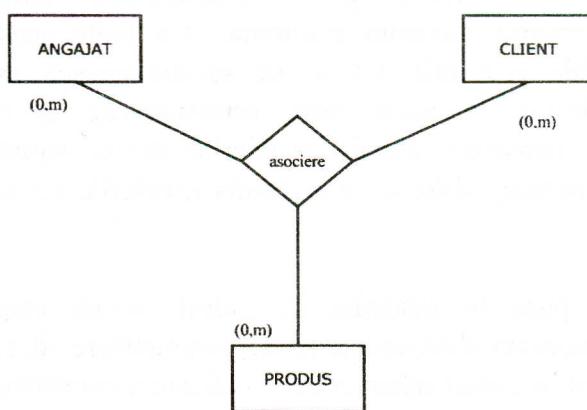


Figura 5.14 Asociere complexă

Acstea entități sunt asociate printr-o legătură complexă, prin care se pot determina caracteristicile activităților comerciale, desfășurate cu ajutorul unor comenzi. În locul unei asocieri complexe se pot utiliza asocieri binare, prin introducerea unor entități intermediare (figura 5.15a.) sau a mai multor entități intermediare (figura 5.15b). Acest procedeu, folosind și notații Oracle sau Bachman a fost prezentat și în figura 5.11.

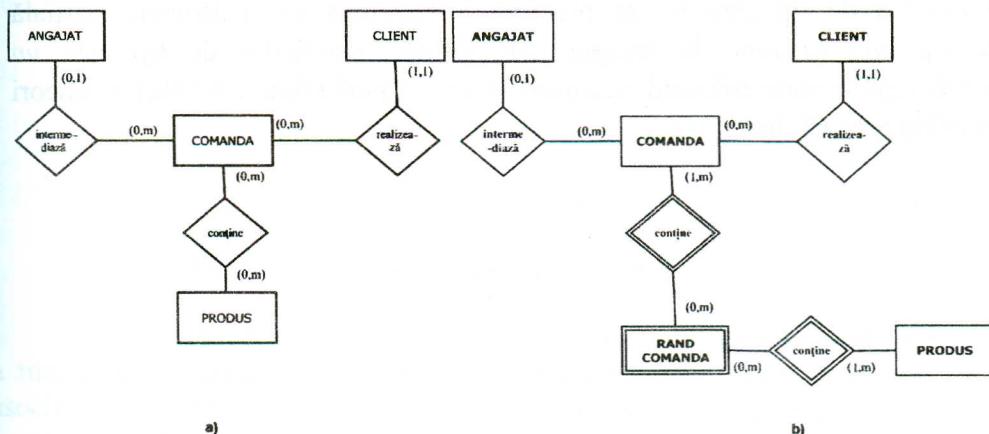


Figura 5.15 Utilizarea unei sau a mai multor entități de legătură

### **Asocieri de tipul *este un* și *este conținut în***

Asocierile dintre entități pot avea semnificații extrem de variate în cadrul unui anumit domeniu economic. Cu toate acestea, în cadrul analizei structurale este util, totuși, să se urmărească asocierile cu o anumită semnificație. În acest sens, este necesar să fie identificate și analizate, cu deosebită atenție asocierile dintre entitățile aflate pe niveluri de abstractizare diferite, mai precis asocierile de tipul *este un* și *este conținut în*.

Entitățile puse în evidență în cadrul primei etape de analiză structurală pot prezenta diferite grade de abstractizare, deci se pot plasa pe diferite niveluri în cadrul schemei de clasificare a entităților (taxonomiei). În acest sens, există *entități generice (clase)*, obținute printr-un proces de *generalizare* a entităților cu anumite caracteristici comune, numite *subclase*. De exemplu, TERTI reprezintă o entitate clasă, care generalizează entitățile ANGAJAT și CLIENT. Angajatul, la rândul său poate fi ANGAJAT PERMANENT sau COLABORATOR (figura 5.16a.). Specializarea reprezintă procesul invers generalizării. Aceleași asocieri de tip *este un* din cadrul figurii 5.16 pot fi interpretate ca reprezentând specializarea clasei TERT în subclasele CLIENT și ANGAJAT, precum și specializarea entității ANGAJAT în entitățile PERMANENT și COLABORATOR. Asocierea de tip *este un* se prezintă întotdeauna ca o asociere parțială de tip *unu la unu*. În notația E-R Oracle, asocierile de tip *este un* pot fi reprezentate folosind subtipuri și supertipuri (figura 5.16b.) și uneori folosind arce (figura 5.16c.).

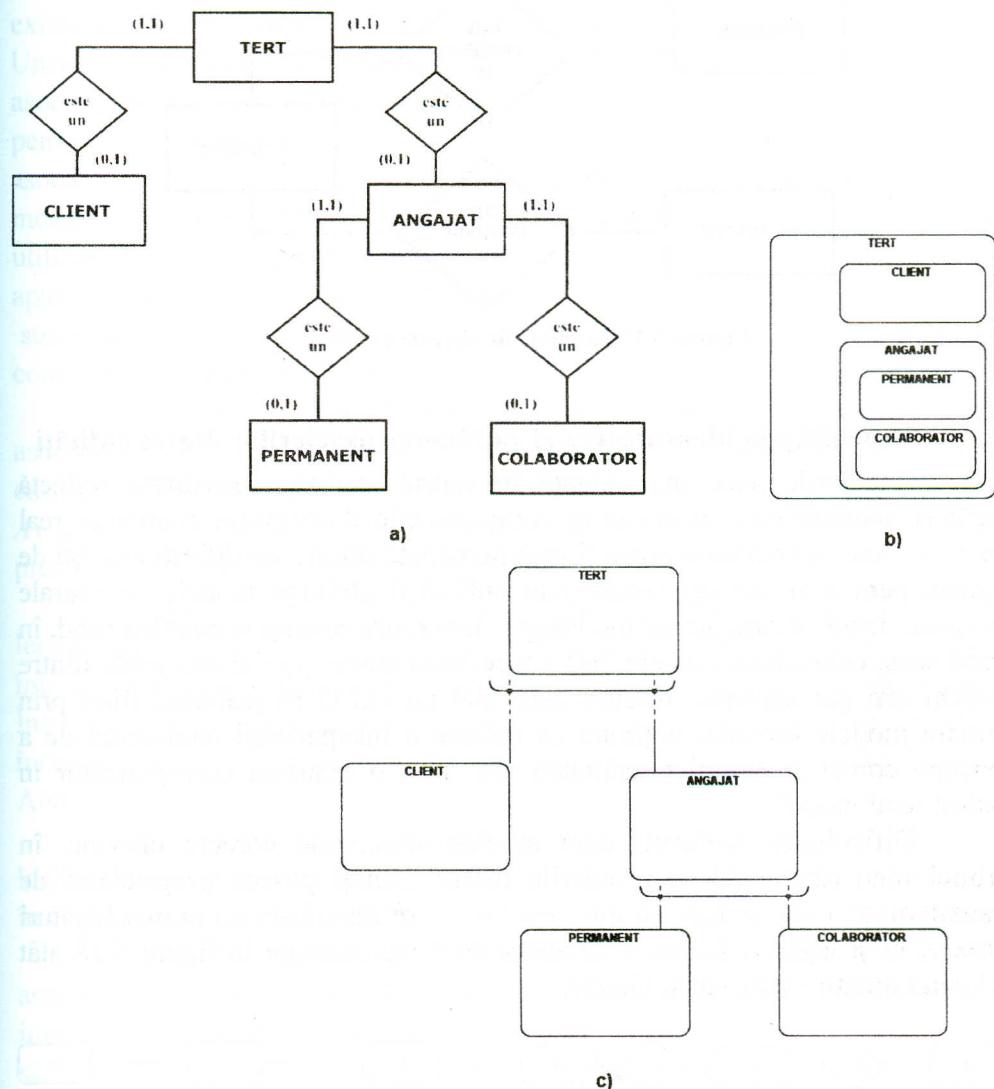
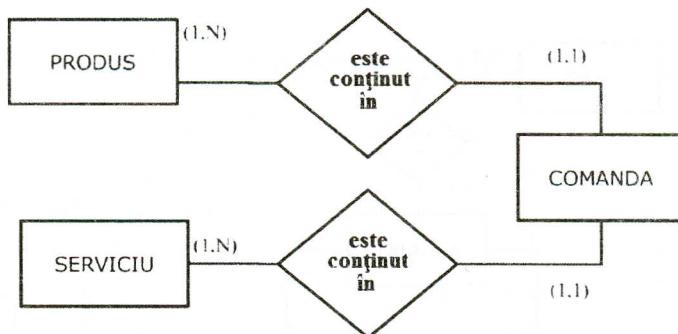


Figura 5.16 Asociieri de tipul *este un*, subtipuri, arce

Asocierea *este conținut în* exprimă un proces de *aggregare*, de grupare a mai multor entități într-o entitate de nivel superior. Spre deosebire de asocierea de tipul *este un*, asocierea *este conținut în* este calificată, după cardinalitate drept asociere de tipul *unu la mulți* sau *mulți la mulți*, totală sau parțială (figura 5.17).

Figura 5.17 Asocieri de tip *este conținut în*.

### Dificultăți în identificarea și calificarea asocierilor dintre entități

Asocierile puse în evidență în cadrul analizei structurate reflectă legături naturale care există între componentele domeniului economic real analizat. Aceeași realitate poate fi însă percepță diferit, de diferiți analiști de sistem, pentru un același sistem real putând fi obținute modele structurale distințe. Unele dintre aceste modele pot încorpora aceeași semantică fiind, în acest sens, echivalente. Altele însă pot reflecta numai parțial asocierile dintre entități sau pot exprima asocieri care nici nu există în realitate, fiind prin urmare modele eronate, obținute ca urmare a incapacității analistului de a percepe corect și complet realitatea sau de a exprima corespunzător în cadrul unui model.

Dificultatea realizării unei analize structurale corecte provine, în primul rând din faptul că asocierile dintre entități posedă proprietatea de tranzitivitate, ceea ce face ca între entități să se manifeste nu numai legături directe, ci și legături indirecte. Acestea sunt reprezentate în figura 5.18 atât folosind notății Chen, cât și Oracle.

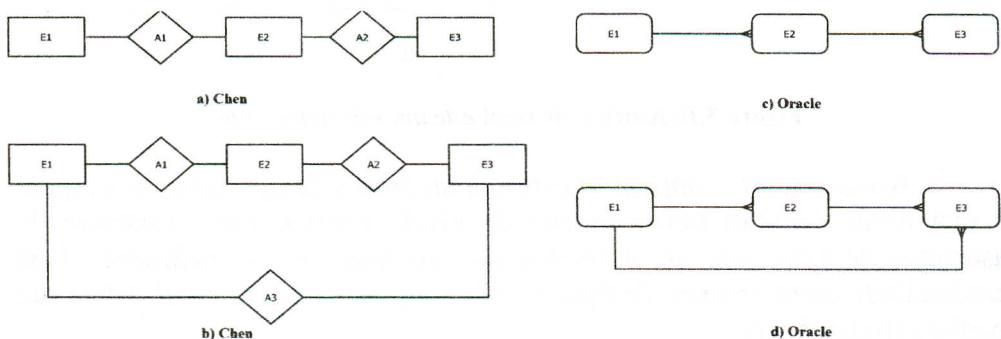


Figura 5.18 Legături directe și indirekte între entități

Între entitățile  $E1$  și  $E2$ , respectiv  $E2$  și  $E3$  din cadrul figurii 5.18 există asocierile directe  $A1$  și  $A2$ . Între  $E1$  și  $E3$  există și o legătură indirectă. Uneori, se preferă evidențierea legăturii între  $E1$  și  $E3$  și sub forma unei asocieri directe  $A3$  (figura 5.18b, 5.18d.).  $A3$  este redundantă atunci când pentru asigurarea unei legături corecte între  $E1$  și  $E3$  este suficientă asociera indirectă. Nu se recomandă includerea de legături redundante în modelul structural încărcat acestea complică modelul, făcându-l greu de utilizat. Există însă situații în care  $A3$  nu este redundantă. Aceste situații apar atunci când asocierile directe  $A1$  și  $A2$  nu sunt plasate într-o succesiune sau nu sunt definite într-un mod care să permită funcționarea corespunzătoare a legăturii indirecte.

În figura 5.19a sunt utilizate notațiile Chen. În situația descrisă, asociera  $A3$  poate fi considerată redundantă încărcat legătura dintre entitățile FIRMĂ și ANGAJATI este asigurată prin intermediul asocierilor  $A1$  și  $A2$ . Aceste asocieri permit determinarea angajaților unei anumite firme, precum și firma în care lucrează un anumit angajat.

În figura 5.19b, asociera  $A3$  nu este redundantă, deoarece pe baza legăturilor  $A1$  și  $A2$  nu se pot determina angajații care lucrează într-un anumit departament sau departamentul de care aparține un angajat. Încărcat asociera indirectă nu funcționează corespunzător este necesară încorporarea unei legături directe între entitățile DEPARTAMENT și ANGAJAT.

Figura 5.15c prezintă varianta cea mai bună pentru reflectarea legăturilor dintre entitățile FIRMA, DEPARTAMENT și ANGAJAT în cadrul modelului structural.

Figura 5.19d. ilustrează varianta cea mai bună de reflectare a asocierilor folosind notații Oracle. Exemplul prezentat evidențiază importanța unei corecte înlănuiri a legăturilor directe dintre entități, care să ducă la obținerea unor legături indirecte posibil de utilizat în mod eficient și care să permită simplificarea modelului structural, prin reducerea numărului de legături evidențiate explicit.

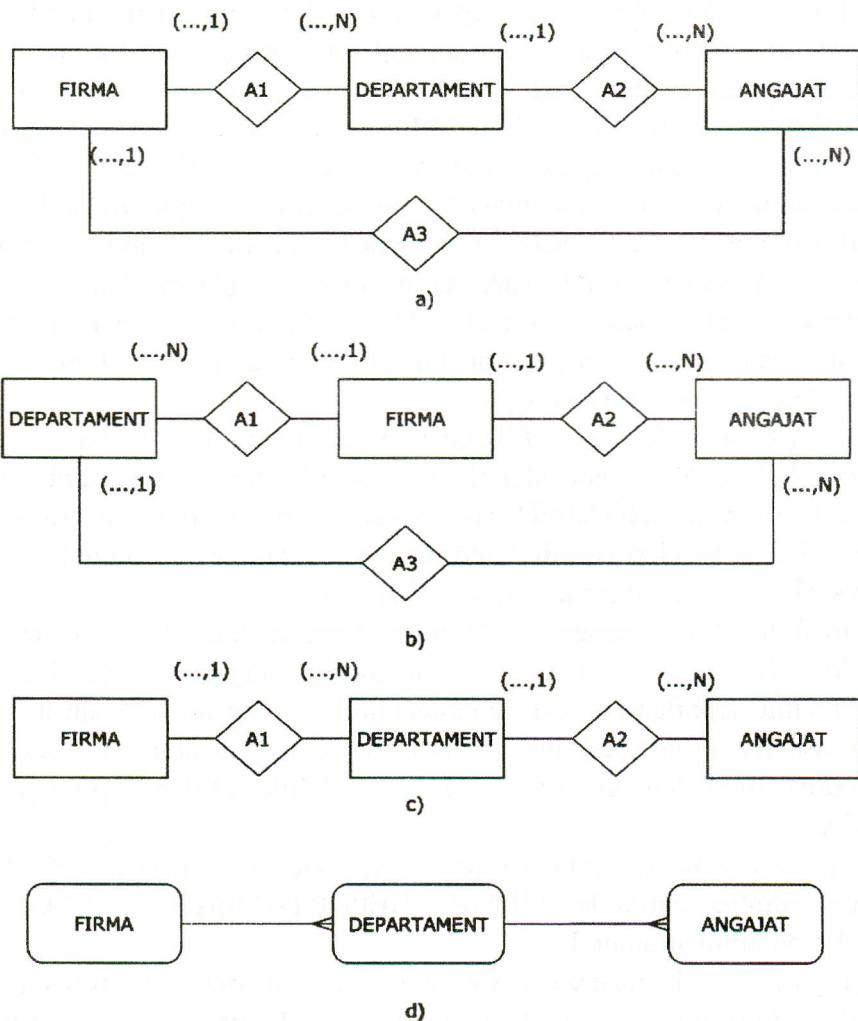
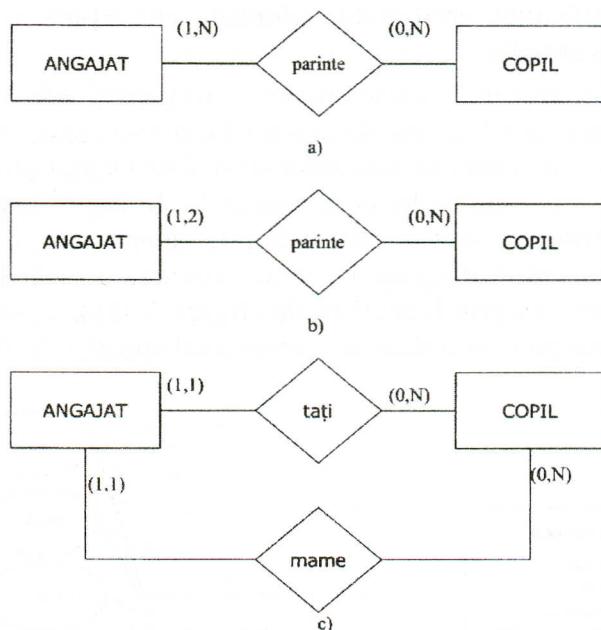


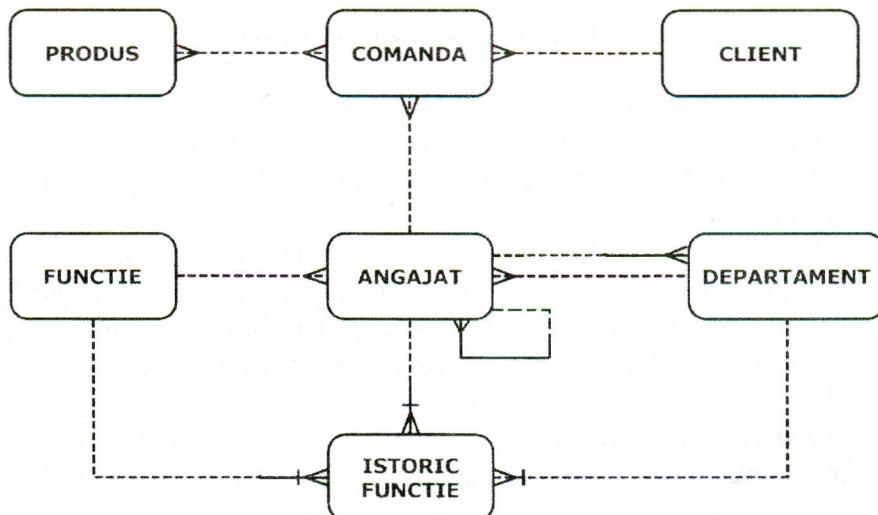
Figura 5.19 Asocieri directe și indirecte între entități

O altă dificultate întâmpinată în identificarea și calificarea asocierilor dintre entități constă în posibilitatea caracterizării aceleiași legături în mai multe moduri. De exemplu, legătura dintre entitățile ANGAJAT și COPIL (figura 5.20) poate fi calificată drept o asociere parțială de tip *mulți la mulți* (figura 5.20a), ca asociere parțială de tip *doi la mulți* (figura 5.20b) sau drept o legătură exprimată prin două asociere parțiale de tip *unu la mulți* (figura 5.20c) și care reprezintă altceva decât descompunerea asocierii de tip *mulți la mulți* în asocieri *unu la mulți*.



**Figura 5.20 Asocierea dintre entitățile ANGAJAT și COPIL**

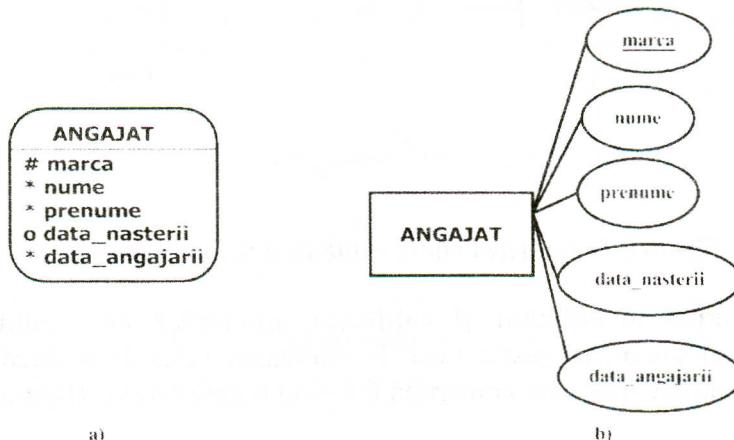
În urma identificării și calificării asocierilor între entitățile din domeniul comercial, se poate trece la realizarea celei de a doua forme a diagramei entitate-asociere, construită folosind notații Oracle (figura 5.21).



**Figura 5.21 Diagrama entitate-asociere pentru domeniul economic (forma a doua)**

### C) Identificarea atributelor aferente entităților și a asocierilor între entități

Atributele exprimă caracteristici, proprietăți ale entităților din domeniul economic analizat sau ale asocierilor dintre aceste entități. În mod curent, attributele sunt asociate entităților. Prin intermediul atributelor se pot descrie însă nu numai entitățile, ci și asocierile. În cazul diagramei E-R cu notații Oracle, attributele sunt reprezentate prin câmpuri în interiorul entității (figura 5.22a). În cadrul diagramei entitate asociere utilizând notații Chen, attributele sunt figurate prin blocuri ovale (figura 5.22b), legate de entitatea, respectiv asocierea pe care o descriu într-un mod specific, în funcție de tipul atributelor.

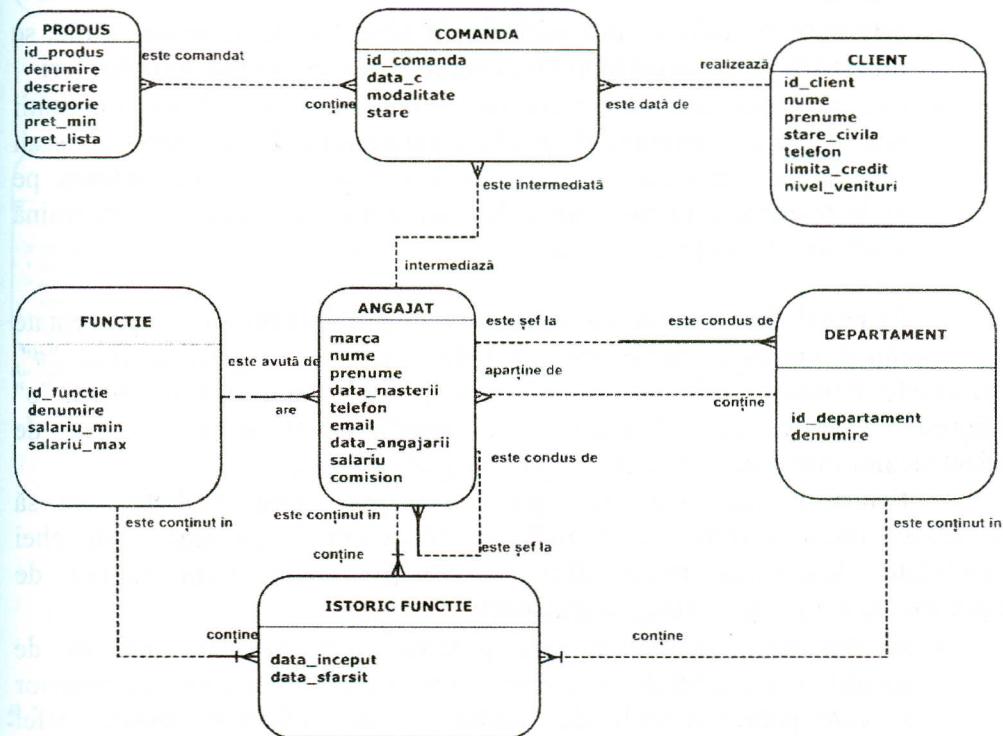


**Figura 5.22 Reprezentarea atributelor entităților**

Un atribut poate fi:

- *atribut compus (bloc)*. Este un atribut constituit din cel puțin două alte attribute. Valoarea sa este reprezentată de valorile atributelor componente;
- *atribut calculat (dedus)*. Reprezintă un atribut a cărui valoare nu este cunoscută direct, ci calculată pe baza valorilor altor attribute (de exemplu, atributul *valoare*, calculat ca produs între *cantitate* și *preț*);
- *atribut simplu*. Reprezintă un atribut care nu este nici compus și nici calculat. Valorile sale sunt valori atomice;
- *atribut repetitiv (multivaloare)*. Este un atribut care, la un moment dat, are mai multe valori care apar sub forma unei liste de valori.

În urma identificării atributelor pentru entitățile și asocierile din cadrul domeniului comercial, diagrama entitate-asociere ajunge în a treia formă, prezentată în figura 5.23.



**Figura 5.23 Diagrama E-R pentru domeniul economic (forma a treia)**

#### D) Stabilirea atributelor de identificare a entităților

Un *atribut de identificare (cheie)* reprezintă un atribut care se caracterizează prin unicitatea valorii sale pentru fiecare instanță a entității. De exemplu, ca attribute de identificare putem să considerăm: marca angajatului (pentru entitatea ANGAJATI), numărul de înmatriculare (pentru entitatea AUTOTURISME) etc.

În cadrul diagramelor entitate-asociere, attributele de identificare se marchează distinct. În scopul stabilirii atributelor de identificare pentru entitățile din cadrul sistemului (domeniului) economic analizat este necesar să se examineze mai întâi *potențialitatea* fiecărui atribut de a se constitui

drept atribut de identificare (cheie). Un atribut poate fi considerat de identificare dacă satisfacă o serie de *cerințe* și anume:

- oferă o identificare unică a realizărilor (instanțelor) de entitate;
- posedă o semnificație;
- este ușor de utilizat; atributele de tip bloc, lipsite de semnificație, se utilizează greu, favorizând producerea de greșeli la manipularea lor;
- este scurt; de cele mai multe ori atributul de identificare a entității apare și în alte entități, drept cheie externă. Pe de altă parte, indecșii pentru accesul direct la date se construiesc, cel mai adesea pe atributele cheie, cheile lungi (alcătuite din multe caractere) determină scăderea eficienței accesului.

În cazul diagramelor cu notații Oracle, atributele sunt reprezentate în interiorul entității. Identifierul unic este marcat prin semnul "#", atributele obligatorii prin semnul "\*", iar cele optionale folosind semnul "o" (figura 5.22a). În cazul diagramelor cu notații Chen, atributul cu rol de identifier unic este subliniat (marca în figura 5.22b.).

Pentru o aceeași entitate pot exista mai multe atrbute care să servească drept atrbute de identificare, adică pot exista mai multe chei candidate. Selectarea uneia dintre candidații cheie drept atrbut de identificare a entității (cheie) se realizează astfel:

- se determină atrbutele care potențial pot constitui atrbute de identificare a entității, deci care respectă cerințele menționate anterior și care poartă numele de candidați cheie. Dacă nu există astfel de atrbute se introduce un nou atrbut (grup de atrbute) drept candidat cheie;
- dacă există un singur candidat cheie, se va selecta acesta drept atrbut de identificare a entității;
- dacă există mai mulți candidați cheie, se selectează unul, cu ajutorul unor *euristici*, precum:
  - se preferă atrbutele ale căror valori sunt mai puțin volatile;
  - se preferă atrbutele ale căror valori sunt mai scurte.

În urma identificării atrbutorilor cheie pentru domeniul comercial, diagrama entitate-asociere se prezintă ca în figura 5.24.

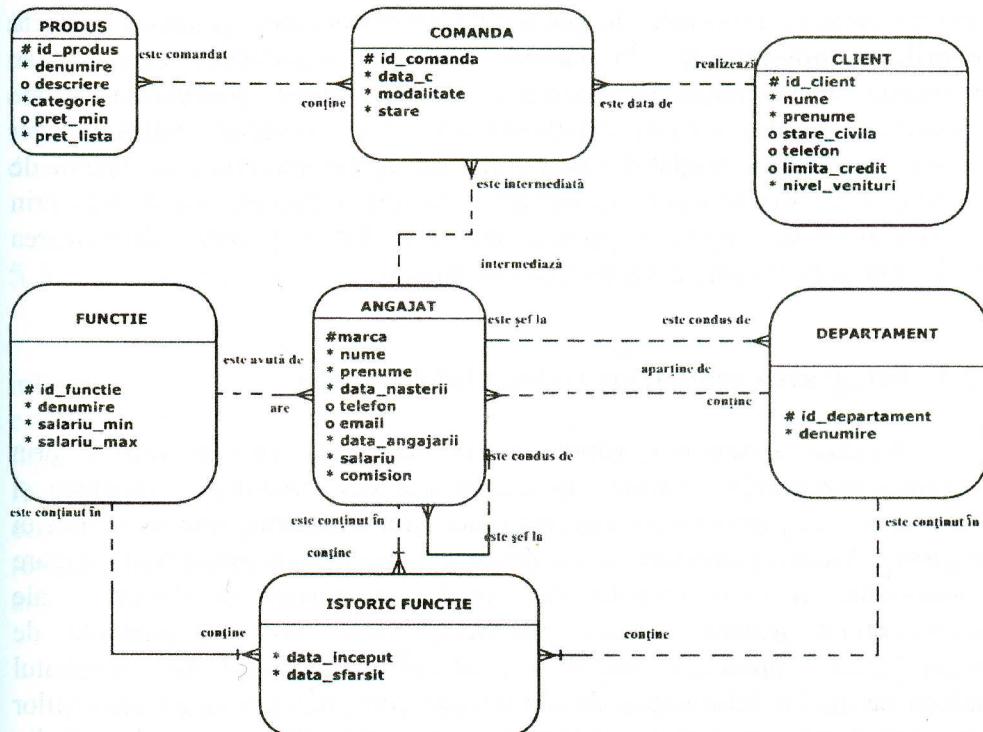


Figura 5.24 Diagrama entitate-asociere pentru domeniul economic (forma finală)

## 5.2.2 Analiza dinamică (de comportament) și funcțională

**Analiza dinamică** are drept scop explicarea comportamentului entităților din domeniul economic analizat. În urma acestei analize se obține modelul dinamic al sistemului. Construirea modelului dinamic presupune identificarea stăriilor în care se pot afla componentele sistemului, identificarea evenimentelor care determină trecerea unei componente dintr-o stare în alta, precum și stabilirea succesiunii (fluxului) de evenimente și construirea unei diagrame care să reflecte tranzițiile de stare pentru componentele sistemului (diagrama de flux a evenimentelor). În realizarea diagramei de flux a evenimentelor este necesar să se țină cont de restricțiile dinamice ale sistemului care servesc la identificarea tranzițiilor admisibile între două stări.

**Analiza funcțională** are drept scop determinarea transformărilor de date care se produc în cadrul sistemului în scopul satisfacerii cerințelor informaționale aferente acestui sistem. Transformările de date se vor prezenta sub forma unei diagrame de flux a prelucrărilor (modelul funcțional), în care

nodurile reflectă procesele de prelucrare informațională și arcele reflectă fluxurile informaționale. În cadrul analizei funcționale, accentul se deplasează de la realitatea analizată către cerințele informaționale ale utilizatorilor, cerințe a căror satisfacere constituie obiectivul realizării bazei de date. Construirea modelului funcțional presupune identificarea datelor de intrare și a datelor de ieșire din sistem, construirea diagramelor de flux prin care sunt reflectate legăturile procedurale dintre intrări și ieșiri, identificarea restricțiilor și precizarea criteriilor de optimizare.

### 5.2.3 Integrarea modelelor sistemului economic .

Analiza sistemului (domeniului) economic se finalizează prin integrarea rezultatelor obținute în cadrul analizei structurale, dinamice și funcționale, mai precis prin integrarea modelului structural, dinamic și a celui funcțional. Modelul structural și cel dinamic sunt obținute printr-o investigare a sistemului real, a proprietăților intrinseci, statice și dinamice ale componentelor acestui sistem, proprietăți care sunt independente de aplicațiile care operează asupra lor. Modelul funcțional este rezultatul analizei cerințelor informaționale ale utilizatorilor, mai precis a tranzacțiilor (aplicațiilor) prin care pot fi satisfăcute aceste cerințe. Perspectiva diferită din care este realizată analiza explică de ce rezultatele obținute pot să difere, fiind necesară o coordonare, deci o integrare a acestor rezultate.

În cadrul etapei de integrare a modelelor sistemului economic se stabilește în ce măsură modelul structural și cel dinamic satisfac necesitățile diferitelor aplicații, verificându-se completitudinea (existența elementelor informaționale solicitate) și consistența lor (în ce măsură componentele modelelor sunt necesare și suficiente în raport cu procesele de prelucrare). Se verifică dacă relațiile dintre componentele sistemului sunt stabilite în mod corespunzător, pentru a face posibilă regăsirea informațiilor din mai multe entități. Se determină, de asemenea, dacă legăturile dintre entități asigură coerentă informațiilor, posibilitatea efectuării de actualizări concomitente asupra datelor redundante. Se urmărește ca toate elementele informaționale participante la diferențele tranzacții să fie asignate, ca atrbute ale diverselor entități. Pe baza acestei analize integrate se efectuează adăugările și/sau corelările necesare între modelele sistemului. În final, se ajunge ca modelul structural și cel dinamic să nu mai fie complet independente față de aplicații, iar modelul funcțional să nu mai fie orientat exclusiv pe aplicații. Modelarea orientată exclusiv pe aplicații are dezavantajul înglobării unor cerințe eterogene, care complică artificial modelul și oferă posibilități scăzute de

adaptare a modelului la noi cerințe informaționale. Pe de altă parte, modelarea complet independentă de aplicații presupune o analiză costisitoare, complexă care solicită resurse considerabile, dezvoltarea aplicațiilor reclamând, de asemenea, un efort substanțial. Integrarea analizei structurale dinamice și a celei funcționale elimină aceste dezavantaje.

### 5.3 Proiectarea structurii bazei de date relaționale

Modelele obținute în urma analizei sistemului economic sunt modele informaționale, adică modele ale datelor despre sistem. O caracteristică esențială a acestor modele (numite și *modele conceptuale* sau *semantice*) este faptul că sunt independente de SGBDR care le face să devină operaționale. Este extrem de important ca în etapa de analiză a sistemului economic și a cerințelor informaționale asociate, activitatea de modelare a datelor să se realizeze independent de un SGBDR. Orientarea pe concepțele proprii unui anumit SGBDR prezintă numeroase dezavantaje, dintre care se pot menționa:

- schimbarea SGBDR impune reproiectarea bazei de date;
- concepțele tehnice ale SGBDR pot influența negativ activitatea de analiză (și deci de modelare), prin restricțiile impuse de acestea, care pot încuraja sau descuraja anumite reprezentări;
- fixând ca punct de plecare facilitățile unui SGBDR, utilizatorul final care nu stăpânește acest SGBDR nu își poate exprima cerințele în deplină cunoștință de cauză.

Trecerea la proiectarea structurii bazei de date impune luarea în considerare a SGBDR cu ajutorul căruia va fi implementată și exploatață baza de date.

Structura bazei de date reprezintă, de fapt un model al datelor, exprimat în concepțele specifice unui anumit SGBDR, ceea ce face ca proiectarea structurii bazei de date să reprezinte transpunerea modelelor conceptuale în termenii unui model al datelor suportat de un anumit SGBDR. Tendința care se manifestă în prezent, de nuanțare și îmbogățire a facilităților de modelare a datelor pe care le oferă SGBDR, face ca trecerea de la etapa de analiză la etapa de proiectare a structurii bazei de date să fie din ce în ce mai simplă și mai facil de realizat.

Proiectarea structurii bazei de date constă din următoarele activități:

- a) proiectarea schemei conceptuale;
- b) proiectarea schemei externe (subschemei);
- c) proiectarea schemei interne (de memorare).

*Proiectarea schemei conceptuale* presupune stabilirea colecțiilor de date și definirea detaliată a conținutului acestora; determinarea legăturilor dintre colecțiile de date și a modului de reprezentare a acestora în cadrul schemei conceptuale; testarea schemei obținute și revizuirea acesteia, dacă este cazul și, în final, descrierea schemei conceptuale în limbajul de descriere a datelor și încărcarea acestei descrieri în baza de date.

*Proiectarea schemei externe* a bazei de date relaționale este realizată, în principal, cu ajutorul tabelelor virtuale (*views*) și al mecanismelor de acordare a drepturilor de acces la BDR.

*Proiectarea schemei interne* a bazei de date relaționale presupune stabilirea modului în care este liniarizată schema conceptuală, în vederea asigurării stocării datelor pe suportul fizic.

În proiectarea bazei de date relaționale se utilizează frecvent termenul de *schemă conceptuală optimală*, prin care se înțelege acea schemă conceptuală care înălătură posibilitatea apariției de anomalii în lucrul cu baza de date și asigură, totodată, facilități și performanțe sporite la încărcarea, exploatarea și întreținerea bazei de date. Anomaliiile apar, în principal, la întreținerea bazei de date, fiind cunoscute și sub numele de anomalii de actualizare a datelor. Aceste anomalii sunt puse în legătură cu dependențele care se manifestă ca restricții de integritate ale modelului relațional. Un asemenea mod de abordare a permis, pe de o parte caracterizarea riguroasă a relațiilor după gradul de perfecțiune pe care îl prezintă și, pe de altă parte, a făcut posibilă definirea unor tehnici pentru înălăturarea anomalieiilor de actualizare, practic de obținere a relațiilor în anumite forme normale. Modelul relațional oferă o tehnică de lucru formalizată, cunoscută sub numele de *tehnica normalizării relațiilor* care asigură obținerea schemei conceptuale optimale a bazei de date, cu un risc minim de apariție a disfuncționalităților la momentul exploatarii acesteia.

### **5.3.1 Anomaliiile de actualizare și formele normale ale relațiilor**

Anomaliiile care apar în lucrul cu bazele de date relaționale se produc din cauza dependențelor nedorite care se manifestă între datele din cadrul relațiilor bazei de date. Aceste dependențe determină creșterea redundanței datelor și reducerea flexibilității structurii bazei de date, făcând extrem de dificil lucrul cu baza de date.

### a) Anomalii cauzate de existența dependențelor funcționale parțiale

Să considerăm, de exemplu relația  $R_1$  din tabelul 5.2.

**Tabelul 5.2 Relația  $R_1$**

Codfurn:D <sub>1</sub>	Codprod:D <sub>2</sub>	Cant:D <sub>3</sub>	Codloc:D <sub>4</sub>	Denloc:D <sub>5</sub>
F <sub>1</sub>	P <sub>1</sub>	5000	C <sub>1</sub>	D <sub>1</sub>
F <sub>2</sub>	P <sub>1</sub>	4000	C <sub>2</sub>	D <sub>2</sub>
F <sub>2</sub>	P <sub>3</sub>	50000	C <sub>2</sub>	D <sub>2</sub>
F <sub>3</sub>	P <sub>2</sub>	15000	C <sub>2</sub>	D <sub>2</sub>

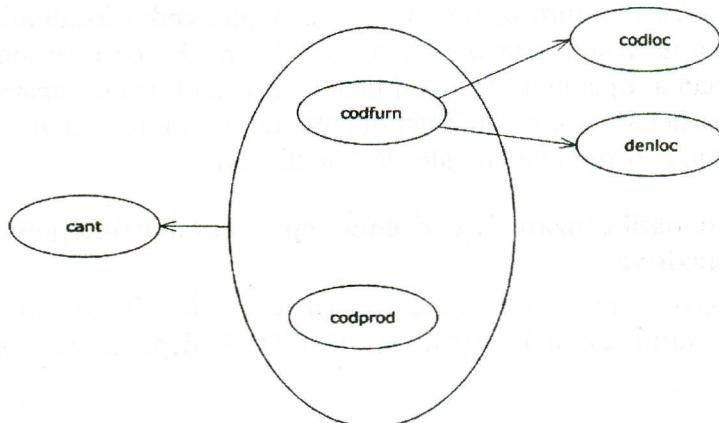
Cheia primară a relației  $R_1$  este compusă din atrbutele *Codfurn* și *Codprod*. Între datele din cadrul relației  $R_1$  se manifestă o serie de dependențe funcționale (figura 5.25) și anume:

$(Codfurn, Codprod) \rightarrow Cant$

$Codfurn \rightarrow Codloc$  – dependență funcțională parțială

$Codfurn \rightarrow Denloc$  – dependență funcțională parțială

$Codloc \rightarrow Denloc$  – dependență tranzitivă



**Figura 5.25 Dependențe funcționale în cadrul relației  $R_1$  din tabelul 5.2**

Dependențele funcționale parțiale determină, la actualizarea relației, producerea următoarelor anomalii:

- *limitarea posibilităților de adăugare a datelor.* În relația  $R_1$  nu pot fi introduse date despre un anumit furnizor (cod localitate, denumire localitate) decât atunci când se cunoaște codul unui produs pe care îl

oferă (conform restricției de integritate a entității, care impune ca într-o relație atributele cheie să nu aibă valoarea *NULL*);

- *pierderi de date la ștergere.* În mod normal, prin operația de ștergere trebuie să se eliminate din baza de date numai datele pe care dorim să le ștergem. Atunci când, concomitent cu aceste date sunt șterse și altele, care nu mai pot fi obținute (reconstituite) din BDR spunem că la operația de ștergere se produc pierderi de date. De exemplu, în situația în care furnizorul  $F_1$  nu mai livrează produsul  $P_1$ , tuplul cu valoarea cheii ( $F_1, P_1$ ) trebuie șters. Ștergerea acestui tuplu din relația  $R_1$  determină pierderea informațiilor cu privire la furnizorul  $F_1$ , întrucât acesta nu mai livrează un alt produs, deci nu mai apare descris în vreun alt tuplu în cadrul relației  $R_1$ . Pierderile de date la ștergere afectează posibilitățile de lucru, în condiții normale, cu baza de date;
- *apariția unor inconsistențe la modificarea datelor.* Dependentele funcționale parțiale semnalează o redundanță nejustificată a datelor în cadrul unei relații. În relația  $R_1$ , de exemplu descrierea furnizorului  $F_2$  se realizează de mai multe ori, și anume pentru fiecare produs pe care îl livrează. În mod normal, descrierea furnizorului trebuie făcută o singură dată, lucru posibil de realizat dacă, de exemplu, datele despre furnizori ar figura într-o relație separată. Modificarea unei caracteristici a furnizorului  $F_2$  (de exemplu, codul localității) trebuie operată în toate tuplurile în care este descris  $F_2$ , ceea ce îngreunează efectuarea operației de modificare, existând riscul apariției unor inconsistențe a datelor (menținerea unor valori neactualizate ale codului, concomitent cu valorile actualizate).

### b) Anomalii cauzate de existența dependențelor funcționale tranzitive

Să considerăm, de exemplu, aceeași relație  $R_1$  din tabelul 5.2. Între datele din cadrul acestei relații se manifestă dependența funcțională tranzitivă:

$$\text{Codloc} \rightarrow \text{Denloc}.$$

Dependențele funcționale tranzitive determină apariția următoarelor anomalii la actualizarea datelor:

- *anomalii la adăugarea datelor.* În relația  $R_1$ , de exemplu, nu se poate adăuga denumirea unei localități cu un anumit cod, decât atunci când în această localitate există un furnizor care livrează un anumit produs;

- *anomalii la ștergerea datelor.* Ca și în cazul dependențelor funcționale parțiale, din cauza dependențelor funcționale tranzitive se pot produce pierderi de date la ștergere;
- *anomalii la modificarea datelor.* Redundanța semnalată de dependențele funcționale tranzitive face ca la modificarea unei caracteristici, să apară inconsistențe dacă modificarea nu este operată peste tot unde apare respectiva caracteristică.

### c) Anomalii cauzate de existența dependențelor multivaloare

Să considerăm, ca exemplu, relația  $R_2$ , din tabelul 5.3.

**Tabelul 5.3 Relația  $R_2$**

Curs:D <sub>1</sub>	Cap:D <sub>2</sub>	CD: D <sub>3</sub>
cs <sub>1</sub>	cp <sub>1</sub>	cd <sub>1</sub>
cs <sub>1</sub>	cp <sub>1</sub>	cd <sub>2</sub>
cs <sub>1</sub>	cp <sub>2</sub>	cd <sub>1</sub>
cs <sub>1</sub>	cp <sub>2</sub>	cd <sub>2</sub>
cs <sub>1</sub>	cp <sub>3</sub>	cd <sub>1</sub>
cs <sub>2</sub>	cp <sub>3</sub>	cd <sub>2</sub>
cs <sub>2</sub>	cp <sub>4</sub>	cd <sub>3</sub>

În cadrul relației  $R_2$  se manifestă următoarele dependențe multivaloare:

*Curs →→ Capitol*

*Curs →→ Cadrudid*

Dependențele multivaloare determină apariția unei mari redundanțe a datelor. Această redundanță creează dificultăți la:

- *modificarea datelor*, putând duce la apariția unor inconsistențe în baza de date;
- *adăugarea datelor*. În relația  $R_2$ , de exemplu dacă se dorește includerea unui nou cadru didactic la cursul  $cs_1$  este necesar ca în relația  $R_2$  să fie adăugate trei tupluri, întrucât cursul  $cs_1$  figurează în relație cu trei capitole;
- *ștergerea datelor*. Pentru ștergerea capitolului  $cp_1$  din cadrul cursului  $cs_1$ , de exemplu, este necesar să fie șterse două tupluri din relația  $R_2$ , întrucât la cursul  $cs_1$  desfășoară activitate didactică doi profesori:  $cd_1$  și  $cd_2$ .

### d) Anomalii cauzate de existența dependențelor jonctiune

Dependența jonctiune determină, la rândul său o serie de neajunsuri la actualizarea datelor din baza de date. Să considerăm, de exemplu, relația  $R_3$ , din tabelul 5.4.

**Tabelul 5.4 Relația  $R_3$**

$X: D_x$	$Y: D_y$	$Z: D_z$
$x_1$	$y_1$	$z_2$
$x_2$	$y_1$	$z_1$
$x_1$	$y_2$	$z_1$
$x_1$	$y_1$	$z_1$

Dacă se dorește ștergerea tuplului  $\langle x_1, y_1, z_1 \rangle$  din cadrul relației  $R_3$  este necesar ca împreună cu acesta să se mai șteargă și un alt tuplu (oricare din cele rămase în relație), întrucât altfel, dependența jonctiune interzice ștergerea tuplului  $\langle x_1, y_1, z_1 \rangle$ .

**Formele normale ale relațiilor** dintr-o bază de date relațională sunt definite în raport de anomalii care pot apărea în lucrul cu aceste relații, deci în funcție de dependențele nedorite care se manifestă între datele din cadrul acestora.

**Forma normală unu (FN1).** O relație  $R$  este în FN1 dacă domeniile pe care sunt definite atributele relației sunt constituite numai din valori atomice (elementare). În plus, un tuplu nu trebuie să conțină atrbute sau grupuri de atrbute repetitive. Este forma de bază a relațiilor, care figurează ca cerință minimală în cazul majorității SGBDR. Relațiile  $R_1$ ,  $R_2$  și  $R_3$  din exemplele anterioare sunt în FN1.

**Forma normală doi (FN2).** O relație  $R$  este în FN2 dacă este în FN1 și oricare dintre atrbutele non-cheie este dependent funcțional complet de cheia primară a relației. FN2 interzice manifestarea unor dependențe funcționale parțiale în cadrul relației. Figura 5.26 prezintă exemple de relații în FN2.

Codfurn:D <sub>1</sub>	Codprod:D <sub>2</sub>	Cant:D <sub>3</sub>
F <sub>1</sub>	P <sub>1</sub>	5000
F <sub>2</sub>	P <sub>1</sub>	4000
F <sub>2</sub>	P <sub>3</sub>	50000
F <sub>3</sub>	P <sub>2</sub>	15000

Codfurn:D <sub>1</sub>	Codloc:D <sub>4</sub>	Denloc:D <sub>5</sub>
F <sub>1</sub>	C <sub>1</sub>	D <sub>1</sub>
F <sub>2</sub>	C <sub>2</sub>	D <sub>2</sub>
F <sub>3</sub>	C <sub>2</sub>	D <sub>2</sub>

**Figura 5.26 Relațiile  $R_4$  și  $R_5$**

**Forma normală trei (FN3).** O relație  $R$  este în FN3 dacă este în FN2 și atrbutele non-cheie nu sunt dependente tranzitiv de cheia primară a

relației. FN3 interzice manifestarea dependențelor funcționale tranzitive în cadrul relației.

 Relația  $R_4$  din figura 5.26 este în FN3, în timp ce relația  $R_5$ , din aceeași figură nu se află în FN3, din cauza dependenței funcționale tranzitive:  $Codloc \rightarrow Denloc$ .

**Forma normală Boyce-Codd (BCNF).** O relație este în BCNF dacă dependențele funcționale netriviale care se manifestă în cadrul relației conțin în partea stângă (ca determinant) o cheie candidată.

Să considerăm, de exemplu relația  $R_6$  cu schema:

$$R_6 (K_1:D_{k1}, K_2:D_{k2}, A:D_A, B:D_B)$$

 Să presupunem că  $R_6$  are cheia  $(K_1, K_2)$  și că în cadrul relației se manifestă, alături de alte dependențe funcționale și dependența  $B \rightarrow K_1$ . Relația  $R_6$  nu se află în BCNF, întrucât dependența menționată încalcă cerințele formei normale Boyce-Codd.

 **Forma normală patru (FN4).** O relație  $R$  este în FN4 dacă în această relație nu se manifestă mai mult de o dependență multivaloare. Relația  $R_2$  din tabelul 5.3. nu este în FN4, întrucât în cadrul acestei relații se manifestă două dependențe multivaloare. Relația  $R_4$  din figura 5.26 este în FN4, deoarece conține o singură dependență multivaloare:  $Codfurn \rightarrow Codprod$ .

**Forma normală cinci (FN5).** O relație  $R$  este în FN5 dacă fiecare dependență joncțiune este implicată printr-un candidat cheie a lui  $R$ .

### 5.3.2 Tehnica normalizării relațiilor

Normalizarea relațiilor presupune stabilirea schemei conceptuale inițiale a bazei de date relaționale și rafinarea progresivă a acesteia.

#### a) Stabilirea schemei conceptuale inițiale a BDR

Se pleacă de la modelele conceptuale ale datelor, mai precis de la modelul entitate-asociere. Schema conceptuală inițială a BDR se poate obține în două moduri, și anume:

- se realizează maparea entităților și a asocierilor în relații. Este o variantă *top-down* de utilizare a tehnicii de normalizare a relațiilor;
- se constituie o relație unică, aşa numita *relație universală*, din atributele tuturor entităților și asocierilor. Este varianta *bottom-up*, care deși are avantajul obținerii rapide a schemei conceptuale inițiale,

este mai puțin utilizată în practică încărcăt dificultățile întâmpinate în optimizarea acestei scheme conceptuale sunt cu mult mai mari decât cele de optimizare a schemei conceptuale inițiale obținută în varianta *top-down*.

Regulile de mapare sunt următoarele:

- o entitate devine o tabelă, numele tabelei fiind pluralul numelui entității;
- o instanță devine un tuplu al relației;
- un atribut devine o coloană. Pentru atributelor obligatorii se vor declara restricții de tip NOT NULL;
- un identificator primar devine o cheie primară;
- un identificator secundar devine cheie unică;
- o legătură se transformă într-o coloană pe care se declară o cheie externă. Dacă participarea entității referite este obligatorie, pe lângă cheie externă se declară și o restricție de tip NOT NULL.

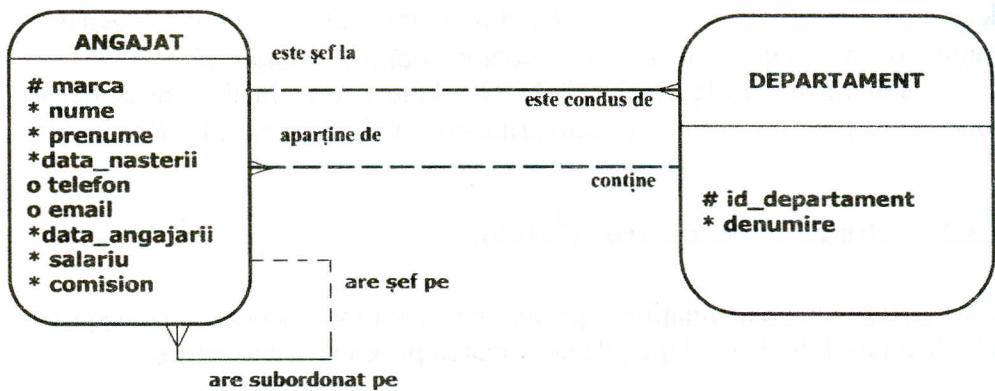


Figura 5.27 Entitățile ANGAJAT și DEPARTAMENT

În tabelul 5.5 sunt prezentate tabelele ANGAJATI și DEPARTAMENTE, rezultate din maparea entităților ANGAJAT și DEPARTAMENT din figura 5.27.

**Tabelul 5.5 Tabelele rezultate din mapare**

<b>ANGAJATI</b>	
Marca	Chei primară
Nume	Not Null
Prenume	Not Null
Data_nasterii	Not Null
Telefon	
Email	
Data_angajarii	
Salariu	Not Null
Comision	Not Null
Id_departament	Chei externă, face legătura cu tabela DEPARTAMENTE
Id_manager	Chei externă, face legătura cu coloana marca a tableei, arată care este șeful fiecărui angajat.
<b>DEPARTAMENT</b>	
Id_departament	Chei primară
Denumire	Not null
Id_manager	Chei externă, face legătura cu coloana marcă a tablei ANGAJATI, se arată care este managerul fiecărui departament. Restricție NOT NULL.

### b) Rafinarea progresivă a schemei conceptuale a BDR

Modul de obținere a schemei conceptuale inițiale (varianta *top-down* sau *bottom-up*) nu afectează procesul de rafinare a acesteia, decât sub aspectul volumului de muncă solicitat. Aceasta este mai mare atunci când se recurge la aşa numita *relație universală*, ca schema conceptuală inițială a BDR (varianta *bottom-up*) și este mai mic atunci când, la obținerea schemei inițiale se utilizează regulile de mapare prezentate anterior (varianta *top-down*). În acest din urmă caz, schema conceptuală inițială prezintă un grad de „perfecțiune” suficient de ridicat, pentru ca uneori nici să nu mai fie necesară optimizarea sa. Așa se explică faptul că uneori, regulile de mapare menționate sunt considerate suficiente pentru obținerea unei scheme conceptuale acceptabile, care nu mai trebuie ameliorată. Cum însă cerințele de performanță și flexibilitate formulate de utilizator și administratorul bazei de date pot depăși facilitățile oferite de schema conceptuală inițială, optimizarea acesteia reprezintă o etapă care trebuie avută în vedere în mod obligatoriu la proiectarea BDR.

Rafinarea schemei conceptuale a unei BDR înseamnă trecerea succesivă a relațiilor componente prin formele normale cunoscute, până la aducerea lor în forma normală stabilită ca fiind optimă pentru baza de date relațională. Trecerea unei relații dintr-o formă normală în alta presupune eliminarea unui anumit tip de dependențe (dependențe funcționale parțiale, tranzitive, dependențe multivaloare sau joncțiune), dependențe care determină apariția anomaliei de actualizare. Aceste dependențe nedorite sunt, de fapt, transformate în dependențe admisibile, care nu provoacă anomalii, adică în dependențe funcționale complete. Transformarea dependențelor se realizează, de regulă, prin descompunerea relației în două sau mai multe relații.

Procesul de optimizare a schemei conceptuale a BDR trebuie să satisfacă o serie de cerințe și anume:

- să garanteze conservarea datelor. Acest lucru înseamnă că în schema conceptuală finală trebuie să figureze toate datele din cadrul schemei inițiale;
- să garanteze conservarea dependențelor dintre date. În cadrul schemei conceptuale finale, fiecare dependență trebuie să aibă atât determinantul, cât și determinatul în schema aceleiași relații;
- să reprezinte o descompunere minimală a relațiilor inițiale. Nici una dintre relațiile din schema conceptuală ameliorată finală nu trebuie să fie conținută într-o altă relație din această schemă.

Procesul de rafinare a schemei conceptuale a BDR este prezentat în figura 5.28. Pentru exemplificarea acestor etape s-a recurs la o relație  $R$ , a cărei schemă este prezentată în tabelul 5.6. Relația  $R$  are două chei candidate și anume:

(*Codreper, Codmasina, Codoper*)  
(*Codreper, Codmasina, Nroper*)

Dintre acestea, prima se alege drept cheie primară a relației  $R$ .

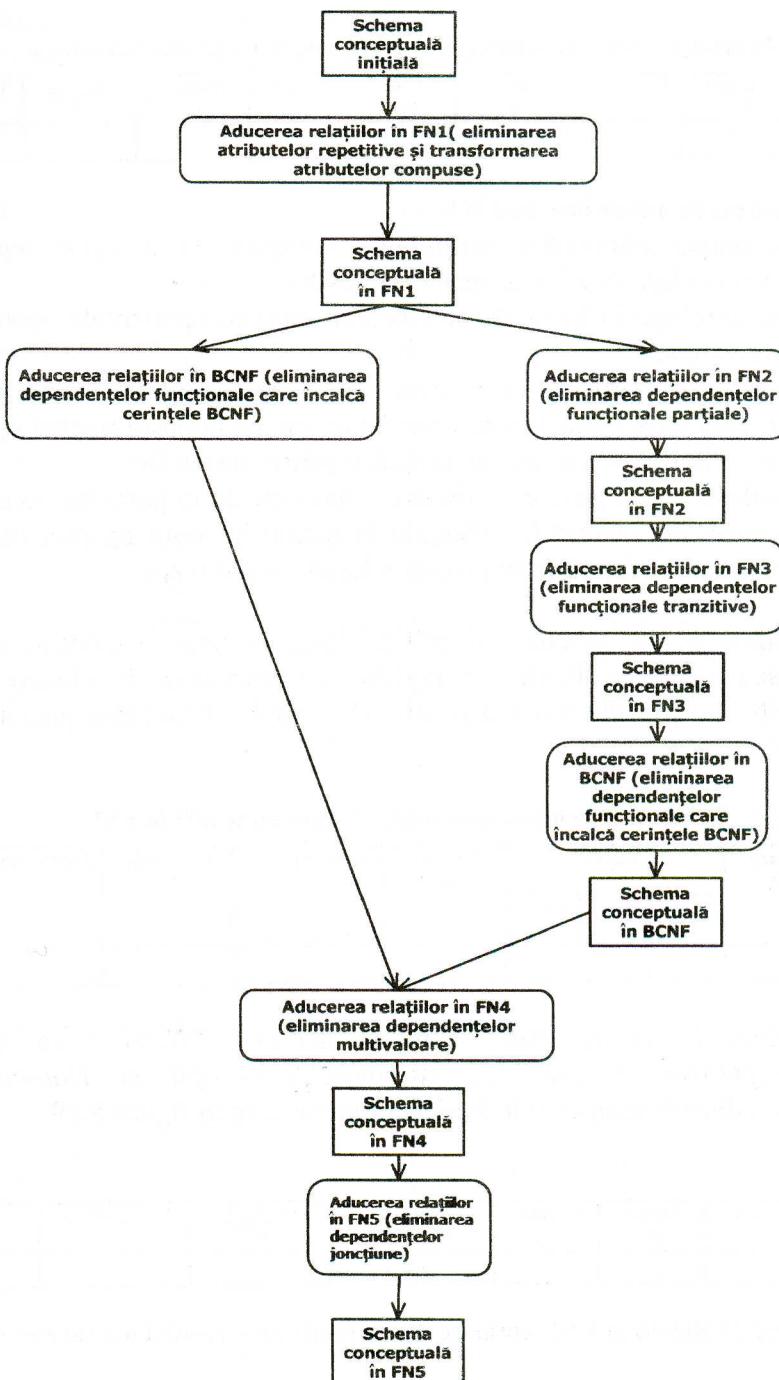


Figura 5.28 Etapele procesului de optimizare a schemei conceptuale a BDR

**Tabelul 5.6 Schema relației R, cu date despre un proces tehnologic**

Codprod: D <sub>1</sub>	Codreper: D <sub>2</sub>	Codsectie: D <sub>3</sub>	Codmasină: D <sub>4</sub>	Nroper: D <sub>5</sub>	Codoper: D <sub>6</sub>	Categoper: D <sub>7</sub>	Timppreg: D <sub>8</sub>	Timpexec: D <sub>8</sub>

**Aducerea relațiilor în FN1**

Presupune eliminarea atributelor compuse și a celor repetitive. Aducerea unei relații în FN1 se realizează astfel:

- se trec în relație în locul atributelor compuse componentele acestora, ca attribute simple;
- se plasează grupurile de attribute repetitive, fiecare în câte o nouă relație;
- se introduce în schema fiecărei noi relații create la pasul b) cheia primară a relației din care a fost extras grupul repetitive respectiv;
- se stabilește cheia primară a fiecărei relații create în pasul b). Aceasta va fi compusă din attributele adăugate la relație în pasul c), precum și din unul sau mai multe attribute proprii relației respective.

Relația *R*, din tabelul 5.6 este în FN1, încât nu conține attribute compuse sau repetitive. Pentru exemplificarea procesului de aducere a unei relații în FN1 se consideră o altă relație, *P*, a cărei schemă este prezentată în tabelul 5.7.

**Tabelul 5.7 Schema unei relații P, care nu se află în FN1**

Marca: D <sub>1</sub>	Numep: D <sub>2</sub>	Adresa:D <sub>3</sub>			Prencopil <sub>1</sub> : D <sub>2</sub>	Datanașterec <sub>1</sub> : D <sub>6</sub>	Prencopil <sub>2</sub> : D <sub>2</sub>	Datanașterec <sub>2</sub> : D <sub>6</sub>	... ...
		Str: D <sub>4</sub>	Loc: D <sub>4</sub>	Cod: D <sub>5</sub>					

Relația *P* are un atribut compus, denumit *Adresa* și un grup de attribute repetitive, format din attributele *Prencopil* și *Datanașterec*. Rezultatele aducerii relației *P* în FN1 sunt prezentate în figura 5.29.

P	Marca: D <sub>1</sub>	Numep: D <sub>2</sub>	Stradr: D <sub>4</sub>	Locadr: D <sub>4</sub>	Codadr: D <sub>5</sub>

C	Prencopil: D <sub>2</sub>	Datanasterec: D <sub>6</sub>	Marca: D <sub>1</sub>

**Figura 5.29 Relații în FN1, obținute din normalizarea relației din tabelul 5.7**

### Aducerea relațiilor în FN2

Presupune eliminarea dependențelor funcționale parțiale, din cadrul relațiilor aflate în FN1.

*Procesul de aducere a unei relații din FN1 în FN2 se desfășoară astfel:*

- pentru fiecare dependență funcțională parțială din cadrul relației se creează o nouă relație, cu schema constituită din determinantul și determinantul acestei dependențe, eliminându-se totodată, din cadrul relației inițiale atributul/atributele care formează determinantul dependenței. Dacă în relația inițială există mai multe dependențe funcționale parțiale cu același determinant, pentru toate acestea se creează o singură relație cu schema constituită din determinantul luat o singură dată și din determinații dependențelor considerate;
- se determină cheia primară a fiecărei noi relații creată în pasul a), ca fiind formată din atributul/atributele care reprezintă determinantul dependenței funcționale parțiale, care a stat la baza constituirii relației;
- se analizează relațiile rezultate la pasul a). Dacă aceste relații conțin dependențe funcționale parțiale se reia procesul de aducere în FN2. Dacă relațiile nu conțin dependențe funcționale parțiale, procesul de aducere în FN2 a luat sfârșit.

În relația din tabelul 5.7 se manifestă următoarele dependențe funcționale parțiale:

$$\text{Codoper} \rightarrow \text{Categoper}$$

$$\text{Nroper} \rightarrow \text{Categoper}$$

Prin aplicarea operațiilor de aducere a acestei relații în FN2 se obțin relațiile din figura 5.30.

$R_1$	Codprod: D <sub>1</sub>	Codreper: D <sub>2</sub>	Codsectie: D <sub>3</sub>	Codmasina: D <sub>4</sub>	Nroper: D <sub>5</sub>	Codoper: D <sub>6</sub>	Timpreg: D <sub>8</sub>	Timpexec: D <sub>8</sub>

$R_2$	
Codoper:D <sub>6</sub>	Categoper:D <sub>7</sub>

$R_3$	
Nroper:D <sub>5</sub>	Categoper:D <sub>7</sub>

Figura 5.30 Relațiile obținute prin aducerea în FN2 a relației  $R$  din tabelul 5.6

Dependențele funcționale parțiale din relația din tabelul 5.7 au fost transformate în următoarele dependențe funcționale complete:

$Codoper \rightarrow Categoper$ , în cadrul relației  $R_1$

$Nroper \rightarrow Categoper$ , în relația  $R_2$

Dacă se notează cu  $R$  o relație în FN1 care trebuie adusă în FN2 și cu  $(A, B)$  cheia relației  $R(\underline{A}:D_A, \underline{B}:D_B, \underline{C}:D_C, \underline{D}:D_D, \dots)$  și dacă se consideră dependența  $B \rightarrow C$  singura dependență funcțională parțială care se manifestă în  $R$ , atunci procesul de aducere a lui  $R$  în FN2 determină descompunerea lui  $R$  în două relații  $R_1$  și  $R_2$ , cu schemele:

$$R_1(\underline{A}:D_A, \underline{B}:D_B, D:D_D, \dots)$$

$$R_2(B:D_B, C:D_C)$$

### Aducerea relațiilor în FN3

Se realizează prin eliminarea dependențelor funcționale tranzitive care se manifestă în cadrul relațiilor aflate în FN2.

*Procesul de aducere a unei relații din FN2 în FN3* conține următorii pași:

- pentru fiecare dependență funcțională tranzitivă din cadrul relației considerate se transferă atributele implicate în dependență tranzitivă într-o nouă relație;
- se determină cheia primară a fiecărei noi relații creată în pasul a);
- se introduc în relația inițială cheile primare determinate în pasul b), în locul atributelor transferate;
- se analizează relația inițială. Dacă în cadrul ei se manifestă noi dependențe funcționale tranzitive se reia procesul de aducere în FN3. Dacă nu, procesul a luat sfârșit.

Pentru exemplificare, se consideră relația  $T$ , cu schema prezentată în tabelul 5.8.

**Tabelul 5.8 Schema relației T**

$K_I:D_{K_I}$	$A_I:D_{A_I}$	$A_2:D_{A_2}$

În cadrul relației  $T$  se manifestă dependență funcțională tranzitivă  $A_I \rightarrow A_2$ . Figura 5.31 prezintă rezultatele aducerii relației  $T$  în FN3.

$T$
$K_I:D_{K_I}$
$A_I:D_{A_I}$

$T_1$
$A_I:D_{A_I}$
$A_2:D_{A_2}$

**Figura 5.31 Relațiile obținute prin aducerea la FN3 a relației  $T$  din tabelul 5.8**

Relațiile din figura 5.31 nu conțin dependențe funcționale tranzitive, deci sunt în FN3.

Dacă se notează cu  $R$  o relație în FN2 care trebuie adusă în FN3 și cu  $X$  cheia acestei relații  $R(\underline{X}:D_X, A:D_A, B:D_B, \dots)$  și dacă se consideră dependența  $A \rightarrow B$  singura dependență funcțională tranzitivă care se manifestă în  $R$ , atunci procesul de aducere a lui  $R$  în FN3 determină descompunerea lui  $R$  în două relații  $R_1$  și  $R_2$ , cu schemele:

$$R_1(\underline{X}:D_X, A:D_A, \dots)$$

$$R_2(A:D_A, \underline{B}:D_B)$$

### **Aducerea relațiilor în BCNF**

Presupune eliminarea dependențelor funcționale care încalcă cerințele formei normale Boyce-Codd, și anume a dependențelor ale căror determinanți nu sunt candidați cheie. Aceste dependențe funcționale mai sunt cunoscute și sub numele de dependențe non-cheie. Pentru ca o relație să fie adusă în BCNF nu trebuie în mod obligatoriu să fie în FN3. Se pot aduce în BCNF și relații aflate în FN1 sau FN2. Acest lucru este posibil întrucât dependențele funcționale parțiale și cele tranzitive sunt, de fapt, tot dependențe non-cheie.

Există trei categorii de dependențe non-cheie și anume:

1. dependențe funcționale parțiale;
2. dependențe funcționale tranzitive;
3. dependențe non-cheie, altele decât cele din categoriile 1 și 2.

Într-o relație aflată în FN3 se manifestă numai dependențele non-cheie din categoria 3 (cele din categoriile 1 și 2 au fost eliminate în procesul aducerii relației în FN3). Într-o relație aflată în FN2 se pot manifesta dependențe non-cheie din categoriile 2 și 3, iar într-o relație în FN1 pot exista dependențe non-cheie din toate cele 3 categorii. A aduce o relație în BCNF înseamnă a elimina toate tipurile de dependențe non-cheie care se manifestă în cadrul ei.

În general, se consideră, ca punct de plecare în procesul de aducere la BCNF următoarele tipuri de relații:

- relațiile în FN1, caz în care procedura de aducere în BCNF recurge la un procedeu unitar pentru eliminarea tuturor categoriilor de dependențe non-cheie;
- relațiile în FN3, caz în care procedura de aducere în BCNF utilizează un proces specific de eliminare a dependențelor non-cheie din categoria 3. În acest caz, dependențele non-cheie din cadrul unei

relații se elimină treptat și anume: prin procedura de aducere a relației în FN2, cea de aducere în FN3 și respectiv prin procedura de aducere din FN3 în BCNF.

*Procesul de aducere a unei relații din FN1 în BCNF* este următorul:

- se analizează relația, pentru a se identifica dependențele non-cheie. Astfel, dacă relația conține numai unul sau două atrbute nu pot exista dependențe non-cheie, deci relația se află în BCNF. Dacă relația conține mai mult de două atrbute, se identifică eventualele dependențe non-cheie. Dacă există astfel de dependențe se trece la pasul b). Dacă nu, relația este în BCNF și procesul a luat sfârșit;
- se reduce progresiv schema relației inițiale și se aplică operațiile de identificare a dependențelor non-cheie de la pasul a). Ori de câte ori prin reducerea schemei relației inițiale se obține o relație în BCNF se consideră că aceasta face parte din descompunerea relației inițiale, în procesul aducerii ei la BCNF.

*Procesul de aducere a unei relații din FN3 în BCNF* se desfășoară astfel:

- se analizează relația, pentru a se identifica dependențele non-cheie. Astfel, dacă relația conține unul sau cel mult două atrbute nu pot exista dependențe non-cheie, deci relația este în BCNF și procesul a luat sfârșit. Dacă relația conține mai mult de două atrbute în cadrul ei pot exista dependențe non-cheie și se trece la identificarea lor. Dacă nu există astfel de dependențe, relația este în BCNF și procesul a luat sfârșit. Dacă există dependențe non-cheie se trece la pasul b);
- pentru fiecare dependență non-cheie  $X \rightarrow Y$  se creează două relații, una cu schema formată din atrbutele reprezentate prin  $X$  și  $Y$  și cealaltă cu schema constituită din toate atrbutele relației inițiale, mai puțin atrbutele reprezentate prin  $Y$ . Aceste două relații reprezintă descompunerea relației inițiale în procesul aducerii ei în BCNF;
- se reia procesul de aducere în BCNF, pe cele două relații obținute în pasul b).

Relațiiile din figura 5.30 nu prezintă dependențe non-cheie, deci sunt în BCNF. Pentru exemplificarea procedurilor de aducere a unei relații în BCNF se consideră relația  $V$ , cu schema redată în tabelul 5.9.

**Tabelul 5.9 Schema relației  $V$ , aflată în FN3**

$K_1:D_{K1}$	$K_2:D_{K2}$	$B:D_B$	$D:D_D$

Presupunem că în cadrul relației  $V$  se manifestă următoarele dependențe:

$(K_1, K_2) \rightarrow B$ , dependență funcțională completă

$(K_1, K_2) \rightarrow D$ , dependență funcțională completă

$D \rightarrow K_1$ , dependență non-cheie (categoria 3)

Relația  $V$  se află în FN3 și prin aplicarea procedurii de aducere în BCNF se obțin rezultatele din figura 5.32. Se observă că descompunerea relației  $V$  în relațiile  $V_1$  și  $V_2$  conservă datele și este minimală, dar nu conservă dependențele între date. De exemplu, nicio relație nu înglobează dependența funcțională  $(K_1, K_2) \rightarrow B$ .

<u><math>D:D_D</math></u>	$K_1:D_{K_1}$

$K_2:D_{K_2}$	$B:D_D$	$D:D_D$

Figura 5.32 Relațiile  $V_1$  și  $V_2$  obținute prin aducerea la BCNF a relației  $V$  din tabelul 5.9

#### Aducerea relațiilor în FN4

Presupune eliminarea dependențelor multivaloare, atunci când sunt mai mult de una în cadrul unei relații.

Procesul de aducere a unei relații din BCNF în FN4 cuprinde următorii pași:

- se identifică dependențele multivaloare  $X \rightarrow\rightarrow Y$  din cadrul relației considerate;
- se izolează fiecare atribut multivaloare  $Y$ , împreună cu atributurile care depind funcțional de acesta într-o relație separată.

În cadrul relației  $R_1$  din figura 5.30 se manifestă următoarele dependențe multivaloare:

*Codreper*  $\rightarrow\rightarrow$  *Codprodus*

*Codmașină*  $\rightarrow\rightarrow$  *Codsecție*

Rezultatele aducerii relațiilor din figura 5.30 în FN4 sunt prezentate în figura 5.33.

<i>R</i> <sub>1</sub>	Codreper:D <sub>2</sub>	Codmasina:D <sub>4</sub>	Nroper: <sub>5</sub>	Codoper:D <sub>6</sub>	Timpreg:D <sub>8</sub>	Timpexec:D <sub>8</sub>
<i>R</i> <sub>2</sub>	Codoper:D <sub>6</sub>	Categoper:D <sub>7</sub>				
<i>R</i> <sub>4</sub>	Codreper:D <sub>2</sub>	Codprod:D <sub>1</sub>				
<i>R</i> <sub>5</sub>	Codmasina:D <sub>4</sub>	Codsectie:D <sub>3</sub>				

**Figura 5.33 Relațiile obținute prin aducerea în FN4 a relațiilor din figura 5.30**

### **Aducerea relațiilor în FN5**

Presupune eliminarea dependențelor jonctiune care se manifestă în cadrul relațiilor aflate în FN4.

*Procesul de aducere a unei relații din FN4 în FN5* se desfășoară astfel:

- se identifică dependențele jonctiune. Între mulțimile de atrbute *A*, *B* și *C* din cadrul relației considerate există o dependență jonctiune atunci când există dependențe multivaloare între fiecare dintre perechile de mulțimi: (*A,B*), (*B,C*) și (*A,C*). Prin urmare, o dependență jonctiune poate exista numai în cadrul acelor relații în FN4 care prezintă chei compuse și atrbute comune în chei. Dacă există dependențe jonctiune în cadrul relației considerate se trece la pasul b). Dacă nu, procesul de aducere a relației în FN5 a luat sfârșit;
- se descompune relația inițială, în scopul obținerii FN5. Considerând că schema relației conține mulțimile de atrbute *A*, *B*, și *C* și că între fiecare pereche (*A,B*), (*B,C*), (*A,C*) există dependențe multivaloare, relația trebuie descompusă în trei relații, cu schemele: *R*<sub>1</sub>(*A:D<sub>A</sub>, B:D<sub>B</sub>*), *R*<sub>2</sub>(*B:D<sub>B</sub>, C:D<sub>C</sub>*) și *R*<sub>3</sub>(*A:D<sub>A</sub>, C:D<sub>C</sub>*).

### **Formalizarea procesului de optimizare a schemei conceptuale a bazei de date relaționale**

Procedurile de trecere a unei relații dintr-o formă normală în alta se pot formaliza în scopul creșterii gradului de rigoare a prelucrărilor precum și în scopul automatizării acestor prelucrări. Procesul de formalizare impune introducerea unor noțiuni, precum: închiderea unui grup de atrbute în raport cu un ansamblu de dependențe funcționale, acoperirea minimală a unui ansamblu de dependențe funcționale, proiecția unui ansamblu de dependențe pe un grup de atrbute. Cu ajutorul acestor noțiuni se pot formaliza atât

### 5.3.3 Proiectarea schemei conceptuale

Proiectarea schemei conceptuale a unei BDR presupune determinarea formei normale la care trebuie aduse relațiile din baza de date (a nivelului de perfecțiune impus schemei conceptuale) și stabilirea relațiilor care să facă parte din baza de date, în forma normală stabilită. Proiectarea schemei conceptuale presupune definirea schemei relațiilor și a restricțiilor de integritate asociate.

#### Determinarea formei normale la care trebuie aduse relațiile din baza de date relațională

Relațiile aflate în forme normale superioare determină apariția unui număr redus de anomalii în lucrul cu BDR, comparativ cu relațiile aflate în primele forme normale. Acest lucru ar putea sugera faptul că totdeauna este preferabil să se lucreze cu relații în FN3 și următoarele și nu cu relații în FN1 sau FN2, ceea ce în realitate, nu se confirmă. În majoritatea cazurilor, bazele de date operaționale sunt constituite din relații în FN1 și FN2. Acest lucru se explică prin faptul că formele normale superioare, deși reduc dificultatea de realizare a operațiilor de actualizare reduc, în același timp și performanțele operațiilor de regăsire a datelor. Relațiile aflate în forme normale superioare conțin, de regulă un număr mic de atribute, deoarece cu cât numărul atributelor dintr-o relație este mai mare, cu atât este mai mare și probabilitatea ca la aceste relații să se manifeste dependențe nedorite, deci șansa lor de a fi într-o formă normală superioară este mai mică. Pentru memorarea unui ansamblu dat de atribute în baza de date (a unui anumit dicționar de informații) cu cât numărul de atribute din relații este mai mic, cu atât numărul de relații din BDR este mai mare. Dispersarea atributelor într-un număr mare de relații favorizează operațiile de actualizare a datelor, dar îngreunează procesul de regăsire a datelor, întrucât satisfacerea cererilor de date impune interogarea simultană a mai multor relații, realizabilă prin efectuarea de joncțiuni, care sunt operații costisitoare în termenii resurselor de calcul reclamate.

În determinarea formei normale la care trebuie aduse relațiile din BDR se vor avea în vedere:

- ponderea operațiilor de interogare și a celor de actualizare în lucrul cu baze de date relaționale;
- exigențele de performanță și flexibilitatea impuse de utilizatorii finali la interogarea, respectiv la actualizarea bazei de date relaționale.

## Stabilirea relațiilor din baza de date relațională

În stabilirea relațiilor din BDR se pleacă de la modelele conceptuale ale sistemului pentru care se construiește baza de date, modele obținute în faza de analiză a sistemului și a cerințelor informaționale asociate. Nu există o corespondență strictă între entitățile din modelele semantice și relațiile din baza de date. Din considerente de optimizare a lucrului pe colecțiile de date, se poate decide spargerea unei entități în două sau mai multe relații, ceea ce duce la creșterea flexibilității în operarea colecțiilor respective. Îmbunătățirea performanțelor în manipularea entităților nu presupune, în mod obligatoriu mărirea numărului de relații în cadrul schemei conceptuale. Nu se poate admite o creștere nelimitată a numărului de colecții de date, încrucișând un număr mare de colecții determină creșterea dificultăților în satisfacerea cerințelor informaționale (căi de acces la date mai lungi și, prin urmare, un timp de răspuns ridicat). Legăturile între un număr mare de colecții determină creșterea redundanței datelor în cadrul bazei de date și, în consecință, o utilizare ineficientă a suportului de stocare. De aceea, obținerea unei scheme conceptuale eficiente poate impune reducerea numărului de colecții de date.

Pentru creșterea performanțelor în lucrul cu baza de date se pot introduce relații pentru memorarea unor rezultate obținute prin prelucrarea datele aflate în bază, atunci când aceste rezultate provin din calcule costisitoare și sunt solicitate în mod frecvent de către utilizatori. Prin memorarea acestor rezultate crește redundanța datelor, dar se evită repetarea unor prelucrări.

Un alt tip de redundanță admisă în baza de date îl constituie acela care apare din cauza includerii unor atrbute, concomitent în mai multe colecții ale bazei de date, în scopul scurtării căilor de acces la date. Să considerăm, de exemplu două colecții de date: una privind personalul dintr-un institut de cercetări și una referitoare la proiectele de cercetare aflate în derulare. Fiecare colecție conține date specifice entităților vizate. Astfel, colecția vizând cercetătorii științifici conține datele personale, precum și datele referitoare la activitatea științifică a cercetătorilor, respectiv identificatorii proiectelor la care lucrează aceștia. Colecția privind proiectele de cercetare conține, pentru fiecare proiect în parte date referitoare la tema și natura proiectului, sursă de finanțare etc., precum și informații de legătură cu personalul de cercetare care activează în cadrul proiectului. Dacă se dorește obținerea unor informații complete despre un anumit proiect de cercetare, inclusiv datele personale ale cercetătorilor implicați în proiectul respectiv trebuie să se realizeze accesul la ambele colecții. Presupunem că pentru un proiect se dorește obținerea numelui și titlului științific ca date personale pentru cercetătorii care lucrează

la proiect. Pentru scurtarea căii de acces la date se poate recurge la includerea acestor date, atât în colecția referitoare la personalul de cercetare, cât și în colecția referitoare la proiecte. Similar se poate proceda și pentru colecția de date privind cercetătorii științifici din cadrul institutului.

### **Determinarea legăturilor dintre colecțiile de date și a modului de reprezentare a acestora**

Se realizează, în principal, pe baza legăturilor dintre entitățile identificate în cadrul etapei de analiză a sistemului și a cerințelor informaționale. Este necesar să se determine, de asemenea, legăturile dintre colecțiile care nu au un corespondent direct în entitățile (componentele) sistemului, dar care la rândul lor se află în asociere, unele cu altele. Modelul relațional reprezintă legăturile dintre colecțiile de date (relații) cu ajutorul cheilor externe sau cu ajutorul unor colecții de date distințe. Această reprezentare uniformă a datelor și a asocierilor între date prin intermediul relațiilor constituie o caracteristică a modelului relațional, care conferă acestuia o mare simplitate și flexibilitate.

### **Testarea schemei conceptuale**

Presupune verificarea completitudinii și consistenței schemei conceptuale, adică determinarea gradului în care schema conține elementele informaționale necesare satisfacerii cerințelor informaționale ale diferiților utilizatori și măsura în care legăturile stabilite între aceste elemente informaționale reflectă raporturile naturale dintre componentele sistemului real. De asemenea, prin testarea schemei conceptuale trebuie să se verifice dacă redundanța datelor este la un nivel minim și poate fi controlată. Testarea schemei conceptuale permite identificarea unor eventuale erori de proiectare care fac necesară revizuirea schemei. În acest caz se va relua etapa de proiectare a structurii bazei de date, uneori chiar și etapa de analiză a sistemului și a cerințelor informaționale.

### **Descrierea schemei conceptuale în limbajul de descriere a datelor și încărcarea în baza de date**

Descrierea schemei conceptuale se realizează în limbajul de descriere a datelor al SGBDR. Rezultatul acestei descrieri îl constituie *schema bazei de date*.

### 5.3.4 Proiectarea schemei externe

Schema externă a bazei de date reprezintă forma sub care apare schema conceptuală pentru un utilizator oarecare. Programele de aplicație operează asupra elementelor schemei conceptuale prin intermediul schemei externe, având acces doar la acele elemente care sunt incluse în schema externă.

În general, elementele care compun schema externă sunt similare celor care compun schema conceptuală, depinzând totuși de tipul de SGBD utilizat. În concepția CODASYL, schema externă reprezintă o parte a schemei conceptuale. Articolele care compun grupurile și înregistrările din schema externă pot差别 de articolele care compun grupurile și înregistrările din schema conceptuală. În cazul BDR, schema externă este realizată, în principal cu ajutorul tabelelor virtuale (*views*) și al mecanismelor de acordare a drepturilor de acces la BD.

### 5.3.5 Proiectarea schemei interne a bazei de date

Este cunoscut faptul că schema conceptuală încarcă diferite forme de structurare a datelor: liniară, arborescentă, rețea, relațională. Memorarea datelor pe suportul fizic îmbracă numai forma unei structuri liniare. De aceea, la proiectarea schemei interne a bazei de date se pune problema modului în care să fie liniarizată schema conceptuală.

Metoda de liniarizare a schemei conceptuale este specifică SGBD utilizat. O serie de SGBD, de exemplu INGRES, fac apel la metodele de memorare a datelor pe suportul de informație pe care le folosesc sistemele de operare gazdă. Alte SGBD utilizează metode proprii de stocare a datelor pe suportul fizic. Acestea depind mai puțin de sistemele de operare gazdă, ceea ce le imprimă o portabilitate sporită, comparativ cu SGBD din prima categorie.

## 5.4 Încărcarea datelor în baza de date

Aceasta este etapa în care se realizează popularea masivă cu date a bazei de date relaționale. Deși conținutul acestei etape este relativ simplu, activitatea putând fi considerată drept o activitate de rutină, fără dificultatea și creativitatea reclamate de activitățile de analiză și proiectare, încărcarea datelor în baza de date relațională reprezintă totuși o activitate dificil de realizat din cauza volumului mare de date care se transferă de la diferite surse de date. Popularea cu date trebuie să garanteze numai încărcarea datelor

corecte și aceasta cu un minim de efort, respectiv cu un minim de parcurgeri ale ciclului: validare-corectare.

Sursele de alimentare cu date a bazei pot fi:

- documente primare;
- colecții de date, gestionate prin diverse instrumente informatiche, de exemplu sisteme de gestiune a fișierelor.

Indiferent de sursa datelor, se recomandă ca în scopul încărcării bazei de date relaționale să se constituie colecții temporare de date (fișiere). În situația în care datele se preiau din documentele primare este necesară utilizarea unor colecții temporare pentru a se deplasa activitatea de validare a datelor cât mai devreme în procesul de încărcare a datelor în baza de date. Programele de încărcare a bazei de date, scrise în limbajul de manipulare a datelor de care dispune SGBDR trebuie să conțină cât mai puține validări întrucât acestea încetinesc mult execuția programelor și determină apariția unor *puncte albe* în baza de date, de exemplu, legături neconstituite din cauza inexistenței datelor.

Pe de altă parte, în situația în care datele se preiau din colecții gestionate prin alte instrumente informatiche este necesară utilizarea colecțiilor temporare pentru a se putea adapta cât mai bine structura acestor colecții la modul de organizare a datelor în baza de date relațională. Programele de încărcare vor fi în acest caz mai simple și mai robuste, asigurând un transfer mai rapid al datelor în cadrul bazei de date.

## 5.5 Exploatarea și întreținerea bazei de date relaționale

**Explotarea bazei de date relaționale** de către diferiți utilizatori finali este realizată în scopul satisfacerii cerințelor de informare ale acestora. SGBDR sprijină utilizatorii finali în exploatarea bazei de date, oferind o serie de mecanisme și instrumente cum ar fi, de exemplu, limbajele de manipulare a datelor care servesc la descrierea cerințelor de date.

**Întreținerea bazei de date relaționale** reprezintă o activitate complexă, care se referă la actualizarea datelor din cadrul bazei de date și la reproiectarea structurii acesteia. Activitatea este realizată în principal de către administratorul bazei de date. Întreținerea bazei de date relaționale este facilitată, ca și exploatarea acesteia, de către SGBD utilizat, de exemplu prin intermediul limbajelor de manipulare a datelor, al utilitarelor pentru generarea de statistică privind activitatea BDR etc. În momentul în care efortul de întreținere este foarte ridicat se poate lua decizia abandonării bazei de date și realizării uneia noi, care să o înlocuiască pe cea veche.

## Rezumat

Capitolul prezintă procesul etapizat de realizare a bazelor de date relaționale.

Prima etapă, analiza de sistem presupune realizarea analizei structurale sau statice, în urma căreia se obține modelul structural (static) al sistemului, analizei temporale (de comportament), prin care se obține modelul dinamic (temporal) al sistemului precum și analizei cerințelor informaționale, în urma acestei activități obținându-se modelul funcțional (transformațional) al sistemului economic. Se realizează de asemenea integrarea modelelor sistemului economic (structural, dinamic și funcțional) în scopul corelării și completării lor.

A doua etapă este cea de proiectare. Aceasta presupune proiectarea schemei conceptuale, a schemei externe precum și a schemei interne a bazei de date relaționale. Formele normale ale relațiilor dintr-o BDR sunt definite în raport de anomaliiile care pot apărea în lucrul cu aceste relații, deci în funcție de dependențele nedorite care se manifestă între datele din cadrul acestor relații. Normalizarea relațiilor presupune stabilirea schemei conceptuale inițiale și optimizarea progresivă a acesteia.

A treia etapă, încărcarea datelor în baza de date, este etapa în care se realizează popularea masivă cu date a BDR. Încărcarea datelor reprezintă o activitate dificil de realizat din cauza volumului mare de date care se transferă, cel mai frecvent, din diferite surse de date.

Ultima etapă este cea de exploatarea și întreținerea bazei de date relaționale.

## Note bibliografice:

Noțiunile legate de realizarea bazelor de date relaționale sunt prezentate pe larg în lucrările autorilor [LUSA03], [LUNG05a]. De asemenea, la realizarea acestui capitol s-au valorificat și surse bibliografice externe, precum cercetările publicate în [CHEN76], [CHEN99] și [CODD79].

## **Capitolul 6**

# **BAZE DE DATE ORIENTATE-OBIECT**

*Realizarea unor baze de date unitare, complexe și complete reprezintă o cerință importantă a economiei digitale. În momentul de față, în cazul în care există, acestea sunt în mare majoritate de tip relațional, caracterizate prin simplitate, implementare relativ ușoară și facilități de regăsire facilă a datelor prin intermediul unor limbaje de interogare puternice. Cu timpul, complexitatea lumii reale a condus către încercări de reprezentare a realității prin modele simple. Limitele sistemelor relaționale au determinat cercetări într-o nouă direcție, dominantă în domeniul programării, și anume tehnologia orientată-obiect. S-a ajuns astfel la o nouă generație de baze de date, cele orientate-obiect, capabile de a trata performant volume foarte mari de date, de complexitate ridicată, întâlnite cel mai adesea în noile tipuri de aplicații informaticice (multimedia, Internet etc.).*

**Cuvinte-cheie:** model de date orientat-obiect, bază de date orientată-obiect, clasă de obiecte, obiect, metode, atribute, moștenire, încapsulare, polimorfism.

*The development of unitary, complete and complex database is an important requirement of the digital economy. Currently, the databases are in most type relational – the most widely used since the '80s, those are characterized by simple, relatively easy implementation and data retrieval facilities through powerful query languages . Over time, the complexity of the real world has led to attempts to represent reality through simple models. The limits of the relational systems led researches in a new direction, dominant in programming, namely object-oriented technology. It was thus a new generation of databases, namely the object-oriented. The development of the object-oriented model can handle very large data volume with high complexity, encountered most often in new types of applications (multimedia, Internet etc.).*

**Keywords:** object-oriented model, object-oriented database, class, object, methods, attributes, inheritance, encapsulation, polymorphism.