BAZE DE DATE

CURS 1

Concepte şi problematica bazelor de date

1.1 Aspecte privind organizarea datelor
1.2 Definirea unei baze de date
1.3 Arhitecturi standardizate pentru bazele de date
1.4 Limbaje pentru baze de date
1.5 Avantajele utilizarii bazelor de date
1.6 Clasificarea sistemelor de baze de date

1.1 Aspecte privind organizarea datelor

Organizarea datelor ocupă un loc important în proiectarea sistemelor informatice. De modul în care sunt organizate datele depinde eficienţa sistemului. Procesul de organizare a datelor presupune următoarele activităţi:
- definirea, structurarea, ordonarea şi gruparea în colecţii de date;
- stabilirea relaţiilor între date, între elementele unei colecţii sau între colecţii de date;
- reprezentarea lor pe un suport informaţional

Pentru reprezentarea unui fenomen al lumii reale în vederea procesării informaţiilor ataşate, este nevoie să se definească un *model de date*. Definirea unui model de date impune precizarea următoarelor elemente:
- structurile de date şi stabilirea relaţiilor între date;
- operatorii care acţionează asupra structurilor de date;
- restricţii pentru menţinerea corectitudinii datelor, numite şi restricţii de integritate

Multitudinea relaţiilor care se pot stabili între date este de mare varietate. Din punct de vedere al modului în care sunt legate între ele prin relaţiile respective, se deosebesc următoarele tipuri de relaţii:

• relaţia de tip *unu- la- unu* sau *one-to-one*, notat (1→1);

De exemplu, relaţia dintre apartamentele construite cu credit de la stat şi proprietari este o relaţie one-to-one. Într-adevăr, un asemenea apartament poate avea un singur proprietar, iar un proprietar poate beneficia de un singur apartament construit cu credit de la stat.

• relaţia de tip *unu- la- mulţi* sau *one-to-many*, notat (1→n);

De exemplu, un elev poate face parte dintr-o singură clasă, o clasă poate avea mai mulţi elevi.

• relaţia de tip *mulţi- la- mulţi* sau *many-to-many*, notat (m→n).

De exemplu, un produs este achiziţionat de mai mulţi clienţi şi un client poate achiziţiona mai multe produse.

Operatorii care acţionează asupra structurilor de date reprezintă cel de-al doilea element al unui model de date şi pot fi de *citire, inserare, modificare, join* etc.

Restricţiile de integritate, cel de-al treilea element al unui model de date sunt restricţii menite să asigure corectitudinea datelor. Exemple de astfel de restricţii: să nu se accepte memorarea valorilor asociate atributelor unui produs dacă nu se cunoaşte valoarea cheii lui, sau să nu se permită ştergerea unui produs dacă clientul nu l-a achitat.

În funcţie de modul în care se definesc elementele de mai sus, modelele de date se clasifică astfel: *modele ierarhice* sau *arborescente*, *modele reţea, modele relaţionale, modele distribuite* (acestea se bazează pe primele trei modele cu particularităţi legate de distribuirea datelor), *modele orientate obiect, modele logice de date* (bazate pe logica de ordinul întâi).

1.2 Definirea unei baze de date

Evoluția metodelor şi tehnicilor de organizare a datelor a fost determinată de necesitatea de a avea un acces cât mai rapid şi uşor la un volum din ce în ce mai mare de informații precum şi de perfecționarea echipamentelor de culegere, memorare, transmitere şi prelucare a datelor.

Scopul principal al unei baze de date constă în stocarea datelor în vederea satisfacerii facile a cerințelor utilizatorului, utilizând tehnica de calcul. Deci, baza de date apare ca un sistem de înmagazinare, regăsire, actualizare şi întreținere a datelor necesare procesului de fundamentare a deciziei.

Definiția 1. *O **bază de date** este o colecție structurată de date ataşate unui fenomen al lumii reale pe care încercăm să îl modelăm.*

Exemplul 1. Pentru o facultate pot fi pastrate de exemplu pe perioade mari de timp informatii privind studentii, personalul, salile, planul de invatamant, aparatura si alte elemente despre care diferite persoane pot cere informatii la un moment dat. Intre aceste elemente exista diferite relatii cum ar fi: unii studenti fac anumite cursuri, unele cursuri se tin in anumite sali, unele aparate se afla in anumite sali, unele persoane pot tine cursuri si alte relatii asemanatoare.

Definirea sistemului de gestiune a bazei de date

O bază de date este o colecție de date stocate pe memorii adresabile, folosită de o mulțime de utilizatori. Însă o bază de date care nu are asociat un sistem de gestiune al acesteia nu are sens, ea nu îşi poate atinge obiectivele pentru care a fost creată.

Definiția 2. ***Sistemul de gestiune a bazei de date*** *(SGBD)* reprezintă software-ul care asigură realizarea următoarelor activități:
- definirea structurii bazei de date;
- încărcarea datelor în baza de date;
- accesul la date (interogare, actualizare);
- întreținerea bazei de date (colectarea şi refolosirea spațiilor goale, refacerea bazei de date în cazul unui incident);
- reorganizarea bazei de date (restructurarea datelor şi modificarea strategiei de acces);
- securitatea datelor.

Aşadar, sistemul de gestiune al bazei de date apare ca un sistem complex de programe care asigură interfața între o bază de date şi utilizatorii acesteia.

Privită într-un sens larg, baza de date implică patru componente:

- date
- hardware
- software
- utilizatori

Datele sunt memorate pe purtători tehnici de informații, hardware-ul se compune din volume de memorie pe care rezidă baza de date, iar software-ul asociat bazei de date se numeşte sistem de gestiune a bazei de date (SGBD). Toate cererile de acces la baza de date sunt tratate prin intermediul SGBD.

Figura 1. reprezintă o viziune simplificată a unei baze de date BD formată din colecțiile de date C_i, gestionată sub controlul unui produs software care este sistemul de gestiune SGBD, iar P_i sunt programe de aplicații pentru baza BD.

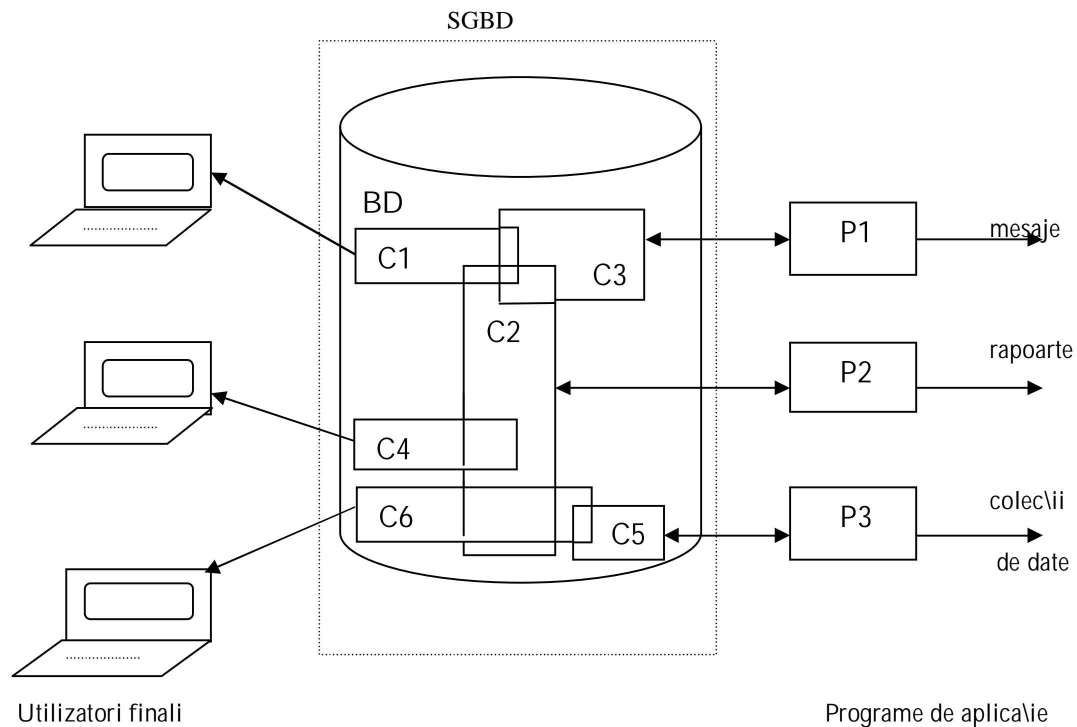


Figura 1. Arhitectura simplificată a unei bazei de date

Utilizatorii bazei de date pot fi grupați în trei mari categorii:

- *utilizatorii finali* sunt cei ce interacționează cu baza de date prin intermediul unui limbaj de interogare, sau care apelează programe scrise de programatorii de aplicații;

- *programatorii de aplicații* sunt cei care realizează programele de aplicații ale bazei de date, utilizând un limbaj de manipulare a datelor;

- *administratorul bazei de date* este o persoană sau un grup de persoane responsabil cu controlul general al SGBD-ului.

Un rol important în proiectarea şi implementarea unui SGBD îl are administratorul de sistem. Acesta are următoarele responsabilități mai semnificative:

• decide conținutul informațiilor din baza de date (defineşte schema conceptuală).

• decide structura de memorare şi strategia de acces, definind schema internă. Prin definirea structurii de memorare se stabileşte modul de descriere a stocării datelor pe suportul extern, care se poate referi la:

- fişiere care compun baza de date (nume, dimensiune, mod de organizare, etc.);
- articole care compun aceste fişiere (lungime, câmpuri componente, mod de plasare, etc.);
- căile de acces la articole (tabele de indecşi, înlănțuiri, etc.).
- stabileşte legăturile cu utilizatorii, definind schemele externe;
- defineşte verificările de autorizare şi procedurile de validare;
- defineşte strategia de refacere a bazei de date după incidente;
- monitorizează performanțele şi realizează schimbările cerute pentru a mări performanța.

Pentru îndeplinirea sarcinilor administratorul de sistem are nevoie de o serie de instrumente şi anume:

• rutine de încărcare, pentru crearea primei versiuni a bazei de date;

• rutine de reorganizare, prin care se realizează colectarea şi eliminarea golurilor, rezultate în urma ştergerilor de înregistrări din baza de date. Tot în cadrul acestor rutine intră şi

componentele care asigură reorganizarea bazei de date atunci când apar modificări în schema conceptuală;
• rutine de jurnalizare, pentru refacerea bazei de date după incidente hardware sau software. Jurnalele reprezintă fişiere în care sunt înregistrate, în timpul lucrului cu baza de date, informaţii privind tranzacţiile de actualizare, operaţiile acestor tranzacţii asupra bazei de date, înregistrările pe care urmează să le modifice şi înregistrările modificate. Toate aceste informaţii sunt utilizate la refacerea bazei de date;
• rutine de refacere, pentru a realiza din când în când copii ale bazei de date pe un suport de memorie externă.
• rutine de analiză statistică pentru înregistrarea datelor statistice privind funcţionarea şi performanţele sistemului de baze de date.

Un ultim instrument puternic pentru administratorul bazei de date îl reprezintă dicţionarul datelor, ce conţine informaţii privind structurile conceptuale, externe şi interne ale datelor, restricţiile de integritate, securitatea datelor, etc.

1.3 Arhitecturi standardizate pentru bazele de date

Pe plan internaţional există mai multe grupuri specializate în standardizarea conceptelor ce apar în dezvoltarea bazelor de date, cele mai importante fiind DBTG, CODASYL, ANSI/X3/SPARC, grupul IBM.

Arhitectura bazelor de date evidenţiază componentele acestora şi a fost standardizată internaţional. În Figura 1.1.2 se prezintă o arhitectură detaliată a unei baze de date în sens larg în cadrul căreia, se evidenţiază o serie de componente cum ar fi: nivele de definire cu schema corespunzătoare fiecărui nivel, utilizatorii bazei de date, administratorul bazei de date, ca utilizator special, sistemul de gestiune a bazei de date (SGBD), etc.

În general, o arhitectură cuprinde următoarele componente:
• baza de date propriu-zisă în care se memorează colecţia de date;
• sistemul de gestiune al bazei de date, care este un ansamblu de programe ce realizează gestiunea şi prelucrarea complexă a datelor;
• un dicţionar al bazei de date, ce conţine informaţii despre date, structura acestora, elemente de descriere a semanticii, statistici, etc;
• mijloace hard utilizate (comune, specializate, etc.);
• personal implicat: utilizatori finali (neinformaticieni) sau de specialitate (administratorul de sistem), programatori şi operatori.

Conform standardului ANSI/X3/SPARC datele pot fi definite pe trei nivele:

- nivelul intern
- nivelul conceptual
- nivelul extern

Nivelul intern corespunde structurii interne de memorare a datelor. Schema internă permite descrierea datelor unei baze sub forma în care sunt stocate în memoria calculatorului. Sunt definite fişierele care conţin aceste date, articolele din fişiere, metodele de acces la aceste articole. La acest nivel schemele descriu o bază de date.

Nivelul conceptual este nivelul central. Acesta corespunde structurii canonice a datelor ce caracterizează procesul de modelat, adică structura semantică a datelor fără implementarea pe calculator.

La *nivel extern* schemele descriu doar o parte din date şi anume, cele ce prezintă interes pentru un utilizator sau un grup de utilzatori. Schema externă reprezintă o descriere a unei părţi a bazei de date ce corespunde viziunii unui program sau utilizator. Modelul extern folosit este dependent de limbajul utilizat pentru manipularea bazei de date.

Pot apărea diferenţe între definirile din schema externă şi cele din schema conceptuală (altă reprezentare, alt nume, altă ordine, etc.). Prin urmare este necesară o *transformare extern/conceptual* care va defini corespondenţa dintre viziunea conceptuală şi baza de date memorată. Dacă se schimbă structura de memorare a bazei de date, se va schimba corespunzător această transformare, astfel încât schema conceptuală şi aplicaţiile să rămână neschimbate. Structurarea bazei de date pe cele trei nivele (extern, conceptual şi intern) asigură independenţa datelor.

O bază de date are mai multe sheme externe, o singură schemă conceptuală şi o singură schemă internă, corespunzător celor trei nivele de structurare a datelor.

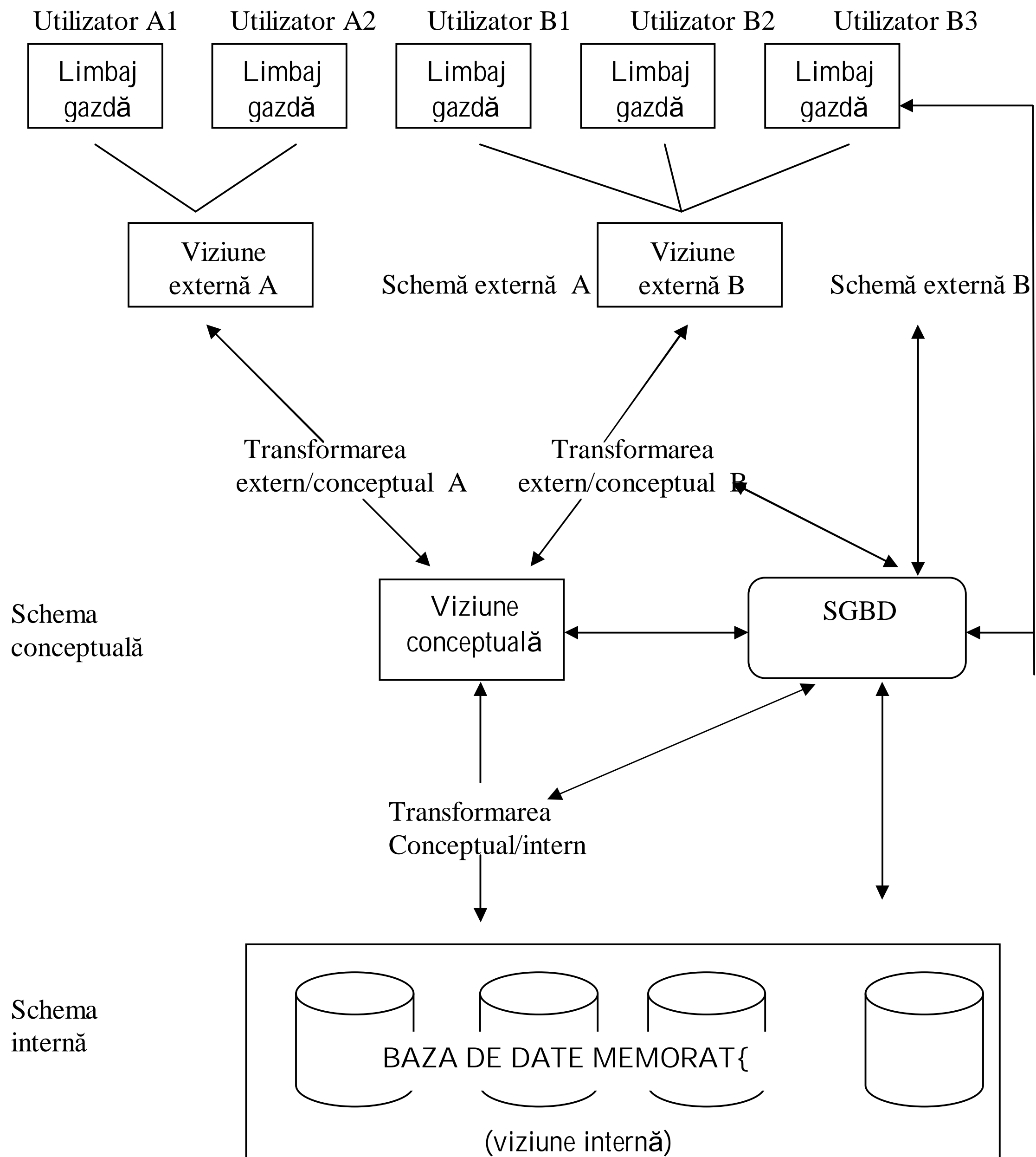


Figura .2 Arhitectura unui sistem de baze de date

1.4 Limbaje pentru baze de date

La limbajele de programare uzuale (Pascal, C, Fortran), declarațiile şi instrucțiunile executabile aparțin aceluiaşi limbaj. În lumea bazelor de date, funcțiile de declarare şi de manipulare a datelor sunt realizate cu ajutorul unor limbaje diferite.

1. Limbaje pentru definirea datelor (LDD)

Descrierea concretă a unui LDD este specifică fiecărui sistem de gestiune, dar funcțiile principale sunt aceleaşi. La nivel conceptual, LDD realizează definirea entităților şi a atributelor acestora prin nume, formă de memorare, lungime- descriind astfel natura datelor. Sunt precizate relațiile dintre date şi strategiile de acces la ele, sunt stabilite criterii de confidențialitate şi de validare automată a datelor utilizate. Acest limbaj perimite obținerea meta-datelor stocate în dicționarul de date.

2. Limbaje pentru manipularea datelor (LMD)

Operațiile pe baze de date solicită un limbaj specializat, în care comenzile se exprimă prin fraze ce descriu acțiuni asupra bazei.

În general, o comandă are următoarea structură:

- operația, care poate fi calcul aritmetic sau logic, editare, extragere, manipulare (adăugare, ştergere, modificare, etc.);
- criterii de selecție (for, while, where, etc.);
- mod de acces (secvențial, indexat, etc.);
- formă de editare.

3. Limbaje pentru controlul datelor (LCD)

Controlul unei baze de date se referă la asigurarea confidențialității şi integrității datelor, la salvarea informației în cazul unor incidente, la obținerea unor performanțe, la rezolvarea unor probleme de concurență.

În Figura 3 se prezintă modul cum un program de aplicație poate interacționa cu o bază de date. Datele locale aparțin programului de aplicație şi sunt manipulate de acesta. Aceste date pot fi utilizate pentru a insera, modifica sau extrage informații în/din baza de date.

Interfața între un utilizator şi un SGBD poate fi realizată în două moduri:

• cu ajutorul unui mecanism de apel inserat în programul aplicație. Acest mecanism poate fi un CALL sau au alt cuvânt cheie. Un SGBD care permite acest tip de mecanism se numeşte SGBD cu *limbaj gazdă.*

• cu ajutorul unor comenzi speciale, utilizate independent. În acest caz, SGBD se numeşte *autonom*. Există totuşi o interfață specială, care este în măsură să interpreteze comenzile limbajului de cereri.

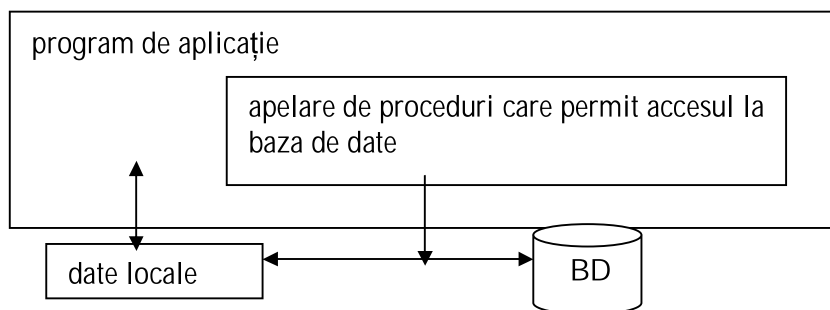


Figura 3. Interacțiunea program de aplicație-bază de date.

1.5 Avantajele utilizării bazelor de date

Memorarea datelor într-o bază de date oferă posibilitatea realizării unui control centralizat asupra acestora. Un astfel de control asupra datelor oferă următoarele avantaje:

• poate fi redusă *redundanța* datelor. Nu se recomandă eliminarea totală a redundanței; uneori, pentru a realiza performanțe sporite sub aspectul vitezei de regăsire a datelor sau din alte considerente de ordin practic (datorate de cele mai multe ori caracteristicilor de implementare a SGBD-ului), se acceptă un anumit grad de redundanță. Deci, într-o bază de date se recomandă o redundanță minimă şi controlată a datelor.

• se poate evita *inconsistența* datelor. Acest avantaj este un corolar al punctului anterior. Când redundanța este înlăturată nu pot apărea inconsistențe, iar când există, este controlată şi sistemul asigură propagarea actualizării la fiecare copie a aceleiaşi date.

• datele pot fi *partajate*. Partajarea datelor trebuie înțeleasă nu numai sub aspectul asigurării accesului mai multor utilizatori la aceleaşi date ci şi sub aspectul dezvoltării de noi aplicații fără să se modifice structura bazei de date.

• se poate obține *standardizarea*. Utilizând baza de date, administratorul bazei de date poate asigura aplicarea standardelor în reprezentarea datelor. Aceste standarde sunt: standardele întreprinderii, ale echipamentelor de tehnică de calcul, ale ramurii de activitate, ale economiei naționale, internaționale, etc.

• se pot aplica *restricții de securitate* a datelor. Având o jurisdicție completă asupra datelor operaționale, administratorul bazei de date poate asigura că accesul la baza de date se face numai prin canale corespunzătoare. În acest sens se pot defini verificări de autorizare, care să fie executate oricând se încearcă un acces la anumite date.

• poate fi menținută *integritatea* datelor prin existența unor proceduri de validare sau a unor protocoale de control concurent precum şi a unor proceduri de refacere a bazei de date după incidente.

• pot fi echilibrate *cerințele conflictuale*. Cunoscând cerințele de ansamblu ale întreprinderii, în opoziție cu cerințele fiecărui utilizator individual, administratorul bazei de date poate structura baza de date în aşa fel încât să satisfacă cerințele tuturor utilizatorilor în condiții de redundanță minimă şi controlată a datelor, pe de o parte, iar pe de altă parte, pot fi definite o serie de criterii de regăsire care să permită un acces rapid pentru aplicațiile mai importante.

• *independența datelor*. Multe din avantajele de mai sus sunt evidente. Un aspect nu aşa de evident, care este mai degrabă un obiectiv decât un avantaj, îl constituie asigurarea *independenței* datelor. Se spune că o aplicație este dependentă de date, dacă este imposibil de schimbat structura de memorare a datelor (modul cum sunt înregistrate fizic datele) sau strategia de acces la date fără a afecta aplicația.

Într-o bază de date nu se doreşte ca aplicațiile să fie dependente de date, cel puțin din următoarele motive:

- diferite aplicații au nevoie de viziuni diferite ale aceloraşi date. De exemplu, o aplicație utilizează acelaşi câmp de dată drept o dată zecimală în timp ce o altă aplicație utilizează acelaşi câmp de dată ca fiind reprezentat în binar. Sistemul va asigura automat conversia între reprezentarea internă a acelei date şi reprezentarea necesară fiecărei aplicații.

- administratorul bazei de date trebuie să aibă libertatea să schimbe structura de memorare sau strategia de acces, ca răspuns la cerințele de schimbare (întreprinderea trebuie să-şi schimbe standardele, prioritățile aplicațiilor, unitățile de memorie, etc.), fără să modifice aplicațiile existente.

• organizarea datelor pe două niveluri, *fizic* şi *logic*.

1.6 Clasificarea sistemelor de baze de date

Se pot lua în considerație mai multe criterii de clasificare ale sistemelor de baze de date.

Clasificare după modelul de date. Majoritatea sistemelor de baze de date actuale sunt realizate în modelul de date relațional sau în modelul de date obiect. Dezvoltarea continuă a acestor modele a condus către o nouă categorie de baze de date, numite *obiect-relaționale*, care combină caracteristicile modelului relațional cu cele ale modelului obiect. De asemenea, mai sunt încă în funcțiune baze de date în modele mai vechi (*modelul ierarhic* sau *modelul rețea*). Modelele de date utilizate de sistemele SGBD vor fi prezentate în secțiunea următoare.

Clasificare după numărul de utilizatori. Majoritatea sistemelor de baze de date sunt sisteme *multiutilizator*, adică permit accesul concurent (în acelaşi timp) a mai multor utilizatori la aceeaşi bază de date. Un număr redus de sisteme de baze de date sunt de tip *monoutilizator*, adică suportă accesul doar al unui singur utilizator (la un moment dat).

Clasificare după numărul de stații pe care este stocată baza de date. Există două categorii de sisteme de baze de date: centralizate şi distribuite.

Un sistem de baze de date *centralizat* (Centralized Database System) este un sistem de baze de date în care datele şi sistemul de gestiune sunt stocate pe o singură stație (calculator).

Un sistem centralizat poate suporta unul sau mai mulți utilizatori, dar, în orice situație, datele şi sistemul de gestiune rezidă în întregime pe o singură stație.

Un sistem de baze de date *distribuit* (Distributed Database System) poate avea atât datele, cât şi sistemul de gestiune, distribuite în mai multe stații interconectate printr-o rețea de comunicație.

Sistemele de baze de date pot fi reprezentate din punct de vedere al funcționării lor printr-o arhitectură de tip *client-server.*

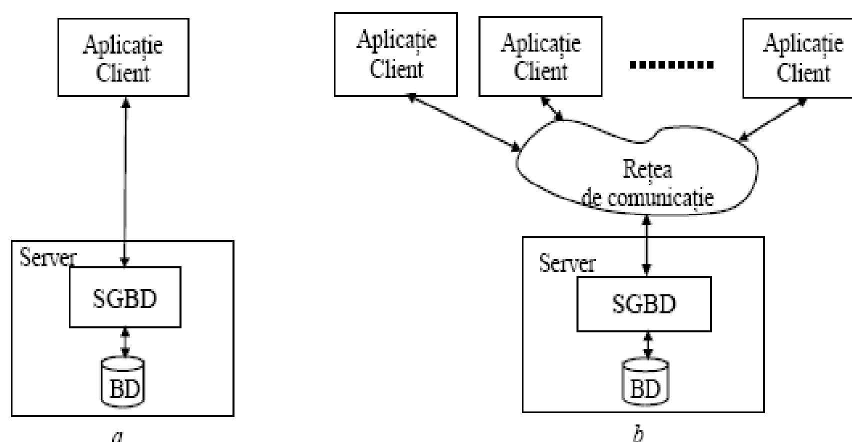


Fig. 4. Sisteme de baze de date centralizate: *a*- monoutilizator; *b*- multiutilizator.

Într-un sistem centralizat (fig. 4) există un singur *server*, care este chiar sistemul SGBD, ce răspunde cererilor unui singur *client* (în sistemele mono-utilizator, fig. 4, *a*) sau mai multor *clienți* (în sistemele multi-utilizator, fig. 4, *b*), care accesează baza de date respectivă. *Clienții* sunt programe de aplicații oferite de furnizorul sistemului de gestiune sau dezvoltate de programatori.

Aplicațiile client pot fi executate pe stații diferite, conectate printr-o rețea de comunicație cu stația pe care rulează serverul. Această arhitectură permite o prelucrare distribuită a datelor şi, mai mult, o configurare a sistemului adaptată cerințelor de calcul

particulare. Astfel, serverul bazei de date poate fi un sistem puternic, echipat corespunzător (cu volum mare de memorie secundară), în timp ce fiecare client este o stație personală, cu putere de calcul adecvată aplicației executate.

Sistemele de baze de date distribuite pot fi reprezentate într-un mod asemănător din perspectiva structurării client-server (fig. 5).

O bază de date distribuită este o colecție de date care aparțin din punct de vedere logic aceluiaşi sistem, dar care pot să fie, din punct de vedere fizic, memorate în mai multe stații de calcul (locații - *sites*) conectate printr-o rețea de comunicație. Sistemul software care gestionează o astfel de bază de date se numeşte Sistem de Gestiune a Bazei de Date Distribuite - SGBDD - (*Distributed Database Management System - DDBMS*). Aplicațiile client rulează pe alte stații din rețea şi solicită servicii de la sistemul de gestiune distribuit.

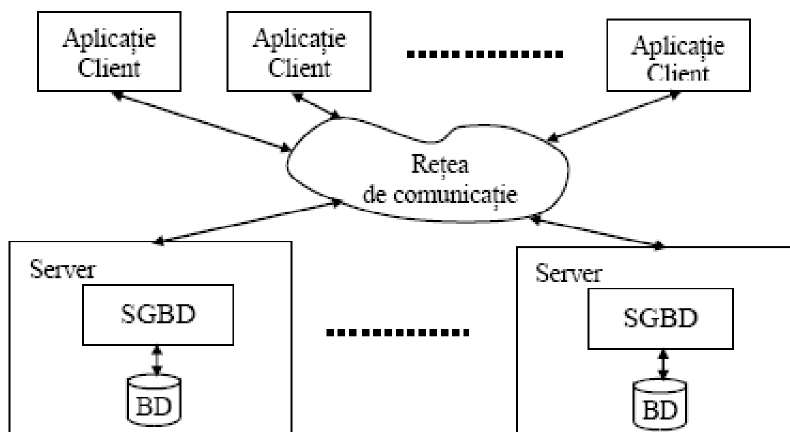


Fig.5 Sistem de baze de date distribuit

Există numeroase avantaje ale sistemelor de baze de date distribuite (creşterea capacității de stocare şi prelucrare a datelor, creşterea disponibilității şi a partajării datelor, etc.), dar şi o creştere considerabilă a complexității acestora.

Cea mai importantă cerință pe care trebuie să o îndeplinească sistemele de gestiune a bazelor de date distribuite este de a asigura administrarea transparentă a datelor. Transparența se referă la capacitatea unui sistem distribuit de a ascunde detaliile de implementare, astfel încât utilizatorii să poată accesa datele pe baza unui model de nivel înalt, fără a fi necesară cunoaşterea exactă a modului de amplasare, replicare sau comunicare a datelor.

Sistemele de gestiune a bazelor de date distribuite comerciale nu oferă în momentul de față un nivel suficient de transparență a localizării datelor, dar dezvoltarea continuă a acestora va putea să asigure în viitor această cerință.

BAZE DE DATE

CURS 2

ASPECTE PRIVIND PROIECTAREA BAZELOR DE DATE

2.1 Proiectarea schemei conceptuale

În ultima decadă a mileniului trecut s-a realizat o creştere rapidă a numărului de sisteme de baze de date (SGBD), dar mai ales a sistemelor care permit stocarea a mari cantități de informații şi dezvoltarea de aplicații complexe. Această cerere continuă de sisteme complexe şi de mare precizie a fost stimulată de concepte de nivel înalt, de instrumente şi de tehnici de proiectare şi dezvoltare a bazelor de date. Metodologia de proiectare a bazelor de date, ce a fost dezvoltată în ultimii ani, constituie pentru proiectant un mijloc de modelare a unei întreprinderii la un nivel înalt de abstractizare, mai înainte de a proceda la proiectarea logică şi fizică detaliată a bazelor de date.

Pentru baze de date de dimensiuni mici, care sunt folosite de un singur utilizator sau de un număr redus de utilizatori, proiectarea este destul de simplă. Însă, pentru baze de date de dimensiuni medii sau mari, care fac parte din sistemul informatic al unei organizații extinse, proiectarea este mult mai complicată. Astfel de baze de date, care trebuie să satisfacă cerințele a numeroase aplicații şi utilizatori trebuie să fie proiectate cu multă grijă şi testate cât mai riguros. Numeroase instituții şi servicii ca: asigurările, serviciile bancare, serviciile de transport, companiile de comunicații, etc. sunt total dependente de funcționarea corectă şi neîntreruptă a sistemelor lor de baze de date.

În organizațiile mari comerciale, industriale sau guvernamentale, sistemul de baze de date face parte din sistemul informatic al organizației respective. **Sistemul informatic** (Information system) include toate resursele unei organizații care sunt implicate în colectarea, administrarea, utilizarea şi diseminarea informațiilor.

Crearea, utilizarea şi întreținerea sistemelor de baze de date comportă mai multe etape, care pot fi prezentate succint astfel:

a) *Definirea sistemului:* definirea scopului sistemului de baze de date, a utilizatorilor şi a aplicațiilor acestuia.

b) *Proiectarea sistemului:* în această etapă se realizează proiectul logic şi proiectul fizic al sistemului, pentru un anumit SGBD ales.

c) *Implementarea:* este etapa în care se scriu definițiile obiectelor bazei de date (tabele, vederi, etc.) şi se implementează aplicațiile software.

d) *Încărcarea (sau conversia) datelor,* popularea bazei de date, fie prin încărcarea directă a datelor, fie prin conversia unor date existente sub diferite alte forme (fişiere, alte sisteme de gestiune a bazelor de date).

e) *Conversia aplicațiilor,* toate aplicațiile software existente în sistemele informatice precedente ale organizației se convertesc în noul sistem.

f) *Testarea şi validarea:* noul sistem de baze de date este testat şi validat cât mai riguros posibil.

g) *Operarea:* sistemul de baze de date este pus la dispoziția utilizatorilor săi, cu toate aplicațiile realizate, în cadrul sistemului informatic al organizației.

h) *Monitorizarea şi întreținerea:* pe tot parcursul etapei de operare sistemul de baze de date trebuie să fie în mod permanent monitorizat şi întreținut, pentru a asigura consistența şi securitatea datelor şi pentru a permite atât creşterea conținutului de date cât şi dezvoltarea de noi aplicații software. Pot fi necesare, la anumite intervale de timp, revizii şi reorganizări ale sistemului de baze de date.

Etapele de conversie (d) şi (e) nu sunt necesare dacă baza de date şi aplicaţiile sunt noi. În continuarea acestei secţiuni se vor detalia diferite aspecte cu caracter general ale etapei (b), de proiectare a bazelor de date şi a aplicaţiilor software pentru acestea.

Etapa de proiectare se referă în general la *modelarea semantică* ale datelor pentru aplicaţiile bazei de date. Modelele de date sunt esenţiale în tehnologia bazelor de date şi constituie o bază formală pentru dezvoltarea limbajelor şi sistemelor de implementare a sistemelor de baze de date.

Modelarea datelor este un proces în două etape, care implică:

1. **fază de proiectare conceptuală**, când se proiectează schema conceptuală ce constituie o abstractizare a universului real şi a situaţiei considerate.
2. **proiectarea structurii logice a datelor,** ce constituie schema care va fi implementată.

Pe parcursul activităţii de proiectare trebuie să fie satisfăcute mai multe cerinţe, multe dintre ele fiind contradictorii şi dificil de îndeplinit: obţinerea unui timp de răspuns la interogări cât mai mic, şi, în acelaşi timp, utilizarea unui spaţiu de memorare cât mai redus; asigurarea unui mod de acces la date cât mai simplu dar intuitiv, etc.

Problema proiectării bazelor de date este complicată şi de faptul că, de cele mai multe ori, procesul de proiectare începe cu cerinţe foarte generale şi imprecis formulate. Prin contrast, proiectul rezultat trebuie să conţină schema bazei de date precis definită, dat fiind că, după implementarea aplicaţiilor, modificarea bazei de date este mult mai dificilă.

În general, se consideră că etapele de proiectare şi implementare a unei baze de date se pot diviza, la rândul lor, în mai multe faze:

- Colectarea şi analiza cerinţelor.
- Proiectarea conceptuală a bazei de date.
- Alegerea unui SGBD.
- Proiectarea logică a bazei de date.
- Proiectarea fizică a bazei de date.
- Implementarea bazei de date.

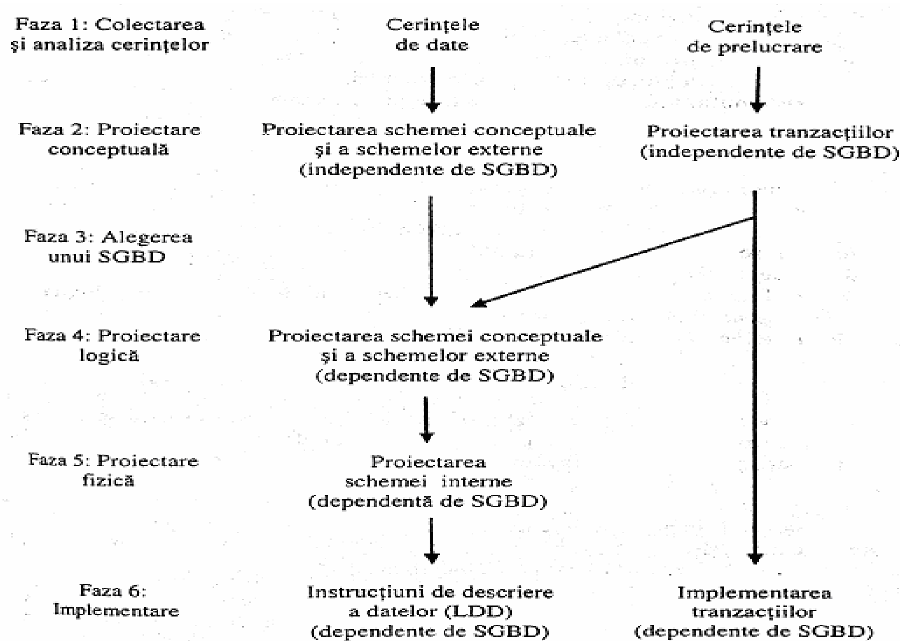


Figura 2.1 Etapele proiectarii şi implementarii unei baze de date

În mod tipic, proiectarea unei baze de date constă din desfăşurarea a două categorii de activităţi paralele, aşa cum se poate vedea în figură.

Prima categorie de activități se referă la proiectarea *structurii şi a conținutului bazei de date,* iar cea de-a doua categorie se referă la proiectarea modului de *prelucrare a datelor.*

Cele şase faze de proiectare şi implementare enumerate mai sus nu se desfăşoară strict într-o singură secvență. În multe cazuri este necesară modificarea proiectului dintr-o fază inițială într-una din fazele ulterioare, pentru a se obține rezultatele dorite.

Fazele cele mai importante de proiectare a unei baze de date sunt fazele 2, 4 şi 5. Faza 1, în care se colectează informațiile despre cerințele de utilizare a bazei de date şi faza 6, de implementare a datelor şi a tranzacțiilor, pot să nu fie considerate ca parte a procesului de proiectare a bazei de date, ci ca parte a ciclului de viată al sistemului informatic din care face parte baza de date respectivă. Faza 3, de alegere a sistemului de gestiune a bazei de date, este mai puțin o fază de proiectare, ci mai curând o fază de decizie.

Terminologia folosită în domeniul proiectării bazelor de date este destul de variată. Cel mai frecvent, pentru proiectul conceptual al unei baze de date se folosesc denumirile de *schemă conceptuală de nivel înalt* sau *schemă conceptuală independentă de SGBD* sau, chiar mai simplu, *schemă conceptuală.* Proiectul logic al unei baze de date este denumit *schemă logică* sau *schemă conceptuală dependentă de SGBD*.

2.1.1 Cerințe generale

Înainte de a se proiecta efectiv o bază de date, este necesar să se cunoască ce rezultate se aşteaptă utilizatorii potențiali să obțină de la baza de date respectivă şi ce informații primare sunt disponibile pentru aceasta. De asemenea, este necesar să se cunoască ce aplicații se vor efectua (aplicații de gestiune a stocurilor, aplicații contabile, aplicații de urmărire a consumurilor, aplicații de salarizare, etc).

În această fază de colectare şi analiză a cerințelor, se desfăşoară următoarele activități:

Identificarea grupurilor de utilizatori potențiali şi a aplicațiilor. De regulă, persoana cea mai avizată din cadrul fiecărui grup de utilizatori este cooptată ca participant în activitățile ulterioare de colectare şi analiză a cerințelor.

Revederea documentației existente privind aplicațiile dorite. În afară de documentațiile aplicațiilor dorite se studiază şi alte documentații (diagramele de organizare a întreprinderii, formularele existente de introducere a datelor, rapoartele utilizate în controlul activității respective, etc), pentru a decide dacă acestea influențează cerințele bazei de date.

Analiza mediului de operare şi a cerințelor de prelucrare a datelor. Această activitate include analiza fluxului de informații în cadrul sistemului, precum şi analiza tipurilor de tranzacții şi a frecventei de lansare a acestora. Deosebit de importantă este şi stabilirea volumului de date conținute în mod tipic de baza de date, a volumului şi frecvenței datelor actualizate precum şi a volumului datelor returnate de interogări şi a frecvenței acestora.

Chestionare şi interviuri. Se colectează răspunsuri scrise de la utilizatorii potențiali la diferite seturi de întrebări şi se organizează interviuri cu persoanele care reprezintă diferitele grupuri de utilizatori. Această fază este puternic consumatoare de timp, dar este crucială pentru succesul sistemului informatic.

Faza de proiectare conceptuală a bazelor de date implică două activități paralele: proiectarea *schemei conceptuale şi a schemelor externe* ale bazei de date şi *proiectarea tranzacțiilor.*

a. Proiectarea schemei conceptuale

Scopul proiectării schemei conceptuale este înțelegerea cât mai completă a structurii bazei de date, a asocierilor şi a constrângerilor de către proiectanți şi programatori. Acest deziderat se obține mult mai bine independent de un anumit SGBD, deoarece un model de date de nivel înalt este mult mai general şi mai expresiv, în timp ce fiecare SGBD are propriile restricții şi soluții particulare, care nu trebuie să influențeze proiectul schemei conceptuale.

Schema conceptuală de nivel înalt independentă de SGBD este o descriere stabilă şi inavuabilă a bazei de date. Alegerea unui SGBD şi deciziile ulterioare de proiectare se pot schimba fără ca aceasta să se schimbe.

Descrierea diagramatică a schemei conceptuale poate servi ca un excelent mijloc de comunicaţie între analiştii, proiectanții, programatorii şi utilizatorii bazei de date, deoarece modelele de date de nivel înalt sunt bazate pe concepte mai uşor de înțeles decât un model de date de nivel mai scăzut, specific unui anumit SGBD.

Pentru proiectarea schemei conceptuale se identifică elementele esențiale ale acesteia, tipurile de entități şi atributele lor precum şi asocierile dintre aceste tipuri. Acest proiect conceptual de nivel înalt este realizat pe baza cerințelor definite în prima etapă de proiectare şi se reprezintă, în general printr-o diagramă Entitate-Asociere (extinsă).

Există două aspecte privind modul de **abordare** a proiectării conceptuale:

1. *proiectarea prin integrarea cerințelor* (care se mai numeşte şi *proiectare centralizată)*; se realizează mai întâi integrarea (combinarea) tuturor cerințelor de proiectare într-un singur set de cerințe, pe baza căruia se proiectează schema conceptuală a bazei de date. Din această schema conceptuală (unică) proiectată se deduc schemele externe (vederile) corespunzătoare diferitelor grupuri de utilizatori şi aplicații.

2. *proiectarea prin integrarea vederilor (schemelor) externe*; se proiectează câte o schemă corespunzătoare fiecărui grup de utilizatori şi aplicații, pe baza cerințelor acestora, după care se combină aceste scheme într-o singură schemă conceptuală globală, pentru întreaga bază de date.

Fiind dat un set de cerințe, pentru un singur utilizator sau pentru mai mulți utilizatori ai bazei de date, proiectarea schemei conceptuale care să satisfacă aceste cerințe se poate aborda într-o varietate de strategii, dintre care cele mai relevante sunt proiectarea descendentă *(top-down)* şi proiectarea ascendentă *(bottom-up)*.

În proiectarea descendentă a bazelor de date se porneşte de la o schemă conceptuală dezvoltată în modelul Entitate-Asociere (extins) în care sunt cuprinse aspectele de bază (tipuri de entități şi asocieri ale acestora). Dezvoltarea şi rafinarea acestui model se face prin adăugarea de noi atribute şi constrângeri, crearea unor subtipuri (prin specializare) şi asocierea acestora cu tipurile de bază.

În proiectare ascendentă a bazelor de date se porneşte de la o *schemă conceptuală universală* care conține toate atributele, care sunt apoi grupate pentru a forma tipuri de entități şi asocierile dintre acestea. Proiectul poate fi rafinat prin grupări ulterioare şi creare de supertipuri şi subtipuri ale tipurilor existente.

Aşadar, în această fază de proiectare se obține schema conceptuală de nivel înalt a bazei de date, care este independentă de orice model de date specific (ierarhic, rețea, relațional, etc), şi de orice sistem de gestiune care poate fi folosit pentru realizarea bazei de date.

b. Proiectarea tranzacțiilor

Atunci când se proiectează o bază de date, se cunosc în mare parte şi ce tipuri de aplicații se vor executa. Un aspect important în proiectarea bazelor de date este specificarea caracteristicilor funcționale ale acestor aplicații cât mai devreme în cursul procesului de proiectare, astfel încât schema bazei de date să includă toate informațiile necesare cu privire la aceste aplicații. In plus, cunoaşterea importanței relative a diferitelor tranzacții executate de aplicațiile bazei de date, precum şi a frecvenței de invocare a acestora, permite luarea unor decizii în etapa de proiectare fizică a bazei de date.

După implementarea sistemului de baze de date vor mai fi identificate şi implementate noi tranzacții. Totuşi, cele mai importante tranzacții sunt cel mai adesea cunoscute înainte de implementarea sistemului.

Una din tehnicile cele mai obişnuite de specificare a tranzacțiilor la nivel conceptual şi independent de sistemul de gestiune este de a identifica *intrările şi ieşirile* lor şi *comportarea*

funcțională. Tranzacțiile sunt grupate în trei categorii: tranzacții de interogare, tranzacții de actualizare şi tranzacții mixte, care execută atât interogări cât şi actualizări.

2.1.2 Metode de proiectare conceptuală bazate pe descompunerea funcțională.

Metoda TOP-DOWN a fost aplicată cu mult înainte de existența informaticii ca ştiință. Abordarea acesteia în informatică a fost determinată de schimbările iminente ce trebuiau să se producă pentru propulsarea informaticii spre stadiul în care se află astăzi şi pentru rezolvarea problemelor majore cu care se confruntau proiectanții de software, în sensul că produsele pe care le realizau erau de nestăpânit sub aspectul complexității lor. Dezvoltarea metodei a fost determinată de lucrările lui Jacopini, Djikstra, Hoare, Mills, Wirth şi Dahl care au impus-o şi au definit-o riguros, rezultând submetode specifice ariei de aplicabilitate, cum ar fi:

- metoda top-down structurată;
- metoda deciziilor multicriteriale în condiții de certitudine;
- metoda de căutare în adâncime;
- metoda deciziilor top-down structurate ş.a.m.d.

Pentru elaborarea modelului metodei TOP-DOWN vom considera simbolurile din figura 2.2.

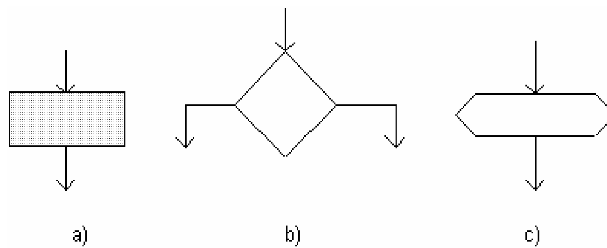


Figura 2.2 Simboluri pentru schemele TOP-DOWN.

a) secvența terminată logic, direct programabilă;

b) decizie care defineşte problemele de tratat;

c) subproblemă care va fi tratată ulterior, dar care acum se defineşte.

Metoda constă în descompunerea problemei de rezolvat în subprobleme legate între ele prin marcarea punctelor decizionale care se constituie ca repere în abordarea problemei. Această descompunere este întemeiată pe definirea pentru fiecare subproblemă în parte a unei funcții specifice care o individualizează în contextul dat. Pe baza simbolurilor o problemă abstractă de proiectare, ca în figura 2.3, poate fi descompusă în blocuri corelate între ele prin puncte decizionale şi aşa poate fi evaluat stadiul în care se află abordarea problemei.

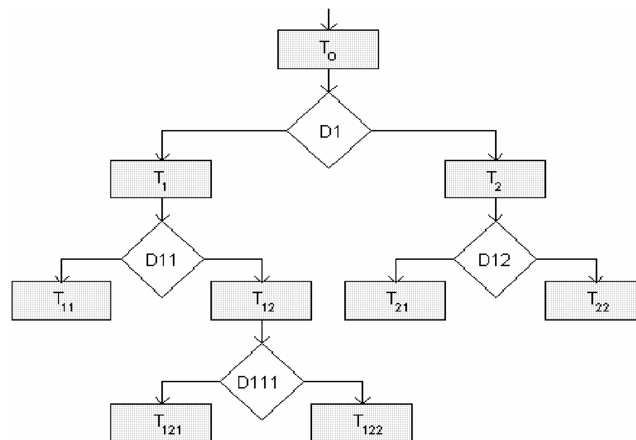


Figura 2.3. Exemplu de descompunere a unei aplicații.

Abordarea **top-down** a problemei definite în figura 2.3, presupune reprezentarea din fig. 2.4.

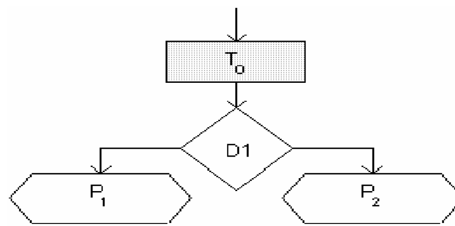


Figura 2.4. Abordarea TOP-DOWN

Observaţie: Toate schemele de mai sus şi cele care urmează pot fi descrise în pseudocod, folosind numai enunţurile de

- atribuire: **v←<expresie>**;
- ramificare: **IF <expresie logică> THEN <Instr1>**
  ELSE <Instr2>;
- apelare proceduri: **CALL <nume procedură>;**

Pentru o abordare **bottom-up** a aceleiaşi probleme, reprezentăm succesiunea de etape prin figurile 2.5. - 2.6.

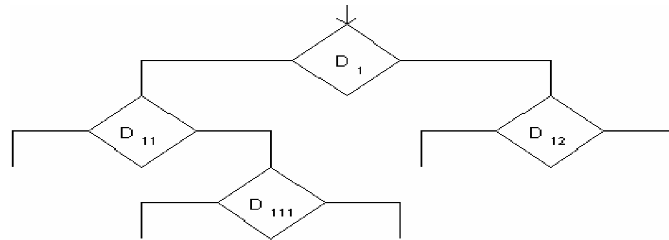


Figura 2.5. Abordarea BOTTOM-UP. Etapa 1.

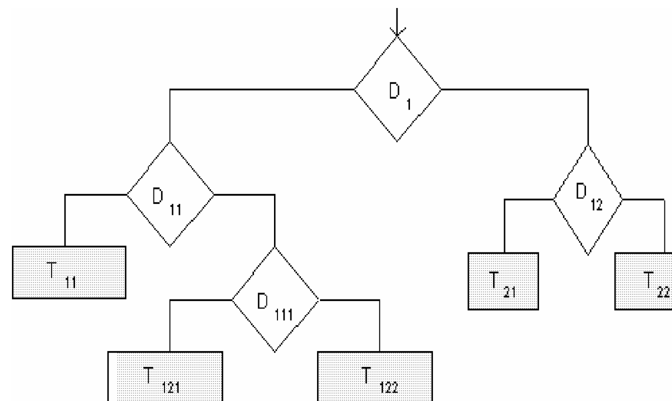


Figura 2.6. Abordarea BOTTOM-UP. Etapa 2.

2.1.3 Metode de proiectare conceptuală bazate pe structuri de date

Metoda Jackson se bazează pe corespondenţa dintre structura datelor şi structura programelor. În elaborarea ei s-a pornit de la premiza că programele corecte nu pot fi obţinute pe baza testării unor programe incorecte. Prin corespondenţa care se stabileşte între structura

datelor şi structura programelor se ajunge la o descompunere a programelor în procese distincte.

Această metodă îmbină armonios principiile programării modulare, ale programării structurate cu principiul corespondenţei între structura datelor şi structura programelor. Prin procedee constructive, structura programelor este derivată din structura datelor.

Componentele specifice unui program structurat sunt folosite de către Jackson nu numai pentru structura programelor ci şi pentru structura datelor. De aici, derivă faptul că adaugă în construirea programelor un principiu suplimentar faţă de principiile programării structurate, şi anume: structura programelor se bazează în întregime pe structura datelor prelucrate. Pentru reprezentarea proiectului software se foloseşte:

- reprezentarea grafică sub forma unor diagrame de structură atât pentru date cât şi pentru programe;
- reprezentarea textuală sub forma unui pseudocod specific.

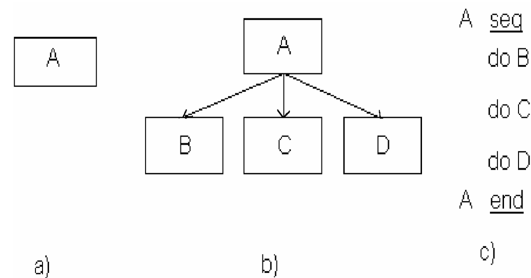


Figura 2.7 Reprezentarea secvenţei în metoda Jackson:

a) reprezentarea grafică a unei secvenţe simple;
b) reprezentarea grafică a unei secvenţe compuse;
c) reprezentarea în pseudocod a secvenţei.

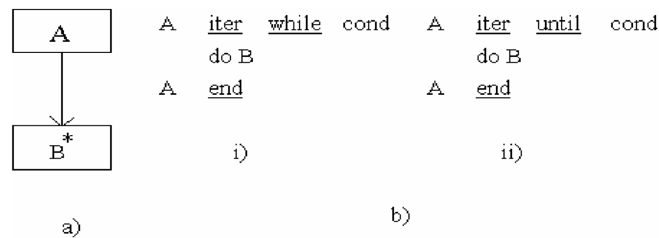


Figura 2.8. Reprezentarea iteraţiei în metoda Jackson:

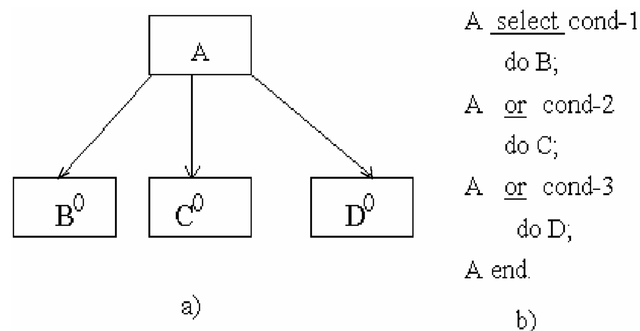


Figura 2.9. Reprezentarea selecţiei în metoda Jackson:

În ceea ce priveşte proiectarea, metoda Jackson este constructivă, în sensul că poate fi descompusă în paşi distincţi, executarea corectă a fiecărui pas trebuind să asigure executarea corectă a întregii metode, deci şi corectitudinea produsului software care rezultă prin folosirea metodei. Dificultatea majoră rezidă din faptul că trebuie ca execuția corectă a unui pas să fie verificabilă fără a face referiri la paşii care nu au fost executați de către proiectant. În forma cea mai simplă aplicarea metodei poate fi evidențiată în trei paşi:

- descrierea structurii datelor de intrare şi de ieşire sub forma unor diagrame de structură;
- compunerea structurii datelor într-o structură de program;
- listarea operațiilor executabile necesare şi introducerea lor în locul corect din structura programului.

2.1.4. Metode de proiectare conceptuală bazate pe structura programelor şi flux de date

Această metodă de proiectare a fost propusă de G. J. Myers şi este legată de proiectarea structurată, diferențele dintre cele două metode constând în reprezentarea proiectului. Atributele fundamentale ale unui program, conform proiectării compuse, sunt următoarele:

- *structura*, a cărei definire se realizează pe baza fluxului datelor;
- *funcția*, care descrie caracteristicile programului;
- *performanța*, care descrie modul în care programul realizează funcția, folosind ca mărimi: viteza de execuție, dimensiunea memoriei, utilizarea resurselor, fiabilitatea ş.a.m.d.

Proiectarea compusă referă structura programelor, exprimată prin module, date, interfețe între module, într-un proces de descompunere. Ca rezultat al acestei abordări se obțin programe de complexitate minimală, ceea ce determină îmbunătățirea fiabilității, mentenabilității şi adaptabilității programelor.

Aplicarea *principiului modularității* permite descompunerea unei aplicații în mai multe module puternic independente. Independența modulelor poate fi atinsă prin metode de optimizare, cum ar fi: *maximizarea relațiilor în interiorul fiecărui modul şi minimizarea relațiilor dintre module*.

Proiectarea compusă cuprinde un proces numit *analiză compusă*, care implică o analiză a structurii problemei şi a modului în care datele sunt transformate. Această informație determină descompunerea problemei într-un set de module, fiecare modul fiind considerat ca subproblemă, analiza fiind aceeaşi ca şi problema inițială. Analiza compusă este un proces de proiectare top-down. Fiecare modul are trei atribute de baza:

1. *Înțelegerea exactă a funcției unui modul* este un element cheie pentru înțelegerea proiectării compuse. De regulă, funcția se identifică cu transformarea intrărilor în ieşiri, care are loc la apelul modulului.
2. *Logica* se referă la controlul fluxului din interiorul modulului şi arată cum se realizează funcția la apelul modulului;
3. *Interfața.*

Consistența unui modul este o mărime folosită pentru a aprecia relația dintre elementele componente ale unui modul individual. A maximiza consistența unui modul înseamnă a organiza elementele componente în aşa fel încât elementele care sunt strâns legate să facă parte din acelaşi modul, iar elementele nelegate să facă parte din module distincte. în practică modulele pot avea una dintre următoarele tipuri de consistențe:

- *Consistența întâmplătoare* apare în cadrul unui modul atunci când nu există nici un fel de relație între elementele acestuia
- *Consistența logică* apare în cazul unui modul care, la fiecare apel, execută o funcție selectată din mai multe funcții similare.

- *Consistența clasică* este similară cu consistența logică. Un modul cu consistență clasică este un modul cu consistentă logică în care funcțiile componente se execută secvențial. Aceste funcții sunt controlate în timp.
- *Consistența procedurală* este similară cu consistența clasică; se execută mai multe funcții corelate, iar corelarea acestora este făcută în raport cu o procedură şi nu în raport cu o procedură a programului ca în cazul consistenței clasice. De exemplu, un modul care afişează graficul unei funcții x şi tipăreşte un raport y are o consistență procedurală, în sensul că cele două funcții nu sunt legate între ele, dar corelate prin faptul că în cadrul problemei care a stat la originea programului ele apar una după alta.
- *Consistența comunicațională* presupune construirea de module cu consistență procedurală în care funcțiile comunică între ele. Comunicarea poate fi de două tipuri:

1. toate funcțiile pot referi acelaşi set de date definite în cadrul modulului;
2. datele rezultate dintr-o funcție sunt folosite ca date de intrare pentru o altă funcție.

Cuplajul modulelor caracterizează mecanismul de transmitere a datelor şi a atributelor acestora între module. A maximiza independența modulelor înseamnă a reduce cuplajul dintre acestea. *Cuplajele* dintre module pot fi ordonate după cum urmează:

1.*Cuplajul prin conținut* apare între două module dacă modulul apelant referă direct elemente din cuprinsul modulului apelat.

2**.** *Cuplajul prin date* presupune a transmite ca parametrii între module doar date elementare.

Mărimile consistență şi cuplaj pot fi folosite pentru evaluarea unui proiect existent sau pentru a evalua diversele soluții de proiectare pentru o aplicație nouă. Sintetic, relația dintre cuplaj şi atribute (independența între module, susceptibilitatea la erori, reutilizarea modulelor în alte contexte, extensibilitatea aplicațiilor) poate fi prezentată astfel:

Cuplaj prin	**Independență față de erori**	**Susceptibilitate la aplicații**	**Reutilizare în alte module**	**Extensibilitate**
Date	mare	mică	mare	Mică
Structuri de date	medie	medie	medie	Medie
Date de control	medie	medie	medie	Medie
Date extreme	mică-medie	mare	mică-medie	Mică
Date comune	mică	mare	mică	Mică
Conținut	foarte mică	foarte mare	mică	Mică

Tabelul 2.1. Proprietățile cuplajului.

Pe lângă consistență şi cuplaj există şi alte mărimi, care alese convenabil, pot avea un efect pozitiv asupra independenței modulelor:

- **dimensiunea modulelor** poate îmbunătăți claritatea aplicațiilor şi poate elimina dificultățile testării acestora;
- **simplitatea soluției** care presupune proiectarea problemei de rezolvat fără aplicarea generalității;
- **recursivitatea** care simplifică logica modulelor;
- **predictibilitatea execuției**; un modul este predictibil dacă funcția lui este independentă de evoluția utilizării lui anterioare, ceea ce presupune că pentru intrări identice va opera identic de fiecare dată când este apelat. În proiectarea compusă acest element asigură independența modulelor de context.

- **structura de decizie**. Ori de câte ori execuția unor module este afectată de o decizie, este de dorit ca modulele afectate direct de această decizie să fie subordonate direct modulului care conține decizia. În acest fel se evită utilizarea unor parametrii speciali reprezentând decizii.
- **accesul la date** se cuantifică prin cantitatea potențială de date pe care un modul le poate referi. Această mărime trebuie minimizată în procesul de proiectare, astfel încât fiecare modul să nu poată referi decât datele de care are nevoie. Utilizarea modulelor cu consistență informațională și evitarea cuplajelor prin date comune, date externe și structuri de date constituie principalele căi prin care acest parametru poate fi minimizat.
- **izolarea operațiilor de intrare-ieşire** într-un număr mic de module este un obiectiv important al proiectării, în sensul că toate funcțiile primitive relative la intrări pot fi cuprinse într-un modul cu consistența informațională. Această strategie conduce la programe portabile şi extensibile.

Pe parcursul procesului de proiectare, în metoda proiectării compuse, se folosesc trei tipuri de diagrame pentru reprezentarea proiectului:

1. Diagrame de flux de date sunt reprezentări grafice neprocedurale ale transformărilor pe care le suferă datele în structura problemei. O astfel de diagramă este compusă din săgeți şi cercuri. Săgețile definesc transferuri de date între procesele de transformare, iar cercurile reprezintă transformările pe care le suferă datele în structura problemei (figura 2.10.).

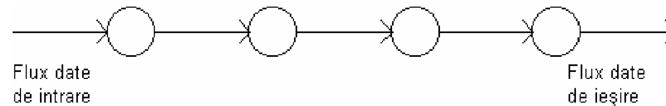


Figura 2.10. Diagrame pentru urmărirea fluxului datelor.

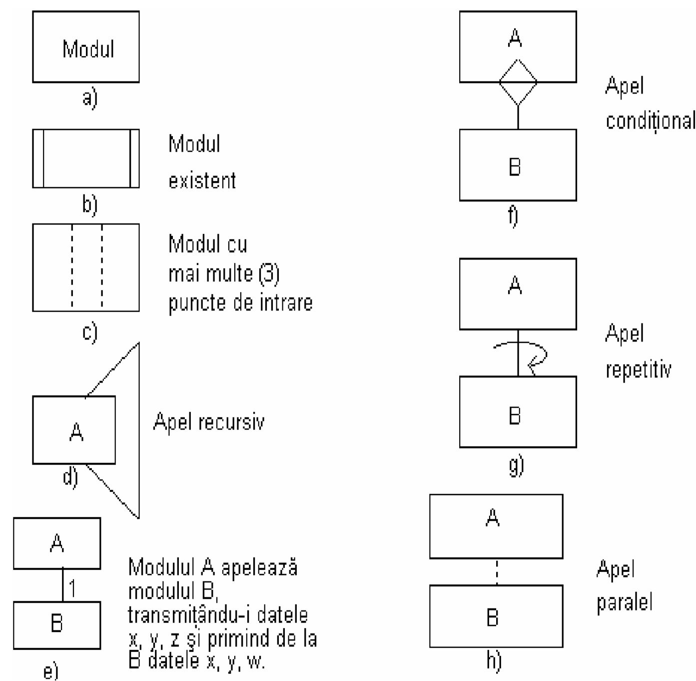


Figura 2.11. Simboluri pentru reprezentarea diagramelor de structură.

2. Diagrame cu structura modulelor se definesc pe baza diagramelor fluxurilor de date, fiecare modul având o funcție care descrie transformările ce au loc atunci când acesta este apelat. Pentru construirea acestor diagrame se folosesc simbolurile din figura 2.11.

3. Diagrame ce definesc interfețele dintre module sunt tabele în care fiecare intrare corespunde unei interfețe din diagrama de structură asociată. Fiecare intrare cuprinde două

părți: o parte care definește datele care sunt transmise modulului apelat de către modulul apelant și o parte pentru datele transmise modulului apelant de către modulul apelat după terminarea funcției acestuia (figura 2.12.).

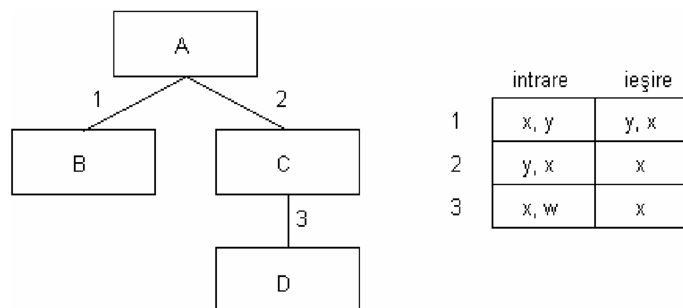


Figura 2.12. Diagrame pentru definirea interfețelor.

Procesul prin care, plecând de la structura unei probleme se ajunge la structura unei aplicații sau sistem informatic, se numește analiză compusă. Acest proces presupune analiza structurii problemei și mai exact a modulului în care datele sunt transformate. Această informație este folosită pentru descompunerea problemei într-un set de module, funcția unui modul corespunzând unei transformări pe fluxul datelor. Fiecare modul este considerat apoi ca o problemă de sine stătătoare, analiza fiind repetată după aceleași principii.

2.1.5 Metode bazate pe abstracții

Procesul de abstractizare joacă un rol important în cadrul activității de realizare a aplicațiilor, constituind unul dintre cele mai însemnate mijloace pentru controlul complexității programelor. Se poate spune că tehnica programării structurate are ca bază procesul de abstractizare.

Elaborarea proiectului unui program complex nu se poate realiza dintr-o dată. De regulă, se procedează la prezentarea problemei printr-un *enunț general.* Prin abstractizare și descompunere în pași succesivi se ajunge la *enunțul de detaliu*, adică la date și operații care pot fi descrise direct cu ajutorul unui limbaj de programare. Dacă notăm cu N nivelul limbajului putem spune că el permite programatorului să descrie problema pentru o mașină abstractă M care va fi simulată pe o mașină reală R. Noțiunea de abstractizare se găsește sub două forme:

- *forma implicită,* bazată pe elemente predefinite în cadrul limbajelor, cum ar fi: masivele, structurile de control if-then-else ș.a.m.d., care uneori constituie bariere pentru procesul de abstractizare;
- *forma explicită*, bazată pe elemente introduse de proiectant, folosind mecanisme speciale, numite și mecanisme de extensie. Această formă duce la o creștere considerabilă a productivității muncii de proiectare și programare.

Prin asocierea la un limbaj a unui mecanism de extensie se pot defini tipuri noi de enunțuri și de date specifice domeniului de aplicație, prin urmare noi obiecte abstracte. Ca urmare a aplicării conceptului de abstractizare se ajunge la definirea unei clase de *obiecte abstracte* caracterizată în mod complet de operațiile posibile asupra acestora.

Procesul de abstractizare, aplicat la nivel funcțional se manifestă în mod curent în activitatea de programare prin utilizarea conceptelor de procedură sau subrutină. Procedura poate fi considerată ca o "cutie neagră" ce realizează o funcție abstractă. Prin abstractizare se separă ceea ce se face, de cum se face. În programul apelant este suficient să se știe ce se face. Prin subrutinele imbricate se poate dezvolta o ierarhie de abstracții.

La nivelul funcţional procesul de abstractizare este pus în valoare de tehnicile de proiectare top-down. Astfel, în cadrul elaborării prin rafinare în paşi succesivi ai programului se face abstracţie de modul în care este realizată o anumită funcţie sau acţiune şi se reţin numai specificaţiile efectelor sale.

Ceea ce se reţine de la început este îndeosebi determinarea comportamentului unei proceduri numai în funcţie de efectele sale externe. Abstractizarea presupune delimitarea unei ierarhii de nivele, rafinarea sau detalierea acestora făcându-se până la nivelul cel mai de jos, respectiv nivelul operaţiilor executabile de către maşină. Important este ca abstractizarea funcţională să pună cât mai bine în relaţie aspectele funcţionale cu cele procedurale şi să asigure eficienţa în dialogul utilizator - informatician pe linia elaborării specificaţiilor şi programelor.

Actualele limbaje de definire a specificaţiilor sunt rezultate ca urmare a preocupărilor pe linia extinderii abstractizării în programare.

Subrutinele sunt foarte potrivite pentru a descrie funcţii, operaţii sau evenimente abstracte, dar nu convin pentru descrierea obiectelor abstracte. Ca urmare a faptului că aplicaţiile manipulează o cantitate apreciabilă de date, aceasta contribuie la creşterea complexităţii algoritmilor. Noţiunea de dată abstractă se introduce pentru a facilita procesul proiectării sau pentru a asigura generalitatea algoritmilor.

2.2. Proiectarea schemei externe

Schema externă a BD reprezintă forma sub care apare schema conceptuală pentru un utilizator oarecare. Programele de aplicaţie 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 totuşi de tipul de SGBD utilizat.

În concepţia CODASYL, schema externă reprezintă o parte a schemei conceptuale. Articolele care compun grupurile şi înregistrările din schema externă pot diferi de articolele care compun grupurile şi înregistrările din schema conceptuală.

În cazul BDR, schema externă este realizată, în principal cu ajutorul viziunilor (view-urilor) şi al mecanismelor de acordare a drepturilor de acces la BD.

2.3. Proiectarea schemei interne

Este cunoscut faptul că schema conceptuală încarcă diferite forme de structurare a datelor: liniară, arborescentă, reţea, relaţională.

Memorarea datelor pe suportul fizic îmbracă numai forma unei structuri liniare. De aceea, la proiectarea schemei interne a BD se pune problema modului în care să fie liniarizată schema conceptuală.

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

BAZE DE DATE

CURS 3

PROIECTAREA LOGICĂ A BAZELOR DE DATE

3.1 Introducere
3.1.1 Clasificarea generală a modelelor de date
3.1.2 Modele logice orientate pe înregistrări
3.1.3 Scheme şi instante
3.2 Modele logice de date
3.2.1 Obiectivele modelarii logice
3.2.2 Utilizarea modelarii logice a datelor
3.2.3 Caracteristicile tehnicilor de modelare logică a datelor
3.3 Tehnici de modelare a datelor. Modelul Entitate-Asociere
3.3.1 Entitate, atribut şi asociere
3.3.2 Modelul entitate-asociere
3.3.3 Modelul entitate-asociere extins

3.1. Introducere

3.1.1. Clasificarea generală a modelelor de date

Modelarea este o tehnică importantă pentru a defini cerinţele de a analiza proiectările alternative. Un model reprezintă caracteristici de interes particular suprimând detalii considerate relativ neimportante. Diferitele tehnici de modelare se concentrează asupra diferitelor tipuri de caracteristici ale unui sistem, de exemplu asupra datelor, sau asupra proceselor sau asupra dinamicii.

Modele de date logice. Un model de date logice este o reprezentare riguroasă a semnificaţiei datelor într-un anumit domeniu de interes. Se mai numeşte model semantic de date deoarece accentul modelului cade pe semnificaţia datelor (i.e. semantică). Un model de date logice tipic reprezintă entităţi, atribute şi relaţii. O entitate este ceva care există în lumea reală, un concept, o persoană, un loc, un lucru sau un eveniment despre care noi vrem să păstrăm fapte. Un atribut este o proprietate sau o caracteristică a unei entităţi. Un model de date logice este reprezentat tipic folosind o tehnologie grafică. De exemplu, cutiile pot reprezenta entităţi iar săgeţile pot reprezenta relaţii între entităţi.

Modele de date fizice. Modelele de fizice reprezintă implementarea structurilor de date şi vizează optimizarea resurselor, cum ar fi spaţiul de memorie şi timpii de acces. Elementele lor includ, în general, înregistrări fizice, fişiere, adrese şi pointeri.

Modele de activitate. Modelele de date logice şi fizice pot fi contrastate cu modele de activitate (numite şi modele funcţie sau modele proces) care reprezintă procese (mai de grabă decât date) şi relaţiile lor. Aceste modele se reprezintă tipic printr-o notaţie grafică în care cutiile (sau cercurile) reprezintă activităţi iar arcele între cutii reprezintă fluxul activităţilor.

Un model de actvitate se focalizează pe procese mai mult decât pe fluxul între procese. Modelele de date, pe de altă parte, se focalizează pe semnificaţiile entităţilor, atributelor şi a relaţiilor de date. Tehnicile de modelare a datelor reprezintă, uzual activitităţile care folosesc datele. Dacă nu indicăm altfel, vom folosi termenul de model de date pentru a referi un model de date logic (mai de grabă decât model de date fizic) ceea ce este consistent cu terminologia general folosită în domeniul bazelor de date.

3.1.2. Modele logice orientate pe înregistrări

Modelele logice orientate pe înregistrări descriu datele la nivel logic şi extern. Sunt utilizate trei modele de acest tip:

· **modelul relaţional** - datele şi legăturile care le unesc sunt organizate sub forma unui tabel. Ilustrăm acest model luând, ca exemplu, baza de date Conturi-Clienţi din figura 3.1:

Client			
NUME	**STRADA**	**ORAS**	**NUMAR**
Lupu I.	Oltului 2	Craiova	2020
Ursu D.	Jiului 17	Timisoara	3030
Ursu D.	Bega 22	Timisoara	4040
Popa C.	Dunării 5	Bucuresti	5050
Popa C.	Someş 10	Cluj	6060

Cont	
NUMAR	**POZITIE**
2020	300
3030	1300
4040	1000
5050	2000
6060	600

Figura 3.1. Relaţiile client şi cont

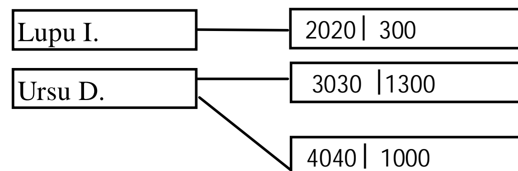


Figura 3.2. Exemple de legături

· **modelul reţea** - datele sunt organizate în reţea prin înregistrări, iar relaţiile dintre ele sunt reprezentate prin legături, care pot fi considerate ca pointeri. Înregistrările sunt afişate ca o colecţie de grafuri oarecare. O structură reţea este dată în figura 3.2.

· **modelul ierarhic** - datele şi relaţiile sunt reprezentate prin înregistrări şi legături. Înregistrările sunt organizate sub formă de arbori.

3.1.3. Scheme şi instanţe

În orice model este important să se facă o distincţie între descrierea bazei de date şi baza de date însăşi.

Descrierea bazei de date formează schema bazei de date. Schema bazei de date este specificată în timpul proiectării BD şi nu se modifică niciodată. Cele mai multe modele cuprind anumite convenţii şi metareguli pe baza cărora se construiesc schemele bazei de date, care sunt numite *diagrame schemă*.

Fiecare element din diagrama schemei se numeşte *constructor de schemă* sau *componentă constructivă*. Conceptul de schemă dintr-o bază de date corespunde noţiunii de „

declarație de tip" dintr-un limbaj de programare. Când o variabilă de un tip dat ia o anumită valoare la un moment dat se spune că ea este o *instanțiere*.

În figura 3.3 se dă diagrama schemei unei BD cu patru instanțe.

O diagramă a schemei cuprinde numai anumite aspecte ale schemei: numele fişierului, elemente de date şi anumite tipuri de restricții. Totuşi, diagrama nu specifică tipul fiecărui element de date şi alte restricții, de exemplu, că un student din anul I trebuie să urmeze cursul de algoritmi. Datele actuale dintr-o bază de date pot fi frecvent schimbate, de exemplu, pentru baza de date din figura 3.3 se adaugă noi note.

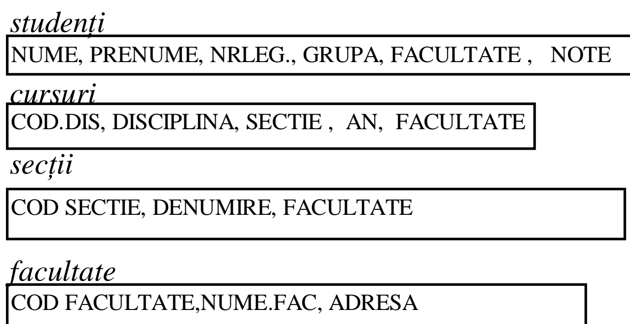


Figura 3.3. Diagrama schemei BD studenți

Datele dintr-o bază de date la un anumit moment de timp formează o *instanță* a bazei (*stare*, *ocurență*). O mulțime de instanțe a bazei corespunde la o *schemă a bazei de date*.

În orice moment când vom insera, modifica valori sau şterge înregistrări vom schimba *instanța* bazei de date într-o altă instanță a bazei de date. Când definim o nouă bază de date, vom specifica schema bazei de date. În acest moment instanța bazei de date este vidă (nu are nici o altă dată). Datele încărcate prima dată în baza de date constituie *instanța inițială*.

3.2. Modele logice de date

3.2.1 Obiectivele modelării logice

Modelele de date au două scopuri principale: de a facilita comunicațiile despre date şi de a ajuta în descoperirea semanticii datelor.

Comunicarea semnificației datelor. Doarece ele sunt reprezentări riguroase, modelele de date pot fi folosite pentru a transmite unei alte persoane înțelegerea semnificației datelor. Atâta timp cât persoanele ce comunică sunt familiare cu notația folosită în model, comunicarea va fi înțeleasă.

Descoperirea semanticii datelor. Construirea unui model de date cere a se răspunde la întrebări despre entități şi relații. Făcând aşa, modelatorii descoperă semnificația datelor organizației, care există indiferent dacă se întâmplă sau nu ca ele să fie înregistrate într-un model formal de date. Entitățile şi relațiile lor sunt fundamentale pentru toate organizațiile; totuşi, semnificația datelor poate rămâne nedescoperită (şi probabil prost înțeleasă) până când organizația nu le documentează cu succes.

3.2.2 Utilizarea modelării logice de date

Facilitând comunicațiile şi descoperind semantica datelor, modelarea datelor este utilă pentru:

- creşterea înțelegerii datelor unei organizații şi a operațiilor asupra lor
- documentarea resurselor de date
- folosirea pachetelor software

- proiectarea sistemelor informatice
- integrarea resurselor de date
- proiectarea şi implementarea bazelor de date

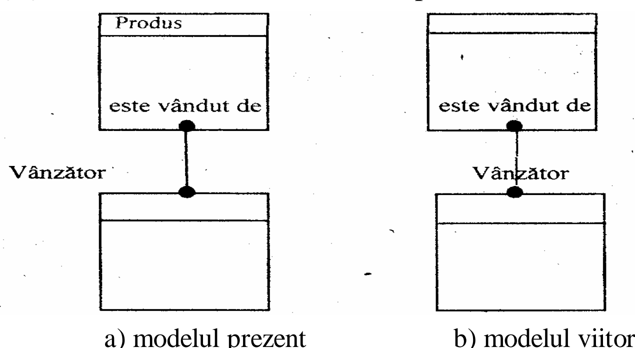
Înțelegerea datelor. Atunci când se dezvoltă modele de date, atât personalul de prelucrare date cât şi utilizatorii care sunt experți în domeniul modelului, sunt necesari. Un model de date construit de personalul care prelucrează datei şi făcut pentru a portretiza date despre produse este diferit de unul făcut de ingineri, producători şi distribuitori. De fapt, modelul de date construit numai de personalul de prelucrare date este incorect şi/sau inexact. Astfel utilizatorii sunt cruciali pentru dezvoltarea cu succes a modelelor de date.

Documentarea datelor. Un model de date este adesea un bun mod de a explica documentat datele. Dacă tehnica de modelare este capabilă şi clară, atunci modelele sale pot fi considerabil mai valabile ca şi documentare decât limbajele naturale.

Evaluarea software-ului. Pe măsură ce pachetele software devin mai populare, companiile se întreabă tot mai des: " Se va potrivi acest software în mediul nostru?". Răspunsul depinde de ce face software-ul şi de modelul de date rezultat. De exemplu, un pachet de contabilitate care presupune că un angajat este plătit dintr-un singur cont (al proiectului la care lucrează) nu se va potrivi bine într-o companie ai cărei angajați sunt plătiți din mai multe conturi.

Planificarea sistemelor informatice. Deoarece facilitează comunicațiile şi descoperirea, modele de date sunt instrumente puternice pentru planificarea sistemelor informatice şi a bazelor de date. Modelele de date sunt folosite în **BSP** (Business Systems Planning - IBM), **SPM** (Strategic Planning Methodology - Janus Martin) şi **RAP** (Requirements Analysis and Planning - DACOM) pentru a identifica ariile principale de date ale unei întreprinderi.

Proiectarea sistemelor informatice. Un proiect de implementare poate adăuga la un model de planificare date de nivel înalt detalii despre atribute, volume şi frecvențe de acces pentru datele din domeniul său. Modelul detaliat rezultat poate fi transformat într-o proiectare de bază de date fizică. *Modelarea datelor poate fi folosită efectiv pentru a documenta mediul existent de date şi de a ajuta la proiectarea viitorului mediu de date.* Modelarea datelor folosită pentru a captura mediul prezent este un proces de descoperire, iar modelarea datelor folosită pentru a proiecta viitorul mediu este o invenție. De exemplu, Figura 3.4. ilustrează în mod simplu modelele prezent şi viitor. Fără a explica tehnica de diagramare în acest punct, figura scoate în evidență un mediu nou descoperit în care un distribuitor vinde numai un singur tip de produs (de ex. maşini de scris) şi un mediu în care responsabilitățile distribuitorului au fost expandate pentru a include mai multe tipuri de produse (de ex. maşini de scris, procesoare de texte, calculatoare). În general, un model de date pentru viitor nu este o pură invenție ci el împărtăşeşte mult din modelul rădăcină al prezentului.



a) modelul prezent b) modelul viitor

Figura 3.4. Modelele prezent şi viitor

Integrarea resurselor de date. Modelele de date pot ajuta, de asemenea, la integritatea resursei de date şi sunt folosite pentru a reprezenta schemele conceptuale. Dacă

modelele produse de către proiectele de implementare folosesc o sintaxă consistentă şi clară, atunci aceste modele împreună dau o vedere de ansamblu despre modul în care se potrivesc datele lor împreună, iar inconsistentele şi redundanțele pot fi identificate şi rezolvate.

Modelul de date integrate dezvoltat de către o întreprindere este schema sa conceptuală. Această vedere neutră de date poate fi mapată pe diferite structuri de baze de date fizice (scheme interne), aşa cum sunt ele implementate şi pe vederi utilizator (scheme externe). Poate fi dificil a integra resursele de date fără a folosi modele de date. Structurile fizice, de exemplu, nu prezintă o imagine completă.

Susținerea proiectării bazelor de date fizice. Un model de date dă proiectantului de baze de date fizice informații despre ce date ar trebui incluse în baza de date, ce tipuri de relații structurează baza de date şi cum se leagă baza de date de alte baze de date. A construi un model de date înseamnă în primul rând a introduce date corecte în baza de date. Tehnicile de proiectare bază de date fizice se folosesc apoi pentru a asigura un acces eficient la date.

3.2.3. Caracteristicile tehnicilor de modelare logică a datelor

O tehnică de modelare logică a datelor trebuie:

- să producă diagrame grafice;
- să ofere o reprezentare explicită a semanticii;
- să aibă un nivel potrivit de detaliere;
- să fie independentă de Sistemul de Gestiune a Bazei de Date (SGBD);
- să aibă un suport automatizat;
- să fie uşor de învățat şi folosit.

Diagrame grafice. Majoritatea tehnicilor de modelare de date aflate în folosință astăzi oferă atât un limbaj de modelare (cu o sintaxa bine definită, cuvinte cheie, etc.) cât şi diagrame grafice ale modelelor. Pictogramele constau din dreptunghiuri sau cercuri pentru a reprezenta entități şi din arce sau linii pentru a reprezenta relații. Unele tehnici folosesc forme diferite pentru categorii diverse de entități: dreptunghiuri şi dreptunghiuri rotunjite pot fi folosite în acelaşi model pentru a distinge între entități independente de identificator şi entități dependente de alte entități.

Anumite tehnici folosesc convenții distincte pentru linii pentru diferite tipuri de relații; de exemplu, linii unice pentru grupări şi linii duble pentru relații "*...este un fel de...*".
Alte tehnici folosesc un simbol special, ca de exemplu un cerc, pentru a arăta atributele.

Reprezentarea explicită a semanticii. Există un domeniu de preferință personală pentru a reprezenta grafic cât mai bine semantica datelor. Anumite persoane preferă o mulțime de simboluri diferite, fiecare cu o semnificație specială, pe când alții preferă un munăr mai mic de simboluri, care fac ca diagramele modelului să apară mai simple. Tehnica ideală de modelare de date produce diagrame model care reprezintă clar semnificațiile datelor. Diagramele pot să fie neschimbătoare, dar un simbol nu poate avea mai multe semnificații diferite.

Nivelul potrivit de detaliere. O tehnică de modelare trebuie să ofere detalii la nivelul potrivit pentru folosința utilizatorului. Modelele de date pentru planificarea sistemelor informatice de nivel înalt (la nivelul companiei sau diviziei) trebuie să păstreze mai puține detalii decât modelele pentru proiectarea bazelor de date fizice. Tehnicile de modelare de date cele mai flexibile suportă diferite nivele de detalii. De exemplu, ele pot fi folosite pentru a construi modele care arată numai entitățile şi relațiile de nivel înalt, dar şi modele care arată toate entitățile, atributele detaliate şi relațiile rafinate.

Independența SGBD-ului. O tehnică de modelare de date trebuie să fie independentă de orice SGBD particular şi trebuie să fie capabilă să reprezinte modele care pot fi suportate de o varietate de SGBD-uri. Multe din tehnicile de modelare a datelor sunt independente de SGBD. Uneori, tehnicile de modelare a datelor folosite cu un SGBD particular sunt considerate de nivel general, aşa cum sunt modelul ierarhic, modelul rețea şi modelul relațional.

Problema majoră a acestor trei tehnici este existența de limitări în semantica datelor pe care le pot reprezenta.

Suport automatizat. Tehnica de modelare de date ideală are suport automatizat pentru a desena diagrame, a găsi inconsistențe şi redondanță în modele, a combina modele din mai multe surse; a raporta pe model componente şi caracteristici, şi aşa mai departe. A avea suport automatizat înseamnă, în general, a oferi atât un limbaj pentru modele specificate cât şi a oferi interfețe utilizator bazate pe grafică. Tehnicile de modelare de date aflate astăzi în folosință sunt parțial automatizate.

3.3. Tehnici de modelare a datelor. Modelul Entitate-Asociere.

3.3.1 Entitate, atribut şi asociere

Un model este o abstractizare a unui sistem, care captează cele mai importante trăsături caracteristice ale sistemului (concepte), relevante din punct de vedere al scopului pentru care se defineşte modelul respectiv. Tehnica de identificare a trăsăturilor caracteristice esențiale ale unui sistem se numeşte *abstractizare*.

Un model de date stabileşte regulile de organizare şi interpretare a unei colecții de date. În proiectarea bazelor de date se folosesc, de regulă, mai multe modele de date, care se pot clasifica în două categorii: *modele conceptuale de nivel înalt* şi *modele specializate*.

Un model conceptual de nivel înalt al datelor conține o descriere concisă a colecțiilor de date care modelează activitatea dorită (numită *schemă conceptuală de nivel înalt*), fără să detalieze modul de reprezentare sau de prelucrare a datelor.

Modelele specializate de date (cum sunt: modelul ierarhic, modelul rețea, modelul relațional, etc.) impun anumite structuri speciale de reprezentare a mulțimilor de entități şi a asocierilor dintre acestea, structuri pe baza cărora sunt dezvoltate sistemele de gestiune a bazelor de date. Într-un astfel de model de date, o bază de date este reprezentată printr-o schemă conceptuală (logică) specifică. Trecerea de la modelul conceptual de nivel înalt la un model de date specific reprezintă etapa de *proiectare logică a bazei de date* care asigură corespondența dintre schema conceptuală de nivel înalt a bazei de date şi schema conceptuală specifică modelului de date respectiv.

Cel mai utilizat model conceptual de nivel înalt este modelul Entitate-Asociere (E-A) care reprezintă schema conceptuală de nivel înalt a bazei de date prin mulțimi de entități şi asocieri dintre acestea. Dezvoltarea acestui model, astfel încât să permită extinderea tipurilor de entități, este cunosută sub numele de model Entitate-Asociere Extins (E-AE). Proiectarea modelului E-A sau al modelului E-AE este, de regulă, una din primele etape în proiectarea bazelor de date, etapă numită *proiectarea schemei conceptuale*

Modelul *Entitate-Asociere* (Entity-Relationship Model), introdus în 1976 de P.S. Chen, este un model conceptual de nivel înalt al unei baze de date, care defineşte mulțimile de entități şi asocierile dintre ele, dar nu impune nici un mod specific de structurare şi prelucrare (gestiune) a datelor.

Elementele esențiale ale modelului Entitate-Asociere folosit în proiectarea bazelor de date sunt *entitățile* (*entities*) şi *asocierile* dintre acestea (*relationships*).

O entitate (*entity*) este „orice poate fi identificat în mod distinctiv"; o entitate se referă la un aspect al realității obiective care poate fi deosebit de restul universului şi poate reprezenta un obiect fizic, o activitate, un concept abstract, etc. Ea se identifică în mod unic printr-un nume. Orice entitate trebuie definită fără ambiguități şi este descrisă şi identificată în mod unic prin *atributele* sale.

Un atribut (attribute) este o proprietate care descrie un anumit aspect al unei entități.

Atributele prin care este descrisă o entitate se aleg pe baza criteriului *relevanței* relativ la domeniul de interes pentru care se defineşte modelul respectiv, astfel încât să asigure diferențierea acelei entități față de restul universului.

Toate entitățile similare, care pot fi descrise prin aceleaşi atribute, aparțin unui acelaşi *tip de entitate* (*entity type*), iar colecția tuturor entităților de acelaşi tip dintr-o bază de date constituie o *mulțime de entități* (*entities set*). În general, în modelul E-A se foloseşte aceeaşi denumire atât pentru un *tip de entitate* cât şi pentru *mulțimea entităților* de acel tip.

De exemplu, tipul de entitate „angajat" (al unei instituții) reprezintă orice persoană angajată a instituției, care are o anumită funcție şi primeşte un anumit salariu. Acest tip de entitate poate fi descris prin mai multe atribute, dintre care o parte sunt atribute de identificare a persoanei (Nume, Prenume, DataNasterii, Adresa), iar altele sunt atribute legate de activitatea acesteia în instituția respectivă (Functie,Salariu).

În proiectarea bazelor de date se consideră două categorii de entități: entități normale (puternice, obişnuite - *regular entities*) şi entități slabe (dependente - *weak entities*).

Entitățile normale au o existență proprie în cadrul modelului, în timp ce entitățile slabe nu pot exista decât dacă există o entitate normală (puternică) cu care sunt asociate. De exemplu, o entitate „dependent" poate să reprezinte o persoană care depinde de un angajat al unei instituții (adică se află în întreținerea acestuia). O entitate „angajat" este o entitate puternică, deoarece ea există în mod normal în modelul activității instituției, în timp ce o entitate "dependent" este o entitate slabă: nu se va înregistra o astfel de persoană decât dacă părintele (susținătorul) acesteia este angajat în acea instituție.

În proiectarea bazelor de date se definesc *asocieri* între mulțimile de entități componente, pentru a reprezenta anumite aspecte ale realității pe care baza de date o modelează.

O asociere (*relationship*) este o corespondență între entități din două sau mai multe mulțimi de entități.

Gradul unei asocieri este dat de numărul de mulțimi de entități asociate. Asocierile pot fi binare (de gradul 2, între două mulțimi de entități) sau multiple (între k mulțimi de entități, $k > 2$).

Asocierile binare sunt, la rândul lor, de trei categorii, după numărul elementelor din fiecare dintre cele două mulțimi puse în corespondență de asocierea respectivă (fig. 3.5). Fiind date două mulțimi de entități, E_1 şi E_2, se definesc următoarele categorii de asocieri binare:

• *Asocierea "unul-la-unul"* (*one-to-one*) este asocierea prin care unui element (entitate) din mulțimea E_1 îi corespunde un singur element din mulțimea E_2, şi reciproc; se notează cu 1:1.

• *Asocierea „unul-la-multe"* (*one-to-many*) este asocierea prin care unui element din mulțimea E_1 îi corespund unul sau mai multe elemente din mulțimea E_2, dar unui element din E_2 îi corespunde un singur element în mulțimea E_1; se notează cu 1:*N*.

• *Asocierea „multe-la-multe"* (*many-to-many*) este asocierea prin care unui element din mulțimea E_1 îi corespund unul sau mai multe elemente din mulțimea E_2 şi reciproc; se notează cu *M*:*N*.

Cardinalitatea (multiplicitatea) unei asocieri față de o mulțime de entități (*cardinality, multiplicity*) *este numărul maxim de elemente din acea mulțime care pot fi asociate cu un element din altă mulțime a asocierii.*

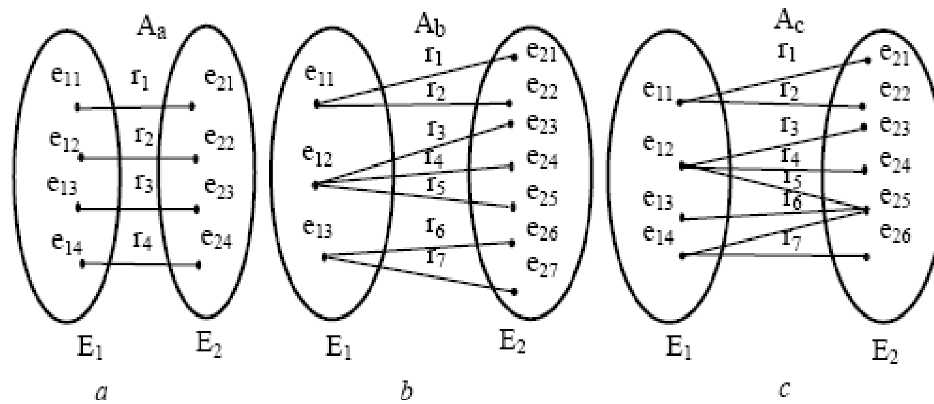


Fig. 3.5. Categorii de asocieri între două mulțimi de entități:
a - asociere 1:1; *b* - asociere 1:*N*; *c*- asociere *M*:*N*.

De exemplu, asocierea 1:*N* dintre mulțimile E_1 și E_2 prezintă multiplicitatea 1 față de mulțimea E_1 și multiplicitatea *N* (se înțelege o valoare oarecare $N > 1$) față de mulțimea E_2. Raportul dintre valorile cardinalităților unei asocieri binare față de cele două mulțimi de entități se numeşte raport de cardinalitate (*cardinality ratio*). Se poate observa că cele trei categorii de asocieri descrise mai sus diferă între ele prin raportul de cardinalitate.

Asocierile multiple (*k*-are, $k > 2$) prezintă câte un raport de cardinalitate pentru fiecare pereche de mulțimi de entități pe care le asociază.

O asociere între două sau mai multe mulțimi de entități este, în acelaşi timp, o asociere între tipurile de entități corespunzătoare.

3.3.2 Modelul Entitate-Asociere

Diagrama Entitate-Asociere (*Entity-Relationship Diagram*) reprezintă modelul Entitate-Asociere prin mulțimile de entități şi asocierile dintre acestea.

Există numeroase variante de notații pentru redarea diagramei *E-A*. Una dintre cele mai folosite notații reprezintă un tip de entitate (precum şi mulțimea de entități de acel tip) printr-un dreptunghi, iar atributele tipului de entitate prin elipse conectate printr-o linie continuă la acesta (fig. 3.6). Pentru entitățile puternice se utilizează un dreptunghi încadrat cu o linie simplă, iar pentru entitățile slabe se utilizează un dreptunghi încadrat cu linie dublă.

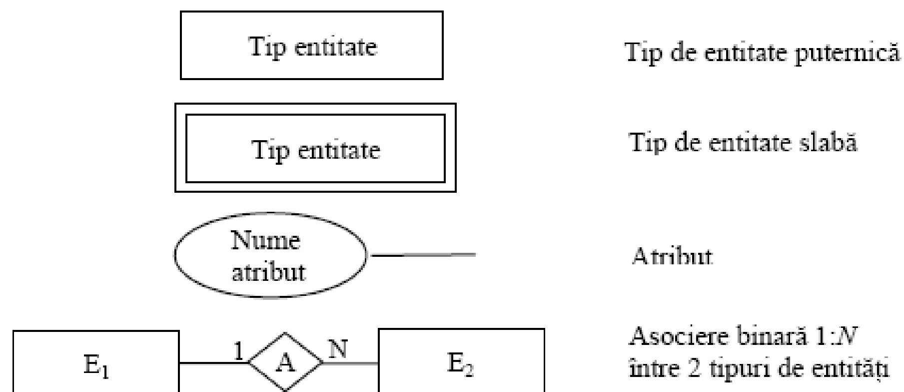


Fig. 3.6. Notațiile diagramei Entitate-Asociere (*E-A*).

O asociere (tip de asociere) dintre două sau mai multe tipuri de entități se reprezintă printr-un romb conectat prin link-uri (linii continue, formate din unul sau mai multe segmente)

la tipurile de entități asociate. O asociere poate să aibă sau nu un nume; dacă are un nume, acesta poate fi înscris în rombul respectiv sau în vecinătatea acestuia. Categoria asocierii se notează prin înscrierea multiplicității pe fiecare link care conduce la un tip de entitate. Este posibil ca o asociere să prezinte ea însăşi atribute, şi aceste atribute se reprezintă prin elipse conectate la asocierea respectivă.

Exemplul 3.1. În continuare se exemplifică dezvoltarea modelului conceptual de nivel înalt al unei baze de date a unei instituții şi diagrama *E-A* corespunzatoare, definind câteva tipuri de entități şi asocierile între acestea. Diagrama *E-A* a acestui mic model de bază de date este prezentată în figura. 3.7

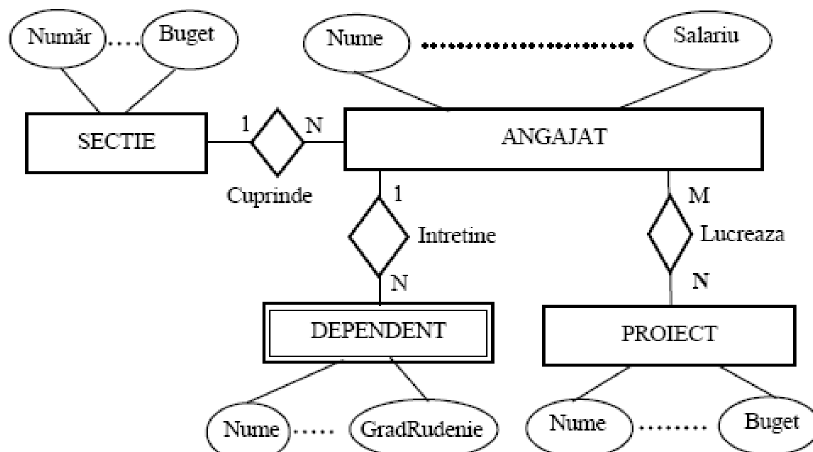


Fig. 3.7. Exemplu de diagramă *E-A*.

Tipul de entitate „secție" reprezintă o unitate de organizare a instituției şi este un tip de entitate puternică (de sine stătătoare). Acest tip se caracterizează prin mai multe atribute, de exemplu, un număr al secției, numele secției şi bugetul alocat. Mulțimea de entități care grupează toate entitățile de acest tip se poate denumi SECTIE sau SECTII (oricare variantă poate fi folosită) şi este caracterizată prin aceleaşi atribute care caracterizează tipul entității:

SECTIE(Numar, Nume,Buget)

Tipul de entitate „angajat" reprezintă o persoană angajată în instituție şi este de asemenea un tip de entitate puternică. Acest tip de entitate, ca şi mulțimea de entități care grupează toate entitățile de acest tip, se poate defini astfel

ANGAJAT(Nume, Prenume, DataNasterii, Adresa, Functie, Salariu)

Tipul de entitate „proiect" reprezintă o activitate a instituției, şi este de asemenea un tip de entitate puternică, care poate fi caracterizat prin numele proiectului, data începerii, termen de realizare, bugetul proiectului:

PROIECT(Nume, DataInceperii, Termen,Buget)

Tipul de entitate „dependent" reprezintă o persoană care depinde de un angajat al instituției (adică se află în întreținerea acestuia). Acest tip de entitate este un tip de entitate slabă: nu se va înregistra o astfel de persoană decât dacă întreținătorul acesteia este angajat în acea instituție. Acest tip se poate defini astfel:

DEPENDENT(Nume, Prenume, DataNasterii, GradRudenie)

Asocierea SECTIE-ANGAJAT este o asociere 1:*N*, dacă se consideră că o secție *cuprinde* mai mulți angajați, iar un angajat aparține unei singure secții.

Asocierea ANGAJAT-PROIECT este o asociere *M*:*N*, dacă se consideră că la fiecare proiect *lucrează* mai mulți angajați, şi fiecare angajat poate lucra la mai multe proiecte.

Asocierea ANGAJAT-DEPENDENT este o asociere de tipul 1:*N*, deoarece un angajat poate *întreține* mai multe persoane (fii, părinți etc.), iar o persoană dependentă este în întreținerea unui singur susținător.

Raportul de cardinalitate al unei asocieri este stabilit de proiectant astfel încât să reflecte cât mai corect modul de organizare a activității modelate. De exemplu, asocierea ANGAJATI-PROIECTE are raportul de cardinalitate *M*:*N* dacă în instituția respectivă se admite ca un angajat să lucreze la mai multe proiecte; dacă s-ar impune ca un angajat să lucreze la un singur proiect, atunci asocierea respectivă ar avea raportul de cardinalitate *N*:1. În ambele situații se admite că la un proiect lucrează mai mulți angajați.

Sunt de remarcat câteva **caracteristici generale** ale modelului *E-A*:

a) Modul de stabilire a tipurilor de entități şi a asocierilor dintre acestea nu este unic, deoarece granița dintre entități şi asocieri nu este, în general, una bine precizată. Aceeaşi funcționalitate se poate obține printr-o varietate de diagrame E-A, depinzând de felul în care proiectantul dezvoltă modelul conceptual. O asociere poate fi considerată şi ca un tip de entitate. De exemplu, pentru baza de date a unei facultăți (şcoli) se definesc tipurile (mulțimile) de entități:

STUDENTI(Nume, Prenume, Adresa,...)
DISCIPLINE(Denumire, Credite,...)

Între aceste mulțimi de entități se poate defini asocierea STUDENTI-DISCIPLINE, cu raportul de cardinalitate *M*:*N*. Această asociere reprezintă (în general) studierea unor discipline de către studenți, cu atribute ca: Nota (examenului la disciplina respectivă), DataExamen, etc. Dar, la fel de bine, este posibil să se definească tipul de entitate NOTE, aflat în asociere N:1 cu fiecare din tipurile de entități STUDENTI şi DISCIPLINE (fig. 3.8).

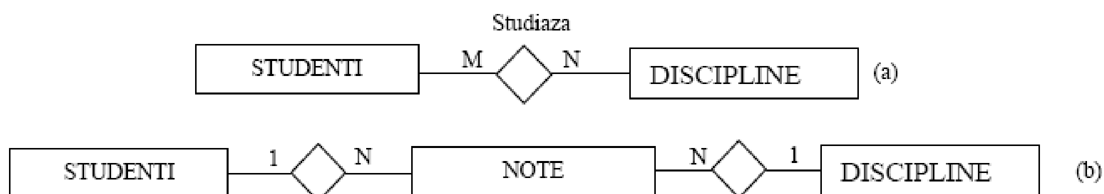


Fig. 3.8. Diferite moduri de definire a tipurilor de entități şi a asocierilor:
a- asociere *M*:*N* între mulțimile de entități STUDENTI şi DISCIPLINE;
b - mulțimea de entități EXAMENE este asociată cu raportul de cardinalitate *N*:1
cu fiecare din mulțimile de entități STUDENTI şi DISCIPLINE.

b) În modelul *E-A*, tipul de entitate (şi mulțimea de entități corespunzătoare) semnifică un substantiv, în timp ce o asociere semnifică un verb. Bineînțeles, nu este obligatoriu ca numele dat unei asocieri să fie un verb (şi, de cele mai multe ori, nici nu este), dar o asociere reprezintă o interacțiune între tipurile de entități (şi mulțimile de entități corespunzătoare), care poate fi exprimată printr-un verb. De exemplu, în diagrama *E-A* din figura 3.7, asocierea SECTIE-ANGAJAT poate fi exprimată prin verbul *cuprinde*, asocierea ANGAJATI-DEPENDENTI poate fi exprimată prin verbul *întreține*, asocierea ANGAJATI-PROIECTE poate fi exprimată prin verbul *lucrează* etc.

c) Modelul *E-A* nu precizează modul cum sunt realizate asocierile între mulțimile de entități. Acest aspect depinde de modelul de date specializat utilizat pentru definirea bazei de date. De exemplu, în modelele ierarhic şi rețea, asocierile sunt realizate explicit, prin pointeri de la o entitate la entitățile asociate, în timp ce în modelul relațional asocierea se realizează prin egalitatea valorilor unor atribute comune ale entităților (chei).

3.3.3 Modelul Entitate-Asociere Extins

Modelul Entitate-Asociere Extins (*Enhanced Entity-Relationship Model*) permite definirea de subtipuri ale unui tip de entități, care moştenesc atribute de la tipul de entitate pe care îl extind (şi care, în acest context, se numeşte supertip) şi au în plus atribute specifice semnificației lor. Prin definirea tipurilor şi a subtipurilor de entități se pot crea ierarhii de tipuri de entități pe mai multe niveluri.

Modelul *E-A* prezentat mai sus este suficient pentru modelarea aplicațiilor de baze de date „tradiționale", adică bazele de date utilizate pentru activități financiare şi industriale, în care se folosesc tipuri de date simple. Odată cu dezvoltarea sistemelor de baze de date, domeniile în care acestea se folosesc au devenit tot mai numeroase, ca, de exemplu: telecomunicațiile, proiectarea tehnologică, sistemele de informații geografice, seviciul Web, etc. Tipurile de entități definite în astfel de baze de date sunt mult mai complexe şi pentru reprezentarea lor cât mai intuitivă şi mai compactă au fost propuse mai multe concepte noi, care au fost introduse în modelul *E-A* extins.

Modelul *E-A* extins se reprezintă printr-o diagramă *E-A* extinsă. Ierarhiile de tipuri se pot crea prin *specializare* sau *generalizare*.

Specializarea (*specialization*) *este un proces de abstractizare a datelor prin care, pornind de la un tip de entitate dat, se definesc unul sau mai multe subtipuri, diferențiate între ele în funcție de rolul specific pe care îl au în modelul de date.*

De exemplu, pornind de la tipul de entitate ANGAJAT se definesc prin specializare subtipurile SECRETARA, TEHNICIAN, INGINER, pentru a diferenția funcțiile pe care angajații le pot avea în întreprinderea respectivă (fig. 3.9). Litera "d" din marcajul de specializare a tipurilor indică o constrângere de disjuncție impusă specializării, care va fi descrisă mai jos.

Subtipurile de entități moştenesc atribute ale tipului inițial şi fiecare dintre ele are atribute suplimentare, specifice rolului lor.

De exemplu, atributele (Nume, Prenume, DataNasterii, Adresa, Salariu) ale tipului de entitate ANGAJAT sunt moştenite de fiecare din subtipurile acestuia. Atributul Functie nu este moştenit, deoarece specializarea subtipurilor s-a efectuat după acest atribut. Ca atribute specifice, subtipul SECRETARA are atributul VitezaRedactare, care este o măsură a calificării, subtipul TEHNICIAN are atributul Calificare, care reprezintă gradul de calificare, iar subtipul INGINER are atributul Specialitate, care este o precizare a domeniului în care lucrează (mecanic, electric, etc.).

Generalizarea (*generalization*) *este procesul de abstractizare invers specializării, prin care se crează un supertip de entitate pornind de la mai multe tipuri de entități.*

Pentru definirea unei generalizări, se identifică atributele comune ale mai multor tipuri de entități şi aceste atribute vor caracteriza supertipul de entitate, iar atributele care diferă de acestea rămân specifice fiecărui tip.

De exemplu, dacă au fost definite tipurile de entități:

AUTOMOBIL (NrInregistrare, Marca, VitezaMaxima, Pret, NumarPasageri) şi

CAMION (NrInregistrare, Marca, VitezaMaxima, Pret, Tonaj)

se poate defini un supertip al acestor tipuri:

VEHICUL(NrInregistrare,Marca, VitezaMaxima, Pret).

Acest tip va cuprinde toate atributele comune, iar tipurile inițiale, AUTOMOBIL şi CAMION, devin subtipuri ale tipului VEHICUL, fiecare conținând atributele specifice (NumarPasageri pentru tipul AUTOMOBIL şi Tonaj pentru tipul CAMION).

Rezultatul obținut prin generalizare este, ca şi în cazul specializării, o ierarhie de tipuri de entități; ceea ce diferă este modul în care se definesc nivelurile ierarhiei.

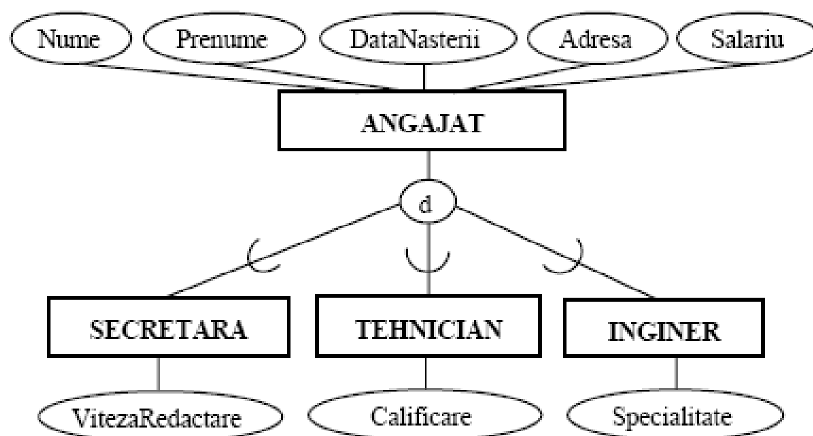


Fig. 3.9. Diagrama E-A extinsă cu ierarhie de tipuri de entități.

Moştenirea atributelor. Proprietatea principală a ierarhiilor de tipuri de entități create prin specializare sau generalizare este *moştenirea atributelor*: atributele tipurilor de entități de nivel ridicat (supertipuri) sunt moştenite de tipurile de entități de nivel scăzut (subtipuri).

Moştenirea dintre un subtip de entități şi supertipul acestuia se reprezintă în diagrama E-A extinsă printr-o legătură (*link*) între subtip şi supertipul de entitate corespunzător pe care este plasat un semicerc orientat către subtip (aşa cum se poate vedea în figura 3.9).

Este evidentă asemănarea dintre ierarhiile de tipuri de entități din modelul E-A extins şi ierarhiile de clase din modelul obiect-orientat, dar modelul *E-A* extins este un model de date mult mai general (de nivel inalt), care poate fi transpus în diferite modele de date specializate, inclusiv modelul obiect-orientat. Aceste transpuneri se fac în funcție de suportul oferit de modelul specializat respectiv pentru reprezentarea entităților, asocierilor, moştenirilor, etc.

BAZE DE DATE

CURS 4

BAZE DE DATE CU STRUCTURI IERARHICE ŞI REȚEA

4.1. Modelul ierarhic şi baze de date ierarhice
4.2. Modelul retea şi baze de date retea

4.1. Modelul ierarhic şi baze de date ierarhice

Modelul ierarhic reprezintă unul dintre primele modele de date şi are ca structură de bază, structura arborescentă. Proprietățile acestei structuri de date determină caracteristicile şi restricțiile modelului ierarhic. În modelul ierarhic fiecărui nod al arborelui îi corespunde un *tip de înregistrare,* format din unul sau mai mute câmpuri, reprezentând atribute ce descriu entități. Descrierea acestei structuri poate fi făcută utilizând *diagrame de structură.*

În Figura 4.1 se observă că un nod părinte poate avea subordonate mai multe noduri copil, în timp ce un nod copil poate avea un singur părinte, ceea ce înseamnă că relația părinte-copil între tipurile de înregistrări va fi 1 la m ("unu la mulți") şi cea copil-părinte va fi de tipul 1 la 1 ("unu la unu").

Unui tip de înregistrare din diagrama de structură a modelului ierarhic îi corespunde în baza de date un anumit număr de *realizări.* Relațiile între realizările unui cuplu părinte-copil sunt indicate de arcele care leagă cele două componente ale diagramei. Astfel, arcul ce leagă TIP_ÎNREG_1 de TIP_ÎNREG_2 indică faptul că unei realizări a tipului părinte (TIP_ÎNREG_1) i se pot asocia "n" realizări ale tipului copil (TIP_ÎNREG_2) şi că o realizare a tipului copil e asociată cu o singură realizare a tipului părinte. Între TIP_ÎNREG_3 şi TIP_ÎNREG_4, relația de 1 la 1 este reciprocă. Acestea sunt singurele tipuri de relații permise între realizările tipurilor de înregistrări ale modelului ierarhic.

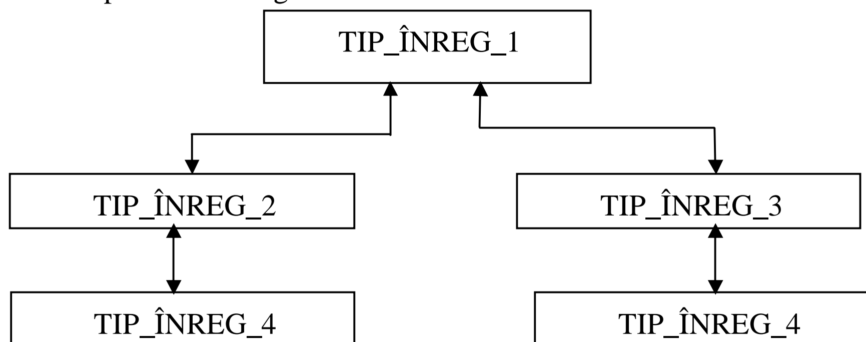


Figura 4.1. Diagramă de structură ierarhică

Definiția 4.1. *O **bază de date ierarhică** poate fi definită ca fiind o mulțime ordonată de **realizări** ale unui singur **tip arbore**.*

Un *tip arbore* constă dintr-un singur tip de înregistrare *"*rădăcină*"*, la care se adaugă o mulțime ordonată alcătuită din unul sau mai multe tipuri de subarbori dependenți. Un *tip subarbore* constă dintr-un tip de înregistrare rădăcină şi o mulțime ordonată alcătuită din unul sau mai multe tipuri de subarbori dependenți ş.a.m.d.

Exemplul 4.1. Considerăm o bază de date cu informații despre sistemul de instruire a angajaților unei mari companii industriale. Compania are un departament de învățământ care

desfăşoară cursuri de pregătire pentru angajați. Fiecare curs poate fi organizat la filiale diferite ca localizare geografică.Baza de date conține informații despre cursurile desfăşurate deja şi despre cele programate a se desfăşura, informații structurate ca în Figura 4.2.

Tipul arbore pentru această bază de date are ca tip înregistrare rădăcină CURS şi două tipuri subarbore: PRELIMINAR şi PROGRAMARE. Această mulțime formată din două tipuri subarbore este ordonată, adică tipul PRELIMINAR precede tipul PROGRAMARE. Tipul PRELIMINAR e alcătuit doar din rădăcină. Tipul subarbore PROGRAMARE are două tipuri subarbore dependente, ambele alcătuite din rădăcină: PROF şi ANGAJAT (de asemenea, ordonate astfel încât PROF precede ANGAJAT). Deci, baza de date conține cinci tipuri de înregistrări (entități): CURS, PRELIMINAR, PROGRAMARE, PROF şi ANGAJAT, în care CURS este tipul de înregistrare rădăcină, iar celelalte sunt tipuri de înregistrări dependente. CURS este părinte pentru tipurile PRELIMINAR şi PROGRAMARE. PROGRAMARE este părinte pentru tipurile PROF şi ANGAJAT.

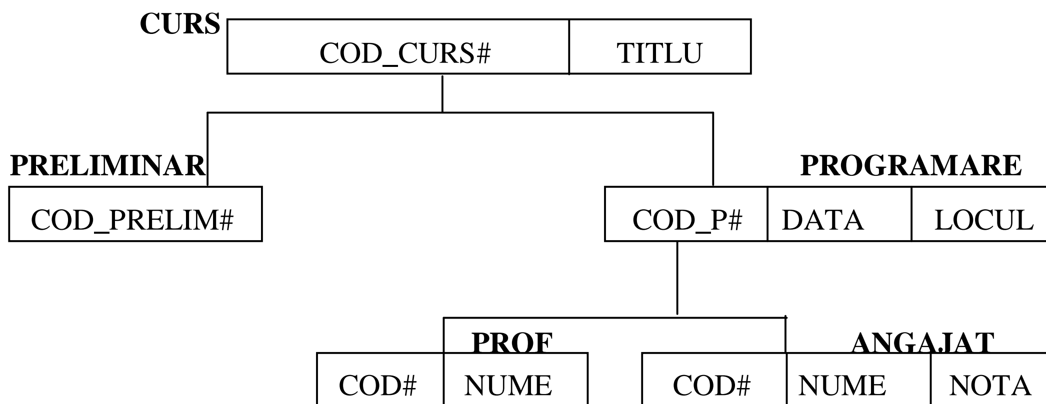


Figura 4.2 Schema conceptuală a bazei de date cu informații despre sistemul de instruire a angaților unei companii

O *realizare a arborelui* constă dintr-o singură realizare a tipului înregistrare rădăcină împreună cu o mulțime ordonată alcătuită din una sau mai multe realizări ale fiecărui tip subarbore imediat dependent de tipul rădăcină.

Deci, pentru orice realizare a unui tip înregistrare părinte, există *n* realizări ale tipurilor înregistrare copil ($n \geq 0$).

De exemplu, o realizare părinte CURS poate avea două realizări copil PRELIMINAR şi trei realizări copil PROGRAMARE (Figura 4.3).

Prima realizare PROGRAMARE este la rândul ei părinte, cu o realizare copil PROF şi trei realizări copil ANGAJAT. Celelalte două realizări PROGRAMARE nu au copii.

Toate realizările unui tip copil care au aceeaşi realizare părinte, se numesc *gemeni*. Deci, cele trei realizări PROGRAMARE sunt gemeni. Observăm că realizările PRELIMINAR nu sunt gemeni cu realizările PROGRAMARE, deoarece sunt de tip diferit, deşi au acelaşi părinte.

Considerăm un (sub)arbore T cu rădăcina R şi subarborii $S_1, S_2,\ldots,S_n$ în această ordine. Fie t o realizare a lui T cu rădăcina r (o realizare a lui R) şi subarborii $s_1, s_2,\ldots,s_n$, realizări ale lui $S_1, S_2,\ldots, S_n$.

Definim recursiv *secvența ierarhică* pentru t, ca fiind secvența obținută alăturând înregistrării r, toate înregistrările lui $s_1, s_2,\ldots,s_n$ luate în ierarhia lor.

Fiecare arbore individual din baza de date poate fi privit ca un subarbore al unei înregistrări rădăcină inițiale. Deci baza de date poate fi considerată ca un singur arbore.

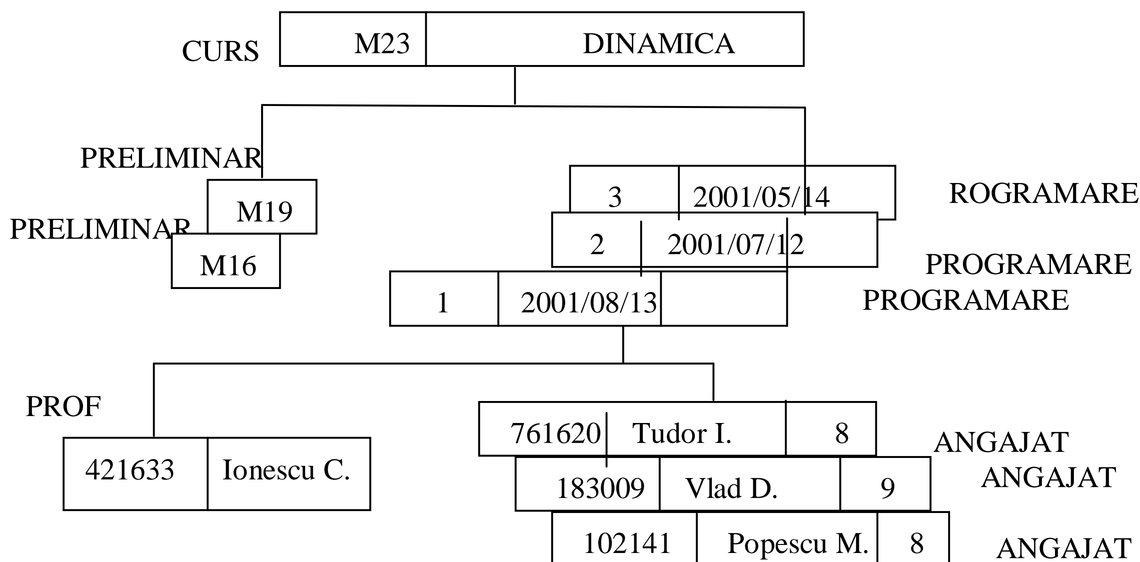


Figura 4.3 Realizare a arborelui asociat bazei de date din Exemplul 4.1

Secvența ierarhică pentru arborele din Figura 4.3 este:

CURS M23
PRELIMINAR M16
PRELIMINAR M19
PROGRAMARE 1
PROF 421633
STUDENT 102141
STUDENT 183009
STUDENT 761620
PROGRAMARE 2
PROGRAMARE 3

Un limbaj de manipulare a datelor ierarhice constă dintr-o mulțime de *operatori* de prelucrare a datelor reprezentate arborescent. Exemple de operatori: operatori pentru localizarea unui anumit arbore în baza de date (exemplu: localizarea arborelui pentru cursul M23); operatori pentru trecerea de la un arbore la altul în baza de date (exemplu: trecerea de la arborele M23 la cel care urmează în secvența ierarhică în baza de date); operatori de trecere de la o înregistrare la alta într-un arbore, cu posibilități de parcurgere în sus sau în jos pe diferite căi ale ierarhiei (exemplu: operator de trecere de la înregistrarea pentru cursul M23 la prima înregistrare PROGRAMARE); operatori de trecere de la o înregistrare la alta corespunzător secvenței ierarhice din baza de date (exemplu: operator de trecere de la o înregistrare PROF din cadrul PROGRAMARE la o înregistrare ANGAJAT din aceeaşi realizare PROGRAMARE); operatori pentru inserarea unei noi înregistrări la o poziție specifică într-un astfel de arbore (exemplu: inserarea unei noi realizări PROGRAMARE); operatori de ştergere a unei înregistrări specificate (exemplu: ştergera unei realizări PROGRAMARE).

O caracteristică a modelului ierarhic este aceea că nici o realizare a unui nod copil nu poate să existe fără a avea asociată o realizare a nodului părinte. Aceasta înseamnă că dacă un nod părinte este şters, automat sistemul va şterge întregul subarbore ce are ca părinte acel nod precum şi că o realizare a nodului copil nu poate fi inserată dacă o realizare a nodului părinte nu există deja.

Având în vedere aceasta, precum şi relaţiile permise între realizările tipurilor de înregistrări, pot fi enunţate următoarele *restricţii de integritate* ale modelului ierarhic:
• o realizare copil este întotdeauna asociată unei singure realizări părinte;
• dacă un tip de înregistrare nu are realizări, atunci nici tipurile descendente de înregistrări nu au realizări.

Pentru ilustrarea acestor restricţii de integritate, să considerăm o bază de date cu informaţii despre comenzile lansate de clienţi unei firme (Figura 4.4).

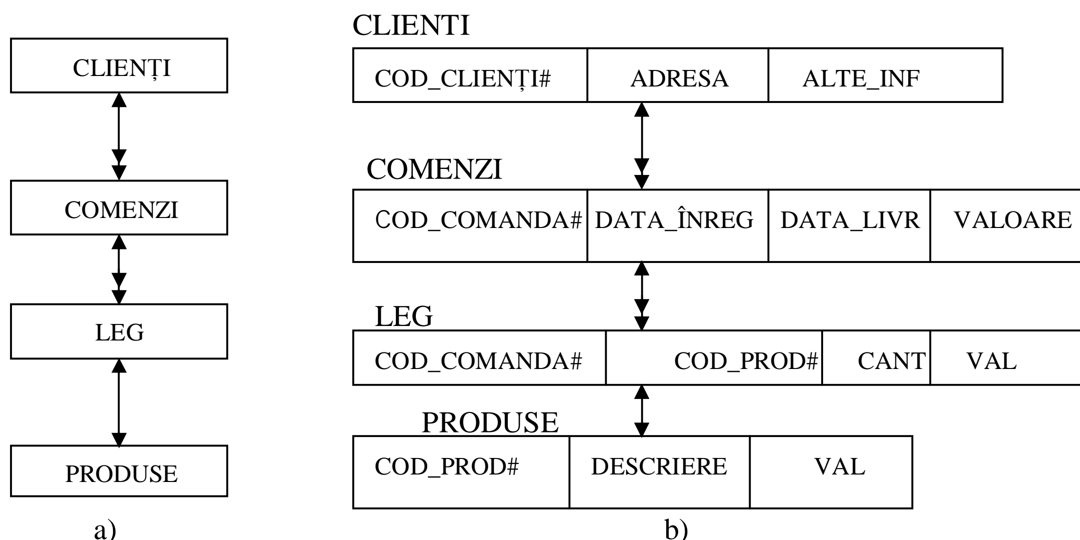


Figura 4.4 Schema conceptuală a BD

Nodul rădăcină este CLIENTI, înregistrarea copil este COMENZI, iar relaţia părinte-copil este de tipul 1: m ("unu la mulţi"), adică un client poate lansa mai multe comenzi firmei considerate. O comandă poate să includă mai multe produse iar acelaşi produs poate apare în mai multe comenzi ale aceluiaşi client sau ale unor clienţi diferiţi. Deci, între tipurile de înregistrare COMENZI şi PRODUSE relaţia este de tip n : m ("mulţi la mulţi").

Deoarece modelul ierarhic nu permite reprezentarea relaţiei de acest tip, s-a optat pentru transformarea acesteia în două relaţii, prin introducerea tipului de înregistrare LEG, astfel: o relaţie de tip 1:n între COMENZI şi LEG şi o alta de tip 1 la 1 între LEG şi PRODUSE (se respectă astfel restricţia 1 conform căreia o realizare copil poate fi asociată unei singure realizări părinte).

Se observă că într-o realizare un nod copil poate apare de 0,1 sau n ori, în timp ce în diagrama de structură, tipul de înregistrare corespunzător apare o singură dată.

În acelaşi timp, dacă în COMENZI nu există realizări care să se refere la un produs T, nu vor exista informaţii despre acest produs în PRODUSE. Deci informaţiile despre produse vor putea fi stocate în baza de date a firmei, numai dacă există comenzi pentru acele produse.

4.2. Modelul reţea şi baze de date reţea

Modelul reţea utilizează ca structură de date, structura reţea. Reţeaua este un graf orientat alcătuit din *noduri* conectate prin *arce*. Nodurile corespund *tipurilor de înregistrare* şi arcele *pointerilor*. Modelul reţea foloseşte înregistrările pentru a reprezenta entităţile şi pointerii între înregistrări pentru a reprezenta relaţiile dintre entităţi.

Structura de date reţea (Figura. 4.5) seamănă cu structura de date arborescentă, cu diferenţa că un nod dependent (copil) poate avea mai mult decât un singur părinte.

O bază de date rețea constă dintr-un număr oarecare de *tipuri de înregistrări*. O înregistrare e constituită dintr-un număr oarecare de *câmpuri*. Un câmp este cea mai mică unitate de date care are un nume. Fiecare câmp are un tip de dată asociat. Câmpul corespunde unui atribut şi înregistrarea unei entități (Figura 4.6).

În Figura 4.7 este reprezentată, prin intermediul unei *diagrame de structură*, structura rețea a unei baze de date cu informații despre proiectele unei firme şi angajații antrenați în realizarea lor. Diagrama corespunde unui graf rețea în care nodurile au fost înlocuite de dreptunghiuri care reprezintă înregistrările. Arcele reprezintă relații de tip 1:1 sau 1:n între înregistrări.

O trăsătură a modelului este conceptul de *set* utilizat ca modalitate de exprimare a relațiilor. Un *tip set* (Figura 4.8) constă dintr-un singur tip de *nod proprietar* şi unul sau mai multe tipuri de noduri dependente legate de acesta, numite *tipuri membre*. Figura 4.8 ilustrează un tip set numit DEPT_STU în care DEPT este tipul propritar şi STU tipul membru. Acest set reprezintă în fapt o relație 1:m între înregistrările DEPT şi STU.

O *realizare* a setului este o colecție de înregistrări având o realizare proprietar şi un număr oarecare de realizări membre asociate. Deci, există o realizare a setului DEPT-STU pentru fiecare realizare a înregistrării DEPT în baza de date (Figura 4.9).

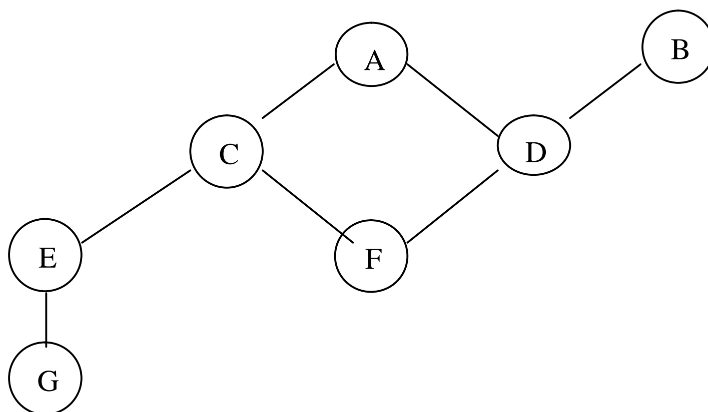


Figura 4.5 Structură rețea

a)STUDENT

CODS#	NUMES	CODC#	PUNCTE

b)STUDENT

CODS#	NUMES	ADR_STU	CODC#	PUNCTE
		STR ORAS COD		

Figura 4.6 Exemplu de înregistrare

Dacă există un departament fără studenți avem o realizare cu proprietar şi fără membrii. O astfel de realizare a setului se numeşte *vidă*.

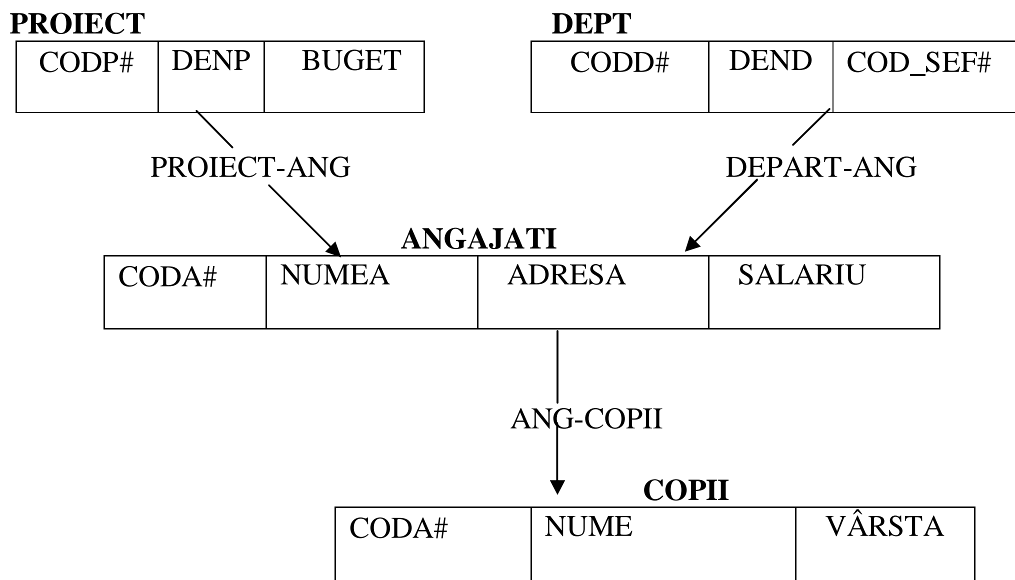


Figura 4.7 Diagramă rețea

Modelul de rețea impune restricția conform căreia o înregistrare nu poate fi membră a două realizări ale aceluiaşi tip set. Rezultă că, în exemplul considerat, un student nu poate fi inclus în mai multe departamente. Totuşi, o înregistrare poate să aparțină mai multor tipuri set. Un tip de înregistrare poate fi un tip proprietar într-un set şi un tip membru în altul. De exemplu, în diagrama de structură din Figura 4.7, ANGAJAT este membru a două tipuri set, PROIECT_ANG şi DEPART_ANG şi proprietar al unui tip set ANG_COPII.

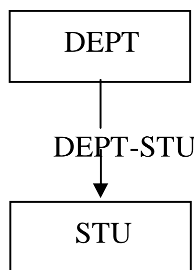


Figura 4.8 Tipul SET

Astfel, o anume înregistrare ANGAJAT poate fi conectată cu înregistrările PROIECT, DEPT şi COPII.

Din setul DEPT-STU prezentat în Figura 4.8, putem spune cărui departament îi aparține un student cunoscând in ce realizare a setului DEPT-STU este înregistrarea STU.

Un *set esențial* transmite informații păstrând o conexiune logică datorată existenței sale. Alternativa unui set esențial este să ținem valoarea unuia sau mai multora dintre câmpurile realizărilor înregistrărilor proprietar, în realizarea membrului. Normal, acest câmp este o cheie externă. Un set cu astfel de informații în înregistrările membre se numeşte *set pe bază de valoare*. Dacă dorim să reducem redundanța pentru a asigura integritatea datelor şi a folosi eficient spațiul, alegem *seturi esențiale*. Dacă nu dorim ca informațiile să fie pierdute sau distruse, alegem *seturi pe bază de valoare.*

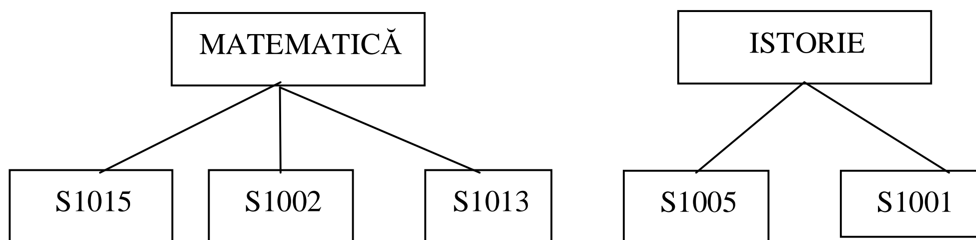


Figura 4.9 Exemple de realizări

Pentru că seturile sunt doar metode de reprezentare a relaţiilor dintre înregistrări şi pentru că un set poate avea un singur proprietar, nu există o cale directă de a reprezenta relaţiile de tip m:n în acest model. Deci, nu putem reprezenta relaţia STU-CURS din Figura 4.10a) în mod direct. Totuşi există un mod indirect pentru soluţionarea acestei situaţii. Astfel, când între două tipuri de înregistrări există o relaţie de tip m:n, pentru reprezentarea acesteia, vom crea un nou tip de înregistrare, numită *înregistrare de intersecţie* sau *legătură,* constând din cheile asociate, plus informaţia de intersecţie, adică atribute descriptive ale căror valori depind de asociere. În Figura 4.11 apare înregistrarea de legătură creată pentru fiecare interacţiune între o înregistrare STU şi un CURS. Această diagramă arată trei tipuri de înregistrări şi două tipuri de seturi. Fiecare înregistrare de legătură aparţine celor două seturi.

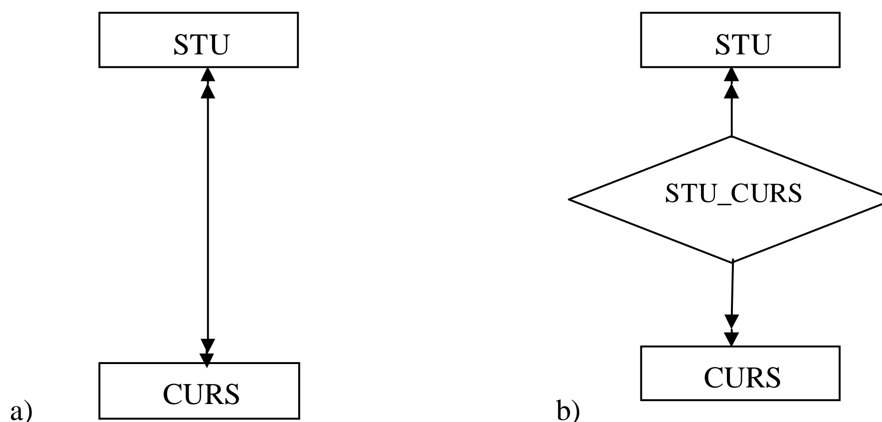


Figura 4.10 Reprezentarea relaţiilor.

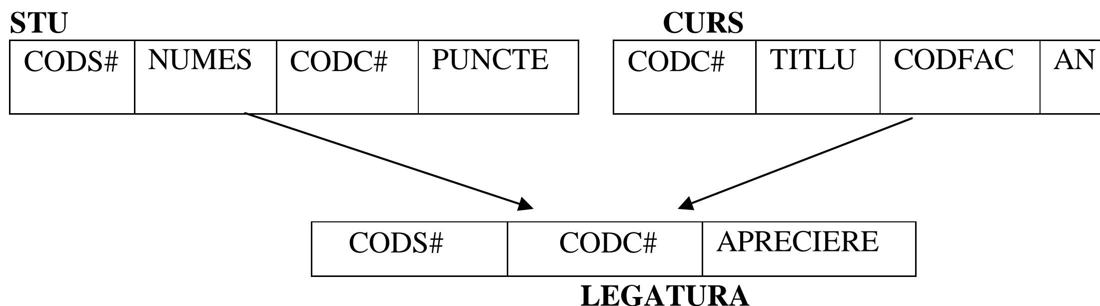


Figura 4.11 Diagramă cu două tipuri de seturi

Modelul rețea, este un model complex, dificil de folosit. Implementarea lui se face pentru limbaje de prelucrare secvențială a înregistrărilor, ceea ce determină o prelucrare dificilă a bazelor de date, o viteză redusă de lucru şi spațiu de memorie ocupat ineficient.

Deoarece o rețea este o extensie a unei ierarhii, proprietățile semantice ale modelului rețea sunt similare cu cele ale modelului ierarhic. De aceea, dependențele din rețea sunt puțin clare, din cauza posibilității existenței mai multor părinți/proprietari.

Modelul rețea nu posedă un mod explicit de tratare a agregării semantice, relațiile între înregistrări fiind folosite atât pentru reprezentarea semantică cât şi pentru alte scopuri.

BAZE DE DATE

CURS 5

BAZE DE DATE RELAȚIONALE

5.1 Considerații generale privind modelul relațional al datelor
5.2 Structura relațională a datelor. Relații, domenii, atribute şi schema unei relații.
5.3 Operații informatice şi booleene
5.4 Constrângeri de integritate ale relațiilor
5.5 Indexarea relațiilor

5.1 Considerații generale privind modelul relațional al datelor

Modelul relațional al datelor a fost acceptat aproape fără rezerve de atât de specialiştii din domeniul bazelor de date cât şi de utilizatori, încă de la apariția primului articol al lui Codd E. F., prin care erau puse bazele acestui model. Ideea unui model asamblist al datelor a fost lansată de către Childs D. F. în 1968, care a subliniat faptul că orice structură de date poate fi reprezentată printr-una sau mai multe tabele de date, în cadrul cărora este necesar să existe şi informații de legătură, în vederea stabilirii unor legături între acestea. Acest model beneficiază de o solidă fundamentare matematică (bazat pe teoria matematică a relațiilor) şi pentru care abordările teoretice sunt nu numai posibile, dar şi extrem de utile.

S-a constatat că, prin utilizarea sistemelor relaționale este posibilă atingerea unor obiective importante ale organizării datelor în comparație cu modelele ierarhice şi rețea:

1. Asigurarea unui grad sporit de independență a programelor de aplicație față de modul de reprezentare internă a datelor şi metodele de acces la date. În precizarea prelucrărilor asupra datelor, programele nu fac apel la pointeri sau alte elemente ale schemei interne a bazei de date.

2. Furnizarea unor metode şi tehnici eficiente de control a coerenței şi redundanței datelor. Modelul relațional, prin tehnica normalizării relațiilor permite definirea unei structuri conceptuale optime a datelor, prin care se minimizează riscurile de eroare la actualizare, reducându-se redundanța datelor.

3. Oferirea unor facilități multiple de definire şi manipulare a datelor. În primul rând modelul relațional oferă posibilitatea utilizării unor limbaje procedurale, bazate pe algebra relațională, precum şi a unor limbaje neprocedurale ce au la bază calculul relațional.

4. Ameliorarea integrității şi confidențialității datelor. Modelul relațional realizează acest lucru prin mecanisme flexibile şi eficace de specificare şi utilizare a restricțiilor de integritate şi a relațiilor virtuale.

Componentele modelului relațional sunt: *structura relațională a datelor*, prin care datele sunt organizate sub forma unor tablouri bidimensionale (tabele) de date, numite *relații*, *operatorii modelului relațional*, ce definesc operațiile care se pot efectua asupra relațiilor, în scopul realizării funcțiilor de prelucrare asupra bazei de date, respectiv consultarea, inserarea, modificarea şi ştergerea datelor, precum şi *restricțiile de integritate* care permit definirea stărilor coerente ale bazei de date.

5.2. Structura relațională a datelor. Relații, domenii, atribute şi schema unei relații

Prezentarea structurii relaționale a datelor impune definirea noțiunilor de domeniu, atribut, relație şi schemă a unei relații.

În Figura 5.1 se prezintă un model relațional ce corespunde unei mulțimi concrete de caracteristici despre lumea reală. Orarul de zbor al avioanelor posedă o structură de date relațională. Pentru fiecare linie aeriană din orarul de zbor sunt date caracteristicile: *numărul cursei, aeroportul de decolare, aeroportul de aterizare, ora decolării, ora aterizării.*

Fiecare avion este determinat de mulțimea valorilor de fiecare linie. Trebuie să ne limităm numai la acele date care pot sa apară în definirea coloanei. Coloana cu numele, Punct de decolare (PD) conține numele aeroporturilor liniilor aeriene considerate. Coloana Ora decolarii (OD) (respectiv, Ora aterizării (OA)) exprimă ora la care are loc decolarea (respectiv, aterizarea). Coloana cu numele Punct de aterizare (PA) conține denumirea aeroportului unde se aterizează.

orar

Nr. Zbor(NR)	**Punct de Decolare(PD)**	**Punct de Aterizare(PA)**	**Ora de Decolare(OD)**	**Ora de Aterizare(OA)**
70	Bucureşti	Craiova	16:59	17:50
75	Craiova	Bucuresti	07 :25	08 :25
80	Bucureşti	Timişoara	17 :30	19 :30
85	Timişoara	Bucureşti	07 :15	09 :25
90	Timişoara	Craiova	10 :15	13 :20

Figura 5.1 Orarul de zbor al avioanelor unei companii

Definiția 5.1 ***Domeniul*** *reprezintă o mulțime de valori, caracterizat printr-un nume şi este definit fie explicit prin enumerarea tuturor componentelor sale, fie printr-o proprietate distinctivă a valorilor din domeniul respectiv.*

Pentru tabelul din Figura 5.1 se pot da următoarele exemple:

D_1={Bucureşti, Craiova, Timişoara}

D_2={x / x∈N, x ∈[1,100]}

Atributul reprezintă coloana unei tabele de date, caracterizată printr-un nume. Numele coloanei (atributului) exprimă de obicei semnificația valorilor din cadrul coloanei respective. Fiecărui nume de atribut îi corespunde un domeniu D_i numit *domeniul atributului* A_i, $1\leq i\leq n$ şi se va nota cu $dom(A_i)$. Pentru a diferenția coloanele care conțin valori ale aceluiaşi domeniu şi a elimina astfel dependența de poziție în cadrul tabelei, se asociază fiecărei coloane un nume distinct.

Pentru tabelul din Figura 5.1 avem atributele NR, PD, PA, OD, OA şi domeniile asociate dom(NR), dom(PD), dom(PA), dom(OD), dom(OA).

De exemplu, dom(PD)= {Bucureşti, Craiova, Timişoara}.

Fie $D_1,D_2, \ldots ,D_n$ domenii finite, nu neapărat disjuncte. *Produsul cartezian* D_1 ´D_2 ´... ´D_n al acestora este definit de mulțimea tuplurilor $<v_1,v_2, \ldots ,v_n>$, unde $v_1\in D_1$, $v_2\in D_2$, … ,$v_n\in D_n$. Numărul n defineşte *aritatea* tuplului.

Definiția 5.2 *O* ***relație*** *r pe mulțimile* $D_1,D_2, \ldots ,D_n$ *este o submulțime a produsului cartezian* D_1 ´D_2 ´... ´D_n, *deci o mulțime de tupluri.*

Există şi un alt mod de a defini o relație, şi anume, ca o mulțime finită de funcții. Asociem fiecărui domeniu D_i un atribut A_i şi definim relația r ca fiind o mulțime de tupluri $\{t_1,t_2, \ldots ,t_n\}$, unde t_i: $\{A_1,A_2, \ldots ,A_n\}\rightarrow D_1\cup D_2\cup \ldots \cup D_n$ şi $t_i(A_j) \in D_j$ pentru orice valori ale lui i şi j. Într-o relație, tuplurile trebuie să fie distincte (nu se admit duplicări ale tuplurilor). De obicei, vom nota relația cu r sau $r[A_1,A_2, \ldots ,A_n]$.

Orarul din Figura 5.1 reprezintă un exemplu de relație pe care o numim *orar*. Conținutul informațional al liniei nu depinde de ordinea coloanelor, de exemplu coloanele PD si PA pot fi interschimbate.

Definirea unei relații ca o mulțime de tupluri sau ca o mulțime de funcții se referă la mulțimi care variază în timp (se adaugă, se şterg sau se modifică elemente). Pentru a caracteriza o relație este nevoie de un element învariant în timp, iar acest invariant este dat de structura relației (schema relației).

Definiția 5.3 *Mulțimea tuturor atributelor* $R=\{A_1,A_2, \dots ,A_n\}$ *corespunzătoare unei relații r o numim* ***schema relației*** *şi o notăm* $r(A_1,A_2, \dots ,A_n)$.

Schema relației *orar* se defineşte astfel: *orar*(NR, PD, PA, OD, OA).

Schema unei relații mai este cunoscută şi sub numele de *intensia* unei relații, ca expresie a proprietăților comune şi invariante ale tuplurilor care compun relația. Spre deosebire de intensie, *extensia* unei relații reprezintă mulțimea tuplurilor care compun la un moment dat relația, mulțime care este variabilă în timp. De obicei, extensia unei relații este stocată fizic în spațiul asociat bazei de date, caz în care relația se numeşte *relație de bază*.

Există situații în care extensia nu este memorată în baza de date. Este cazul aşa numitelor *relații virtuale*, cunoscute şi sub numele de *relații derivate* sau *viziuni*. Acestea sunt definite implicit, pe baza altor relații, prin intermediul unei expresii relaționale iar stabilirea tuplurilor care o compun se face prin evaluarea expresiei.

Aşadar, putem reprezenta o relație printr-un tabel bidimensional în care fiecare linie corespunde unui tuplu şi fiecare coloană corespunde unui domeniu din produsul cartezian. Numărul atributelor defineşte *gradul* relației, iar numărul de tupluri *cardinalitatea* relației.

Fiecare linie a relației este o multime de valori, câte una pentru fiecare nume de atribut. Linia relației se numeşte *tuplu*. În Figura 5.1 relația *orar* este formată din 5 tupluri. Unul dintre acestea, notat cu *t*, este definit astfel:

t(NR)=75, t(PD)=Craiova, t(PA)=Bucureşti, t(OD)=7.25, t(OA)=8.25

Valoarea concretă a tuplului *t* pentru atributul *A* o numim *A-valoarea tuplului t*, iar dacă *t* este considerată ca funcție, atunci *A*-valoarea tuplului *t* o vom nota cu $t(A)$. Pentru $X \subseteq R$, restricția tuplului *t* la *X* o notăm cu t/X sau $t(X)$ şi o vom numi *X-valoarea tuplului t*.

Pentru relația *orar*, considerăm un tuplu *t* oarecare, de exemplu primul tuplu din relație. {PD, PA}- valoarea tuplului *t* este tuplul *t'* pentru care t'(PD)=Bucureşti, t'(PA)=Craiova.

5.3 Operații informatice şi booleene

Asupra tuplurilor unei relații se pot efectua următoarele operații informatice:

1. **Adăugarea**. Permite adăugarea de noi tupluri la o relație.

Astfel, pentru relația $r[A_1A_2, \dots ,A_n]$ operația are forma:

$$ADD\ (r : A_1=d_1,\ A_2=d_2, \dots ,A_n=d_n)$$

De exemplu, adăugarea unui tuplu la relația *orar* se face astfel:

ADD (*orar* : NR =99, PD=Oradea, PA=Bucuresti, OD=20, OA=22).

Când ordinea atributelor este fixată aceasta poate fi scrisă în forma:

ADD (*orar* : 99, Oradea, Bucuresti, 20,22)

Scopul operației de adăugare este de a adăuga un tuplu la o anumită relație *r*, dar rezultatul adăugării nu este conform cu scopul acesteia în următoarele cazuri :

- tuplul de adăugat nu corespunde schemei relației ;
- anumite valori ale tuplului nu aparțin domeniului respectiv;
- tuplul de adăugat coincide după cheie cu un tuplu din relație.

De exemplu, adăugarea în relația *orar,* a tuplului:

ADD(*orar* : NR:90, PD:Iaşi, PA:Sibiu, PD:16, PA:12)
nu este permisă, deoarece nu se respectă prima condiţie.

2.Ştergerea. Această operaţie se introduce pentru a elimina anumite tupluri din relaţie.

Pentru o relaţie *r*, operaţia are forma :

$$DEL(r: A_1=d_1, A_2=d_2, ... ,A_n=d_n)$$

Atunci când numele atributelor sunt ordonate, se poate scrie mai simplu:

$$DEL(r:d_1, d_2, ... ,d_n)$$

De exemplu, pentru relaţia *orar*, ştergerea primului tuplu, se realizează astfel:

DEL(*orar* : 70, Bucureşti, Craiova, 16:59, 17:50)

Deoarece, într-o relaţie, tuplurile sunt identificate unic prin valoarea unei chei, este suficient pentru a realiza ştergerea, să definim numai valoarea cheii.

Dacă $K=\{ B_1,B_2,...,B_n\}$ este o cheie, atunci se poate utiliza următoarea formă directă:

$$DEL(r : B_1=c_1, B_2=c_2, ... ,B_n=c_n)$$

De exemplu, varianta scurtă a operaţiei de ştergere din relaţia *orar* este:

DEL(*orar* : 70)

Dacă tuplul ce se doreşte a fi şters, nu există în relaţia *r* atunci se generează o eroare.

3. Modificarea. Se referă la faptul că anumite valori dintr-un tuplu se pot modifica.

Pentru o relaţie oarecare *r* şi pentru submulţimea $\{C_1, C_2, ... ,C_p\} \subseteq \{A_1, A_2, ... ,A_n\}$, operaţia de modificare are forma :

$$CH (r : A_1=d_1,A_2=d_2,...,A_n=d_n ; C_1=c_1,....,C_p=c_p)$$

Dacă $K=\{B_1, ... ,B_n\}$ este o cheie, atunci operaţia de modificare se poate scrie:

$$CH (r : B_1=d_1, ... ,B_m=d_m; C_1= c_1,...,C_p=c_p)$$

Pentru relaţia *orar*, operaţia de modificare a primului tuplu are forma :

CH(*orar* : NR=70, PD=Bucureşti, PA=Craiova, OD=16:59,OA=17:50;OD=18, OA=19)
sau utilizând numai cheia relaţiei: *CH* (*orar* : NR=70, OD=18, OA= 19).

Asupra relaţiilor dintr-o bază de date se pot efectua următoarele operaţii booleene:

1. **Reuniunea**. Este o operaţie definită pe două relaţii *r* şi *s* cu aceeaşi schemă R şi constă în construirea unei noi relaţii *q*, cu aceeaşi schemă R şi având drept extensie tuplurile din *r* şi *s* luate împreună o singură dată.

Notaţiile uzuale pentru această operaţie sunt: r∪s, OR(r,s), UNION(r,s).

Considerând tuplurile ca transformări, avem următoarea definiţie formală a reuniunii:

$$r\cup s=\{t \mid t\in r\}\cup\{t \mid t\in s\}$$

Exemplul 5.1 Fie *orar1* şi *orar2* două relaţii cu aceeaşi schemă R(NR, PD, PA, OD, OA).

orar1

NR	PD	PA	OD	OA
75	Craiova	Bucuresti	07:15	08 :25
80	Bucureşti	Timişoara	17:30	19 :30
85	Timişoara	Bucureşti	07:15	09 :25
90	Timişoara	Craiova	10 :15	13 :20

orar2

NR	PD	PA	OD	OA
75	Craiova	Bucuresti	07 :15	08 :25
80	Bucureşti	Timişoara	17 :30	19 :30
95	Timişoara	Arad	11 :15	11 :25
96	Timişoara	Oradea	12 :15	13 :20

Prin operația de reuniune a celor două relații se obține:

($orar1 \cup orar2$)

NR	**PD**	**PA**	**OD**	**OA**
75	Craiova	Bucuresti	07 :15	08 :25
80	Bucureşti	Timişoara	17 :30	19 :30
85	Timişoara	Bucureşti	07 :15	09 :25
90	Timişoara	Craiova	10 :15	13 :20
95	Timişoara	Arad	11 :15	11 :25
96	Timişoara	Oradea	12 :15	13 :20

2. Diferența. Reprezintă o operație definită pe două relații *r* şi *s* cu aceeaşi schemă R, şi constă în construirea unei noi relații *q*, cu aceeaşi schemă R şi având drept extensie tuplurile din *r* care nu se regăsesc în *s*.

Notațiile uzuale pentru această operație sunt: r-s, REMOVE(r,s), MINUS(r,s).

Considerând tuplurile ca transformări, avem următoarea definiție formală a diferenței:

$$r\text{-}s = \{t \mid t \in r\} - \{t \mid t \in s\} \text{ sau}$$

$$r\text{-}s = \{t \mid t \in r \wedge t \notin s\}$$

Diferența relațiilor *orar1* şi *orar2* din Exemplul 5.1 ne conduce la următoarea relație:

NR	PD	PA	OD	OA
85	Timişoara	Bucureşti	07 :15	09 :25
90	Timişoara	Craiova	10 :15	13 :20

3. Intersecția. Reprezintă o operație definită pe două relații *r* şi *s* cu aceeaşi schemă R, şi constă în construirea unei noi relații *q*, cu aceeaşi schemă R şi având drept extensie tuplurile comune din *r* şi *s*.

Notațiile uzuale pentru această operație sunt: $\cap$, INTERSECT(r,s), AND(r,s).

Considerând tuplurile ca transformări, avem următoarea definiție formală a diferenței:

$$r \cap s = \{t \mid t \in r \wedge t \in s\}$$

Intersecția relațiilor *orar1* şi *orar2* din Exemplul 5.1, ne conduce la relația:

NR	**PD**	**PA**	**OD**	**OA**
75	Craiova	Bucuresti	07 :15	08 :25
80	Bucureşti	Timişoara	17 :30	19 :30

Intersecția reprezintă o operație derivată, care poate fi exprimată prin intermediul diferenței astfel: $r \cap s = r-(r-s)$ sau $r \cap s = s-(s-r)$.

4. Produs cartezian. Reprezintă o operație definită pe două relații *r* şi *s* de schemă R, respectiv S, şi constă în construirea unei noi relații *q*, a cărei schemă Q, se obține din concatenarea schemelor relațiilor *r* şi *s* iar extensia lui *q* se obține din toate combinațiile tuplurilor din *r* cu cele din *s*.

Notațiile uzuale pentru această operație sunt: r×s, PRODUCT(r,s), TIMES(r,s).

Considerând tuplurile ca transformări, avem următoarea definiție formală a produsului cartezian:

$$r \times s = \{t \mid (\exists t_1 \in r) \wedge (\exists t_2 \in s) \wedge (t(R)=t_1) \wedge (t(S)=t_2)\} \text{sau}$$

$$r \times s = \{t \mid t=t_1 t_2 \wedge t_1 \hat{I}\, r \wedge t_2 \hat{I}\, s\}$$

Fie *pilot* o relație cu următoarea extensie:

pilot

NUME	VARSTA
Popa	35
Vigu	40

Produsul cartezian al relațiilor *orar1* din Exemplul 5.1 şi *pilot*, ne conduce la următoarea relație:

NR	PD	PA	OD	OA	NUNE	VARSTA
75	Craiova	Bucuresti	07 :15	08 :25	Popa	35
80	Bucureşti	Timişoara	17 :30	19 :30	Popa	35
85	Timişoara	Bucureşti	07 :15	09 :25	Popa	35
90	Timişoara	Craiova	10 :15	13 :20	Popa	35
75	Craiova	Bucuresti	07 :25	08 :25	Vigu	40
80	Bucureşti	Timişoara	17 :30	19 :30	Vigu	40
85	Timişoara	Bucureşti	07 :15	09 :25	Vigu	40
90	Timişoara	Craiova	10 :15	13 :20	Vigu	40

5.4 Constrângeri de integritate ale relațiilor

Restricțiile de integritate definesc condiții pe care trebuie să le satisfacă datele din baza de date, pentru a fi considerate corecte, coerente în raport cu lumea reală la care se referă.

Restricțiile de integritate reprezintă principalul mod de integrare a semanticii datelor în cadrul modelului relațional al datelor, mecanismele de definire şi verificare a acestor restricții reprezentând principalele instrumente pentru controlul semantic al datelor. Există două tipuri de restricții, şi anume *restricții structurale* care sunt inerente modelării datelor şi *restricții de funcționare (comportament)* care sunt specifice unei anumite baze de date. Restricțiile structurale sunt de patru tipuri: de cheie, de referință, de entitate şi de dependentă între date, din care primele trei, constituie mulțimea minimală de restricții de integritate pe care trebuie să le respecte un SGBD relațional. Aceste restricții sunt definite în raport de noțiunea de *cheie* a unei relații.

Definiția 5.4 *O* ***cheie*** *a unei relații r, este o mulțime* $K \subseteq R$*, astfel încât:*

(i) *pentru orice două tupluri* t_1, t_2 *ale lui r* $\Rightarrow t_1(K) \neq t_2(K)$*;*

(ii) *nu există nici o submulțime proprie a lui K cu proprietatea (i).*

Altfel spus, cheia reprezintă o mulțime minimală de atribute ale căror valori identifică în mod unic un tuplu într-o relație.

Fiecare relație are cel puțin o cheie. Dacă există mai multe chei posibile, ele se numesc *chei candidat*. Una din cheile candidat va fi aleasă de administratorul bazei de date pentru a identifica efectiv tupluri şi ea va primi numele de *cheie primară*. Cheia primară nu poate fi reactualizată. Restul cheilor vor purta numele de *chei alternative* sau *alternate*.

Atributele care reprezintă cheia primară pot fi subliniate sau urmate de semnul # în schema relației respective. Un grup de atribute din cadrul unei relații care conține o cheie a relației se numeşte *supercheie*.

Exemplul 5.2 În relația *orar* din Figura 5.1 mulțimile {NR} şi {PD, PA} sunt chei. Dacă se alege {NR}drept cheie primară, atunci {PD, PA} devine cheie alternativă. Acest fapt se reprezintă astfel : *orar*(<u>NR</u>, PD, PA, OD, OA) sau *orar*(NR#, PD, PA, OD, OA). Mulțimea de atribute {NR, PA} este o supercheie.

Considerăm relația *local*, ce contine o mulțime de oraşe cu anumite caracteristici:

local

Punct de Decolare(PD)	**Cod Localitate(CL)**	**Județ (JD)**
Craiova	1100	Dolj
Bucureşti	1200	Ilfov
Timişoara	1300	Timiş

O cheie identifică tupluri şi este diferită de un index care localizează tupluri. O *cheie secundară* este folosită ca index pentru a accesa tupluri. Fie schemele relaționale *orar*(NR#, PD, PA, OD, OA) şi *local*(PD#, CL, JD), unde NR şi PD sunt chei primare respectiv secundare pentru *orar*, iar PD este cheie primară pentru relația *local*. În acest caz vom spune că PD este *cheie externă* pentru *orar*. În acest context, *orar* este denumită relație care referă, în timp ce *local* poartă numele de *relație referită*. O cheie primară poate conține o cheie externă. De asemenea, valorile atributului PD din relația *orar*, care reprezintă o cheie externă pentru această relație, trebuie ori să corespundă la o valoare a cheii primare din relația *local*, ori să aibă valoarea *null*. De multe ori un atribut este necunoscut sau neaplicabil. Pentru a reprezenta acest atribut a fost introdusă o valoare convențională în relație, şi anume valoarea *null*.

Modelul relațional respectă trei restricții de integritate structurală:

- *unicitatea cheii* - cheia primară trebuie să fie unică şi minimală, adică pentru o relație r cu cheia K, oricare ar fi tuplurile t_1 şi t_2, să avem $t_1(K) \neq t_2(K)$;

- *integritatea entității* - atributele cheii primare trebuie să fie diferite de valoarea *null*, deoarece unicitatea cheii impune ca la încărcarea unui tuplu, valoarea cheii trebuie să fie cunoscută pentru a putea verifica dacă tuplul figurează deja în baza de date;

- *integritatea referirii* - într-o relație r_1 care referă o relație r_2 valorile cheii externe să figureze printre valorile cheii primare din relația r_2 sau să fie *null*.

În categoria, alte tipuri de restricții se pot menționa restricțiile de comportament şi dependențele funcționale. Pentru o anumită bază de date, utilizatorii pot defini mai multe tipuri de restricții de comportament: de domeniu, temporale, etc. De exemplu, în relația *local* o restricție de domeniu se poate referi la atributul CL, şi care impune ca valorile acestui atribut să se încadreze între anumite limite.

5.5 Indexarea relațiilor

Unul din avantajele sistemelor de gestiune este acela de a elibera proiectantul bazei de date de necesitatea de a cunoaşte detaliile de organizare a datelor, prin asigurarea *independenței datelor*. Totuşi, această independență nu este completă şi, în stadiul actual de realizare a sistemelor de baze de date, programatorii de aplicații trebuie să ia în considerație influența pe care modul de stocare şi de regăsire a datelor îl are asupra performanțelor aplicațiilor.

Într-o relație, privită ca o colecție de elemente în care nu sunt admise elemente duplicat (deoarece este o mulțime), operațiile de bază sunt, ca în orice colecție de elemente, *căutarea, inserarea* şi *ştergerea* unui element, cărora, desigur, le corespund operațiile (interogare, inserare, ştergere) din limbajele de manipulare a datelor în relații. Relațiile unei baze de date sunt memorate în fişiere pe disc, iar comenzile de manipulare a datelor sunt transformate de SGBD în operații asupra fişierelor care stochează relațiile bazei de date. Modul şi timpul de execuție a acestor operații de bază depind de modul de reprezentare a mulțimii de elemente (tupluri) ale relației.

În cazul unei mulțimi reprezentate printr-o colecție neordonată de elemente, timpul de căutare a unui element creşte proporțional cu numărul de elemente ale mulțimii, deoarece, în

cazul cel mai defavorabil, este necesar să fie parcurse toate elementele colecției pentru a găsi elementul dorit.

Timpul de căutare a unui element poate să fie micşorat semnificativ dacă elementele mulțimii sunt ordonate; de exemplu, la reprezentarea unei mulțimi ca arbore binar ordonat, limita asimptotică a timpului de căutare este în O(log N).

Pentru *inserarea* unui element într-o mulțime, trebuie mai întâi să fie căutat elementul respectiv în colecția (ordonată sau nu) prin care este reprezentată mulțimea respectivă; dacă există deja un element identic, se interzice inserarea noului element; dacă nu există, atunci se introduce noul element. Timpul de inserare a unui element într-o mulțime este compus din timpul de căutare al unui element, la care se adaugă timpul de memorare propriu-zis. Observații similare se pot face privind timpul de *ştergere* a unui element dintr-o mulțime.

În general, operațiile de *căutare, inserare* şi *ştergere* a elementelor într-o mulțime se execută mai rapid dacă elementele mulțimii sunt reprezentate printr-o *colecție ordonată*. Astfel se ajunge la situația că, deşi o relație nu presupune ordonarea tuplurilor sale, pentru accelerarea operațiilor de căutare, inserare şi ştergere, se folosesc colecții ordonate, ca de exemplu arbori binari ordonați sau tabele de dispersie (*hash table*).

Un index al unei relații (*index*) *este o structură auxiliară memorată în baza de date care permite accesul rapid la înregistrările* (*tuplurile*) *relațiilor prin ordonarea acestora.*

La definirea unei relații se stabilesc două categorii de indexuri: indexul primar al relației, care determină localizarea tuplurilor în fişierele bazei de date, şi zero, unul sau mai multe indexuri secundare, care nu modifică localizarea tuplurilor, dar sunt folosiți pentru regăsirea tuplurilor după un criteriu dat.

Index primar

Indexul primar (*primary index*) *se defineşte pe unul sau mai multe atribute ale relației şi reprezintă cheia* (*eticheta*) *după care ordonează tuplurile relației.*

Fiecare SGBD prevede o anumită modalitate de reprezentare a indexului primar. În continuare se va folosi termenul de *etichetă de ordonare*, iar termenul de *cheie de ordonare* va fi evitat, pentru a nu se confunda cu *cheia relației*, întrucât este posibil să nu reprezinte acelaşi lucru, deşi, în general, sistemele SGBD definesc în mod implicit indexul primar pe cheia primară a relației.

Operațiile de căutare sau ştergere în care se specifică valoarea atributului index primar se execută de asemenea eficient: se calculează mai întâi grupul căruia îi aparține tuplul, prin aplicarea funcției de dispersie h asupra valorii atributului index, apoi se caută poziția tuplului în lista înlănțuită corespunzătoare grupului. După aceasta, se continuă cu execuții specifice operației: se şterge sau se citeşte tuplul găsit. De exemplu, pentru rezolvarea interogării "Care este oraşul de domiciliu al studentului cu identificatorul 200?" se găseşte tuplul (200,Marin,Dan,Craiova) folosind valoarea indexului primar (200).

Dacă într-o operație de căutare sau ştergere nu se specifică valoarea atributului index primar, atunci căutarea unui anumit tuplu presupune parcurgerea (în cazul cel mai defavorabil) a tuturor listelor tuturor grupurilor tabelei de dispersie. De exemplu, interogarea "Care este oraşul de domiciliu al studentului cu numele Marin?" găseşte tuplul (200, Marin, Dan, Craiova) parcurgând toate tuplurile şi comparând valoarea atributului Nume cu constanta Marin, până este găsit tuplul care îndeplineşte această condiție.

Astfel de situații sunt frecvent întâlnite în aplicații, dat fiind că interogările se formulează de obicei folosind un nume, nu un identificator (care este probabil cheie primară, deci conține indexul primar, dar este necunoscut utilizatorilor). Pentru rezolvarea mai eficientă a unor astfel de interogări se pot defini indexuri secundare pe acele atribute care intervin frecvent în interogări.

Index secundar

Un index secundar pe un atribut A *al unei relații* (*secondary index*) *este o structură care conține o mulțime de perechi* (v,L) *ordonate după* v; *fiecare pereche corespunde unui tuplu al relației, v este valoarea atributului* A, *iar* L *este adresa tuplului respectiv în structura indexului primar al relației.*

Un index secundar nu modifică adresa de memorare a unui tuplu (care este conținută în structura indexului primar), dar conține informații suplimentare care permit identificarea rapidă a unui tuplu după valoarea atributului indexului.

Din punct de vedere al eficienței operațiilor de căutare în relații este avantajos să fie definite oricât de multe indexuri secundare, dar *fiecare index secundar ocupă spațiu de memorare suplimentar şi trebuie să fie actualizat* la fiecare operație de inserare şi ştergere a unui tuplu, ceea ce înseamnă timp de execuție suplimentar. În general, se recomandă utilizarea unui număr cât mai mic de indexuri secundare, definiți pe atributele care intervin cel mai frecvent în condițiile de interogare. Această decizie o ia proiectantul bazei de date prin analiza cerințelor din domeniul pentru care se dezvoltă baza de date şi aplicațiile corespunzătoare.

Dat fiind că indexurile sunt folosite în special pentru îmbunătățirea performanțelor bazelor de date şi nu fac parte din modelul relațional de bază, ei nu sunt cuprinşi în standardul limbajului SQL. Însă majoritatea sistemelor SGBD încorporează indexuri, conținând instrucțiuni pentru crearea indexurilor de forma:

CREATE [optiuni] INDEX nume_index ON tabel (lista_atribute);

Forma exactă şi opțiunile acestei instrucțiuni variază de la un sistem SGBD la altul. Una din opțiunile care se pot introduce în instrucțiunea CREATE INDEX este opțiunea UNIQUE, care specifică faptul că nu pot exista două tupluri cu aceeaşi combinație de valori ale atributelor indexului, deci acele atribute reprezintă o cheie unică a relației. Unele sisteme SGBD adaugă câte un index secundar pentru fiecare cheie unică, definită prin constrângerea UNIQUE din comanda CREATE TABLE.

BAZE DE DATE

CURS 6

LIMBAJE DE INTEROGARE A DATELOR PENTRU MODELUL RELAȚIONAL

6.1 Algebra relationala si extensiile sale
6.1.1 Operatii ale algebrei relationale
6.2 Calculul relational
6.2.1 Calculul relational orientat pe tupluri
6.2.2 Calculul relational orientat pe domenii
6.3 Criterii de optimizare a interogarilor

6.1 Algebra relațională şi extensiile sale

Modelul relațional oferă două colecții de operatori pe relații, şi anume: *algebra relațională (AR) şi calculul relațional (CR).* E. F. Codd a introdus algebra relațională (AR) ca o colecție de operații pe relații, fiecare operație având drept operanzi una sau mai multe relații şi având ca rezultat o relație.

Unele operații ale AR pot fi definite prin intermediul altor operații. În acest sens, putem vorbi de operații bază, precum: reuniunea, diferența, produsul cartezian, etc. dar şi de operații derivate: intersecția, diviziunea, etc.

Ulterior, la operațiile standard au fost adăugate şi alte operații, numite extensii ale AR standard, precum: complementarea, splitarea (spargerea), închiderea tranzitivă, etc.

În general, operațiile AR pot fi grupate în:

- operații tradiționale pe mulțimi (reuniunea, intersecția, diferența, produsul cartezian);
- operații relaționale speciale (selecția, proiecția, join-ul etc.).

În continuare vom prezenta principalele operații ale AR, precum şi modul lor de utilizare.

6.1.1 Operații ale algebrei relaționale

1. Selecția. Reprezintă o operație definită asupra unei relații *r*, şi constă în construirea unei relații *s*, cu schema identică cu cea a relației *r* şi a cărei extensie este constituită din acele tupluri din *r* care satisfac o condiție menționată explicit în cadrul operației. Condiția este în general de forma: < *atribut operator de comparație valoare*>.

Notațiile uzuale pentru această operație sunt: $\sigma_{conditie}(r)$, r[conditie] sau RESTRICT(r,conditie).

Considerând tuplurile ca transformări, operatorul de selecție se poate defini astfel:

$$s_{A=a}(r)=\{ t\in r \mid t(A)=a\}$$

Selecția $\sigma_{PD=Timisoara}$(*orar1*) aplicată relației *orar1* din Exemplul 5.1, ne conduce la următoarea relație:

NR	PD	PA	OD	OA
85	Timişoara	Bucureşti	07 :15	09 :25
90	Timişoara	Craiova	10 :15	13 :20

2. Proiecția. Reprezintă o operație definită asupra unei relații *r* şi constă în construirea unei relații *s*, în care se regăsec numai acele atribute din *r* specificate explicit în

cadrul operației. Suprimarea unor atribute din *r* poate avea ca efect apariția unor tupluri duplicate ce vor trebui eliminate. Prin operația de proiecție se trece de la o relație de grad *n* la o relație de grad *m*, mai mic decât cel inițial.
Notațiile uzuale pentru această operație sunt: $\Pi_{A1, A2, \dots, Am}(r)$, PROJECT(r, A_1, A_2, … ,A_m).
Considerând tuplurile ca transformări, operatorul de proiecție se poate defini astfel:

$$\Pi_X=\{ t(X) \mid t\in r\}$$

Aplicarea operatorului $\Pi_{PD,PA}(orar1)$ relației *orar1* din Exemplul 5.1, ne conduce la următoarea relație:

PD	PA
Craiova	Bucuresti
Bucureşti	Timişoara
Timişoara	Bucureşti
Timişoara	Craiova

3. Join(Joncțiunea sau unirea). Reprezintă o operație definită pe două relații *r* şi *s*, operație care constă din construirea unei noi relații *q*, prin concatenarea unor tupluri din *r* cu tupluri din *s*. Se concatenează acele tupluri din *r* şi *s* care satisfac o anumită condiție. Extensia relației *q* va conține acele tupluri care satisfac condiția de concatenare.

Notațiile uzuale pentru această operație sunt: r><$_{conditie}$s sau JOIN(s, r, condiție).

În general condiția de concatenare are următoarea formă:

< atribut din r operator de comparație atribut din s>.

În funcție de operatorul de comparație, join-ul poate fi de mai multe tipuri. Cel mai important tip, în sensul celei mai frecvente utilizări este *echijoin*-ul, care reprezintă o operație de join, dirijată de o condiție de forma următoare: *<atribut din r = atribut din s>.*

Definiția 6.1 *Fie relațiile r şi s de schemă R respectiv S, cu* A_i *Î R şi* B_i *Î S,* $dom(A_i)=dom(B_i)$*, i=1, ... ,m. Relația:*

$$q(RS)=\{ t : \exists\, t_r \in r,\ t_s \in s,\ \text{astfel încât}\ t(R)= t_r ,\ t(S)= t_s ,\ t(A_i)=t(B_i),\ i=1, \dots ,m \}$$

se numeşte ***echijoin-ul*** *relațiilor r şi s în raport cu* $A_1=B_1=,\dots,=A_m=B_m$ *şi se notează cu* $r[A_1=B_1=,\dots,=A_m=B_m]\ s$.

Considerăm relația *oras*, de forma următoare:

oras

PD	JUDET
Timisoara	Timis
Craiova	Dolj
Oradea	Bihor

Operația *orar1*><$_{PD=PD}$*oras* aplicată relațiilor *orar1* din Exemplul 5.1 şi *oras* definită mai sus, ne conduce la următoarea relație:

NR	PD	PA	OD	OA	PD	JUDET
75	Craiova	Bucuresti	07 :15	08 :25	Craiova	Dolj
85	Timişoara	Bucureşti	07 :15	09 :25	Timişoara	Timiş
90	Timişoara	Craiova	10 :15	13 :20	Timişoara	Timiş

Operația de join se poate exprima cu ajutorul operațiilor de produs cartezian şi selecție, rezultatul unui join fiind acelaşi cu rezultatul unei selecții operate asupra unui produs cartezian, adică:

JOIN(r,s,condiție)=RESTRICT(PRODUCT(r,s), condiție)

Produsul cartezian reprezintă o operaţie laborioasă şi foarte costisitoare, ceea ce face ca în locul produsului să fie utilizat join-ul ori de câte ori acest lucru este posibil.

În cazul echijoin-ului, schema relaţiei rezultat, conţine toate atributele celor doi operanzi şi de aceea vor exista cel puţin două valori egale în fiecare tuplu. *Join-ul natural* elimină această redundanţă, fiind definită în mod similar cu echijoin-ul cu observaţia că atributele cu acelaşi nume se trec o singură dată în relaţia rezultat iar extensia se compune din concatenarea tuplurile lui *r* cu tuplurile lui s care prezintă aceleaşi valori pentru atributele cu acelaşi nume. Notaţia uzuală pentru această operaţie este: r><s.

Joinul natural al relaţiilor *orar1* din Exemplul 5.1 şi *oras* definită mai sus, ne conduce la următoarea relaţie:

NR	PD	PA	OD	OA	JUDET
75	Craiova	Bucuresti	07 :15	08 :25	Dolj
85	Timişoara	Bucureşti	07 :15	09 :25	Timiş
90	Timişoara	Craiova	10 :15	13 :20	Timiş

Join extern. Atunci când relaţiile care participă la join nu au proiecţii identice pe atributul de joncţiune (atributul nu are aceleaşi valori în relaţiile care se joncţionează), operaţia de join conduce la pierderea de tupluri, cel puţin dintr-o relaţie. Pentru a evita pierderile de informaţie a fost definit join-ul extern, ca operaţie prin care din două relaţii *r* şi *s* se obţine o nouă relaţie *q* prin join-ul relaţiilor *s* şi *r* , relaţie la care se adaugă şi tuplurile din *r* şi *s* care nu au participat la join. Aceste tupluri sunt completate în relaţia *q* cu valori "null" pentru atributele relaţiei corespondente (*r*, respectiv *s*).

Notaţiile uzuale pentru desemnarea unei join extern sunt: r><•s sau EXT-JOIN(r, s).

Join-ul extern al relaţiilor *orar1* şi *oras* conduce la următoarea relaţie:

NR	PD	PA	OD	OA	JUDET
75	Craiova	Bucuresti	07 :15	08 :25	Dolj
80	Bucureşti	Timisoara	17.30	19.30	Null
85	Timişoara	Bucureşti	07 :15	09 :25	Timiş
90	Timişoara	Craiova	10 :15	13 :20	Timiş
Null	Oradea	Null	Null	Null	Bihor

Semi-join. Această operaţie a fost introdusă de Bernstein P. A., fiind necesară la definirea procesului de optimizare a interogărilor. Semi-joncţiunea conservă atributele unei singure relaţii participante la join şi reprezintă o operaţie pe două relaţii *r* şi *s* în urma căreia rezultă o nouă relaţie ce conţine numai tuplurile din relaţia *r* care participă la join. Notaţiile uzuale pentru această operaţie sunt: r >< s sau SEMIJOIN(r,s).

Astfel, *orar1* >< *oras* conduce la următoarea relaţie:

NR	PD	PA	OD	OA
75	Craiova	Bucuresti	07 :15	08 :25
85	Timişoara	Bucureşti	07 :15	09 :25
90	Timişoara	Craiova	10 :15	13 :20

4. Diviziunea. Reprezintă o operaţie din AR definită asupra unei relaţii *r* de schemă: $R(A_1:D_1, \ldots ,A_p:D_k,A_{p+1}:D_1, \ldots ,A_n:D_m)$, operaţie care constă din construirea, cu ajutorul unei relaţii $s(A_{p+1}:D_1, \ldots ,A_n:D_m)$ a relaţiei $q(A_1:D_1, \ldots ,A_p:D_k)$. Tuplurile relaţiei *q*, concatenate cu tuplurile relaţiei *s* permit obţinerea tuplurilor relaţiei *r*.

Notațiile uzuale pentru această operație sunt: r ÷s sau DIVISION(r,s).

Definiția 6.2 *Fie r şi s două relații de schemă R respectiv S, cu* $S \subset R$ *şi* $R'=R - S$. *Relația:* $r'(R')=\{t: \forall\ t_S \in s,\ \exists\ t_r \in r$ *astfel încât* $t_r(S)=t_S,\ t_r(R')=t\}$ *se numeşte* ***diviziunea*** *relației r la s.*

Operația de diviziune este o operație derivată, care se poate exprima prin intermediul diferenței, produsului cartezian şi proiecției astfel:

$$r \div s=\Pi_{A1, A2, \ldots ,Ap}(r) - \Pi_{A1, A2, \ldots ,Ap}((\Pi_{A1, A2, \ldots ,Ap}(r) \times s) -r)$$

Considerăm relațiile *drept* şi *tip* de forma următoare:

drept

Pilot	Tip avion
Dan	AIRBUS
Dan	TU154
Ion	TU154
Ion	AIRBUS
Mihai	TU154

tip

Tip avion
AIRBUS
TU154

Dacă dorim să obținem piloții care au drept de zbor pe toate tipurile de avioane, calculăm *drept¸tip*, şi rezultă relația:

Pilot
Dan
Ion

5. Complementarea. Complementul unei relații reprezintă mulțimea 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 să fie destul de rar utilizată.

Notațiile uzuale pentru această operație sunt: ¬r, NOT(r) sau COMP(r).

Să considerăm, de exemplu, pentru relația *drept* definită la operația de diviziune, următoarele valori ale domeniilor:

Pilot={Dan, Ion, Mihai, Andrei}

Tip avion={AIRBUS, TU154, IAR500}

Complementul relației *drept* este:

Pilot	Tip avion
Dan	IAR500
Ion	IAR500
Mihai	AIRBUS
Mihai	IAR500
Andrei	IAR500
Andrei	AIRBUS
Andrei	TU154

6. Splitarea (spargerea). Este o operație adițională din AR definită asupra unei relații *r*, şi care, pe baza unei condiții definite asupra atributelor lui *r* permite construirea a două relații r_1 şi r_2, cu o aceeaşi schemă cu *r*. Extensia lui r_1 conține tuplurile din *r* care îndeplinesc condiția specificată, iar r_2 pe cele care nu verifică această condiție.

Pentru relația *drept* definită mai sus şi condiția Pilot="Dan", operația de splitare produce ca rezultat relațiile *drept1* şi *drept2*:

drept1

Pilot	Tip avion
Dan	AIRBUS
Dan	TU154

drept2

Pilot	Tip avion
Ion	TU154
Ion	AIRBUS
Mihai	TU154

7. Închiderea tranzitivă. Este o operație adițională din AR prin care se pot adăuga noi tupluri la o relație. Operația de închidere tranzitivă presupune efectuarea în mod repetat a secvenței de operații: join-proiecție-reuniune. Numărul de execuții depinde de conținutul relației. Închiderea tranzitivă se defineşte asupra unei relații *r*, a cărei schemă conține două atribute A_1 şi A_2 cu acelaşi domeniu, şi constă în adăugarea la relația *r* a tuplurilor care se obțin succesiv prin tranzitivitate, în sensul că, dacă există în *r* tuplurile <*a*, *b*> şi <*b*, *c*> se va adăuga la *r* şi tuplul <*a*, *c*>. Notațiile uzuale pentru această operație sunt: $\tau(r)$, r^+, CLOSE(r).

Prezentăm mai jos, o relație *legatura* ce ne arată legăturile aeriene între anumite localități precum şi închiderea tranzitivă a relației *t(legatura)*.

Legatura

PD	PA
Bucuresti	IaşI
Bucureşti	Timişoara
Timişoara	Arad
Timişoara	Craiova

t(legatura)

PD	PA
Bucuresti	Iaşi
Bucureşti	Timişoara
Timişoara	Arad
Timişoara	Craiova
Bucureşti	Arad
Bucureşti	Craiova

Algebra relațională

O *expresie* a AR este constituită dintr-o serie de relații, legate prin operații din AR. Să considerăm, de exemplu două relații *r* şi *s* cu schemele *r*(*A*, *B*) şi *s*(*B*, *C*). Cu ajutorul acestor relații putem defini o expresie E_1, astfel: $E_1=\Pi_C(\sigma_{A=a}(r><s))$.

În formularea unei expresii se pot introduce relații intermediare. De exemplu, expresia E_1 se poate reprezenta cu ajutorul unor relații intermediare X_1 şi X_2, astfel:

X_1= r><s

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

$E_1=\Pi_C(X_2)$

Prin calcularea unei expresii algebrice *E* se obține o relație unică. Prin urmare, *E* reprezintă o transformare a unei mulțimi de relații într-o singură relație. În expresii se admit paranteze rotunde şi se presupune că nici un operator binar nu are prioritate în execuție față de un altul cu excepția intersecției fața de reuniune.

Fie *U* mulțimea tuturor atributelor care descriu un univers, *D* o mulțime de domenii şi fie *dom* o funcție completă din *U* în *D*. Fie în continuare $\boldsymbol{R}=\{R_1, R_2,..., R_p\}$ o mulțime de scheme de relații diferite, unde $R_i \subset U$, $1 \leq i \leq p$ şi $d=\{r_1, r_2,..., r_p\}$ o mulțime de relații astfel încât relația r_i este de schemă R_i. Fie Q o mulțime de comparatori definiți pe domeniile din *D* care conține cel puțin o relație de egalitate şi de inegalitate pentru fiecare domeniu.

Definiția 6.3. *Se numeşte algebra relațională în raport cu U, D, dom, R, d şi Q septetul B=(U, D, dom, R, d,**Q**, **O**), unde **O** este mulțimea operatorilor enunțați mai sus ce folosesc atributele din U şi relațiile din d.*

Definiția 6.4. *Se numeşte expresie algebrică în raport cu B orice expresie corect construită din relații care aparțin lui d şi relații constante cu schemă din U care folosesc operatori din **O**.*

Schema unei expresii depinde de schemele relațiilor care o compun.

Notăm cu *sch(E)* schema expresiei algebrice *E*, care se poate defini recursiv conform următoarelor reguli :

1. Dacă E este r_i atunci $sch(E)=R_i$.
2. Dacă $E=E_1 \cup E_2$, $E_1 \cap E_2$, $E_1 - E_2$, sau $\sigma_c(E_1)$ unde c reprezintă condiții, atunci $sch(E)=sch(E_1)$.
3. Dacă $E=\Pi_x(E_1)$ atunci $sch(E)=X$.
4. Dacă $E=E_1 >< E_2$, atunci $sch(E)= sch(E_1) \cup sch(E_2)$.
5. Dacă $E=E_1 \div E_2$ atunci $sch(E)= sch(E_1) - sch(E_2)$.

Două expresii sunt echivalente, dacă în urma evaluăriilor se obține ca rezultat aceeaşi relație.

De exemplu, expresiile:

$E_1=\Pi_C(\sigma_{A=a}(r_1><r_2)$

$E_3=\Pi_C(\Pi_B(\sigma_{A=a}(r_1))><r_2)$

sunt echivalente, considerând $r_1(A, B)$ şi $r_2(B, C)$.

6.2 Calculul relațional

Calculul relațional (CR) reprezintă o adaptare a calculului cu predicate la domeniul bazelor de date relaționale. Ideea de bază o constituie identificarea unei relații 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 AR, CR permite definirea neprocedurală, "declarativă" a relațiilor, în sensul precizării lor prin intermediul proprietăților tuplurilor şi nu prin maniera de derivare efectivă acestor tupluri.

6.2.1 Calculul relațional orientat pe tuplu (CRT)

Reprezintă varianta inițială, introdusă de Codd E., în care CR utilizează variabile definite asupra relațiilor, variabile ale căror valori reprezintă tupluri de relație. Din acest motiv, variabilele au fost denumite *variabile tuplu*, iar calculul relațional a primit numele de *calculul relațional orientat pe tuplu*.

Cea mai simplă construcție a calculului relațional se numeşte *atom* (sau *formulă atomică*). Un atom este constituit din termeni (constante, variabile tuplu şi operatori) şi poate avea una din următoarele forme:

- *r(v)*, unde *r* numele unuei relații, *v* variabilă tuplu reprezentând un tuplu al relației *r*. De exemplu, *orar(z)*.
- *v[i] comp w[j]*, unde *v* şi *w* sunt variabile tuplu iar *comp* este un operator de comparare (<, =, <=, >, >=, <>). Semnificația atomului este: a *i*-a componentă a tuplului *v* se află în relația *comp* cu a *j*-a componentă a tuplului *w*. De exemplu, v[2]=w[3].

• $v[i]$ *comp* k sau k *comp* $v[i]$, unde v variabilă tuplu, *comp* este un operator de comparare iar k o constantă. Semnificația atomului este: a i-a componentă a tuplului v se află în relația *comp* cu constanta k. De exemplu, $v[2]>5$ sau $5< v[2]$.

Pe baza atomilor cu ajutorul unor operatori se pot construi formule mai complexe. În cadrul calculului relațional orientat pe tuplu sunt utilizați următorii operatori: conectorii uzuali (conjuncția, disjuncția, negația) precum şi cuantificatorii universali ($\forall$) şi existențiali ($\exists$).

Se numeşte *variabilă tuplu liberă* o variabilă asupra căreia nu acționează nici un cuantificator. O *variabilă tuplu legată* reprezintă o variabilă asupra căreia acționează un cuantificator universal sau existențial.

Dacă F_1 şi F_2 sunt formule, atunci $F_1 \wedge F_2$, $F_1 \vee F_2$, $\neg F_1$, $\neg F_2$, $(\exists s F_1)$, $(\exists s F_2)$, $(\forall v F_1)$ şi $(\forall v F_2)$ sunt formule, în care s şi v sunt variabile tuplu care apar în F_1 respectiv F_2.

Se numeşte *expresie* a calculului relațional orientat pe tuplu o construcție E de forma:

$$E=\{ t \mid F(t)\}$$

unde F reprezintă o formulă din calculul relațional orientat pe tuplu, iar t este o variabilă tuplu şi anume singura variabilă tuplu liberă din formula F.

Ca şi expresiile din AR, expresiile din calculul relațional orientat pe tuplu reprezintă definiții ale unor relații. În forma prezentată anterior, aceste expresii permit exprimarea unor relații infinite, adică relații cu un număr infinit de tupluri.

De exemplu, expresia: $E_i=\{ t \mid \neg r(t)\}$ semnifică relația formată din toate tuplurile care nu aparțin lui r. Deoarece este imposibil de precizat "toate tuplurile posibile", se impune o definiție mai clară a expresiilor din calculul relațional orientat pe tuplu.

Se numeşte *expresie bine formată* o expresie de forma:

$$E=\{ t \mid F(t)\}$$

unde F reprezintă o formulă din calculul relațional orientat pe tuplu, iar t este singura variabilă liberă din formula F, şi în plus fiecare componentă a lui t este un element al lui DOM(F). DOM(F) reprezintă o mulțime de simboluri care, fie apar explicit în F, fie sunt componente ale tuplurilor unei relații r, menționată în F. Mulțimea DOM(F) este finită.

O expresie din calculul relațional orientat pe tuplu se consideră bine formată dacă satisface următoarele condiții:

- Fiecare componentă a lui t aparține lui DOM(F).
- Dacă într-o expresie de forma: $(\exists v)F(v)$, fiecare componentă a variabilei v aparține lui DOM(F), atunci v satisface F.
- Dacă într-o expresie de forma: $(\forall v)F(v)$, fiecare componentă a variabilei v aparține lui DOM(F), atunci v satisface F.

O expresie din calculul relațional orientat pe tuplu reprezintă definiția unei relații, definiție formulată prin intermediul proprietăților pe care le au tuplurile care compun relația. De exemplu, considerând relațiile r_1 şi r_2, cu schemele $R_1(A, B)$ şi $R_2(B, C)$ putem defini o expresie bine formată E_e, astfel:

$$E_e=\{ t \mid (\exists v)\ (\exists s)\ (r(t) \wedge r_1(v) \wedge r_2(s) \wedge (v[1]=a) \wedge (s[1]= v[2]) \wedge (t[1]= s[2])\ \}$$

Expresia E_e reprezintă definiția unei relații care conține ca tupluri acele valori ale atributului C care au asociate în join-ul relațiilor r_1 şi r_2 valoarea "*a*" pentru atributul A. Se observă că expresia E_e exprimă proprietățile tuplurilor care intră în componența unei relații şi nu modul de derivare efectivă a acestei relații, aşa cum este cazul expresiilor E_1 şi E_3 definite mai sus, care sunt definiții procedurale ale relației E_e.

Exemplul 6.1. Considerăm relațiile *orar1* şi *oras* definite mai sus. Dacă dorim să aflăm județul în care se află un anumit Punct de decolare(PD), de exemplu Timişoara, putem să utilizăm expresia E_1. Aceasta se rescrie astfel:

$$E_1=\Pi_{\text{JUDET}}(\sigma_{\text{PD=Timisoara}}(orar1 >< oras))$$

Prin join-ul natural *orar1><oras* se obține relația *cursa*:

cursa

NR	PD	PA	OD	OA	JUDET
75	Craiova	Bucuresti	07 :15	08 :25	Dolj
85	Timişoara	Bucureşti	07 :15	09 :25	Timiş
90	Timişoara	Craiova	10 :15	13 :20	Timiş

Prin selecţia $s_{PD=Timisoara}(cursa)$ se obţine relaţia:

aero

NR	PD	PA	OD	OA	JUDET
85	Timişoara	Bucureşti	07 :15	09 :25	Timiş
90	Timişoara	Craiova	10 :15	13 :20	Timiş

În final, prin $\Pi_{JUDET}(aero)$ se obţine relaţia:

JUDET
Timis

Aceeaşi problemă o putem rezolva şi prin evaluarea expresiei E_e, care se rescrie astfel:

$$E_e=\{ t \mid (\exists v)\ (\exists s)\ (r(t) \wedge orar1(v) \wedge oras(s) \wedge (v[1]=Timisoara) \wedge (s[1]= v[2]) \wedge (t[1]= s[2]) \}$$

Se observă faptul că $s[1]$ identifică atributul PD din relaţia *oras,* $v[2]$ identifică atributul PD, din relaţia *orar1*, deci se poate realiza join-ul natural al celor două relaţii şi apoi pe rezultatul join-ului se aplică selecţia $v[1]$=Timisoara, iar pe acest rezultat se identifică $t[1]$ cu atributul $s[2]$ (adică JUDET) şi proiecţia pe acest atribut conduce la relaţia:

JUDET
Timis

Deci, cele două expresii conduc la acelaşi rezultat.

6.2.2 Calculul relaţional orientat pe domeniu(CRD)

Calculul relaţional orientat pe domeniu utilizează în construcţiile sale aceiaşi operatori ca şi calculul orientat pe tuplu, dar variabilele care apar în aceste construcţii sunt *variabile domeniu*, adică variabile definite asupra domeniilor.

O *formulă atomică* reprezintă o construcţie elementară din calculul relaţional orientat pe domeniu care poate avea una din formele:

· $r(x_1, x_2, \dots ,x_n)$, unde r este o relaţie n-ară şi x_i, i=1, … ,n sunt constante sau variabile domeniu. Semnificaţia atomului este în acest caz, următoarea: "Valorile variabilelor x_i trebuie alese astfel încât $< x_1, x_2, \dots ,x_n>$ să fie un tuplu al relaţiei r".

· x *comp* y, unde x şi y sunt constante sau variabile domeniu, iar "comp" este un operator de de comparaţie (<, =, <=, >, >=, <>). În această formă, atomul are semnificaţia : "Variabilele x şi y trebuie să aibă acele valori care să facă expresia *x comp y* adevărată".

O *formulă compusă* se defineşte similar calculului relaţional orientat pe tuplu.

O *expresie* din calculul relaţional orientat pe domeniu este o construcţie de forma:

$$E=\{ x_1x_2\dots x_n \mid F(x_1x_2\dots x_n \}$$

unde $x_1x_2\dots x_n$ sunt singurele variabile libere din F.

Să considerăm, de exemplu două relaţii binare r_1 şi r_2, cu ajutorul cărora definim următoarea expresie din calculul relaţional orientat pe domeniu:

$$E=\{xy \mid r_1(xy) \wedge (\forall z)\ (\neg r_2(xy) \wedge \neg r_2(yz))\}$$

Expresia E reprezintă definiția unei relații, constituită din acele tupluri ale relației r_1 pentru care nici una din componente nu figurează pe prima poziție în tuplurile din relația r_2.

Astfel, pentru două relații *ruta1* și *ruta2* definite mai jos, expresia E se scrie:

$$E=\{xy / ruta1(xy) \wedge (\forall z)\ (\neg ruta2(xy) \wedge \neg ruta2(yz))\}$$

ruta1

PD	PA
Arad	Cluj
Iasi	Bucuresti
Timisoara	Iasi

ruta2

PD	PA
Timisoara	Bucuresti
Oradea	Bucuresti
Constanta	Oradea

Evaluarea expresiei E conduce la următoarea relație:

ruta

PD	PA
Arad	Cluj
Iasi	Bucuresti

O expresie bine formată din calculul relațional orientat pe domeniu se defineşte similar expresiei bine formată din calculul relațional orientat pe tuplu.

După cum se poate observa, între formulele din CRD şi CRT nu există deosebiri de fond, ci doar de notație (variabilelor de domeniu din CRD le corespund componentele variabilelor de tuplu din CRT şi invers). Orice formulă din CRT se poate translata într-o formulă din CRD şi deci într-o expresie algebrică. Deci, folosind algebra relațională, sau formule din CRD sau din CRT se pot exprima aceleaşi interogări şi deci toate aceste formalisme sunt echivalente din punctul de vedere al puterii de expresie. Limbajul de interogare SQL este la fel de expresiv ca Algebra relațională, CRT şi CRD.

6.3 Criterii de optimizare a interogărilor

Utilizarea limbajelor relaționale de manipulare bazate pe calculul relațional sau a celor bazate pe mapping face necesară prezența, unor mecanisme care să transforme cererile într-o formă adaptată prelucrărilor eficiente. Aceste mecanisme sunt de fapt nişte mecanisme de rescriere a cererilor de date şi sunt necesare pentru optimizarea prelucrărilor de BD, cât şi la realizarea bazelor de date distribuite, la care se folosesc mai multe tipuri de limbaje relaționale.

Optimizarea cererilor de date se realizează prin parcurgerea următoarelor etape:

- exprimarea cererilor sub forma unor expresii algebrice relaționale;
- aplicarea unor transformări algebrice asupra expresiilor obținute în etapa precedentă, în scopul obținerii unor expresii echivalente celor inițiale, dar care să fie executate mai eficient.

Exprimarea cererilor de date sub forma unor expresii din algebra relațională are la bază echivalența dintre calculul relațional şi algebra relațională.

Procesul de transformare a cererilor de date se realizează conform unor *strategii de optimizare.*

Operațiile din algebra relațională, posedă o serie de proprietăți care permit implementarea strategiilor generale de optimizare.

Aceste proprietăți sunt:

P1. Comutativitatea operațiilor de join şi produs cartezian

$E_1 >< E_2 \equiv E_2 >< E_1$

$E_1 \times E_2 \equiv E_2 \times E_1$

P2. Asociativitatea operațiilor de join şi produs cartezian

$(E_1 ><E_2) ><E_3 \equiv E_1 >< (E_2 ><E_3)$

$(E_1 \times E_2) \times E_3 \equiv E_1 \times (E_2 \times E_3)$

P3. Compunerea proiecțiilor

$\Pi_{A1,\ldots,An} (\Pi_{B1,\ldots,Bm}(E)) \equiv \Pi_{A1,\ldots,An} (E)$

unde: $A_1,..,A_n$ trebuie să aparțină lui $B_1,..,B_m$

P4. Compunerea selecțiilor

$\sigma_{F1} (\sigma_{F2}(E)) \equiv \sigma_{F1 \wedge F2} (E)$

Deoarece $F\,1 \wedge F\,2 = F\,2 \wedge F\,1$, selecțiile se pot comuta:

$\sigma_{F1} (\sigma_{F2} (E)) \equiv \sigma_{F2} (\sigma_{F1} (E))$

P5. Comutarea selecției cu proiecția

Dacă, condiția F implică numai atributele $A_1,..,A_n$, atunci:

$\Pi_{A1,\ldots,An} (\sigma_F (E)) \equiv \sigma_F (\Pi_{A1,\ldots,An} (E))$

Dacă, condiția F implică şi atributele $B_1,\ldots,B_m$, care nu aparțin de $A_1,\ldots,A_n$, atunci:

$\Pi_{A1,..An} (\sigma_F (E)) \equiv \Pi_{A1,\ldots,An} (\sigma_F (\Pi_{A1,..An,\ B1,..,Bm} (E)))$.

P6. Comutarea selecției cu produsul cartezian

Dacă toate atributele menționate în F sunt atribute ale lui E_1 atunci:

$\sigma_F (E_1 \times E_2) \equiv \sigma_F (E_1) \times E_2$

Dacă, în plus F este de forma $F=F_1 \wedge F_2$ şi F_1 implică numai atribute din E_1, iar F_2 implică numai atribute din E_2, atunci: $\sigma_F (E_1 \times E_2) \equiv \sigma_{F1} (E_1) \times \sigma_{F2} (E_2)$.

Dacă F_1 implică numai atributele din E_1, dar F_2 implică atribute aâtât din E_1 cât şi din E_2, atunci:

$\sigma_F (E_1 \times E_2)\equiv \sigma_{F2}(\sigma_{F1}(E_1) \times E_2)$.

P7. Comutarea selecției cu reuniunea

$\sigma_F(E_1 \cup E_2) \equiv \sigma_F (E_1) \cup \sigma_F (E_2)$

P8. Comutarea selecției cu diferența

$\sigma_F(E_1-E_2) \equiv \sigma_F (E_1)-\sigma_F (E_2)$

P9. Comutarea proiecției cu produsul cartezian

Dacă $A_1,\ldots,A_n$ reprezintă o listă de atribute din cadrul a două expresii relaționale: E_1 şi E_2, listă formată din atributele aparținând lui E_1 şi notate cu: $B_1,\ldots,B_m$ şi din atributele aparținând lui E_2 notate cu: $C_1,\ldots,C_k$, atunci:

$\Pi_{A1,..,An}(E_1 \times E_2) \equiv \Pi_{B1,\ldots,Bm}(E_1) \times \Pi_{C1,\ldots,Ck}(E_2))$

P10. Comutarea proiecției cu reuniunea

$\Pi_{A1,..,An}(E_1 \cup E_2) \equiv \Pi_{A1,..,An}(E_1) \cup \Pi_{A1,..,An}(E_2)$

Aceste transformări permit definirea unor strategii generale de optimizare a cererilor de date, strategii care se referă la posibilitățile de creştere a performanțelor de execuție a cererilor de date prin reordonarea operațiilor implicate de aceste cereri. De exemplu, prin deplasarea

operațiilor de selecție cât mai în stânga expresiilor algebrice se reduce numărul de tupluri care trebuie manipulate în procesul de executare a cererii.

Se pot menționa următoarele **strategii generale de optimizare** a cererilor de date:

• Deplasarea operațiilor de selecție înaintea operațiilor de join. Join-ul şi produsul cartezian acționează ca generatori de tupluri. Prin selecție se reduce dimensiunea relațiilor la care se aplică aceşti generatori de tupluri, deci şi a rezultatelor obținute. Pentru deplasarea selecției în fața join-ului se vor aplica proprietățile menționate anterior, ținând cont de faptul că un join poate fi exprimat sub forma unui produs cartezian urmat de o selecție, iar în cazul join-ului natural printr-un produs cartezian urmat de o selecție şi o proiecție.

• Deplasarea operațiilor de proiecție înaintea operațiilor de join. Este realizată cu ajutorul proprietății de comutare a selecției cu produsul cartezian. Prin deplasarea proiecției în fața join-ului se obțin o serie de avantaje şi anume:

- dacă proiecția lasă intactă cheia, operația de join se poate realiza mai rapid întrucât s-a redus numărul de tupluri ce trebuie memorate şi eventual sortate;

- proiecția implică eliminarea tuplurilor duplicate, care se poate realiza relativ uşor, de exemplu cu ajutorul unui index. După eliminarea tuplurilor duplicate, join-ul va fi mai simplu, pentru că relațiile sunt mai mici.

• Combinarea selecțiilor multiple. Se realizează cu ajutorul relației de compunere a selecțiilor. Nu toate selecțiile sunt la fel de uşor de realizat. Din această cauză, numai selecțiile ″mai rapide″ trebuie grupate şi mutate în fața operațiilor de join.

Selecțiile rapide sunt cele realizate cu ajutorul unor tehnici de acces direct(indexare, hash, etc). În acest caz, timpul pentru realizarea selecțiilor va depinde numai de numărul de tupluri selectate, nu şi de mărimea întregii relații.

Nu se recomandă gruparea selecțiilor rapide cu cele lente. De aceea, este necesar, ca înainte de combinarea selecțiilor să se estimeze viteza de realizare a fiecărei selecții în parte.

• Deplasarea selecțiilor în fața proiecțiilor. Regula de mutare a selecțiilor în fața proiecțiilor este dată de proprietatea de comutare a selecției cu proiecția. Realizarea deplasării este utilă atunci când:

- există o serie de facilități în realizarea selecției, precum accesul direct, facilități care ar putea fi inhibate prin operația de proiecție ;

- eliminarea tuplurilor duplicate obținute prin proiecție se realizează prin sortarea tuplurilor. Selecția reduce numărul de tupluri care trebuie sortate, facilitând astfel realizarea operației de proiecție.

BAZE DE DATE

CURS 7

RESTRICȚII DE INTEGRITATE ÎN BAZELE DE DATE

7.1 Dependente functionale
7.1.1 Introducere
7.1.2 Axiome de inferenta
7.1.3 Completitudinea sistemului de inferente
7.1.4 Secvențe derivate
7.2 Dependente multivoce
7.3 Dependente generalizate

7.1 Dependențe funcționale

7.1.1 Introducere

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", şi stau la baza teoriei optimizării structurii relaționale a datelor, respectiv teoria normalizării relațiilor.

Când descriem un univers real sau conceptual printr-un model relațional ne confruntăm cu următoarea problemă: *Care este mulțimea de relații care va fi o reprezentare fidelă a schemei conceptuale şi deci a universului real fără ca să riscăm problemele de consistență.*

Plecând de la regulile semantice care traduc restricțiile universului modelat, proiectantul trebuie să definească "dependențele " şi să le introducă în definiția schemei de relație. Proprietățile dependențelor sunt proprietăți pentru schema de relație a bazei de date şi nu pentru o extensie oarecare, adică aceste dependențe constituie invarianți care trebuie să fie satisfăcute de toate extensiile legale de relații ale schemei.

Crearea unei baze de date are două obiective esențiale: să reducă *excedentul de date* şi să mărească *siguranța* în exploatare. Orice cunoştință apriori despre diferite tipuri de restricții referitoare la totalitatea datelor poate fi de mare folos pentru atingerea acestor obiective.

Un procedeu de formalizare a unor cunoştințe despre date este dat de *dependențe.*În această secțiune vom considera numai un singur tip de dependențe şi anume dependența funcțională, pe care o vom numi F-dependență, care este *o generalizare a noțiunii de cheie.*

Exemplul 7.1. Fie relația grafic(PILOT CURSA DATA ORA-DECOLARE) care arată ce pilot participă la o cursa dată, la o anumită dată şi ora la care are loc decolarea.

grafic(PILOT CURSA DATA ORA-DECOLARE)

PILOT	CURSA	DATA	ORA-DECOLARE
Dragoş	75	9-03	10:15
Dragoş	80	10-03	13:25
Mihai	80	8-03	13:25
Mihai	87	12-03	18:50
Mihai	75	11-03	10:15
Mircea	75	13-03	10:15
Mircea	80	12-03	13:25
Costin	85	9-03	5:50
Costin	85	13-03	5:50
Costin	90	5-03	13:25

Se au în vedere următoarele restricţii:

-o cursă are aceeaşi ora de decolare (cursa 85 decolează întotdeauna la ora 5:50);

- un pilot nu se poate afla decât într-o singură cursă la o anumită ora de decolare.

Aceste restricţii sunt exemple de F-dependenţe. Intuitiv vorbind, o F-dependenţă are loc când valorile tuplurilor după o mulţime de atribute definesc în mod unic valorile unei alte mulţimi de atribute. Astfel, restricţiile arătate mai sus pot fi astfel formulate:

- ORA-DECOLARE depinde funcţional de CURSA ,
- CURSA depinde funcţional de { PILOT DATA ORA-DECOLARE },
- PILOT depinde funcţional de { CURSA DATA }.

În mod obişnuit, ordinea acestor secvenţe se poate schimba şi spunem că:

{ CURSA DATA} defineşte în mod funcţional pe { PILOT }, sau simbolic

{CURSA DATA} $\rightarrow$ { PILOT }.

Vom da o definiţie strictă a dependenţei funcţionale folosind operatorii relaţionali.

Definiţia 7.1. *Fie relaţia r de schemă R, X şi Y submulţimi ale lui R. Relaţia r satisface dependenţa funcţională X®Y dacă* $P_Y(s_{X=x}(r))$ *are nu mai mult de o singură ocurenţă pentru fiecare X-valoare x.*

În F-dependenţa X$\rightarrow$Y submulţimea X se numeşte partea stângă iar Y se numeşte partea dreaptă.

Un procedeu de a verifica că o relatie satisface X$\rightarrow$Y este dat de următoarea proprietate :

Dacă pentu orice două tupluri $t_1, t_2 \in r$*, cu* $t_1(X) = t_2(X)$ *rezultă* $t_1(Y)=t_2(Y)$ *atunci r satisface X®Y.*

Această caracterizare stă la baza procedurii **satisfie** dată mai jos. Presupunem că relaţia r este ordonată după valorile lui X , apoi după ale lui Y.

```
0  procedure satisfie(X,Y;v)
1  v←true
2  Input (t)
3  While not eof (r)
   3.1  Input { t'}
   3.2  If  t(X) = t'(X)
        then
        3.2.1 If  t(Y)≠t'(Y)
              then
               3.2.1.1  v←False
       3.2.2  continue
       else
      3.2.3  t←t'
   3.3   continue
4  Return
```

Tabelul de mai jos arată rezultatul aplicării algoritmului **satisfies** relaţiei **grafic.**

grafic(PILOT CURSA DATA ORA-DECOLARE)

PILOT	CURSA	DATA	ORA-DECOLARE
Dragoş	75	9-03	10:15
Mihai	75	11-03	10:15
Mircea	75	13-03	10:15

Dragoş	80	10-03	13:25
Mircea	80	12-03	13:25
Mihai	80	8-03	13:25
Costin	85	9-03	5:50
Costin	85	13-03	5:50
Mihai	87	12-03	18:50
Costin	90	15-03	13:25

Rezultatul aplicării algoritmului satisfie este True.

7.1.2 Axiome de inferență

Pentru orice relație r(R) există la un moment dat o familie de F-dependențe pe care le satisface r. Trebuie remarcat că, o stare a unei relații poate satisface o mulțime anumită de F-dependențe iar o altă stare poate să nu o satisfacă.

Trebuie să determinăm familia ***F*** de F-dependențe care satisface toate stările admisibile ale relației. Pentru a determina ***F*** sunt necesare cunoştințe semantice despre relația r. De aceea trebuie să considerăm că familia ***F*** de F-dependențe este dată pentru schema de relație R. În acest caz orice relație r(R) trebuie să satisfacă toate F-dependențele din ***F***. Nu este evident care dintre afirmațiile următoare este mai importantă :

- care este mulțimea de stări admisibile ale unei relații r care defineşte o F-dependență,
- care F-dependențe determină restricțiile impuse schemei de relație R.

Numărul de F-dependențe care pot fi aplicate unei relații este finit, deoarece există numai un număr finit de submulțimi ale lui R. Prin urmare întotdeauna se poate determina toate F-dependențele pe care le poate satisface o relație r prin triere cu ajutorul procedurii **satisfie**.

Totuşi acest procedeu necesită mult timp deoarece trebuie generate şi apoi testate toate submulțimile schemei R. Dacă însă sunt cunoscute câteva F-dependențe din ***F*** atunci adesea se pot găsi cele rămase.

O mulțime de F-dependențe ***F*** implică F-dependența X→Y (notată $F \models X \rightarrow Y$) dacă orice relație care satisface ***F*** satisface X→Y.

Definiția 7.2. *O* ***inferență*** *este o regulă care arată că, dacă o relație satisface un grup de F-dependențe atunci ea satisface de asemenea un alt grup.*

Vom introduce şase axiome de inferență pentru dependențele funcționale. În formularea axiomelor se folosesc următoarele notații :

- r - pentru relații şi
- X, Y, Z,W - pentru submulțimi ale lui R.

Axioma I (Reflexivitate): Relația $\Pi_X(\sigma_{X=x}(r))$ are cel mult un tuplu pentru orice X-valoare x, prin urmare are loc **X®X.**

Axioma II ne dă posibilitatea să extindem partea din stânga a unei F-dependențe X→Y.

Axioma II (Augmentare): Dacă $\Pi_Y(\sigma_{X=x}(r))$ are cel mult un tuplu pentru orice X-valoare x şi Z este o submulțime oarecare a lui R atunci : $\Pi_Y(\sigma_{XZ=xz}(r))$ are nu mai mult de un tuplu şi prin urmare **: XZ®Y.**

Exemplul 7.2. Se consideră relația r care satisface F-dependența A→B

r(A	B	C	D)
a_1	b_1	c_1	d_1
a_2	b_2	c_1	d_1
a_1	b_1	c_2	d_1
a_3	b_3	c_2	d_3

si prin urmare rezultă că sunt adevărate următoarele F-dependențe:AB→ B, AC→ B, D→B, ACD→B, ABCD→ B.

Axioma III (Aditivitate): Această axiomă ne permite să unim două F-dependențe care au partea stânga aceeaşi. Dacă relația r satisface F-dependențele X→Y şi X→ Z atunci ambele proiecții $\Pi_Y(\sigma_{X=x}(r))$ şi $\Pi_Z(\sigma_{X=x}(r))$ au cel mult un tuplu pentru orice X-valoare x. Dacă $\Pi_{YZ}(\sigma_{X=x}(r))$ ar avea mai mult de un tuplu atunci cel puțin una din relațiile $\Pi_Y(\sigma_{X=x}(r))$ şi $\Pi_Z(\sigma_{X=x}(r))$ ar avea mai mult de un tuplu penntru orice X-valoare x. Prin urmare X→YZ.

Exemplul 7.3. Fie relația r din exemplul 7.2 care satisface şi F-dependența A→C atunci din axioma din A3 rezultă că r satisface A→ BC.

Axioma IV (Proiectivitate): Dacă r satisface X→YZ, atunci $\Pi_{YZ}(\sigma_{X=x}(r))$ are cel mult un singur tuplu pentru orice X-valoare x. Atunci $\Pi_Y(\Pi_{YZ}(\sigma_{X=x}(r)))= \Pi_Y(\sigma_{X=x}(r))$ are cel mult un tuplu, prin urmare r satisface X→ Y şi $\Pi_Z(\Pi_{YZ}(\sigma_{X=x}(r)))= \Pi_Z(\sigma_{X=x}(r))$, deci X→ Z.

Exemplul 7.4. Relația din exemplul 7.2 satisface relația A→BC. Conform axiomei IV, r satisface relațiile A→B şi A→C.

Axioma V (Tranzitivitate): Fie relația r care satisface F-dependențele X→Y şi Y→Z. Să considerăm tuplurile t_1 şi t_2 din r. Dacă $t_1(X)= t_2(X)$ atunci $t_1(Y)= t_2(Y)$ din care rezultă $t_1(Z)=t_2(Z)$ deci X→Z.

Exemplul 7.5. Relația r dată mai jos satisface F-dependențele A→B şi B→C. Din axioma V rezultă că r satisface A→C.

r(A	B	C	D)
a_1	b_1	c_2	d_1
a_2	b_2	c_1	d_2
a_3	b_1	c_2	d_1
a_4	b_1	c_2	d_3

Axioma VI (**Pseudotranzitivitate**): Fie relația r care satisface F-dependențele X→Y şi YZ→W, şi două tupluri t_1 si t_2 din r. Din condiția $t_1(X)= t_2(X)$ rezultă $t_1(Y)=t_2(Y)$ şi analog din $t_1(YZ)= t_2(YZ)$ rezultă $t_1(W)= t_2(W)$. Din $t_1(XZ)= t_2(XZ)$ rezultă $t_1(X)= t_2(X)$ ceea ce implică $t_1(Y)= t_2(Y)$ şi $t_1(YZ)=t_2(YZ)$ de unde rezultă $t_1(W)=t_2(W)$**.** Prin urmare XZ→W.

Prin urmare , dacă X,Y,Z şi W sunt submulțimi ale lui R atunci pentru orice relație r de schemă R sunt adevărate următoarele axiome de inferență:

A1. **Reflexivitate: X®X**,
A2. **Augmentare**: **X®Y Þ XZ® Y**,
A3. **Aditivitate : X® Y şi X® Z Þ X® YZ**,
A4. **Proiecție: X®YZ Þ X®Y şi X®Z**,
A5. **Tranzitivitate: X®Y,Y®Z Þ X®Z**,
A6. **Pseudotranzitivitate: X®Y ,YZ®W Þ XZ®W.**

Folosind axiomele A1-A6 se poate obține alte F-dependențe.

Exemplul 7.6. Fie relația r de schemă R şi $X,Y \subseteq R$. Din axioma A1 rezultă $Y \rightarrow Y$, aplicând axioma A2 obținem $XY \rightarrow Y$. Din proiecție rezultă că dacă $Y \subseteq X \subseteq R$ atunci $X \rightarrow Y$ pentru orice relație.

Exemplul 7.7. Fie relația r de schemă R şi $X,Y,Z \subseteq R$. Presupunem că r satisface F - dependențele $X \rightarrow Y$ şi $XY \rightarrow Z$ atunci din axioma A6 rezultă că $XX \rightarrow Z$ adică $X \rightarrow Z$.

Exemplul 7.8. Pentru a nega o afirmație referitoare la o F-dependență este suficient să arătăm că , există cel putin o relație care nu satisface această inferență. Dacă negăm afirmația, că din $XY \rightarrow WZ$ rezultă $X \rightarrow Z$ dăm contraexemplul dat de relația r care satisface $AB \rightarrow CD$ dar nu pe $A \rightarrow C$.

r(A	B	C	D)
a_1	b_1	c_1	d_1
a_2	b_2	c_2	d_1
a_1	b_2	c_2	d_1

Anumite axiome pot fi deduse unele din altele. De exemplu tranzitivitatea rezultă din axioma A6 luând $Z=\emptyset$. Axioma de pseudotranzitivitate rezultă din axiomele A1, A2, A3 şi A5.

1 $X \rightarrow Y$ (dată)
2 $YZ \rightarrow W$ (dată)
3 $Z \rightarrow Z$ (A1)
4 $XZ \rightarrow Z$ (A2 la 3)
5 $XZ \rightarrow Y$ (A2 la 1)
6 $XZ \rightarrow YZ$ (A3 la 4 şi 5)
7 $XZ \rightarrow W$ (A5 la 2 şi 6).

În paragraful următor vom arăta că sistemul de axiome A1-A6 este complet, adică orice F-dependență implicată de ***F*** poate fi obținută prin aplicarea de un număr finit de ori a axiomelor A1-A6 unor F- dependețe din ***F***.

7.1.3. Completitudinea sistemului de axiome

Fie ***F*** o mulțime de F-dependențe pentru relația r de schemă R.

Definiția 7.3. Se numeşte închidere a lui ***F***, notată F^+, cea mai mică mulțime de F-dependențe pe R care conține ***F*** şi orice aplicare a axiomelor A1-A6 la elementele ei, nu mai generează o altă F-dependență care să nu o conțină.

Deoarece F^+ este finită o putem calcula plecând de la ***F***, prin aplicarea axiomelor A1,A2,A6 până nu mai rezultă nici o nouă F-dependență. Închiderea lui ***F*** depinde de schema R, de exemplu dacă R=AB atunci F^+ va conține pe $B \rightarrow B$ dar nu şi pe $C \rightarrow C$. Când schema R nu este definită explicit se presupune că este formată din mulțimea tuturor atributelor utilizate în dependențele care compun pe ***F.***

Definiția 7.4. O F-dependență $X \rightarrow Y$ este derivată din ***F*** dacă $X \rightarrow Y \in \boldsymbol{F}^{+}$ (sau că ***F*** determină pe $X \rightarrow Y$).

Definitia 7.5. Spunem că ***F*** implică $X \rightarrow Y$ (şi se notează cu $\boldsymbol{F} \models X \rightarrow Y$) dacă orice relație care satisface ***F*** atunci satisface $X \rightarrow Y$**.**

Din Axiomele lui Armstrong se pot deduce şi alte reguli. Dintre care foarte utile sunt lemele următoare.

Lema 7.1. Este adevărată următoarea formulă (descompunerea): dacă $\boldsymbol{F} \models X \rightarrow Y$ şi $Z \subset Y$, atunci $\boldsymbol{F} \models X \rightarrow Z$

Demonstrație. Din $Z \subset Y$ rezultă prin reflexivitate $Y \rightarrow Z$ şi aplicând tranzitivitatea se obține $X \rightarrow Z$.

Lema 7.2. $X \rightarrow A_1, A_2, \ldots, A_k \Rightarrow X \rightarrow A_i$ pentru $1 \leq i \leq k$.

Demonstrația se face prin inducție după k aplicând lema 7.1 şi A3(aditivitatea).

Deoarece axiomele de inferentă sunt adevărate şi dacă ***F*** determină $X \rightarrow Y$ atunci ***F*** implică $X \rightarrow Y$.

În continuare vom arăta că axiomele A1-A6 ne permit să deducem toate F-dependențele implicate de ***F***. Pentru a demonstra acest rezultat va trebui să arătăm cum se construieşte pentru ***F*** o relație care satisface $\boldsymbol{F}^{+}$ şi nici o altă relație în plus.

Definitia 7.6. Spunem că $X \rightarrow Y$ este o F-dependentă pe R dacă $X, Y \subseteq R$. ***F*** este o mulțime de F-dependențe pe R dacă pentru orice F- dependentă $X \rightarrow Y$ din ***F*** atunci $X, Y \subseteq R$.

Definitia 7.7. Dacă ***F*** este o multime de F-dependențe pe R şi *G* multimea tuturor F-dependențelor posibile pe R, atunci se numeşte exterior a lui ***F*** mulțimea $\boldsymbol{F}^{-} = G - \boldsymbol{F}^{+}$.

F-depenedența $X \rightarrow Y$ pe R se numeşte *trivială* dacă $X \supseteq Y$. Orice relație r(R) satisface dependența trivială $X \rightarrow Y$. Dacă ***F*** este o mulțime de F-dependențe pe R şi X este o submulțime a lui R atunci există o F-dependență $X \rightarrow Y$ în $\boldsymbol{F}^{+}$ astfel încât Y să fie maximală.

Pentru orice altă F-dependență $X \rightarrow Z$ din $\boldsymbol{F}^{+}$ rezultă că $Z \subseteq Y$. Acest rezultat se numeşte **închiderea** lui X în raport cu ***F***, se notează cu $\mathbf{X}^{+}$ şi conține întodeauna pe X.

Lema 7.3. $X \rightarrow Y$ rezultă din axiomele lui Armstrong daca şi numai dacă $Y \subset X^{+}$.

Demonstrația rezultă din definiții şi aplicând lema 7.2.

Exemplul 7.9. Fie $\boldsymbol{F}=\{A \rightarrow D, AB \rightarrow DE, CE \rightarrow G, E \rightarrow H\}$ atunci $AB^{+} = ABDEH$.

Într-adevăr:

$AB \rightarrow AB$

$AB \rightarrow DE$

$AB \rightarrow ABDE$

$AB \rightarrow E$

$E \rightarrow H$

$AB \rightarrow ABDEH$

Teorema 7.1. (completitudine). Sistemul de axiome A1-A6 este complet. Adică $\boldsymbol{F} \models X \rightarrow Y \Leftrightarrow X \rightarrow Y \in \boldsymbol{F}^{+}$.

Demonstrație. Fie $\boldsymbol{F}$ o mulțime de F-dependențe pe R. Pentru orice $X \rightarrow Y \in F^-$ vom determina o relație r(R) care satisface $\boldsymbol{F}^+$ dar nu pe $X \rightarrow Y$. Prin urmare nu există F-depenednță implicată de $\boldsymbol{F}$ care să nu fie derivată din $\boldsymbol{F}$. Fie schema R= $A_1A_2...A_n$ şi a_i,b_i elemente distincte din dom(A_i), $1 \leq i \leq n$ şi relația r formată din două tupluri t şi t'. Tuplul t=< a_1, a_2, …,a_n> iar tuplul t' este definit de

$$t'=\begin{cases} a_i \ \ daca \ \ A_i \in X^+ \\ b_i \ \ daca \ \ A_i \notin X^+ \end{cases}$$

Mai întâi vom arăta că r nu satisfce $X \rightarrow Y$. Din definiția lui r rezultă că t(X)=t'(X). Presupunem prin absurd că t(Y)=t'(Y). Atunci t'(Y) are numai componente a_i şi prin urmare $Y \subseteq X^+$. Dar $X \rightarrow X^+$ şi din proiectivitate rezultă că $X^+ \rightarrow Y$ şi din tranzitivitate rezultă că $X \rightarrow Y \in \boldsymbol{F}^+$ contradicție cu faptul că $X \rightarrow Y \in \boldsymbol{F}^-$.

Vom arăta că r satisface toate F-dependențele din $\boldsymbol{F}^+$. Va trebui să considerăm numai acele F-dependențe $W \rightarrow Z$ în care $W \subseteq X^+$.Din proiectivitate rezultă că $X^+ \rightarrow W$ şi din tranzitivitate rezultă că $X^+ \rightarrow Z$ prin urmare $Z \subseteq X^+$ şi deci t(Z)=t'(Z), deci r satisface $\boldsymbol{F}^+$.

Corolar 7.1. Pentru orice mulțime de F-dependențe $\boldsymbol{F}$ pe schema R există o relație r[R] care satisface $\boldsymbol{F}^+$ dar nu satisface $\boldsymbol{F}^-$.

Pentru a verifica că mulțimea de F-dependențe $\boldsymbol{F} \models X \rightarrow Y$ va trebui să verificăm dacă $X \rightarrow Y \in \boldsymbol{F}^+$. Această variantă durează foarte mult. Este de dorit un procedeu de verificare a apartenenței lui $X \rightarrow Y$ la $\boldsymbol{F}^+$ fără a genera toate dependențele care o compun. Nucleul algoritmului îl reprezintă o procedură de calcul a închiderii lui X în raport cu F. După ce s-a calculat X^+ se verifică dacă $\boldsymbol{F} \models X \rightarrow Y$. Procedura **closure** dată mai jos returnează X^+ în raport cu $\boldsymbol{F}$.

```
1 PROCEDURE Closure(X,F;X⁺)
2 DV←∅
3 DN←X
4 WHILE DV≠DN
    4.1 DV← DN
    4.2 FOR fiecare W→ Z∈F
          IF DN ⊇ W
               THEN
            DN←DN ∪ Z
         CONTINUE
    4.3  CONTINUE
5 X⁺←DN
6 RETURN
```

Procedura **closure** este utilizată în funcția **termen** de testare a apartenenței lui $X \rightarrow Y$ la $\mathbf{F}^+$.

```
0   FUNCTION  termen(F, X→Y)
1   CALL closure(X,F;X⁺)
2   IF Y⊆ X⁺
    THEN
     2.1  termen← true
      ELSE
      2.2  termen← false
3   RETURN
```

Un alt algoritm, utilizat pentru a decide dacă o dependenţă $X \rightarrow Y$ poate fi dedusă logic din ***F***, arată că este suficient să calculăm X^+ şi să vedem, conform lemei 7.3 dacă $Y \subseteq X^+$ sau nu.

Algoritmul 7.1. Fie X o mulţime de atribute şi ***F*** o mulţime de dependenţe funcţionale.

1. Se face $X^{(0)}=X$ şi i=0.
2. Dacă există în F o dependenţă $V \rightarrow W$ cu $V \subset X^{(i)}$ şi $W-X^{(i)} \neq \emptyset$, atunci se face $X^{(i+1)}=X^{(i)} \cup W$; i=i+1 şi se reia pasul 2; altfel STOP($X^+=X^{(i)}$).

Exemplul 7.10. Fie X=BD şi ***F*** având următoarele dependenţe funcţionale:
1)$AB \rightarrow C$, 2)$C \rightarrow A$, 3)$BC \rightarrow D$, 4)$ACD \rightarrow B$, 5) $D \rightarrow EG$, 6) $BE \rightarrow C$, 7)$CG \rightarrow BD$, 8) $CE \rightarrow AG$.

Aplicând algoritmul 7.1, se obţine mai întâi $X^{(0)}=BD$, apoi folosind dependenţa 5), $X^{(1)}=BDEG$, şi în continuare, folosind dependenţa 6), $X^{(2)}=BCDEG$. Folosind dependenţa 2) se obţine $X^{(3)}=ABCDEG$ pentru care nu mai sunt îndeplinite condiţiile din pasul 2.

Deci $(BD)^+=ABCDEG$.

7.1.4. Secvenţe derivate

Dacă $F \models X \rightarrow Y$ atunci $X \rightarrow Y$ fie este conţinută în ***F***, fie poate fi obţinută prin aplicarea unei secvenţe de inferenţe la anumite F-dependenţe din ***F***. Aceaşta secvenţă de aplicare a axiomelor şi a F-dependenţelor rezultate se numeşte **derivata** lui $X \rightarrow Y$ din ***F*.**

Definiţia 7.8. O secvenţă *P* de F-dependenţe pe R este *derivată din **F***, dacă orice F-dependenţă din P este fie un termen din ***F***, fie este rezultată din F-dependenţele care o preced în secvenţă prin aplicarea uneia dintre axiomele de la A1 la A6.

Definiţia 7.9. Se numeşte derivată a F-dependenţei $X \rightarrow Y$ din ***F*** o secvenţă de F-dependenţe derivate din ***F*** care o conţin.

Deci $F \models X \rightarrow Y$ dacă există o derivată din ***F*** care o conţine.

Exemplul 7.10. Fie ***F*** ={$AB \rightarrow C$, $AD \rightarrow G$, $BC \rightarrow E$, $C \rightarrow D$, $DE \rightarrow K$ }. Următoarea secvenţă este o derivată a lui $AB \rightarrow DK$:

1. $AB \rightarrow C$ (dată)
2. $C \rightarrow D$ (dată)
3. $AB \rightarrow D$ (tranzitivitate din 1 şi 2)
4. $B \rightarrow B$ (reflexivitate)
5. $AB \rightarrow B$ (augmentare din 4)
6. $AB \rightarrow BC$ (trazitivitate din 4 şi 5)
7. $BC \rightarrow E$ (dată)
8. $AB \rightarrow E$ (tranzitivitate din 6 şi 7)
9. $AB \rightarrow DE$ (aditivitate din 6 şi 8)
10. $DE \rightarrow K$ (dată)
11. $AB \rightarrow K$ (tranzitivitate din 9 şi 10)
12. $AB \rightarrow DK$ (aditivitate din 3 şi 11)
13. $AB \rightarrow BE$ (aditivitate din 1 şi 8)

În continuare se foloseşte o altă mulţime completă de inferenţe care nu este o submulţime a lui A1...A6 unde inferenţele sunt numite ***B_axiome***. Fie r(R) şi submulţimile X,Y,Z,W ale schemei R şi un atribut A din R Vom arăta apoi că **axiomele lui Armstrong** A1,A2,A6 se deduc din **B_axiomele B1, B2,B3 :**

B1. Reflexivitatea : X®X,
B2. Acumularea : X®YZ, Z®AW Þ X®AYZ,
B3. Proiecţie : X ®YZ Þ X® Z sau **X®Y.**

A1. **Reflexivitatea**, aceeaşi cu B1.
A2. **Augmentarea** :
1: XZ→XZ (B1)
2: X→Y=A_1 A_2 ..A_n (dată)
3: X→XYZ (aplicăm de n ori B2 la 1 şi 2)
4: XZ→Y (aplicam B3 la 3)

A6. **Pseudotrazitivitate** :
1. XZ→XZ (B1)
2. X→Y=A1A2..Am (dată)
3. XZ→XYZ (aplicăm de m ori B2 la 1 şi 2)
4. YZ→W= C1C2..Ck (dată)
5. XZ→XYZW (aplicăm de k ori B2 la 3 şi 4)
6. XZ→W (aplicăm B3 la 5)

Deoarece sistemul de B-axiome este complet întotdeauna se poate găsi o derivare care să utilizeze numai B1,B2,B3 dacă F ⊨X→Y.

Exemplul 7.11. Fie ***F*** mulţimea din exemplul 7.10. Atunci:
1. CE→ CE (reflexivitate)
2. C→D (dată)
3. CE→CDE (acumulare din 1 şi 2)
4. CE→DE (proiectivitate din 3)
5. DE→K (dată)
6. DE→DEK (acumulare din 4 şi 5)
7. AB→ AB (reflexivitate)
8. AB→C (dată)
9. AB→ ABC (acumulare 7 şi8)
10. BC→ E (dată)
11. AB→ABCE (acumulare din 10 şi 11)
12. AB→ABCDE (acumulare din 4 şi 12)
13. AB→ABCDEK (acumulare din 5 şi 12)
14. AB→DK (proiectivitate din 13)

este o derivată pentru care se utilizează numai B-axiome.

7.2 Dependențe multivoce

Există situații în care şi în lipsa dependențelor funcționale se poate da o descompunere a schemei care micşorează redundanța şi conservă informațiile.

Se consideră schema de relație R=[Uzina, Vânzător, Produs] notată pe scurt R=[U,V,P] şi relația

r

U	V	P
u_1	v_1	p_1
u_1	v_2	p_2
u_1	v_1	p_2
u_1	v_2	p_1
u_2	v_2	p_1

Un tuplu t=<u,v,p> din relația *r* arată că uzina *u* fabrică produsul *p* şi îl aprovizionează pe vânzătorul *v*. Se presupune că o uzină fabrică mai multe produse şi aprovizionează mai mulți vânzători. Avem două funcții independente, de fabricare şi vânzare.

fabricare

U	P
u_1	p_1
u_1	p_2
u_2	p_2

vânzare

U	V
u_1	v_1
u_1	v_2
u_2	v_2

Evident că relația r conține o anumită redundanță. Prin urmare, dacă uzina u_1 aprovizionează un nou vânzător v_3 , este necesar să creeze două tupluri pentru fiecare produs.

Definiția 7.10. Fie schema de relație R= $[A_1,A_2,\ldots,A_n]$ şi o partiție [X,Y,Z] a schemei R cu X şi Y care nu se intersectează. Se spune că relația r satisface dependența multivocă (MV-dependența) $X \twoheadrightarrow Y$ sau $X \twoheadrightarrow Z$ dacă din apartenența tuplurilor <x,y,z> şi <x,y',z'> la relația r rezultă că şi tuplurile <x,y',z> şi <x,y,z'> aparțin lui r.

Pentru relația r avem $U \twoheadrightarrow P$ şi $U \twoheadrightarrow V$.

- dacă un vânzător se aprovizionează de la uzină el are în vedere toate produsele fabricate de uzină.
- dacă un produs este fabricat de o uzină el trebuie avut în vedere de toți vânzătorii care se aprovizionează de la uzina respectivă.
- toate produsele fabricate de o uzină sunt comercializabile de toți vânzătorii care se aprovizionează de la uzină.

Dăm în continuare o definiție formală:

Definiția 7.11. Fie schema de relație R şi $X,Y\subset R$,$X\cap Y=\emptyset$, Z=R-XY. Relația r(R) satisface dependența multivocă (pe scurt: MV-dependență) $X \twoheadrightarrow Y$ dacă:

(i) $(\forall)\ t_1, t_2\in r$, $t_1(X)=t_2(X)$ $(\exists)\ t_3\in r$ astfel încât $t_3(X)=t_1(X)$, $t_3(Y)=t_1(Y)$, $t_3(Z)=t_2(Z)$

(ii) $(\forall)\ t_1, t_2\in r$, $t_1(X)=t_2(X)$ $(\exists)\ t_4\in r$ astfel încât $t_4(X)=t_1(X)$, $t_4(Y)=t_2(Y)$, $t_4(Z)=t_1(Z)$.

Exemplul 7.12 Pentru a înțelege mai bine acest tip de dependență între atributele unei relații se va considera o relație **CSL** (Curs-Student-Laborator) cu următoarea ipoteză de lucru: **"Studenții** *utilizează pentru un anumit* **Curs** *acelaşi material pentru* **Laborator"**.

Este evident că în relația CSL nu există dependențe funcționale în schimb setul de valori pentru **Laborator** care corespund unei singure valori pentru **Curs** este independent de valorile pentru atributul **Student**. Această proprietate este considerată o restricție foarte importantă numită dependență multivaloare.

CSL

Curs	Student	Laborator
C1	S1	Ll
C1	S1	L2
C1	S1	L3
C1	S2	Ll
C1	S2	L2
C1	S2	L3
C2	S3	L4
C2	S4	L4

Pentru exemplul considerat dependența mutivaloare implică faptul că în cazul în care apar în relația CSL tuplurile (C1,S1,L1) şi (C1,S2,L3) trebuie să apară obligatoriu şi tuplurile (C1,S2,L1) respectiv (C1,S1,L3). Din punct de vedere semantic acest lucru este posibil dacă acceptăm ipoteza de lucru *"Laboratorul pentru un curs trebuie să fie aceeaşi indiferent de student"*.

Dependența multivaloare poate fi abordată şi din alt punct de vedere, în scopul punerii în evidentă a unei metode de testare a existenței acesteia într-o relație dată.

Fie R(XYZ) o relație cu m+n+r atribute, unde $X=\{X_1,X_2,...,X_m\}$, $Y\{Y_1,Y_2,...,Y_n\}$ şi $Z=\{Z_1,Z_2,...,Z_r\}$ sunt seturi de atribute disjuncte două câte două.

Un m-tuplu $(x_1x_2 ... x_m)$ de valori ale atributelor X_1, X_2, … ,X_m se notează x, un n-tuplu $(y_1y_2...y_n)$ de valori ale atributelor Y_1, Y_2, ... ,Y_n se notează y iar un r-tuplu $(z_1z_2...z_r)$ de valori ale atributelor Z_1, Z_2, ... ,Z_r se notează z. Se notează $Y_{xz}=\{y \mid (x,y,z)\in R\}$

Folosind aceste notații, dependența multivaloare poate fi redefinită:

În relația R(XYZ) *există o dependență multivaloare* $X \twoheadrightarrow Y$ *dacă şi numai dacă în orice moment* $Y_{xz} = Y_{xz'}$ *pentru fiecare x,z şi z' pentru care* Y_{xz} *şi* $Y_{xz'}$ *sunt seturi nenule de atribute.*

Se observă imediat că această definiție implică independența setului de valori pentru atributul Y puse în corespondență cu o valoare a atributului X de valorile atributului Z, deci cele două definiții ale dependenței multivaloare sunt echivalente. Cea de a doua definiție are avantajul că permite testarea imediată a existenței dependenței multivaloare într-o relație dată.

Exemplul 7.13 Se consideră relația CSL din exemplul anterior. Aplicând a doua definiție pe eşantionul de date prezentat, se obține:

$LABORATOR_{C1S1}=\{L1,L2,L3\}$

$LABORATOR_{C1S2}=\{L1,L2,L3\}$

$LABORATOR_{C2S3}=\{L4\}$

$LABORATOR_{C2S4}=\{L4\}$

$$\left.\begin{array}{l} LABORATOR_{C1S1} = LABORATOR_{C1S2} \\ LABORATOR_{C2S3} = LABORATOR_{C2S4} \end{array}\right\} \Rightarrow CURS \twoheadrightarrow LABORATOR$$

Dependențele multivaloare au următoarele **proprietăți**:

P1) Dacă X şi Y sunt mulțimi disjuncte ale atributelor din relația R[XYZ] şi există dependența funcțională X—>Y, atunci există şi dependența multivaloare $X \twoheadrightarrow Y$, deci dependența funcțională este un caz particular de dependență multivaloare.

Demonstrație: Proprietatea rezultă imediat din analiza definițiilor pentru cele două tipuri de dependențe. Din definiția dependenței funcționale rezultă imediat că pentru o valoare dată a lui **X**, **Yxz** va conține întotdeauna o valoare unică. Se poate deci spune că dependența funcțională este un caz particular al dependenței multivaloare şi anume cazul în care setul de valori asociat unui atribut se reduce la o valoare unică.

P2) Dacă în relația R(XYZ) există $X \twoheadrightarrow Y$ atunci obligatoriu există şi $X \twoheadrightarrow Z$.

Demonstrație: $X \twoheadrightarrow Y$ înseamnă că în orice moment valorile lui Y care corespund unei anumite valori a lui X formează o mulțime independentă de valorile lui Z ataşate aceleeaşi valori a lui X. Dar aceasa înseamnă că este adevărată şi afirmația inversă adică mulțimea valorilor lui Z ataşate unei anumite valori a lui X nu depinde de valorile lui Y ataşate aceleeaşi valori a lui X adică $X \twoheadrightarrow Z$.

Observație: Această proprietate justifică notația $X \twoheadrightarrow Y/Z$.

Exemplul 7.14 În relația CSL mulțimea studenților ce audiază un curs şi mulțimea laboratoarelor la un curs sunt independente conform ipotezei de lucru considerate, deci:

CURS $\twoheadrightarrow$ STUDENT şi CURS $\twoheadrightarrow$ LABORATOR

Cele două expresii pot fi scrise într-o singură expresie şi anume:

CURS $\twoheadrightarrow$ STUDENT/LABORATOR

P3) O relație R(XYZ) unde X,Y,Z sunt seturi de atribute, poate fi descompusă fără pierdere de informație din proiecțiile sale R[XY] şi R[XZ] printr-o joncțiune naturală după X dacă şi numai dacă relația R suportă dependența multivocă $X \twoheadrightarrow Y/Z$.

Formal această proprietate poate fi scrisă sub forma:

$X \twoheadrightarrow Y/Z \Leftrightarrow R(XYZ)=R[XY] ><_{X} R[XZ]$

Exemplul 7.15 Fie relația CSL şi proiecțiile sale:

CS

CURS	**STUDENT**
C1	S1
C1	S2
C2	S3
C2	S4

CL

CURS	**LABORATOR**
C1	L1
C1	L2
C1	L3
C2	L4

Atunci FINAL=CS$><_{CURS}$CL

FINAL

CURS	**STUDENT**	**LABORATOR**
C1	S1	L1
C1	S1	L2
C1	S1	L3
C1	S2	L1
C1	S2	L2
C1	S2	L3
C2	S3	L4
C2	S4	L4

P4) În orice relație R(XY) unde $X \cap Y = \emptyset$) există dependențe multivaloare $X \twoheadrightarrow \emptyset$ şi $X \twoheadrightarrow Y$ numite şi dependențe triviale.

Axiome de inferență pentru dependențe multivoce

Primele şase axiome de inferență introduse mai jos sunt analoage cu axiomele pentru F-dependențe, dar numai primele trei sunt afirmații identice cu cele de la F-dependențe.

Fie r o relație de schemă R şi X,Y,Z,W submulțimi ale lui R .

M1. Reflexivitate: Relația r satisface $X \twoheadrightarrow Y$.

M2. Augmentare: Dacă r satisface $X \twoheadrightarrow Y$, atunci ea satisface $XZ \twoheadrightarrow Y$, pentru orice $Z \subset R$.

M3. Aditivitate: Dacă r satiface $X \twoheadrightarrow Y$ şi $X \twoheadrightarrow Z$ atunci ea satisface $X \twoheadrightarrow YZ$.

M4. Proiectivitate: Dacă r satisface $X \twoheadrightarrow Y$ şi $X \twoheadrightarrow Z$ atunci ea satisface $X \twoheadrightarrow Y \cap Z$ şi $X \twoheadrightarrow Y\text{-}Z$.

M5. Tranzitivitate : Dacă r satisface $X \twoheadrightarrow Y$ şi $X \twoheadrightarrow Z$ atunci r satisface $X \twoheadrightarrow Z\text{-}Y$.

M6. Pseudotranzitivitate : Dacă r satisface $X \twoheadrightarrow Y$ şi $YW \twoheadrightarrow Z$ atunci r satisface $XW \twoheadrightarrow Z\text{-}(YW)$.

M7. Complementaritate : Dacă r satisface $X \twoheadrightarrow Y$ şi Z=R-XY atunci r satisface $X \twoheadrightarrow Z$.

Axiomele M1 şi M2 rezultă direct din prima definiție a MV–dependențelor. Arătăm că axioma M3 este adevărată. Fie relația r care conține tuplurile t_1 şi t_2 pentru care $t_1(X)=t_2(X)$. Trebuie să arătăm că r conține tuplul t astfel ca $t(X)=t_1(X)$, $t(YZ)=t_1(YZ)$ şi $t(U)=t_2(U)$ unde U=R=(XYZ). Întrucât r satisface $X \twoheadrightarrow Y$ ea trebuie să conțină tuplul t_3 astfel ca: $t_3(X)=t_1(X)$, $t_3(Y)=t_1(Y)$, $t_3(V)=t_2(V)$ unde V=R-(XY). Din relația $X \twoheadrightarrow Z$ rezultă că există tuplul t_4 astfel ca: $t_4(X)=t_1(X)$, $t_4(Z)=t_1(Z)$, $t_4(W)=t_2(W)$ unde W=R-(XZ. Luăm $t=t_4$. Evident $t(X)=t_4(X)$, $t_4(Z)=t_1(Z)=t(Z)$, $t_4(X \cap W)=t_2(X \cap W)= t_1(X \cap W)=t(X \cap W)$ astfel ca $t_4(XZ)=t(XZ)$.

Întrucât $U \subset W \cap V$ avem $t_4(U)=t_3(V)=t_2(V)=t(V)$. Deoarece R=XYZU, atunci $t_4=t$. Axiomele M3 şi M7 pot fi folosite la demonstrarea axiomei M4. Dacă r satisface $X \twoheadrightarrow Y$ şi $X \twoheadrightarrow Z$ atunci conform axiomei M3 $X \twoheadrightarrow YZ$ şi conform axiomei M7 r satisface $X \twoheadrightarrow V$, V=R-(XYZ). Folosind din nou M3 rezultă că r satisface $X \twoheadrightarrow VZ$. Din ultima aplicație a axiomei M7 rezultă că $X \twoheadrightarrow R\text{-}(XVX)$. Prin schimbare şi ordonare rezultă M4. R-(XVZ)=R-(X{R-(XYZ)}Z)=R-(X{R-Y}Z)= Y-(XZ)=(Y-Z)-X.

Prin urmare r satisface $X \twoheadrightarrow$ (Y-Z)-X, de aici rezultă $X \twoheadrightarrow Y\text{-}Z$. Din $X \twoheadrightarrow Y$ cu ajutorul axiomei M7 obținem $X \twoheadrightarrow W$, unde W=R-(XY). Combinată cu $X \twoheadrightarrow Y\text{-}Z$ cu ajutorul axiomei M3 obținem $X \twoheadrightarrow W(Y\text{-}Z)$. Din nou folosind axioma M7, avem $X \twoheadrightarrow R\text{-}(XW(Y\text{-}Z))$. Schimbând ordinea obținem :

$R\text{-}(WX(Y\text{-}Z))=R\text{-}(X\{R\text{-}(XY)\}(Y\text{-}Z))=R\text{-}(X\{R\text{-}Y\}(Y\text{-}Z))=Y\text{-}(X(Y\text{-}Z))=(Y \cap Z)\text{-}X$.

Prin urmare satisface $X \twoheadrightarrow (Y \cap Z)\text{-}X$ şi prin urmare $X \twoheadrightarrow Y \cap Z$. Pentru demonstrarea axiomei M5 la început arătăm că dacă în r există tuplurile t_1 şi t_2 astfel ca $t_1(X)=t_2(X)$, atunci r conține tuplul t astfel ca $t(X)=t_1(X)$, $t(YZ)=t_1(YZ)$, $t(W)=t_2(W)$. Din $X \twoheadrightarrow Y$ rezultă că există tuplul t_3 pentru care $t_3(X)=t_1(X)$, $t_3(Y)=t_1(Y)$ şi $t_3(V)=t_2(V)$, unde V=R-(XY). Prin urmare $X \twoheadrightarrow Z$ rezultă că există tuplul t_4 pentru care $t_4(Y)=t_1(Y)$, $t_4(Z)=t_1(Z)$ şi $t_4(U)=t_3(U)$, unde U=R-(XZ) .

Întrucât pentru fiecare atribut $A \in X$, tuplurile t_1, t_2, t_3 au aceleaşi valori avem $t_4(X)=t_1(X)$. Evident $t_4(YZ)=t_1(YZ)$. Dar deoarece $W \subseteq U\text{-}X \subseteq V$ atunci $t_4(W)=t_2(W)$. Prin urmare t_3 este tuplul căutat. Noi arătăm că r satisface $X \twoheadrightarrow YZ$. Folosind axioma M4 şi $X \twoheadrightarrow Y$ obținem că $X \twoheadrightarrow Z\text{-}Y$.

Exemplul 7.16. Fie R=ABCDE şi F={$A \twoheadrightarrow BC$,$DE \twoheadrightarrow C$}. Din $A \twoheadrightarrow BC$ cu ajutorul complementarității obținem $A \twoheadrightarrow DE$. Mai departe din tranzitivitate avem $A \twoheadrightarrow C$. Cu

ajutorul axiomei de augmentare obținem că DA$\twoheadrightarrow$C. Prin urmare aplicarea repetată a axiomelor atrage după sine AD$\twoheadrightarrow$BE. Prin urmare F|=AD$\twoheadrightarrow$BE .

Ne vom limita la numai formularea rezultatelor despre completitudinea sitemului de inferențe pentru MV-dependențe.

Teorema 7.2 Sistemul de inferențe M1-M7 pentru MV-dependențe este complet .
Demonstrația este asemănătoare ca la F-dependențe.

Ca şi în cazul dependențelor funcționale, există un criteriu foarte simplu pentru a vedea dacă o descompunere în două componente este fără pierderi la uniune şi în cazul dependențelor multivoce, după cum rezultă din teorema următoare.

Teorema 7.3. Fie *R* o schemă relațională, *D* o mulțime de dependențe funcționale şi multivoce pe mulțimea atributelor lui *R* şi $\boldsymbol{r}=(R_1,R_2)$ o descompunere a lui *R*. Atunci $\boldsymbol{r}$ este fără pierderi la uniune dacă şi numai dacă $(R_1 \cap R_2) \twoheadrightarrow (R_1 - R_2)$.

Demonstrație. Descompunerea $\boldsymbol{r}$ este fără pierderi la uniune dacă şi numai dacă, pentru orice relație *r* ce satisface *D* şi pentru orice două tupluri *t* şi *s* din *r*, există un tuplu astfel încât $u[R_1]=t[R_1]$ şi $u[R_2]=s[R_2]$. Dar tuplul *u* exită dacă şi numai dacă $t[R_1 \cap R_2]=s[R_1 \cap R_2]$, ceea ce duce la dependența multivocă din enunțul teoremei.

Se întâlnesc situații când sunt valabile dependențe multivoce pe componente ale unei descompuneri fără a fi valabile pe relația inițială. Aceste dependențe se numesc *dependențe ascunse*. O relație *r* de tipul schemei relaționale *R* satisface o dependență multivocă ascunsă $X \twoheadrightarrow Y|_z$ dacă dependența multivocă $X \twoheadrightarrow Y$ este satisfăcută de relația $\Pi_{X \cup Z \cup Y}(r)$.

7.3. Dependențe generalizate

Până acum am definit dependențele funcționale şi dependențele multivaloare.
În afară de acestea, se mai poate considera dependența impusă de condiția de descompunere fără pierderi la uniune a unei scheme relaționale, numită *dependență de uniune* şi notată >< $(R_1R_2,...,Rk)$, care este satisfăcută de relația *r* conținut actual al relației $R_1\cup ... \cup R_k$ dacă şi numai dacă uniunea naturală a proiecțiilor lui *r* pe fiecare R_i este egală cu *r*. Toate aceste tipuri de dependențe pot fi exprimate unitar prin dependențele generalizate pe care le definim în continuare.

O *dependență generalizată* peste schema relațională $A_1...A_n$ este o expresie de forma $(t_1,...,t_k)|t$, unde fiecare t_i, este un n-tuplu de simboluri şi *t* este fie un alt tuplu, caz în care avem o *dependență generatoare de tupluri,* fie o expresie $x = y$ cu *x* şi *y* dintre simbolurile ce apar anterior, în acest caz având o *dependență generatoare de egalități.* Numim $(t_1,...,t_k)$ *ipoteza* dependenței şi *t concluzia* dependenței. Un simbol din concluzie care nu mai apare în altă parte se numeşte *simbol unic*. O dependență generalizată se numeşte *ascunsă* dacă are cel puțin un simbol unic şi se numeşte *completă* dacă nu conține nici un simbol unic.

O dependență funcțională nebanală $X \twoheadrightarrow Y$ se exprimă printr-un număr de dependențe generalizate egal cu numărul elementelor mulțimii *Y-X,* cu ipoteza formată din două tupluri ce conțin simboluri comune pentru atributele din *X* şi simboluri distincte pentru celelalte atribute, iar concluzia conține o egalitate între simboluri din cele două tupluri pentru un atribut din *Y-X.*

O dependență multivaloare de tipul $X \twoheadrightarrow Y$ se exprimă ca o dependență generalizată cu ipoteza formată din două tupluri care au aceleaşi simboluri pentru atributele din X şi

simboluri distincte în rest, iar concluzia este un tuplu cu simboluri din primul tuplu pentru atributele XY şi cu simbolurile din al doilea tuplu în rest.

O dependență multivaloare ascunsă $X \longrightarrow\!\!> Y|_Z$ se poate reprezenta ca o dependență generalizată cu ipoteza formată din două tupluri care au aceleaşi valori pentru atributele lui *X* şi valori diferite pentru celelalte atribute, iar concluzia coincide cu ipotezele pe atributele lui *X,* are valorile din primul tuplu pentru atributele lui *Y*, are valorile din al doilea tuplu pentru atributele din *Z* şi simboluri unice în rest.

O dependență de uniune se poate reprezenta ca o dependență generalizată în care ipoteza conține un număr de tupluri egal cu numărul relațiilor din descompunere ce includ simboluri comune pentru aparițiile unui atribut în mai multe relații pentru tuplurile corespunzătoare relațiilor în care apare acel atribut şi simboluri diferite în rest, iar concluzia conține simbolul asociat fiecărui atribut în ipoteză. Evident, concluzia nu are simboluri unice deoarece fiecare atribut apare cel puțin într-o componentă.

Pentru simplificarea notației, în tabelele asociate dependențelor generalizate se convine să nu se mai noteze simbolurile care au o singură apariție (ipoteză + concluzie).

Fie *S* şi *T* două mulțimi de simboluri. Spunem că *h* este o *aplicație de simboluri* dacă pentru fiecare *a* din *S* se defineşte *h(a)* ca fiind un element al lui *T*. Dacă $s = a_1a_2...a_n$ este un tupltu de simboluri, se defineşte *h(s)* ca fiind cuvântul obținut prin concatenarea $h(a_1)h(a_2)...h(a_n)$. Dacă $s_1s_2...s_k$ este o mulțime de tupluri cu simboluri din *S* şi $t_1t_2...t_m$ este o mulțime de tupluri cu simboluri din *T,* vom spune că *există o aplicație de simboluri* de la prima mulțime de tupluri la a doua mulțime de tupluri dacă şi numai dacă există o aplicație de simboluri *h* de la *S* la *T* astfel încât, pentru orice *i*, să existe un *j* cu $h(s_i) = t_j$.

Exemplul 7.17. Fie *A* = {*abc, ade, fbe*} şi *B* = {*xyz, wyz*}. Există mai multe aplicații de simboluri de la *A* la *B.* De exemplu, aplicația *h* cu *h(a) = h(f) = x, h(b)= h(d) = y* şi *h{c) = h(e) = z* are drept imagine, pentru toate cele trei elemente ale lui *A,* elementul *xyz* din *B.* Aplicația *g(a) = x, g(b) = g(d) =y, g(c) = g(e) = z* şi *g(f) = w* transformă *abc* şi *ade* în *xyz* şi pe *fbe* în *wyz.* Nu există o aplicație de simboluri care să ducă *abc* în *xyz* şi *ade* în *wyz* dacă *x* şi *w* sunt simboluri distincte, deoarece astfel lui *a* i s-ar asocia atât *x*, cât şi *w,* ceea ce nu este posibil.

O relație *r satisface* o dependență generalizată $(t_1,...,t_k)|t$, dacă, pentru orice aplicație de simboluri *h* de la mulțimea tuplurilor din ipoteza dependenței generalizate la *r,* se poate extinde *h* la orice simbol unic din *t* astfel încât *h(t)* să aparțină lui *r.* Analog, spunem că *r satisface* dependența generalizată $(t_1,...,t_k)|a=b$ dacă, pentru orice aplicație de simboluri *h* de la ipoteză la *r,* să aibă loc egalitatea *h{ă) = h(b).*

Exemplul 7.18. Fie *d* dependența din fig.7.1.a. şi relația *r* din fig.7.1.b. Deoarece în prima coloană a tuplurilor din ipoteză figurează acelaşi simbol a_1, rezultă că aplicația de simboluri trebuie să transforme tuplurile din ipoteză în tupluri care să aibă pe primul loc aceleaşi valori. Dacă se ia $h(a_1) = 5$, rezultă că ambele tupluri sunt duse în tuplul 514 al lui r şi deci $h(b_1) = h(b_2) = 1$ şi $h(c_1) = = h(c_2) = 4$, care se poate extinde luând $h(a_2) = 5$, obținând $h(a_2b_1c_2) = 514$, care este un tuplu din *r.* Dacă $h(a_1) = 0$, o aplicație de simboluri transformă cele două tupluri din ipoteză în mulțimea primelor trei tupluri ale lui *r* şi deci $h(b_1)$ este 1 sau 3 şi $h(c_2)$ este 2 sau 4, dar pentru combinațiile 12, 32 şi 34 se poate lua $h(a_2)=0$ şi pentru combinația 14 se poate lua $h(a_2)=5$, obținând de fiecare dată pentru $h(a_2b_1c_2)$ un tuplu al lui *r.* Deci *r* satisface *d.*

a)

a_1	b_1	c_1
a_1	b_2	c_2
a_2	b_1	c_2

(the line separates the hypothesis from the conclusion $a_2\ b_1\ c_2$)

b)

0	1	2
0	3	4
0	3	2
5	1	4

Figura 7.1.

Fie dependența $d = (s_1,...,s_k)|a=b$ şi o relație $r= \{t_1,...,t_m\}$. Spunem că *se poate aplica d Ia r* dacă există o aplicație de simboluri h de la mulțimea $s_1s_2...s_k$ la mulțimea $t_1t_2...t_m$. Efectul aplicării lui d la r folosind aplicația de simboluri h este obținut prin identificarea simbolurilor $h(a)$ şi $h(b)$ ori de câte ori apar în r.

Pentru dependențe $d= (s_h...,s_k)|s$ se aplică d la r folosind aplicația de simboluri h, adăugând la r tuplul $h(s)$. Pentru fiecare simbol unic c din s se creează un simbol care nu mai apare în r şi se defineşte $h(c)$ noul simbol creat.

Exemplul 7.19. Aplicarea dependenței generalizate $(abc, ade, fbe)|a=f$ la relația $r = \{xyz, wyz\}$ folosind aplicația de simboluri g din exemplul 7.17 cu $g(a) = x$ şi $g(f) = w$ identifică în r pe w cu x, obținând $r = \{xyz\}$. Dacă se aplică dependența $(abc, ade, fbe)\ |abq$ lui r folosind g, atunci se adaugă lui r tuplul xyu deoarece $g(a)=x$, $g(b) = y$ şi, q fiind un simbol unic, se introduce un nou simbol u şi se defineşte $h(q) = u$.

BAZE DE DATE

CURS 8

MODELAREA BAZELOR DE DATE RELAȚIONALE

8.1 Forme normale în baze de date
8.1.1 Prima forma normala (FN1)
8.1.2 A doua forma normala (FN2)
8.1.3 A treia forma normala (FN3)
8.1.4 Forma normala Boyce-Codd (FNBC)
8.1.5 A patra forma normala (FN4)
8.1.6 A cincea forma normala (FN5)
8.2 Metode şi tehnici de normalizare a relațiilor
8.2.1 Descrierea procesului de ameliorare a schemei conceptuale
8.2.2 Aducerea relațiilor în FN1
8.2.3 Aducerea relațiilor în FN2
8.2.4 Aducerea relațiilor în FN3
8.2.5 Aducerea relațiilor în FNBC
8.2.6 Aducerea relațiilor în FN4
8.2.7 Aducerea relațiilor în FN5

Modelarea bazelor de date relaționale

Modelarea bazei relaționale este una din cele mai importante sarcini ale proiectantului, utilizatorului şi administratorului bazei de date. Ea prezintă două aspecte semnificative:

1. Aspectul static al modelării- se stabileşte structura datelor (relații, filtre), se stabilesc restricții independente de timp (chei, domenii);
2. Aspectul dinamic al modelării- se descriu acțiunile ce operează pe aceste structuri de date.

Procesul modelării este bazat pe tehnica top-down şi are următoarele faze:

- Obținerea şi formalizarea solicitărilor beneficiarului. Se identifică entități, relații, cardinalitate şi proprietăți relevante ale acestora;
- Integrarea şi sinteza acestei solicitări, adică elaborarea unei scheme conceptuale globale neredundantă, coerentă şi unică;
- Normalizarea relațiilor conceptuale, adică obținerea unor relații mai mici, fără a pierde din informație, pentru a elimina redundanța şi anomaliile la actualizare;
- Optimizarea schemei interne care derivă din aspectul dinamic al modelării şi care este specifică reprezentării fizice a bazei de date. Se fac denormalizări, se realizează compuneri, se alege modul de organizare a fişierelor, metode de acces, etc.

8.1. Forme normale în baze de date

În lucrul cu bazele de bate se manifestă o serie de anomalii datorită dependențelor "nedorite" ce se manifestă între datele din cadrul relațiilor bazei. Aceste dependențe determină creşterea redundanței datelor şi reducerea flexibilității structurii bazei de date. *Formele normale* ale relațiilor sunt definite în raport de anomaliile care pot apare în lucrul cu aceste relații, deci în funcție de anumite dependențe "nedorite".

O relație este într-o *formă normală* particulară dacă satisface o mulțime specificată de restricții. Până în prezent se cunosc cinci forme normale ale relațiilor dintr-o bază de date.

Fie $r[A_1, \ldots, A_n]$ o relație și $X=\{A_{i1}, \ldots, A_{ip}\} \subset \{A_1, \ldots, A_n\}$ o mulțime de atribute. Reamintim că, prin *proiecția* relației r pe X se înțelege $r'[A_{i1}, \ldots, A_{ip}] = \Pi_{A_{i1},\ldots,A_{iip}}(r)$ unde pentru $p=(a_1, a_2, \ldots, a_n)\in r$, avem $\Pi_X p=p[X]=(a_{i1}, a_{i2}, \ldots, a_{ip}) \in r'$ (și toate elementele din r' sunt distincte).

Fie relațiile $r(X,Y)$, $s(Y,Z)$ și X, Y, Z mulțimi de atribute, $X\cap Z=\Phi$. Prin *join-ul natural* al relațiilor r și s se înțelege:

$$r><s=\{(\Pi_X(t), \Pi_Y(t), \Pi_Z(v)) \mid t\in r, v\in s, \Pi_Y(t)=\Pi_Y(v)\}$$

O relație r se poate descompune în mai multe relații noi: $r_1, r_2, \ldots, r_m$. Această descompunere este corectă, dacă: $r= r_1><r_2>< \ldots ><r_m$.

Vom da un exemplu de descompunere care nu este corectă. Fie relațiile:

r [NUME, VARSTA, SALARIU, LOCALITATE]

r_1 [NUME, SALARIU]

r_2 [VÂRSTA, SALARIU, LOCALITATE].

și presupunem că pentru r avem următoarea extensie:

NUME	VÂRSTA	SALARIU	LOCALITATE
Ionescu	30	800000	Arad
Popescu	40	1200000	Oradea
Georgescu	60	1500000	Iași
Călinescu	25	1200000	Arad

În acest caz se obține:

r_1

NUME	SALARIU
Ionescu	800000
Popescu	1200000
Georgescu	1500000
Călinescu	1200000

r_2

VÂRSTA	SALARIU	LOCALITATE
30	800000	Arad
40	1200000	Oradea
60	1500000	Iași
25	1200000	Arad

$r_1><r_2$

NUME	VÂRSTA	SALARIU	LOCALITATE
Ionescu	30	800000	Arad
Popescu	40	1200000	Oradea
Popescu	40	1200000	Arad
Georgescu	60	1500000	Iași
Călinescu	25	1200000	Arad
Călinescu	25	1200000	Oradea

8.1.1 Prima formă normală (FN1)

Este posibil, ca în diverse aplicații practice să apară atribute (simple sau compuse), ce au mai multe valori pentru un element din relație. Aceste atribute formează un *atribut repetitiv*. Prin atribut simplu vom înțelege un singur atribut din relație, iar prin atribut compus o mulțime de atribute (cel puțin două).

Considerăm, de exemplu relația:

persoana[NUME, AN -NAȘTERE, PROFESIA, NUME-COPIL, AN-NAȘTERE-COPIL]

cu atributul NUME cheie primară. Perechea {NUME-COPIL, AN-NAŞTERE-COPIL} este un grup repetitiv, deoarece relația poate avea următoarea extensie:

Popa	1970	inginer	Daniel	1992
			Anca	1994
			Viorel	1998
Ionescu	1966	economist	Andrei	1989
			Magda	1993

De asemenea, relația:

Carte[COTA, AUTOR, TITLU, EDITURA, AN-APARITIE, CUVINTE-CHEIE]

cu atributul COTA cheie primară, are atributele repetitive AUTOR şi CUVINTE-CHEIE. O carte poate avea mai mulți autori şi mai multe cuvinte cheie.

Grupele de atribute repetitive creează greutăți în memorarea diverselor relații şi de accea se încearcă eviterea lor, fără a pierde însă din informații. Dacă $r[A_1, \ldots ,A_n]$ este o relație, unde $A_{m+1}, \ldots ,A_n$ formează un grup repetitiv, atunci relația r se poate descompune în două relații fără atribute repetitive. Dacă $A_1, \ldots ,A_p$, $p<m$, este o cheie pentru relația r atunci cele două relații în care se descompune r sunt:

$$r'[A_1, A_2, \ldots, A_m] = \Pi_{A_1, A_2, \ldots, A_m}(r)$$

$$r''[A_1, A_2, \ldots, A_p, A_{m+1}, \ldots, A_n] = \Pi_{A_1, \ldots, A_p, A_{m+1}, \ldots, A_n}(r)$$

Astfel, relațiile *persoana* şi *carte* se descompun în două, respectiv trei relații:

părinte[NUME, AN-NAŞTERE, PROFESIA]
copil [NUME, NUME-COPIL, AN-NAŞTERE-COPIL]
autori [COTA, AUTOR]
cărti [COTA, TITLU, EDITURA, AN-APARITIE]
cuvinte [COTA, CUVÂNT-CHEIE]

Definiția 8.1. *O relație este în **prima formă normală** (FN1) dacă nu conține grupuri (de atribute) repetitive.*

8.1.2 A doua formă normală (FN2)

Următoarele forme normale utilizează noțiunea de *dependența funcțională* între submulțimi de atribute. Stabilirea dependențelor funcționale este sarcina administratorului bazei şi depinde de semnificația datelor care se memorează în relație. Operațiile de actualizare a datelor (adăugare, modificare, ştergere) nu trebuie să modifice dependențele funcționale existente.

Definiția 8.2. *Fie $r[A_1, \ldots ,A_n]$ o relație şi $X,Y \subset \{ A_1, \ldots ,A_n \}$. Atributul Y este **complet dependent funcțional** de X, dacă Y este dependent funcțional de X ($X®Y$) şi nu este dependent funcțional de nici o submulțime de atribute din X (pentru această dependență funcțională trebuie ca X să fie un atribut compus).*

Fie $r[A_1, \ldots ,A_n]$ o relație şi $C \subset A = \{A_1, \ldots ,A_n\}$ o cheie. Presupunem că există $Y \subset A$, $Y \cap C = \Phi$ (Y nu este cheie), Y dependent funcțional de $X \subset C$ (Y este complet dependent funcțional de o submulțime strictă de atribute din cheie). Dependența $X \rightarrow Y$ se poate elimina dacă relația r se descompune în următoarele două relații:

$$r'[X \cup Y] = \Pi_{X \cup Y}(r)$$

$$r''[A - Y] = \Pi_{A \cup Y}(r)$$

Definiția 8.3. *O relație este în **a doua formă normală** (**FN2**) dacă este de prima formă normală şi orice atribut (simplu sau compus) este complet dependent de cheie sau este inclus în cheie.*

Exemplul 8.1. Se consideră următoarea relație (cu rezultatele la examene):

examen[NUME-STUDENT, DISCIPLINA, NOTA, PROFESOR]

în care cheia este {NUME-STUDENT, DISCIPLINA} şi unei discipline îi corespunde un singur cadru didactic, iar unui cadru didactic pot să-i corespundă mai multe discipline, deci avem dependența funcțională DISCIPLINA→PROFESOR.

De aici deducem că atributul PROFESOR nu este complet dependent funcțional de cheie. Atunci, relația *examen* se poate descompune în următoarele două relații:

apreciere[NUME-STUDENT, DISCIPLINA, NOTA] şi

stat-funcții[DISCIPLINA, PROFESOR]

Dacă dependența funcțională DISCIPLINA→PROFESOR nu este respectată, atunci poate apare o inconsistență. Fie două elemente din relație:

T	……..Disciplina…….Profesor
	……………………………..
t_1	… Analiza … Popa
t_2	… Analiza … Popa
	………………………………

Dacă în t_1 valoarea atributului PROFESOR se schimbă, dar în t_2 nu se face schimbarea, atunci dependența funcțională nu este respectată şi apare o inconsistență (la aceeaşi disciplină apar cadre didactice diferite).

8.1.3 A treia formă normală (FN3)

Definiția 8.4 *Un atribut Z este **tranzitiv dependent** de atributul X dacă există Y astfel încât X®Y, Y®Z, iar Y®X nu are loc şi Z nu este inclus în X∪Y.*

Dacă C este o cheie şi Y un atribut tranzitiv dependent de cheie, atunci există un X care verifică $C \to X$ şi $X \to Y$. Deoarece relația este în forma normală FN2, obținem că Y este complet dependent de C, deci $X \cap C = \Phi$ şi există o dependență $X \to Y$, iar X nu este cheie.

Dacă $r[A_1,\ldots,A_n]$ are cheia C şi există atributul $Y \subset \{A_1,...,A_n\}$, tranzitiv dependent de C şi care nu este cheie (adică $Y \cap C = \Phi$), atunci relația r se poate descompune în următoarele relații (se elimină dependența funcțională $X \to Y$):

$r'[X \cup Y] = \Pi_{X \cup Y}(r)$

$r''[A - Y] = \Pi_{A-Y}(r)$

Definiția 8.5. *O relație r este în a **treia formă normală** (**FN3**) dacă şi numai dacă relația r este în a doua formă normală şi fiecare atribut care nu este cheie (nu participă la o cheie) nu este tranzitiv dependent de nici o cheie a lui r.*

Exemplul 8.2 Se consideră următoarea relație (cu rezultatele obținute de absolvenți la lucrarea de diplomă):

diploma[NUME-ABSOLVENT, NOTA,CADRU-DID-ÎNDR, CATEDRA]

cu cheia NUME-ABSOLVENT.

Se observă că avem următoarele dependențe funcționale:
CADRU-DID-ÎNDR→CATEDRA
NUME-ABSOLVENT→CADRU-DID-ÎNDR

Relația inițială se poate, atunci descompune în următoarele două relații:
rezultate[NUME-ABSOLVENT, NOTA, CADRU-DID-ÎNDR]
îndrumatori[CADRU-DID-ÎNDR, CATEDRA].

8.1.4 Forma normală Boyce-Codd (FNBC)

După definiția formei normale FN3 dată de E. F. Codd, ulterior, au mai apărut o serie de noi definiții:

- O relație *r* este în **fomă normală Boyce-Codd** (**FNBC**) dacă orice determinant este cheie (principală sau secundară).
- O relație este în a **treia formă normală C. J. Date** (**FN3 Date**) dacă orice atribut care nu este cheie, nu este tranzitiv dependent de cheia principală.

Exemplul 8.3 Transportul local pe timp de o săptămână dintr-un oraş este specificat de relația:
transport [ZI, NR-TRASEU, NR-MASINA, COND-AUTO]

unde COND-AUTO este numele conducătorului auto (el conduce o singură maşină, dar pe acea maşină o poate conduce şi un alt conducător). Avem cheia: {ZI, NR-TRASEU, NR-MASINA} şi dependența COND-AUTO→NR-MASINA.
Relația definită este în FN3 Date (NR-MASINA) apare în cheie, dar nu este în FNBC şi se poate descompune în următoarele două relații:

traseu[ZI, NR-TRASEU, NR-MASINA]
şoferi[NR-MASINA, COND-AUTO]

Definiția 8.6. *O relație este în forma normală Boyce-Codd dacă dependențele funcționale netriviale care se manifestă în cadrul relației conțin în partea stângă (ca determinant) o cheie candidată.*

Există mai mulți algoritmi pentru aducerea unei relații în forma normală Boyce-Codd, unul dintre aceştia fiind prezentat în secțiunea prind tehnica normalizării relațiilor.
Orice relație în forma normală Boyce-Codd este în a treia formă normală, dar reciproca nu este adevărată, după cum se ovservă din exemplul următor.

Exemplul 8.4. Relația R=ABC cu dependențele funcționale F={*AB*→C, C→A} este în a treia formă normală, dar nu este în forma normală Boyce-Codd, deoarece cheile acestei relații sunt AC şi BC, iar pentru dependența C→A partea stângă nu conține nici una din cele două chei.

8.1.5 A patra formă normală

Definiția 8.7 *Fie relația* $r[A_1, A_2, \ldots, A_n]$ *şi două mulțimi de atribute* $X, Y \subset \{A_1, \ldots, A_n\}$. *Spunem că Y este* ***multiplu dependent funcțional*** *de X* ($X \twoheadrightarrow Y$) *dacă şi numai dacă pentru orice* $t_1, t_2 \in r$ *pentru care* $\Pi_X(t_1) = \Pi_X(t_2)$ *există* t_3 şi $t_4 \in r$ *astfel încât:*

$\Pi_X(t_1) = \Pi_X(t_2) = \Pi_X(t_3) = \Pi_X(t_4)$
$\Pi_Y(t_1) = \Pi_Y(t_3)$; $\Pi_Y(t_2) = \Pi_Y(t_4)$
$\Pi_{A-X-Y}(t_1) = \Pi_{A-X-Y}(t_4)$; $\Pi_{A-X-Y}(t_2) = \Pi_{A-X-Y}(t_3)$

Dependența $X \twoheadrightarrow Y$ se numeşte *dependență funcțională multiplă* sau *dependență multivaloare* şi se poate se poate reprezenta astfel:

	X	Y	A-X-Y
t_1	v	u_1	w_1
t_2	v	u_2	w_2
t_3	v	u_1	w_2
t_4	v	u_2	w_1

Dacă $A=X \cup Y$ sau $Y \subset X$, atunci dependența $X \rightarrow Y$ se numeşte *trivială*.

Definiția 8.7. *O relație r este în* ***a patra formă normală (FN4)****, dacă pentru toate dependențele funcționale multiple, avem* $X \twoheadrightarrow Y$ *este dependență trivială sau X este cheie pentru r sau: O relație R este în FN4 dacă în cadrul ei nu se manifestă mai mult de o dependență multivaloare.*

Această definiție diferă de definiția formei FNBC doar prin folosirea dependențelor funcționale multiple în locul celor simple.

Exemplul 8.5. Considerăm relația *carte* în care se observă că avem următoarele dependențe funcționale:

COTA $\twoheadrightarrow$ AUTOR; COTA $\twoheadrightarrow$ CUVÂNT-CHEIE;
COTA $\twoheadrightarrow$ {TITLU, EDITURA, AN-APARITIE}

carte

AUTOR	COTA	TITLU	EDITURA	AN APARITIE	CUVINTE CHEIE
Popescu I.	1	Mara	ALL	1990	Rom
Slavici I.	1	Mara	ALL	1990	Rom
Popescu I.	1	Mara	ALL	1990	Roman
Slavici I.	1	Mara	ALL	1990	Roman
Tudor P	2	Baze de date	Teora	1993	Bdate
Ioan S.	2	Baze de date	Teora	1993	Bdate
Vigu T.	2	Baze de date	Teora	1993	Bdate
Tudor P.	2	Baze de date	Teora	1993	Rom
Ioan S.	2	Baze de date	Teora	1993	Rom
Vigu T.	2	Baze de date	Teora	1993	Rom

Pentru a fi în forma FN4, vom descompune relația în următoarele relații:

COTA	**TITLU**	**EDITURA**	**AN-APARITIE**
1	Mara	ALL	1990
2	Baze de date	Teora	1993

COTA	AUTOR
1	Popescu I.
1	Slavici I.
2	Tudor P.
2	Ioan S.
2	Vigu T.

COTA	CUVANT-CHEIE
1	Rom
1	Roman
2	Bdate
2	Rom

8.1.6 A cincea formă normală (FN5)

Definiția 8.8 *Fie relația* $r[A_1, A_2, \ldots, A_n]$ și $r_1[X_1], \ldots, r_m[X_m]$ *o descompunere a relației r. Relația r satisface* ***dependența join*** notată $*(r_1, \ldots, r_m)$, *dacă* $r = r_1 >< r_2 >< \ldots >< r_m$.

Dacă una din relațiile r_i este egală cu r, atunci această dependență este *trivială*.

Să considerăm o relație r și o dependență join $*(r_1, r_2)$ unde $r_1[X]$, $r_2[Y]$ sunt relații. Cu aceste presupuneri, avem: $r = r_1 >< r_2$.

Fie $t_1, t_2 \in r$ și valorile lor date prin următorul tabel:

	X-Y	XÇY	Y-X
$\Pi_X(t_1)$	u_1	v	-
$\Pi_X(t_2)$	u_2	v	-
$\Pi_Y(t_1)$	-	v	w_1
$\Pi_Y(t_2)$	-	v	w_2

Dacă se calculează $r_1 >< r_2$, care este egală cu r, rezultă faptul că mai avem două elemente t_3 și t_4 din r cu valorile următoare:

	X-Y	XÇY	Y-X
t_1	u_1	v	w_1
t_2	u_2	v	w_2
t_3	u_1	v	w_2
t_4	u_2	v	w_1

De aici, se deduce că $X \cap Y \rightarrow\rightarrow X$ sau $X \cap Y \rightarrow\rightarrow Y$, deci dependența join $*(r_1, r_2)$ este echivalentă cu dependența funcțională multiplă.

Definiția 8.9. *O relație este în* ***forma normală cinci (FN5)*** *cu respectarea unei mulțimi D de dependențe funcționale multiple sau join, dacă fiecare dependență* $*(r_1, \ldots, r_m)$ *este fie trivială, fie* X_i *este cheie (avem* $r_i[X_i]$*) pentru r, pentru toate valorile lui i.*

Cu alte cuvinte, o relație r este în FN5 dacă orice dependență join definită pe r este implicată de cheile candidat ale lui r.

Exemplul 8.6. Fie relația *cursa* [CP#, CA#, PD, PA], unde CP-codul pilotului, CA-codul avionului, PD și PA punctul de decolare, respectiv aterizare.

În această relație, care este în FN4, nu există dependențe funcționale multiple, dar există o redundanță logică care va ridica probleme la actualizare.

cursa

CP#	CA#	PA	PD
11	100	Sibiu	Iaşi
10	100	Iaşi	Sibiu
10	100	Sibiu	Iaşi
10	101	Sibiu	Iaşi

Descompunem relația *cursa* prin proiecție în:

r_1 (CP#, CA#)
r_2 (CP#, PD, PA)
r_3 (CA#, PD, PA)

Se observă că, $cursa \neq r_1 >< r_2$; $cursa \neq r_2 >< r_3$; $cursa \neq r_1 >< r_3$), dar $cursa = r_1 >< r_2 >< r_3$.

În relația $r_1 >< r_2$ a apărut un tuplu (10, 101, Iaşi, Sibiu) care nu există în *cursa*. Dacă ar fi existat, ar fi avut loc multidependența CP$\twoheadrightarrow$CA şi astfel descompunerea reversibilă a relației *cusa* în r_1 şi r_2.

r_1

CP#	CA#
11	100
10	100
10	101

r_2

CP#	PD	PA
11	Sibiu	Iaşi
10	Iaşi	Sibiu
10	Sibiu	Iaşi

r_3

CA#	PD	PA
100	Sibiu	Iaşi
100	Iaşi	Sibiu
101	Sibiu	Iaşi

$r_1 >< r_2$

CP#	CA#	PD	PA
11	100	Sibiu	Iaşi
10	100	Iaşi	Sibiu
10	100	Sibiu	Iaşi
10	101	Iaşi	Sibiu
10	101	Sibiu	Iaşi

În practică se utilizează destul de rar formele normale 4 şi 5 deoarece acestea conduc la o descompunere a unei relații în multe subrelații, ceea ce măreşte timpul de răspuns la interogări precum şi spațiul de memorare.

8.2. Metode şi tehnici de normalizare a relațiilor

8.2.1 Descrierea procesului de ameliorare a schemei conceptuale

Proiectarea schemei conceptuale a unei baze de date presupune parcurgerea următoarelor etape:

1. Determinarea formei normale în care trebuie să se afle relațiile din baza de date. În majoritatea cazurilor bazele de date relaționale sunt constituite din relații aflate în FN1 sau FN2. Acest lucru se explică prin faptul că formele normale superioare, deşi reduc dificultatea de realizare a operațiilor de actualizare, reduc în acelaşi timp şi performanțele operațiilor de regăsire a datelor. Relațiile aflate în forme normale superioare conțin un număr mic de atribute

şi acest lucru favorizează operațiile de actualizare a datelor, dar îngreunează procesul de regăsire a lor, deoarece satisfacerea cererilor de date impune interogarea simultană a mai multor relații, deci efectuarea unor operații de join, care sunt costisitoare în termenii resurselor de calcul solicitate.

2. Stabilirea relațiilor care să facă parte din BD, în forma normală precizată la etapa anterioară. Presupune definirea schemei relațiilor şi a restricțiilor de integritate. Modul prin care se stabileşte mulțimea de relații din baza de date, se numeşte tehnica normalizării relațiilor.

3. Descrierea schemei conceptuale în limbajul de descriere a datelor utilizat de SGBD-ul relațional ce se utilizează.

În obținerea unei baze de date performantă, un rol important îl are *tehnica normalizării relațiilor*. Această tehnică permite obținerea schemei conceptuale printr-un proces de ameliorare progresivă a unei scheme concepute inițial, prin utilizarea formelor normale. După fiecare etapă de ameliorare, relațiile din bază ating un anumit grad de perfecțiune prin eliminarea unui anumit tip de dependențe nedorite (dependențe funcționale parțiale, tranzitive, multivaloare), deci se află într-o anumită formă normală.

Procesul de ameliorare, trebuie să satisfacă următoarele cerințe:

- să garanteze *conservarea datelor*, adică în schema conceptuală finală trebuie să figureze toate datele din schema inițială;
- să garanteze *conservarea dependențelor* dintre date, adică în schema finală fiecare dependență trebuie să aibă determinantul şi determinatul în schema aceleiaşi relații;
- să reprezinte o *descompunere minimală* a relațiilor inițiale. Nici una din relațiile care compun schema finală nu trebuie să fie conținută într-o altă relație din această schemă.

Necesitatea normalizării este ilustrată în exemplul următor.

Fie schema relațională *avion* (NR, TIP, CAPACITATE, LOCALITATE), cu cheia primară numărul avionului (NR).

avion

NR	TIP	CAPACITATE	LOCALITATE
100	IAR500	90	BRAŞOV
101	IAR500	90	ARAD
102	ROMBAC	100	BUCUREŞTI
103	TU154	200	TIMIŞOARA

Presupunem că în cadrul companiei, există restricția: "toate avioanele de acelaşi tip au aceeaşi capacitate" care este de fapt o dependență funcțională de forma TIP→CAPACITATE.

Datorită acestei dependențe, pot exista redundanțe în date sau pot să apară anomalii la reactualizare. Astfel, în relația de mai sus, avem o redundanță logică (perechea <IAR 500, 90> apare de mai multe ori) precum şi anomalii la reactualizare: dacă dorim să ştergem avionul cu numărul 102, vom pierde informația care ne arată că un avion ROMBAC are capacitatea 100.

De asemenea, dacă modificăm capacitatea avionului IAR 500, de la 90 la 190 de locuri putem întâlni următoarele anomalii: modificând un singur tuplu, relația devine incoerentă (restricția nu mai este verificată), iar dacă modificăm toate tuplurile cu IAR 500, costul modificării creşte semnificativ.

Prezentăm în continuare procedeul de ameliorare a schemei conceptuale inițiale, care constă în aducerea acesteia la diferite forme normale.

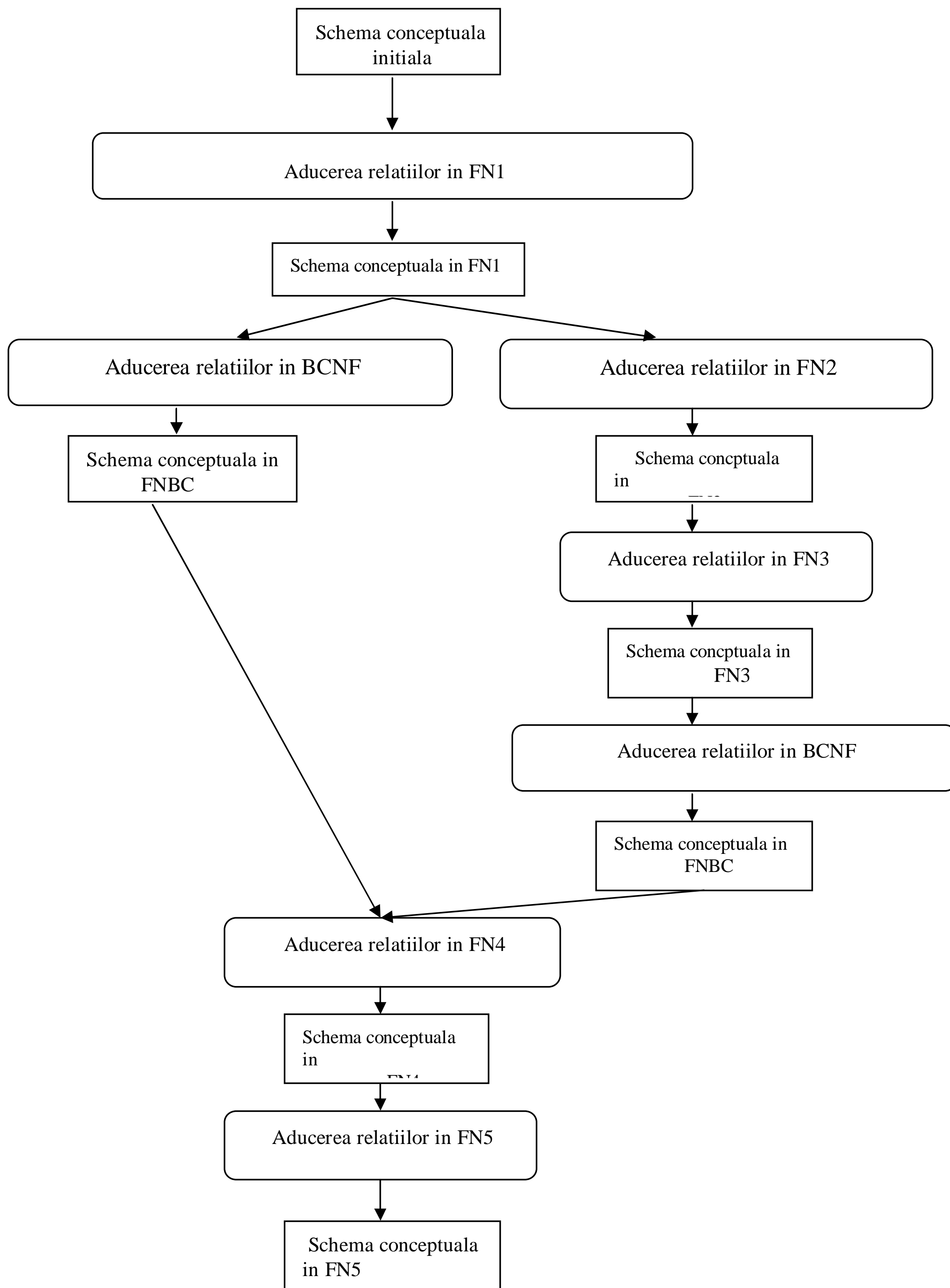


Fig. 8.1 Etapele procesului de ameliorare a schemei conceptuale

8.2.2 Aducerea relațiilor în FN1

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

1. Se trec în relație, în locul atributelor compuse componentele acestora, ca atribute simple.

2. Se plasează grupurile de atribute repetitive, fiecare în câte o nouă relație.

3. Se introduce în schema fiecărei noi relații creată în pasul 2 cheia primară a relației din care a fost extras grupul repetitiv.

4. Se stabileşte cheia primară a fiecărei relații creată în pasul 2. Aceasta va fi compusă din atributele adăugate la relație în pasul 3, precum şi din unul sau mai multe atribute proprii relației.

Pentru exemplificarea procesului de aducere a unei relații în FN1 se consideră relația *Persoana*, cu schema prezentată în fig. 8.2.

Persoana

Marca	Numep	Adresa			$Prenc_1$	Data naşterec$_1$	$Prenc_2$	Data naşterec$_2$	...
		Str	Loc	Cod					

Fig.8.2. Schema relației nenormalizate *Persoana*

Relația *Persoana* are un atribut compus, denumit "Adresa" şi un grup de atribute repetitive, format din atributele "Prenc" şi "Datanaşterec". Rezultatele aducerii relației *Persoana* în FN1 sunt prezentate în fig. 8.3.

Pers

Marca	Numep	Stradr	Locadr	Codadr

Copil

Precopil:	Data nasterec	Marca

Fig. 8.3. Relațiile obținute prin aducerea relației din fig. 8.2. în FN1

8.2.3 Aducerea relațiilor în FN2

Presupune eliminarea dependențelor funcționale parțiale din relațiile aflate în FN1.

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

1. Pentru fiecare dependență funcțională parțială se crează o nouă relație, cu schema constituită din determinantul şi determinatul acestei dependențe.

2. Dacă în relația inițială există mai multe dependențe funcționale parțiale cu acelaşi determinant, pentru toate acestea se creează o singură relație cu schema constituită din determinantul, luat o singură dată şi din determinații dependențelor considerate.

3. Se determină cheia primară a fiecărei noi relații creată în pasul 1. Aceasta va fi formată din atributul/atributele din determinantul dependenței funcționale parțiale, care a stat la baza constituirii relației.

4. Se analizează relațiile rezultate la pasul 1. Dacă aceste relații conțin dependențe funcționale parțiale se reia procesul de aducere în FN2, altfel procesul s-a terminat..

Pentru exemplificare s-a recurs la o relație *Reper*, a cărei schemă este prezentată în fig. 8.4.

Reper

Codprod	Codreper	Codsecție	Codmaşină	Nroper	Codoper	Categoper	Timp preg.	Timp exec.

Fig. 8.4. Schema relației *Reper*

Relația R are două chei candidate şi anume:
(Codreper, Codmaşină, Codoper)
(Codreper, Codmaşină, Nroper)
Dintre acestea, prima se alege drept cheie primară.
În relația din fig. 8.4. se manifestă următoarele dependențe funcționale parțiale:
Codoper $\rightarrow$Categoper
Nroper $\rightarrow$Categoper
Prin aplicarea operațiilor de aducere a acestei relații în FN2 se obțin relațiile din fig. 8.5.

Reper1

Codprod	Codreper	Codsecție	Codmasina	Nroper:	Codoper	Timp preg.	Timp exec.

R_1

Codoper	Categoper

R_2

Nroper	Categoper

Fig. 8.5. Relațiile obținute prin aducerea relației din fig. 8.4. în FN2

Dependențele funcționale parțiale din cadrul relației din fig. 8.4. au fost ransformate în următoarele dependențe funcționale complete:
Codoper $\rightarrow$ Categoper, în cadrul relației R_1
Nroper $\rightarrow$ Categoper, în relația R2

Dacă se notează cu R o relație în FN1 care trebuie adusă în FN2 şi cu (A,B) cheia acestei relații R(A, B, C, D, ...) şi dacă se consideră dependența B $\rightarrow$ C drept singura dependență funcțională parțială care se manifestă în R, atunci aducerea lui R în FN2 determină descompunerea relației în două relații R l şi R2, cu schemele:
R1(A, B, D, ...)
R2(B, C)

8.2.4 Aducerea relațiilor în FN3

Presupune aducerea unei relații FN2 în FN3 prin eliminarea dependențelor tranzitive.

1. Pentru fiecare dependență funcțională tranzitivă se transferă atributele implicate în dependență tranzitivă într-o nouă relație.
2. Se determină cheia primară a fiecărei noi relații creată la pasul 1.
3. Se introduc în relația inițială în locul atributelor transferate, cheile primare determinate la pasul 2.
4. Se reanalizează relația inițială. Dacă în cadrul ei există noi dependențe tranzitive, atunci se face transfer la pasul 1, altfel procesul de aducere la FN3 s-a terminat.

Dacă se notează cu R o relație în FN2 care trebuie adusă în FN3 şi cu X cheia acestei relații R(X, A, B, ...) şi 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 şi R_2, cu schemele:

R_1(X, A, ...)

R_2(A, B)

A treia formă normală poate fi obținută şi cu ajutorul unei ***scheme de sinteză.*** Algoritmul de sinteză construieşte o acoperire minimală F^+ a dependențelor funcționale totale. Se elimină atributele şi dependențele funcționale redundante. Mulțimea F este partiționată în grupuri F_i, astfel încât în fiecare grup F_i sunt dependențe funcționale care au acelaşi membru stâng şi nu există două grupuri cu acelaşi membru stâng. Fiecare grup F_i produce o schemă FN3. Algoritmul realizează o descompunere ce conservă dependențele.

Vom ilustra algoritmul pe un exemplu. Fie $A_1, A_2, ..., A_m$ o mulțime de atribute şi fie E o mulțime de dependențe funcționale $f_1, f_2, .., f_n$ de forma $f_i : X_i \rightarrow Y_j$, unde $X_i = A_{i_1}, A_{i_2}, .., A_{i_k}$ şi $Y_j = A_{j_1}, A_{j_2}, ..., A_{j_l}$.

Concret, fie :

$f_1 : F \rightarrow N; f_2 : F \rightarrow P; f_3 : P, F, N \rightarrow U; f_4 : P \rightarrow C; f_5 : P \rightarrow T; f_6 : C \rightarrow T; f_7 : N \rightarrow F$ o mulțime de dependențe funcționale.

Ideea schemei de sinteză este de a regrupa dependențele funcționale cu acelaşi membru stâng: $F_1 = \{f_1, f_2\}; F_2 = \{f_3\}; F_3 = \{f_4, f_5\}; F_4 = \{f_6\}; F_5 = \{f_7\}$ care conduc la schemele relaționale:

r1(F#, N, P)

r2(P#, F#, N#, U)

r3(P#, C, T)

r4(C#, T)

r5(N#, F)

Aceste relații nu sunt în FN3. De exemplu, N este atribut redundant în f_3 deoarece $F \rightarrow N; r4 \subseteq r3$ şi există tranzitivitatea $P \rightarrow C, C \rightarrow T; r5 \subseteq r1$ şi $F \rightarrow N, N \rightarrow F$. Prin urmare, trebuie eliminate atributele şi dependențele redundante.

Algoritmul de sinteză îşi propune:

1. *Suprimarea atributelor redundante.* Atributul A_i este redundant în dependența funcțională $A_1, ..., A_i, ..., A_n \rightarrow Y$ dacă putem genera dependența funcțională $A_1, ..., A_{i-1}, A_{i+1}, ..., A_n \rightarrow Y$ plecând de la mulțimea inițială E de dependențe funcționale şi de la axiomele lui Amstrong.

Pentru exemplul considerat:

f_1 : F→N; f_3 : P, F, N → U sau N, P, F → U.

Aplicând axioma 3 se obține F, P, F→ U deci P, F → U

2. *Suprimarea dependențelor funcționale redundante*. Dependența funcțională f este redundantă în E dacă $E^+=(E - f)^+$ unde E^+ reprezintă închiderea lui E. În cazul exemplului considerat se observă că f_5 este redundantă, deoarece poate fi obținută din f_4 şi f_6.

La sfârşitul acestei etape se obține:

f_1 : F→N; f_2 : F→P; f_3 : P,F→U; f_4 : P→C; f_6 : C→T; f_7 : N→F

3. *Gruparea dependențelor cu acelaşi membru stâng*. În cazul exemplului considerat:

$F_1=\{f_1, f_2\}$; $F_2=\{f_3\}$; $F_3=\{f_4\}$; $F_4=\{f_6\}$; $F_5=\{f_7\}$

4. Regruparea mulţimilor F_i şi F_j dacă există dependenţe de forma $X \rightarrow Y$şi $Y \rightarrow X$, unde X este partea stângă a dependenţei lui F_i şi Y este partea stângă a dependenţei lui F_j.
Grupând F_1 şi F_5 se obţine:

$$F_1' = \{f_1, f_2, f_3\}; F_2=\{f_3\}; F_3=\{f_4\}; F_4=\{f_6\}$$

5. Generarea relaţiilor FN3. Pentru exemplul considerat, se obţin schemele relaţionale:

r1(F#, N, P), r2(P#, F#, U), r3(P#, C), r4(C#, T).

Algoritmul BFN3 permite aducerea unei relaţii în FN3 şi corespunde schemei de sinteză comentate anterior. Algoritmul solicită determinarea unei acoperiri minimale (algoritm ELIMA şi ELIMF) şi determinarea închiderii A^+ a unei mulţimi de atribute A în raport cu o mulţime de dependenţe funcţionale E (algoritm INCHID).

Algoritmul INCHID

1. Se caută dacă există în mulţimea E dependenţe funcţionale $X \rightarrow Y$ pentru care determinantul reprezintă o submulţime a lui A, iar determinatul nu este inclus în mulţimea A ($X \subset A$, $Y \not\subset A$).
2. Pentru fiecare astfel de dependenţă funcţională se adaugă mulţimii A, atributele care constituie determinantul dependenţei.
3. Dacă nu mai există nici o dependenţă funcţională de tipul dependenţelor de la pasul 1, atunci $A^+=A$.

Fie E o mulţime de dependenţe funcţionale. Un atribut A este redundant dacă prin eliminarea lui din partea stângă a dependenţei funcţionale $X \rightarrow Y$ se obţine dependenţa funcţională $X-\{A\} \rightarrow Y$ care de asemenea este în E.

Algoritmul ELIMA permite eliminarea atributelor redundante din determinantul dependenţelor funcţionale.

Algoritmul ELIMA

Pentru fiecare dependenţă funcţională din E şi pentru fiecare atribut din partea stângă a unei dependenţe funcţionale:
1. Se elimină atributul considerat;
2. Se calculează închiderea părţii stângi reduse;
3. Dacă această închidere conţine toate atributele din determinantul dependenţei funcţionale, atunci atributul eliminat la pasul 1 este redundant şi rămâne eliminat. În caz contrar, atributul nu este redundant şi se reintroduce în partea stângă a dependenţei funcţionale.

Algoritmul ELIMF elimină dependenţelor funcţionale redundante din mulţimea E.

Algoritm ELIMF

Pentru fiecare dependenţă funcţională $X \rightarrow Y$ din E:
1. Se elimină dependenţa din E;
2. Se calculează închiderea X^+, în raport cu mulţimea redusă de dependenţe;
3. Dacă Y este inclus în X^+, atunci dependenţa $X \rightarrow Y$ este redundantă şi rămâne eliminată.

În caz contrar, dependenţa nu este redundantă şi se reintroduce în mulţimea E.

Determinarea acoperirii minimale a unei mulţimi de dependenţe funcţionale presupune:
- eliminarea atributelor redundante (algoritm ELIMA);
- eliminarea dependenţelor funcţionale redundante (algoritm ELIMF).

Acoperirea minimală nu este unică şi depinde de ordinea în care sunt eliminate atributele şi dependenţele funcţionale redundante.

Două mulțimi de atribute X, Y sunt *chei echivalente* dacă în mulțimea de dependențe E există atât dependența $X \rightarrow Y$, cât şi dependența $Y \rightarrow X$

Algoritm BFN3

1. Se determină F o acoperire minimală a lui E.
2. Se descompune mulțimea F în grupuri notate F_i, astfel încât în cadrul fiecărui grup să existe dependențe funcționale ce au aceeaşi parte stângă.
3. Se determină perechile de chei echivalente (X, Y) în raport cu F.
4. Pentru fiecare pereche de chei echivalente:
- se identifică grupurile F_I şi F_j care conțin dependențele funcționale cu partea stângă X şi respectiv Y;
- se formează un nou grup de dependențe F_{ij}, care va conține dependențele funcționale ce au membrul stâng (X, Y);
- se elimină grupurile F_i şi F_j, iar locul lor va fi luat de grupul F_{ij}.
5. Se determină o acoperire minimală a lui F, care va include toate dependențele $X \rightarrow Y$, unde X şi Y sunt chei echivalente (celelalte dependențe sunt redundante).
6. Se construiesc relații FN3 (câte o relație pentru fiecare grup de dependențe funcționale).

8.2.5 Aducerea relațiilor în FNBC

Presupune eliminarea dependențelor funcționale care încalcă cerințele formei normale Boyce-Codd, şi anume a dependențelor a căror determinanți nu sunt chei candidat. Aceste dependențe funcționale mai sunt cunoscute şi sub numele de *dependențe noncheie*.

Pentru ca o relație să fie adusă în FNBC nu trebuie, în mod obligatoriu să fie în FN3. Se pot aduce în FNBC şi relații aflate în FN1 sau FN2. Acest lucru este posibil întrucât dependențele funcționale parțiale şi cele tranzitive sunt de fapt, tot dependențe noncheie.

Există trei categorii de dependențe noncheie şi anume:

- dependențe funcționale parțiale;
- dependențe funcționale tranzitive;
- dependențe noncheie, altele decât cele din categoriile 1 şi 2.

Într-o relație aflată în FN3 se manifestă numai dependențe noncheie din categoria 3 (cele din categoriile 1 şi 2 au fost eliminate în procesul aducerii relației în FN3).

Într-o relație aflată în FN2 se pot manifesta dependențe noncheie din categoriile 2 şi 3, iar într-o relație în FN1 pot exista dependențe noncheie din toate cele trei categorii.

A aduce o relație în FNBC înseamnă a elimina toate tipurile de dependențe noncheie care se manifestă în cadrul ei.

În general, se consideră ca puncte de plecare în procesul de aducere la BCNF, FN1 şi FN3. În cazul relațiilor în FN1, procedura de aducere în FNBC recurge la un procedeu unitar pentru eliminarea tuturor categoriilor de dependențe noncheie.

Când se lucrează cu relații în FN3, procedura de aducere în FNBC utilizează o metodă specifică de eliminare a dependențelor noncheie din categoria 3. În acest din urmă caz, dependențele noncheie din cadrul unei relații se elimină treptat şi anume: prin procedura de aducere a relației în FN2, prin cea de aducere in FN3 şi, respectiv prin procedura de aducere din FN3 în FNBC.

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

1. Se analizează relația, pentru a se identifica dependențele noncheie. Astfel, dacă relația conține numai unul sau două atribute nu pot exista dependențe noncheie, deci relația se află în FNBC. Dacă relația conține mai mult de două atribute, se identifică eventualele

dependențe noncheie. Dacă există astfel de dependențe se trece la pasul următor. Dacă nu, relația este în FNBC şi procesul s-a terminat.

2. Se reduce progresiv schema relației inițiale şi se aplică operațiile de identificare a dependențelor noncheie de la pasul 1. Ori de câte ori, prin reducerea schemei relației inițiale se obține o relație în FNBC, se consideră că aceasta face parte din descompunerea relației inițiale, în procesul aducerii ei la FNBC.

Procesul de aducere a unei relații din FN3 în FNBC se desfăşoară astfel:

1. Se analizează relația, pentru a se identifica dependențele noncheie. Astfel, dacă relația conține unul sau cel mult două atribute nu pot exista dependențe noncheie, deci relația este în FNBC şi procesul a luat sfârşit. Dacă relația conține mai mult de două atribute în cadrul ei pot exista dependențe noncheie şi se trece la identificarea lor. Dacă nu există astfel de dependențe, relația este în FNBC şi procesul a luat sfârşit, altfel se trece la pasul 2.

2. Pentru fiecare dependență noncheie $X \rightarrow Y$ se crează două relații, una cu schema formată din atributele reprezentate prin X şi Y şi cealaltă, cu schema constituită din toate atributele relației inițiale, mai puțin atributele reprezentate prin Y. Aceste două relații reprezintă descompunerea relației inițiale în procesul aducerii ei în FNBC.

3. Se reia procesul de aducere în FNBC pe relațiile obținute la pasul 2.

Pentru exemplificarea procedurilor de aducere a unei relații în BCNF se consideră relația *forma*, cu schema prezentată în fig. 8.6.

forma

$\underline{K_1}$	$\underline{K_2}$	B	D

Fig. 8.6. Relație aflată în FN3

Să presupunem că în cadrul relației *forma* 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 noncheie (categoria 3)

Relația *forma* se află în FN3 şi prin aplicarea procedurii de aducere în BCNF se obțin rezultatele din fig. 8.7.

Se observă că descompunerea relației *forma* în relațiile *forma*$_1$ şi *forma*$_2$ conservă datele şi este minimală, dar nu conservă dependențele între date, întrucât nici una din relații nu înglobează dependența funcțională $(K_1, K_2) \rightarrow B$.

forma$_1$

$\underline{D}$	K_1

forma$_2$

$\underline{K_2}$	B	D

Fig. 8.7. Relațiile obținute prin aducerea relației din fig. 8.6. în FNBC

8.2.6 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 FNBC în FN4 cuprinde următorii pași:

1. Se identifică dependențele multivaloare $X \twoheadrightarrow Y$ din cadrul relației considerate.
2. Se izolează fiecare atribut multivaloare *Y*, împreună cu atributele care depind funcțional de acesta, într-o relație separată.

În cadrul relației *Reper1* din fig.8.5. se manifestă următoarele dependențe multivaloare:

Codreper $\twoheadrightarrow$ Codprodus

Codmaşină $\twoheadrightarrow$ Codsecție

Deci relația *Reper1* nu se află în FN4. Rezultatele aducerii relațiilor din fig. 8.5 în FN4 sunt prezentate în fig. 8.8.

ReperFN4

Codreper	Codmaşină	Nroper	Codoper	Timppreg	Timpexec:

R_1

Codoper	Categoper

R_2

Nroper	Categoper

R_3

Codreper	Codprod

R_4

Codmasina	Codsectie:

Fig. 8.8. Relațiile obținute prin aducerea relațiilor din fig. 8.5. în FN4

8.2.7 Aducerea relațiilor în FN5

Presupune eliminarea dependențelor join din cadrul relațiilor aflate în FN4. Procesul de aducere a unei relații din FN4 în FN5 se desfăşoară astfel:

1. Se identifică dependențele join. Între mulțimile de atribute *A*, *B* şi *C* din cadrul unei relații există o dependență join atunci când există dependențe multivaloare între fiecare dintre perechile de mulțimi: (*A, B*), (*B, C*) şi (*A, C*). Prin urmare, o dependență join poate exista numai în cadrul acelor relații în FN4 care prezintă chei compuse şi atribute comune în chei. Dacă există dependențe join în cadrul relației considerate se trece la pasul 2. Dacă nu, procesul de aducere a relației în FN5 se încheie.
2. Se descompune relația inițială, în scopul obținerii FN5. Considerând că schema relației conține mulțimile de atribute *A*, *B* şi *C* şi că între fiecare pereche (*A, B*), (*B, C*), (*A, C*) există dependențe multivaloare, relația trebuie descompusă în trei relații: $r_1(A,B)$, $r_2(B,C)$ şi $r_3(A,C)$.

BAZE DE DATE

CURS 9

STRUCTURA FIZICĂ A BAZELOR DE DATE

9.1 Consideraţii privind structura fişierelor
9.2 Tipuri de organizare a fişierelor
9.3 Metode de căutare in fişiere
9.4 Metode de memorare pentru înregistrări cu lungime variabila

9.1 Consideraţii privind structura fişierelor

În acest capitol vom descrie nivelul fizic al bazelor de date. Plecând de la necesitatea reprezentării informaţiilor şi a legăturilor între informaţiile ce constituie o bază de date vom descrie metode de organizare şi de operare cu astfel de structuri folosind mediile de memorare.

Nivelul la care se face gestionarea informatiilor de catre sistemele de operare actuale este cel de fişier. După conţinut, fişierele se împart în mai multe clase dintre care cele mai des utilizate sunt cele ce urmează:

- *directoarele* sunt fişierele care dau informatii despre alte fişiere; cu ajutorul directoarelor se pot contrui structuri arborescente ce permit accesul la oricare fişier din sistem plecând de la un director initial numit radacina; orice nod interior al arborelui de structura corespunzator fişierelor este un director; de obicei, directoarele nu sunt frunze în arbore (exceptie sunt doar directoarele vide);
- *fişierele de date* contin informatii ce pot fi prelucrate de programe;
- *fişierele text* contin informatii alfanumerice de informare a utilizatorilor sau diferite documente memorate în sistem;
- *fişierele cod sursa* contin programe scrise intr-un limbaj de programare;
- *fişierele cod obiect* contin programe compilate.
- *fişierele executabile* contin programe ce pot fi lansate în executie; un caz particular il reprezinta *fişierele de comenzi* care contin o succesiune de comenzi ale sistemului de operare sau lansari de alte programe.

Principalele caracteristici ce definesc fiecare fişier sunt numele fişierului, tipul fişierului, lungimea, locul de memorare, modul de acces, data crearii sau a ultimei modificari şi alte informatii. Tipul acestor informatii şi modul lor de reprezentare difera de la sistem la sistem.

Elementele componente ale unui fişier sunt *înregistrările*. Fiecare înregistrare contine informatiile corespunzatoare unui obiect de tipul celor pentru care s-a construit fişierul. Fiecarei informatii îi corespunde un tip, un domeniu de valori posibile, o lungime de reprezentare şi o pozitie în înregistrare. Toate acestea definesc un *câmp* al înregistrării. Structura înregistrărilor este descrisa de formatul înregistrării asociat fiecarui fişier.

Operatiile curente cu un fişier se reduc de cele mai multe ori la patru tipuri: inserare, ştergere, modificare şi căutare. Inserarea presupune introducerea unei noi înregistrări, ştergerea presupune eliminarea unei înregistrări şi modificarea presupune schimbarea unor valori ale unor câmpuri intr-o înregistrare. Aceste operatii nu schimba modul de organizare şi modul de acces asociate fişierului. Căutarea presupune determinarea unor valori sau localizarea unor valori sau o combinatie a lor în functie de anumite calitati sau proprietati pe care trebuie sa le indeplineasca.

Unitatea de transfer de informatii între fişier, memorat pe un mediu şi memoria interna este *blocul*. Un bloc are de obicei lungimea o putere a lui 2 cuprinsă între 2^9 şi 2^{12} octeti.

Fiecarui bloc i se asociaza o *adresa*. Un bloc poate sa contina una sau mai multe înregistrări. De regula o înregistrare nu se poate memora în mai mult decat un bloc cu exceptia înregistrărilor cu lungimi mai mari decat lungimea blocului (caz rar intalnit) dar în acest caz nu pot apare informatii pentru doua înregistrări diferite în acelaşi bloc.

Înregistrările pot sa fie localizate în mai multe feluri: prin *adresa absoluta* pe mediu de memorare care se face indicând cilindrul, pista, sectorul şi adresa în sector a inceputului înregistrării (metoda mai rar utilizata în diferitele limbaje de programare), prin indicarea numarului asociat înregistrării în cadrul fişierului, prin indicarea distantei fata de inceputul fişierului a inceputului înregistrării, prin indicarea blocului şi a adresei relative în bloc (numarul înregistrării din bloc sau distanta pana la capatul blocului). Referirea înregistrărilor (*pointeri la înregistrare*) se poate face prin oricare din metodele de adresare de mai sus. Un alt mod de adresare este prin intermediul valorilor unor câmpuri ale înregistrărilor, de cele mai multe ori ale câmpurilor corespunzatoare *cheilor*.

Performantele unui sistem depind în foarte mare masura de timpul necesar accesului la bloc. Acest timp este determinat de caracteristicile fizice ale sistemului de calcul, de sistemul de operare folosit, de limbajul de programare utilizat şi de modul de organizare a fişierului.

Pentru gestionarea blocurilor şi a înregistrărilor în blocuri, fiecare bloc are la inceput o eticheta care poate sa contina numarul înregistrărilor ocupate, biti de ocupare sau biti de ştergere.

9.2 Tipuri de organizare a fişierelor

În organizarea fişierelor se tine seama de mai mulți factori cum ar fi frecventa cu care se efectueaza anumite tipuri de operatii asupra fişierelor, câmpurile implicate în operatiile de căutare (*cheile fişierului*) sau relatia în care se afla înregistrările fişierului cu alte informatii din sistem. Dacă exita ponteri la unele înregistrări din fişier se spune ca fişierul este cu *înregistrări fixate* şi în acest caz fiecare înregistrare ramane, de obicei pe locul pe care a fost introdusa, locul ocupat de o astfel de înregistrare nu mai poate fi refolosit după eliminarea înregistrării. În caz contrar se spune ca fişierul este cu *înregistrări nefixate*, înregistrările putand fi mutate sau spațiul eliberat prin eliminarea unor înregistrări putand fi reutilizat.

Cele mai des utilizate tipuri de organizare a fişierelor sunt: secvential, cu dispersie, cu index rar, cu index dens, cu structura de B-arbore. Aceste cinci tipuri de organizare a fişierelor vor fi descrise pe scurt în cele ce urmeaza.

9.2.1 Fişiere secvențiale

Fişierele secventiale (heap files) presupun înregistrările memorate ca elementele unei liste liniare de cele mai multe ori în blocuri consecutive, legate între ele prin pointeri sau prin construirea unui tabel separat cu adresele acestor blocuri ce permit accesul la ele.

Inserarea unui element se face de cele mai multe ori prin adaugarea noului element la sfarşitul listei (în special în cazul fişierelor cu înregistrări fixate şi neordonate). Inserarea se mai poate face prin includerea noii înregistrări pe primul loc disponibil pentru fişiere cu înregistrări nefixate şi cu biti de ştergere, prin includerea noii înregistrări intr-un bloc existent sau unul nou şi legarea ei în lista pentru înregistrări fixate sau ordonate, prin deplasarea fizica a unor înregistrări pentru a face loc noii înregistrări în cazul fişierelor cu înregistrări nefixate şi ordonate sau prin alte tehnici asemanatoare.

Ştergerea unui element se face prin inlocuirea elementului care se elimina cu ultimul element al fişierului, prin deplasarea elementelor care urmeaza înregistrării care se elimina cu un element catre inceputul fişierului pentru fişiere cu înregistrări nefixate şi fara bit de ştergere, eventual ordonate, prin modificarea bitului de ştergere sau prin eliminarea din lista cu legaturi şi marcarea înregistrării ca libera. Pentru fişierele cu înregistrări nefixate pentru care unele blocuri nu mai contin înregistrări, acele blocuri sunt eliberate şi pot fi reutilizate.

Modificarea unui element se face de obicei prin citirea înregistrării corespunzatoare, modificarea câmpurilor implicate şi rescrierea în acelaşi loc a valorilor rezultate. Pentru fişierele ordonate pentru care sunt afectate câmpuri ce contribuie la ordonare operatia de modificare presupune citirea valorilor înregistrării implicate, ştergerea înregistrării din fişier, modificarea valorilor câmpurilor şi inserarea unei noi înregistrări cu valorile reactualizate.

Căutarea unui element se poate face prin parcurgerea secventiala a tuturor elementelor fişierelor şi verificarea conditiilor de selectionare a înregistrării examinate. Aceasta metoda presupune în medie examinarea a (N+1)/2 înregistrări la o căutare cu succes şi a N înregistrări la o căutare fara succes, unde N este numarul de înregistrări din fişier. Dacă fişierul este ordonat şi se face căutaredupă cheie (câmpuriledupă care s-a facut ordonarea) se obține o eficienta mai buna în cazul cautarii secventiale la căutarea fara succes şi anume în medie (N+1)/2 înregistrări examinate dar mai eficiente sunt cautarile binare care dau în medie log N înregistrări examinate sau cautarile cu calculul adresei care dau în medie log log N înregistrări examinate.

Avantaje: programe simple de organizare şi utilizare, nefolosirea sau folosirea în mica masura a spațiului suplimentar pentru legaturi sau alte informatii, posibilitati multiple de reorganizare, uneori permit operare simpla utilizabila pentru fişiere dinamice.

Dezavantaje: numarul mare de elemente examinate în medie la căutare, operatii dificile în cazul fişierelor ordonate.

Utilizare: Dacă în cazul unor fişiere mari organizarea secventiala este practic ineficienta, în cazul unor fişiere de dimensiuni mici se poate utiliza cu succes aceasta organizare.

9.2.2. Fişiere cu dispersie

Fişierele cu dispersie (*hashed files*) grupeaza înregistrările în *clase de înregistrări* fiecare clasă fiind memorată în unul sau mai multe blocuri de memorie. Apartenența unei înregistrări la una din clase este determinată rapid (de obicei printr-un proces de calcul) în functie de valoarea pe care o are cheia înregistrării. Funcția care determină aceasta corespondență se numeste *functie de dispersie*. Fiecare clasă este organizată prin metode de organizare secvențială.

Numărul de clase şi modul de calcul al clasei asociate unei înregistrări se aleg în aşa fel încât să asigure pe de o parte o distributie cât mai uniformă în clase şi pe de altă parte ca numărul mediu de înregistrări într-o clasă să nu fie prea mare pentru a da un timp rezonabil la căutare. O funcție de dispersie des utilizată este cea care interpretează şirul de biți corespunzator chei ca un numar natural iar clasa asociata elementului este data de restul împartirii acestui numar la numarul de clase (eventual adunat cu 1 dacă numerotarea începe de la 1 şi nu de la 0).

În implementarea fişierelor cu dispersie se construieste o listă numita *director* cu pointeri la blocul de începere corespunzator fiecarei clase, blocurile unei aceleiaşi clase fiind înlănțuite ultimul bloc având pointer nul. Directorul este memorat în blocuri ale fişierului şi dacă este de dimensiuni mici este incarcat în memoria principala de cate ori se lucreaza cu fişierul respectiv pentru a micsora numarul de accese la memoria externa.

Operatiile cu fişiere cu dispersie se fac analog cu operațiile cu fişiere secventiale, singura deosebire fiind data de localizarea clasei corespunzatoare înregistrării implicate.

Dacă clasele devin prea mari se pune problema reorganizarii fişierului cu mărirea numarului de clase. O buna alegere a funcției de calcul a clasei permite ca în cazul unei măriri de un numar de ori a numarului claselor, noua funcție de calcul să stabileasca drept noi clase partitii ale vechilor clase. De exemplu dacă se dubleaza numarul de clase şi functia de imprastiere este obtinuta ca restul unei impartiri la numarul de clase, atunci orice clasa i se imparte în doua clase distincte şi anume i şi $n+i$ unde n este numarul initial de clase. Deci în acest caz operatia de reorganizare se poate face clasa cu clasa.

Avantaje: timp relativ redus de acces pentru clase de dimensiuni mici, programe relativ simple de gestionare şi utilizare, posibilitati de organizare în cazul fişierelor cu înregistrări fixate.

Dezavantaje: spaţiu suplimentar pentru organizarea claselor, posibile reorganizari, dificila parcurgerea în ordine a înregistrărilor din tot fişierul.

Utilizare: organizare destul de des utilizata în tehnicile de implementare a bazelor de date mai ales pentru modelele retea şi ierarhic; cu posibilitati bune de operare în cazul fişierelor dinamice.

9.2.3. Fişiere cu index rar

Fişierele cu index rar sau indexat secventiale presupun memorarea înregistrărilor intr-un fişier numit *fişierul principal* în ordinea crescatoare a cheilor şi grupate pe pagini. Se adauga un alt fişier, numit *fişierul index* ce contine pentru fiecare pagina din fişierul principal cate o înregistrare cu valoarea celei mai mari chei din pagina şi adresa de inceput a paginii. Fişierul index este ordonat crescator în raport cu valoarea cheii folosind pentru el o metoda de organizare oarecare. De cele mai multe ori pagina corespunde cu un bloc. Înlănţuirea blocurilor în ordinea crescatoare a cheilor permite parcurgerea secventiala ordonata a fişierului. Din necesitati practice se pot înlănţui şi blocurile corespunzatoare fişierului index.

Căutarea se face cu o căutare în fişierul index prin metode adecvate organizarii lui (căutare liniara, căutare binara sau căutare prin calculul adresei) pentru gasirea unei înregistrări ce contine cea mai mica cheie mai mare sau egala cu cheia cautata. Adresa din acea înregistrare da posibilitatea acesului la pagina care poate contine înregistrarea cautata. Apoi se face o căutare secventiala sau prin alta metoda în acea pagina. Eventual se testeaza bitul de existenta sau bitul de ştergere dacă acestia exista.

Inserarea se face urmand procedeul de la căutare pentru determinarea paginii unde urmeaza sa apara noua înregistrare şi dacă mai este loc în pagina se aplica procedeul de inserare în fişier ordonat, altfel se introduce un bloc nou cu distribuirea înregistrărilor implicate şi modificarea corespunzatoare a fişierului index. Dacă cheia noii înregistrări este mai mare decat cea mai mare cheie existenta în fişier trebuie modificata cheia ultimei pagini din index punand cheia ultimei înregistrări introduse.

Ştergerea unui element se face prin eliminarea elementului din lista ordonata corespunzatoare paginii în care apare acea înregistrare. Dacă blocul nu mai contine nici-o înregistrare se elimina blocul respectiv din fişierul principal cu modificarea corespunzatoare a indexului. Se observa ca în cazul în care se elimina înregistrarea cu cheia cea mai mare din pagina sistemul functioneaza corect şi dacă lasam neschimbat indexul în acest caz.

Modificarea se face prin citire, schimbarea valorilor şi rescriere când nu sunt afectate câmpuri ce apartin cheii sau prin ştergere şi inserare în cazul afectarii unor câmpuri ce apar în cheie.

Pentru fişierele cu înregistrări fixate indexul nu se schimba iar noile înregistrări introduse şi care nu mai incap în blocurile corespunzatoare lor în blocuri noi legate de acestea formand clase ca la fişierele prin dispersie. Dacă clasele devin foarte mari trebuie aplicata o reorganizare. Parcurgerea în ordine a înregistrărilor se poate face dacă se asociaza fiecarei înregistrări un pointer la înregistrarea urmatoare.

Avantaje: spaţiu supimentar pentru reprezentarea fişierului index relativ redus, permite determinarea înregistrărilor cu chei intr-un interval dat, bine adaptabil pentru fişiere dinamice, fişierele index nu sunt cu înregistrări fixate ceea ce permite reorganizari mai simple şi permite aflarea înregistrării cu cheia cea mai mica dintre cele cu chei mai mari decat o valoare data (cheia ce acopera o valoare v data).

Dezavantaje: uneori poate sa consume spaţiu suplimentar mult dacă paginile nu sunt incarcate pana aproape de capacitatea maxima, operatii de depasire a capacitatii unei pagini dificil de manevrat.

Utilizare: Se folosesc în special la organizarea unor fişiere statice sau pentru imbunatatirea timpului de acces la alte fişiere index.

9.2.4. Fişiere cu index dens

Organizarea cu index dens presupune asocierea la un fişier numit *fişierul de baza* a unui alt fişier numit *fişier index* cu acelaşi numar de înregistrări ca fişierul de baza. Înregistrările din fişierul index contin, ca şi în cazul fişierelor cu index rar, perechi formate din cheile înregistrărilor fişierului de baza şi pointeri la aceste înregistrări dar de data aceasta pentru toate înregistrările fişierului de baza. Pentru fişierul index se poate folosi oricare dintre tipurile de organizari prezentate.

Avantaje: înregistrările fişierului index sunt nefixate ceea ce permite o organizare mai eficienta, înregistrările fişierului index fiind mai scurte decat ale fişierului de baza numartul de blocuri ce trebuiesc accesate este mai mic pentru diferitele operatii cu fişierul, la acelaşi fişier de baza pot fi asociate mai multe fişiere index corespunzatoare diferitelor chei. Poate fi utilizat în transformarea unui fişier cu înregistrări fixate intr-un fişier cu înregistrări nefixate.

Dezavantaje: spațiu suplimentar ocupat, necesitatea combinarii cu alte metode, neasigurarea unei bune acoperiri a spațiului alocat.

9.2.5. Fişiere cu structura de B-arbore

Pentru fişierele de dimensiuni foarte mari se poate privi indexul din organizarea indexat secventiala ca un fişier de baza şi pentru el sa se organizeze un fişier index rar. Procedand recursiv pana se obține un fişier index ce ocupa un singur bloc obținem o structura indexata pe mai multe nivele foarte flexibila şi eficienta de arbore echilibrat care are drept frunze blocuri ce contin pointeri la înregistrările fişierului sau eventual chiar înregistrările dacă fişierul nu este cu înregistrări fixate. Numele de *B-arbore* al acestor structuri vine de la denumirea în limba engleza a lor *balanced trees*.

Pentru a asigura o ocupare eficienta a spațiului toate operatiile care se fac în B-arbori respecta conditia de a lasa în toate blocurile cu exceptia eventual a radacinii a unui numar de înregistrări mai mare decat jumatate din capacitatea blocului respectiv. Dacă de exemplu un nod intern poate memora $2d$-1 înregistrări (cheie+adresa bloc) şi o frunza poate sa memoreze $2e$-1 înregistrări atunci nodurile interne cu exceptia eventual a radacinii trebuie sa aiba cel putin d înregistrări şi frunzele trebuie sa aiba cel putin e înregistrări.

Se observa ca intr-un B-arbore ultima cheie a fiecarui bloc nu este necesara presupunandu-se ca orice înregistrăre cu cheia mai mare decat penultima cheie a blocului se afla în blocul dat de ultimul pointer.

Căutarea se face incepand de la radacina şi luând în continuare blocul dat de adresa corespunzatoare celei mai mici valori a unei chei dintre cele mai mari sau egale cu cheia cautata (în căutarea liniara este prima cheie mai mare sau egala cu cheia cautata) apoi aplicând acest procedeu recursiv pana se ajunge la o frunza unde poate fi gasita înregistrarea cautata.

Inserarea se face prin localizarea ca la căutare a blocului unde ar trebui sa se afle noua înregistrare şi prin inserarea acelei înregistrări în blocul gasit dacă mai este loc. Dacă toate înregistrările blocului sunt ocupate se creaza un nou bloc cele doua blocuri impartindu-şi în mod egal înregistrările şi apoi inseranu-se la nivelul imediat superior adresa noului bloc impreuna cu cea mai mare cheie a lui (se presupune ca în blocul nou introdus s-au pus înregistrările cu cele mai mici chei. Dacă se ajunge astfel la radacina arborele creste numarul nivelelor cu unu noua radacina având în acest caz doi fii: un bloc nou introdus şi vechea radacina.

Ştergerea se face prin localizarea înregistrării care se elimina ca la căutare şi se elimina aceasta înregistrare din bloc. Dacă în bloc raman mai mult de jumatate înregistrări

ocupate atuci procesul se termina, altfel blocul respectiv se combina cu un bloc vecin fie redistribuind înregistrările fie, dacă şi acel bloc este la limita inferioara se formeaza din cele doua blocuri unul singur. Combinarea a doua blocuri poate produce modificari sau ştergeri şi la nivelele superioare uneori (foarte rar) putand micsora inaltimea arborelui (cazul unei radacini cu numai doi fii care se combina intr-un singur bloc).

Modificarea se face la fel ca şi pentru celelalte metode în functie de faptul dacă sunt afectate sau nu câmpurile cheie.

Pentru operatiile cu B-arbori se fac un numar de citiri/scrieri de blocuri de ordinul lui (log n - log e)/log d unde n este numarul de înregistrări din fişier, o frunza poate sa contina cel mult $2e$-1 înregistrări şi un nod intermediar poate sa contina cel mult $2d$-1 perechi cu chei şi adrese.

Avantaje: aplicabil cu foarte bune rezultate în cazul fişierelor dinamice datorita numarului mic de blocuri accesate în operatii şi a numarului mic de operatii la reactualizari, permite furnizarea listei elementelor în ordinea data de cheie, se poate aplica oricarui fişier unul sau mai mulți B-arbori care sa lucreze ca fişiere index, o buna ocupare a spațiului alocat (în medie circa 75% spațiu ocupat).

Dezavantaje: dificil de programat operatiile cu B-arbori, folosire spațiu suplimentar pentru nodurile interne.

9.3. Metode de căutare în fişiere

Operația cel mai des utilizată în lucrul cu fişierele este operația de căutare. Unele din problemele privind căutarea au fost discutate mai sus. Vom prezenta şi alte probleme specifice pentru bazele de date.

9.3.1. Fişiere cu index secundar

Nu toate cautarile din bazele de date se fac după cheia principală. Dacă un atribut sau un grup de atribute apar des în cereri atunci pentru acel atribut sau grup de atribute se construieste un index numit *index secundar* ce permite accesul rapid la înregistrările corespunzatoare valorilor date. Un fişier cu un index secundar corespunzator unui atribut sau grup de atribute *F,* se spune ca este fişier inversat în raport cu *F*. În indexul secundar înregistrările sunt indicate fie prin pointeri la ele fie prin valorile cheilor principale corespunzatoare lor.

Referirea la înregistrări prin pointeri are avantajul accesului mai rapid la informatie dar produce constrangeri din cauza fixarii înregistrărilor pe locul pe care au fost introduse sau la nivel de bloc sau la nivel de clasa. Referirea prin cheia principala asociata are dezavantajul unui acces mai lent dar nu mai fixeaza înregistrările.

9.3.2. Indicarea parțială a chei de căutare

Dacă ne intereseaza înregistrările care au valorile $a_1,a_2,...,a_k$ pentru atributele $A_1,A_2,...,A_k$ care nu constituie o cheie, aceasta revine la determinarea intersectiei mulțimilor $S_1,S_2,...,S_k$ unde S_i este mulțimea tuturor înregistrărilor care au valoarea a_i corespunzatoare atributului A_i.

O metodă utilizată în acest caz este metoda *indecşilor secundari multipli*. Din indecşii asociati atributelor $A_1,A_2,...,A_k$ se determina mulțimile de pointeri $P_1,P_2,...,P_k$ ale pointerilor catre înregistrările mulțimilor $S_1,S_2,...,S_k$ şi dacă acestea nu au prea multe elemente se face intersectia lor în memoria principala. O varianta este alegerea unui indice i pentru care mulțimea S_i are cele mai putine elemente (de obicei se ia mulțimea pentru care atributul poate lua cele mai multe valori) şi apoi se verifica pentru aceste elemente dacă indeplinesc şi

celelalte conditii. Dacă pointerii sunt la nivel de bloc, metoda intersectiei mulțimilor de pointeri poate sa produca şi elemente false şi se fac verificari suplimentare.

A doua metoda utilizata în acest caz este o generalizare a fişierelor cu dispersie ce folosesc *functii de dispersie partitionate*. În aceasta metoda la calculul clasei unui element contribuie toate câmpurile înregistrării. Cu functii de dispersie bine construite se poate limita numarul de clase în care se poate afla o înregistrare pentru care cunoastem numai o parte dintre câmpuri. Aceasta se obține impartind bitii unui numar de clasa în mai multe parti componente şi atribuind fiecarui atribut posibilitatea de a determina o anumita parte asociata din adresa.

Sa presupunem de exemplu ca numarul de clase este o putere a lui 2 şi anume 2^B, deci adresa unei clase este un şir de B biti. Vom imparti cei B biti în grupe, cate o grupa pentru fiecare atribut (eventual unele grupe nu au nici-un bit). Dacă atributele sunt $A_1,A_2,...,A_k$ şi atributului A_i îi sunt atribuiti b_i biti, determinam clasa înregistrării $(a_1,a_2,...,a_k)$ calculand $h_i(a_i)$ pentru i=1,...,k unde h_i este o functie de dispersie pentru atributul A_i cu valori între 0 şi 2^{bi-1} iar numarul clasei înregistrării se obține prin concatenarea acestor adrese deci este şirul de B biti $h_1(a_1)h_2(a_2)...h_k(a_k)$.

Pentru căutare se construiesc numere de clase cu valori fixate, obținute aplicând functia de dispersie unei valori cunoscute pentru atributele pentru care se cunosc valorile şi luand toate posibilitatile pentru grupele asociate atributelor pentru care nu se cunosc valorile.

Dacă se cunosc probabilitatile cu care atributele apar intr-o cerere atunci se pot stabili proprietati interesante după cum urmează.

Teorema 9.1. Dacă valorile unui atribut sunt egal probabile când se specifica o valoare pentru acest atribut, atunci se obține în medie un numar minim de clase de examinat pentru a obține raspunsul la o cerere dacă, pentru anumite numere $n_1,n_2,...,n_k$ al caror produs este numarul de clase, adresa clasei asociate înregistrării $(a_1,a_2,...,a_k)$ se exprima cu formula

$$h_k(a_k)+n_k(h_{k-1}(a_{k-1})+n_{k-1}(h_{k-2}(a_{k-2})+...+n_2h_1(a_1)...))$$

unde functia de dispersie $h_i(a_i)$ ia valori între 0 şi n_i.

Din aceasta teorema se deduce ca alegand fiecare n_i ca o putere a lui 2 putem sa obținem o aproximatie a unei solutii optime. Alegerea puterilor lui 2 este o alta problema care a fost rezolvata numai în cazuri particulare. Dacă în cereri se considera valoarea unui singur atribut atunci valorile $b_1,b_2,...,b_k$ se aleg după cum urmează din teorema:

Teorema 9.2. Dacă toate cererile specifica doar cate un atribut şi p_i este probabilitatea ca A_i sa fie atributul specificat, atunci, presupunand ca nici-un b_i nu este mai mic decat 0 sau mai mare ca B, numarul mediu de clase cercetate este minim dacă

$$b_i=(B - (\log p_1 + \log p_2 +...+ \log p_k))/k + \log p_i$$

unde p este numarul de atribute, 2^B este numarul de clase şi logaritmii sunt în baza 2.

Demonstratia acestei teoreme se face utilizand metoda multiplicatorilor lui Lagrange şi poate fi gasita în Ullman. Formula din teorema permite aflarea valorilor elementelor b_i aplicând urmatoarele reguli:

- dacă un b_i este mai mare decat B se face acel b_i egal cu B şi se fac 0 celelalte valori;

- dacă o valoare b_i este negativa se elimina atributul corespunzator din calcule şi se reaplica teorema;

- se trunchiaza valorile b_i (luandu-se partea întreaga a lor) şi eventualele unitati ramase se adauga la valorile acelor bi care au partea zecimala cea mai mare.

Exemplul 9.1. Sa consideram un fişier corespunzator entitatii STUDENT având atributele MATRICOLA, NUME şi DATA_NASTERE care sunt cerute cu probabilitatile 0.24, 0.75 şi respectiv 0.01. Presupunem ca formam 512 clase, deci B=9 şi aplicând formula

din teorema obținem b_i=6.03 + log p_i, de unde rezulta b_1=3.98, b_2=5.62 şi b_3=-0.61. Deoarece b_3 este negativ se ia b_3=0 şi se refac calculele numai pentru primele doua câmpuri cu probabilitatile p_1=0.24/(1-0.01)=0.242 şi respectiv p_2=0.75/(1-0.01)=0.758 de unde se obține b_i=5.72+log p_i rezultand b_1=3.68 şi b_2=5.32 apoi prin trunchiere la 3 şi respectiv 5 se obține disponibil o unitate care se adauga lui b_1 care are partea fractionara cea mai mare obtinand în final b_1=4, b_2=5 şi b_3=0.

Un alt caz în care se pot specifica valorile optime pentru b_i este cel în care se cunosc probabilitatile cu care câmpurile sunt mentionate în cereri aceste probabilitati fiind independente de mentionarea altor câmpuri în aceeaşi cerere. Formulele de calcul pentru b_i sunt date de teorema urmatoare:

Teorema 9.3. Dacă p_i este probabilitatea cu care se specifica valoarea atributului A_i intr-o cerere independent de celelalte valori specificate, atunci, presupunand ca b_i nu sunt negativi sau mai mari decat B, numarul mediu de clase de cautat este minim dacă se ia

$$b_i=(B - (\log q_1 + \log q_2 +...+ \log q_k))/k + \log q_i$$

unde $q_i=p_i/(1-p_i)$, i=1,2,...,k, restul conditiilor fiind ca la teorema 9.2.

Algoritmul de calcul al valorilor b_i, i=1,...,k este la fel ca în cazul precedent doar ca în loc de p_i se ia $p_i/(1-p_i)$ în calcule.

Exemplul 9.2. Sa consideram fişierul corespunzator entitatii COMENZI care are atributele NUME, MARFA, CANTITATE şi DATA specificate cu probabilitatile p_1=0.8, p_2=0.5, p_3=0.01 şi p_4=0.2 şi numarul de clase este 512, deci B=9. Facând calculele se obține b_1=5.91, b_2=3.91, b_3=-2.73 şi b_4=1.91. Cum b_3 este negativ se elimina atributul CANTITATE din calcule şi se face b_3=0. Refacând calculele se obține b_1=5, b_2=3, b_3=0 şi b_4=1 şi nu mai sunt necesare reajustari. Numarul de clase cercetate la o cerere este în acest caz de 58.3 în medie.

9.3.3. Cazuri speciale de căutare

În cereri pot sa intervina şi alte tipuri de conditii pentru determinarea unei înregistrări sau unei mulțimi de înregistrări decat cele prezentate pana acum. Un astfel de caz il constituie indicarea limitelor între care trebuie sa se afle valorile corespunzatoare unor câmpuri pentru a selectiona înregistrările. Acest tip de cereri le vom numi cereri după marime.

Pentru cererile după marime se pot aplica metodele anterioare cu o alegere corespunzatoare o tehnicilor de lucru. De exemplu se poate adapta tehnica de dispersie partitionata alegand functiile de dispersie în asa fel incat elementele din clase diferite sa fie în aceeaşi relatie de ordine ca şi adresele lor. Un exemplu de astfel de functie este urmatorul. Dacă numarul de clase este M (deci adresele claselor sunt de la 0 la M-1) şi valorile unui câmp sunt relativ uniform distribuite pe intervalul [a,b) se ia drept clasa pentru valoarea v numarul [(v-a)/(b-a)*M] unde prin [] am notat partea întreaga a numarului respectiv. Pentru distributii neuniforme sau nespecificarea intervalului de variatie a valorilor câmpurilor se pot defini alte functii de dispersie mai bune (eventual tabelate).

O alternativa de rezolvare este şi folosirea B-arborilor construind cate un B-arbore pentru fiecare atribut în parte, selectionand din ei pointeri catre înregistrările care au valori cuprinse între limitele precizate şi apoi facând intersectia acestor mulțimi pentru a identifica toate înregistrările care indeplinesc toate conditiile specificate.

9.4 Metode de memorare pentru înregistrări cu lungime variabila

În practica apar situatii când o înregistrarte nu are o lungime fixa. Aceasta se intampla mai ales în cazul când un câmp sau o combinatie de câmpuri se repeta de mai multe ori. De exemplu, în cazul unei relatii de forma unu-la-mai-mulți de la entitatea E1 la entitatea E2

poate fi implementata logic în felul urmator: fiecarui element al lui E1 îi corespunde o înregistrare a unui fişier care include toate informatiile elementelor entitatii E2 care corespund în relatia data elementului din entitatea E1.

La nivel fizic, fişierele cu înregistrări logice de lungime variabila sunt reprezentate tot prin fişiere cu înregistrări de lungime fixa. De obicei transformarea înregistrărilor logice de lungime variabila se face prin una din urmatoarele trei metode: metoda spațiului rezervat, metoda înlănțuirii sau metoda mixta.

Operatiile cu fişiere cu înregistrări de lungime variabila se fac la fel ca la celelalte fişiere tinand seama de modul de organizare a lor. În acest caz intervin operatii suplimentare ce privesc unele repetari ale grupului repetitiv. Aceste operatii sunt tratate ca operatii pe fişiere obisnuite interpretand repetarile unui grup repetitiv ca un mic fişier.

9.4.1. Metoda spațiului rezervat

În metoda spațiului rezervat se rezervau pentru numarul maxim de ocurente posibile pentru atributul sau grupul de atribute care se repeta. Se poate decide care elemente sunt ocupate şi care nu în ocurentele definite fie prin intermediul unui nou câmp care contine numarul de ocurente ce apar efectiv fie punand o valoare null în locurile neocupate. Aceasta metoda se poate aplica în cazul în care numarul maxim de repetari ale grupului de atribute este destul de apropiat de numarul mediu de repetari, pentru utilizarea eficienta a spațiului de memorie alocat, dar în acelaşi timp este destul de mic pentru a evita scrierea de mai multe ori a unor instructiuni din cauza numelor diferite date câmpurilor.

9.4.2. Metoda înlănțuirii

În metoda înlănțuirii grupurile care se repeta sunt memorate intr-un fişier separat, în fişierul initial se pune legatura la primul bloc din al doilea fişier ce contine ocurente corespunzatoare acelei înregistrări (în cazul când nu exista astfel de ocurente se pune valoarea null). Dacă repetarile ocurentelor depasesc capacitatea unui bloc, se folosesc blocuri suplimentare ce formeaza o lista cu legaturi. Aceasta metoda este utilizata mai ales în cazul în care numarul de ocurente este foarte mare sau numarul de repetari variaza foarte mult pentru înregistrările logice.

9.4.3. Metoda mixtă

Cea de-a treia metoda este o combinatie a precedentelor doua metode şi anume se rezerva loc în înregistrarea initiala pentru un numar nic de repetari ale grupului repetitiv şi un pointer la primul bloc al lantului de blocuri (al altui fişier) ce contine celelalte repetari ce nu au putut fi puse aici. Acesta strategie se aplica mai ales în cazul când cele mai multe repetari sunt apropiate de numarul mediu de repetari în care caz numarul de rezervari de locuri este putin mai mare decat numarul mediu de rezervari urmand sa se apeleze la al doilea fişier numai în cazurile exceptie când numarul de repetari ale grupului repetitiv este mai mare decat numarul spațiilor rezervate.

CURS 10

BAZE DE DATE

INTEGRITATEA ŞI SECURITATEA BAZELOR DE DATE

10.1 Aspecte privind integritatea datelor
10.1.1 Integritatea semantica a datelor
10.1.2 Controlul accesului concurent la bazele de date
10.1.3 Salvarea si restaurarea bazei de date
10.2 Securitatea bazei de date

10.1 Aspecte privind integritatea datelor

Protecția bazelor de date constă într-un set de măsuri umane şi facilități oferite de SGBD prin care se urmăreşte asigurarea integrității datelor, care se concretizează prin corectitudinea datelor introduse şi manipulate, şi a securității datelor, ce vizează interzicerea accesului la date pentru persoanele ce nu au competențe în folosirea lor. Aceasta capătă o importanță deosebită în contextul extinderii folosirii configurațiilor cu număr mare de utilizatori şi cu un volum mare de date de prelucrat.

În ceea ce priveşte protecția datelor, pot fi puse în evidență două tendințe: protecția împotriva unor defecte sau erori accidentale şi protecția completă care realizează în plus față de prima şi protecția contra unor acțiuni voite. Teoretic, toate sistemele ar trebui să asigure protecția completă a datelor. În practică însă, costul protecției, care creşte pe măsură ce sunt reduse posibilitățile de apariție a unor erori şi de violare a confidențialității datelor, este cel care dictează complexitatea metodelor de protecție care vor fi utilizate.

Aspectele protecției bazelor de date ce vor fi prezentate în continuare se bazează pe presupunerea că protecția informației la nivelul sistemului de operare este asigurată.

Corespunzător situațiilor care pot genera apariția unor date incorecte în baza de date, se disting trei aspecte ale asigurării integrității datelor:
1. *Asigurarea integrității semantice a datelor*- presupune prevenirea introducerii unor date incorecte şi a efectuării unor prelucrări greşite. Dacă acest lucru nu va fi împiedicat sau semnalat imediat, datele vor fi utilizate în alte prelucrări, declanşându-se astfel un proces necontrolat de alterare a bazei de date.
Cu cât sesizarea unei erori are loc după o perioadă mai mare, cu atât efectele ei vor fi mai greu sau chiar imposibil de înlăturat.
2.*Controlul accesului concurent la date* - presupune prevenire unor rezultate incorecte din execuția concurentă a unor prelucrări multiutilizator. De exemplu, în cazul a două prelucrări concurente cai în calcularea salariului mediu lunar al angajaților unei firme şi respectiv unui procent de creştere de 10% salariilor angajaților, există riscul ca salariului mediu să intervină valori actualizate şi valori vechi ale salariu, rezultatul fiind evident fără semnificație.
3.*Salvarea şi restaurarea bazei de date* - presupun refacerea baza afectată de funcționarea anormală sau căderea SGBD sau SO sau ca urmare a unor defecte hardware.

10.1.1 Integritatea semantică a datelor

Introducerea unor date incorecte sau prelucrările ce furnizează rezultate greşite trebuie prevenite prin includerea în programele de aplicație a unor secvențe de testare a datelor şi prin utilizarea unor facilități de asigurare a integrității semantice a datelor oferite de SGBD. Concret, orice operație asupra datelor trebuie constrânsă să respecte anumite reguli, reguli care reprezintă restricții de integritate.

Restricțiile de integritate se pot clasifica, după modul în care sunt exprimate în:
- restricții de integritate structurale;

- restricţii de integritate de comportament.

Restricţiile de integritate structurale decurg din caracteristicile modelului utilizat pentru reprezentarea datelor şi din elementele furnizate la structurii bazei de date. Astfel, la introducerea datelor nu vor fi acceptate valorile care nu aparţin tipului de dată specificat pentru câmpurile respective din înregistrare. Dacă există conceptul de cheie unică, la inserare se va verifica unicitatea cheii. Structurile de date pot impune de asemenea rest tipurile de înregistrări. De exemplu, anumite sisteme impun o relaţie de forma 1:N între părinte şi segmente dependente (dacă segmentul rădăcină conţine date despre profesori şi segmentele dependente date despre cursuri, sistemul impune automat restricţia ca fiecare curs să aibă un singur profesor).

În modelul relaţional, există două restricţii de integritate asociate cheilor primare şi celor externe şi anume:

1. Integritatea entităţii (entity integrity) - conform acesteia, nici un atribut care participă la formarea cheii primare a unei relaţii nu poate primi o valoare NULL (valoare diferită de blanc sau 0, considerată în lipsa unei valori explicite pentru câmpul respectiv).

Aceasta este o consecinţă a faptului că o cheie primară trebuie să identifice în mod unic tuplurile unei relaţii.

2. Integritatea referenţială (referenţial integrity) - statuează că orice valoare a unei chei externe din relaţia care referă trebuie să aibă corespondentă o cheie primară cu aceeaşi valoare în relaţia referită sau să fie NULL.

Restricţiile de integritate de comportament pot fi incluse în programele de aplicaţie şi verificate în momentul execuţiei acestora sau pot fi memorate în dicţionarul datelor şi verificate automat de SGBD la fiecare operaţie vizată asupra anumitor date.

Această soluţie, foarte frecventă în practică, prezintă următoarele dezavantaje:

- efortul de programare va fi mai mare şi dimensiunea programelor va creşte prin includerea părţilor de cod pentru testarea restricţiilor de integritate.
- restricţiile de integritate asociate anumitor date trebuie testate de fiecare program care lucrează cu datele respective; o aceeaşi parte de cod va trebui repetată în toate aceste programe.
- programele de aplicaţie nu pot controla operaţiile efectuate de utilizatori în limbajele de interogare de nivel înalt, existând deci în continuare pericolul afectării integrităţii datelor.
- dacă restricţiile de integritate se schimbă, efortul pentru modificarea tuturor programelor implicate va fi foarte mare.

Toate aceste dezavantaje dispar în cazul specificării restricţiilor de integritate la nivelul SGBD. Acestea pot fi exprimate cu ajutorul limbajului de definire a datelor sau al limbajului de manipulare a datelor, în funcţie de particularităţile fiecărui SGBD.

În diferite lucrări de specialitate, o regulă de integritate este reprezentată printr-un tuplu forma:

(o, t, c, p, pa)

unde:

o - reprezintă obiectul asupra căruia se aplică restricţia de integritate;

t - indică tipul operaţiei pentru care restricţia va fi invocată (INSERT, UPDATE, DELETE);

c - este o condiţie care trebuie să fie adevărată pentru ca restricţia de integritate să se aplice obiectului o (de exemplu, dacă restricţia se referă doar la studenţii din facultatea cu codul MAT, condiţia va fi de de forma COD_S = 'MAT');

p - restricţia de integritate, care trebuie să fie adevărată pentru o realizare a obiectului o, pentru ca operaţia cerută să poată fi efectuată;

pa - o procedură auxiliară care specifică ce trebuie să facă sistemul dacă p nu este adevărată (poate fi utilizată pentru efectuarea de verificări de integritate foarte complexe,

pentru realizarea unei jurnalizări selective ca şi pentru întreținerea automată a bazei de date).

De exemplu, regula pentru o restricție de integritate care prevede la inserarea unui tuplu într-o relație cu date despre studenții unei facultăți, incrementarea numărului de studenți este:

o - relația STUDENT
t - INSERT
c - true
p - false
pa - NR_STUDENȚI = NR_STUDENȚI + 1

În practică, complexitatea relațiilor de integritate exprimare şi de implementare diferă de la un SGBD la altul.

10.1.2 Controlul accesului concurent la bazele de date

Sistemele monoutilizator răspund unui număr redus de probleme practice care implică utilizarea bazelor de date. Cea mai mare parte a aplicațiilor trebuie concepute pentru a putea funcționa în regim de lucru multiutilizator şi implementate pe sisteme care prezintă această caracteristică. Pe lângă aspectele prezentate privind asigurarea integrității datelor, în acest caz apare şi necesitatea controlului accesului concurent la date.

În sistemele multiutilizator, sistemul de operare asigură accesul concurent al programelor în execuție la resurse după o anumită disciplină internă. În cazul aplicațiilor care utilizează o aceeaşi bază de date, întreruperea executării unui proces pentru începerea sau continuarea altora poate conduce la alterarea datelor. Asigurarea integrității datelor în acest context presupune existența unor facilități speciale pentru controlul accesului concurent la date la nivelul SGBD. Pentru prezentarea acestora este necesară explicarea conceptului de tranzacție.

Tranzacția este o secvență de operații care din punctul de vedere al SGBD constituie o unitate de prelucrare, aceasta însemnând că se va accepta fie executarea completă fie, în situația în care acest lucru nu este posibil, nu va trebui reținută nici o modificare făcută de ea asupra bazei de date, SGBD efectuând (automat sau la comandă) derularea înapoi a tranzacției.

O tranzacție este caracterizată de punctul său de început şi de punctul de sfârşit. După modul în care acestea sunt definite, tranzacțiile se pot clasifica în:

• Tranzacții implicite - sunt acelea pentru care punctele de început şi sfârşit sunt automat definite. Pentru SQL-Server de exemplu, toate comenzile de modificare a datelor (INSERT, UPDATE şi DELETE) sunt tratate ca tranzacții implicite. În consecință, nici una dintre aceste comenzi nu va putea lăsa baza de într-o stare inconsistentă. Acest lucru este ilustrat prin exemplul 10.1.

Exemplul 10.1 Comanda de inserare

INSERT student VALUES (101, 'Popescu', 221, 12-03-1989, 'Bucuresti') va avea efect asupra bazei de date doar dacă va putea fi executată integral. Dacă însă sistemul cade înaintea terminării execuției acesteia, baza de date nu va fi lăsată în starea în care doar unele dintre valorile coloanelor există, sau în care un index nu este actualizat.

•Tranzacții explicite - presupun folosirea unor comenzi speciale pentru stabilirea punctelor de început şi sfârşit ale tranzacției. În SQL - Server, tranzacțiile explicite permit utilizatorului să grupeze un set de comenzi SQL într-o tranzacție folosind comenzile BEGIN TRANSACTION şi COMMIT TRANSACTION pentru precizarea punctelor de început şi sfârşit. De asemenea, utilizatorul poate defini puncte de salvare (utile în cazul tranzacțiilor foarte mari) folosind comanda SAVE TRANSACTION sau să deruleze înapoi tranzacția până

la punctul de început sau până la puncte de salvare anterioare, folosind comanda ROLLBACK TRANSACTION.

Exemplul 10.2 O tranzacție explicită poate fi reprezentată de înregistrarea unui transfer între două conturi, fapt ce presupune debitarea contului X şi creditarea contului Y. Aceste transformări vor fi grupate între comenzile BEGIN TRANSACTION şi COMMIT TRANSACTION. Tranzacția fiind astfel definită, SGBD va garanta tratarea ei ca o secvență unitară de prelucrări.

```
BEGIN TRANSACTION
Debit -100 din CONTUL X
Credit +100 în CONTUL Y
COMMIT TRANSACTION
```

În anumite condiții, utilizatorul poate comanda derularea înapoi a tranzacției. Un exemplu îl constituie o tranzacție care realizează creşterea selectivă a valorilor dintr-o coloană a unei tabele, urmărindu-se menținerea unei limite pentru media acestor valori. În situația în care această limită nu va fi respectată în urma modificărilor făcute, acestea vor fi anulate

Exemplul 10.3

```
BEGIN TRANSACTION media
IF (SELECT AVG (preț) FROM tabel) < 2000
        BEGIN
        UPDATE tabel
        SET pret=pret*2 WHERE pret<1500
        UPDATE tabel
        SET pret=pret*1.2 WHERE pret<2000
END
IF(SELECT AVG (preț) FROM tabel) > 2000
BEGIN
          PRINT "Aceste modificări de preturi nu se realizează"
          ROLLBACK TRANSACTION
          END
ELSE
         BEGIN
         PRINT  "Media preturilor  este  <   2000"
         COMMIT TRANSACTION media
END
```

Pin controlul accesului concurent la date se urmăreşte păstrarea integrității de date în condițiile execuției concurente a tranzacțiilor.

Efecte ale execuției concurente necontrolate a tranzacțiilor

Efectele generate de execuția concurentă necontrolată a tranzacțiilor vor fi ilustrate prin următoarele exemple.

Considerăm, o tranzacție de actualizare a disponibilului în cont la bancă (DISP) al clientului C, de 5000 RON inițial. Aceasta presupune efectuarea a două operarțiuni asupra înregistrării corespunzătoare din baza de date:

- citirea conținutului câmpului respectiv într-o zonă de lucru din memorie
- modificarea şi scrierea noii valori în baza de date.

Dacă sunt inițiate două tranzacții concurente:

T1- înregistrarea retragerii sumei de 500RON din cont

T2 - înregistrarea depunerii sumei de 1000RON în cont

din motive de performanţă, acţiunile acestor tranzacţii pot fi executate intercalat ca în fig. 10.1.

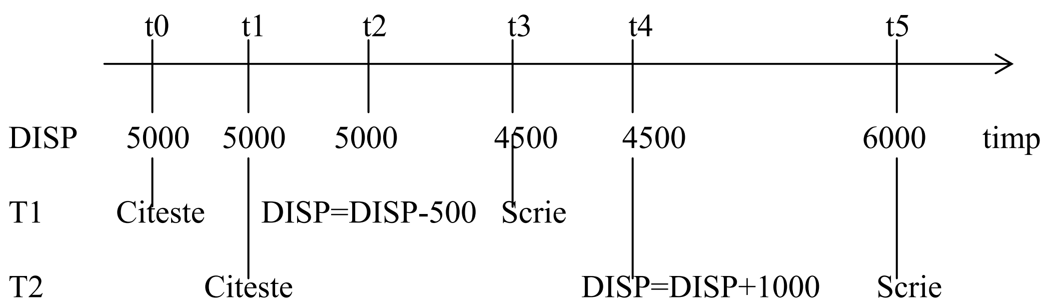


Fig.10.1. Execuţia intercalată a unor tranzacţii

În final, baza de date va conţine pentru clientul C în câmpul DISP valoarea 6000 în loc de 5500 cât ar fi corect. Se observă deci că actualizarea realizată de T1 s-a pierdut ca efect al intercalării acţiunilor tranzacţiilor T1 şi T2. Un alt exemplu de inconsistenţă a datelor generată de lipsa controlului accesului concurent la date este ilustrat de fig. 10.2.

Tranzacţiile TI şi T2 sunt executate serial însă, din anumite motive, transformările efectuate de T1 trebuie anulate. În această situaţie, valoarea utilizată în continuare de T2 nu va fi corectă. Tranzacţiei T2 ar fi trebuit să i se interzică citirea valorii lui x pţnă în momentul în care T1 ar fi fost încheiată.

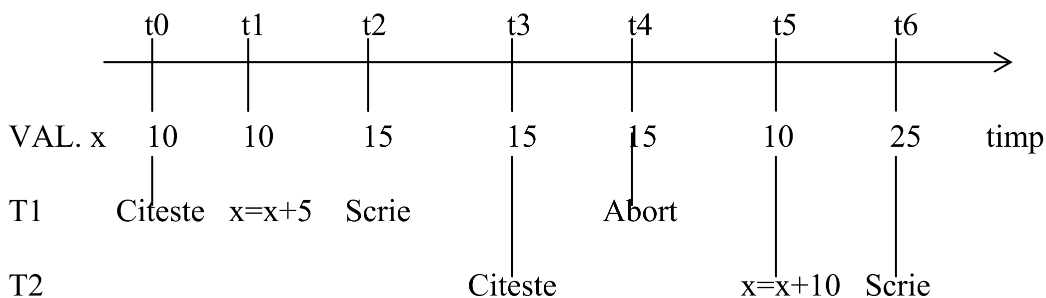


Fig.10.2. Execuţia serială a unor tranzacţii

Asemenea probleme se pot evita dacă S.G.B.D. oferă posibilitatea blocării datelor utilizate la un moment dat de o tranzacţie, înţelegând prin aceasta interzicerea accesului celorlalte tranzacţii concurente la aceste date.

Tehnica blocării

Aspectele prezentate sunt deci o consecinţă a execuţiei concurente necontrolate a tranzacţiilor. Execuţia serială a tranzacţiilor (una după alta), care ar asigura consistenţa bazei de date, nu este posibilă în regim multiutilizator. Ea stă însă la baza criteriului formal de apreciere a efectelor execuţiei concurente a tranzacţiilor. Conform acestuia, o execuţie neserială a unor tranzacţii concurente va fi considerată corectă dacă este serializabilă, adică dacă produce acelaşi rezultat ca o execuţie serială a acelor tranzacţii. Dacă SGBD va asigura doar execuţii serializabile ale tranzacţiilor, atunci consistenţa datelor va fi garantată.

Tranzacţiile considerate trebuie să fie independente, astfel ca orice execuţie serială a lor să furnizeze acelaşi rezultat. În caz contrar, utilizatorul trebuie să se asigure că ele sunt date sistemului în secvenţa dorită.

Tehnica utilizată de SGBD pentru a asigura execuția serializabilă a tranzacțiilor tehnica blocării. În cea mai simplă formă, blocarea unor date de către o tranzacție interzice celorlalte tranzacții accesul la aceste date. Blocarea se poate aplica la nivelul întregii baze de date, al unui fişier, grup de înregistrări sau înregistrare sau chiar câmp. Generic, în continuare vom vorbi despre resursa sau obiectul blocat.

Soluția de blocare prezentată este prea restrictivă, ea fiind însă folosită de anumite SGBD. În exemplele prezentate, observăm că trebuie urmărite două aspecte:

- accesul la datele pe care un utilizator intenționează să le actualizeze să fie interzis celorlalți utilizatori până la completarea procesului de actualizare;
- accesul la datele pe care un utilizator le citeşte fără a le actualiza să fie interzis utilizatorilor pentru operații de actualizare şi permis pentru operații de citire.

Deci tipul de blocare trebuie diferențiat în funcție de operația care va fi executată asupra datelor respective astfel:

- blocare pentru citire (partajabilă) - datele vor putea fi folosite şi de către ceilalți utilizatori însă numai pentru operații de citire;
- blocare pentru scriere (exclusivă) - datele nu vor putea fi accesate de nici un utilizator.

Efectiv, blocarea se realizează prin emiterea de către o tranzacție a unei cereri (explicite sau implicite) de blocare pentru SGBD. Cererea explicită poate fi exprimată în una din următoarele forme:

- în cadrul unei comenzi de manipulare a datelor:

```
UPDATE  ANGAJAȚI   WITH   EXCLUSIVE   LOCK
SET   SALARIU   =   SALARIU   *1.1
```

- printr-o comandă de început a unei tranzacții:

```
BEGIN   TRANSACTION  WITH   EXCLUSIVE   LOCK
BEGIN  TRANSACTION WITH   SHARED  LOCK
```

- printr-o comandă de deschidere a unui fişier :

```
OPEN  FIŞIER  FOR  EXCLUSIVE  UPDATE
OPEN   FIŞIER  FOR   SHARED   READ
```

Dacă cererea este admisă, tranzacția va continua. Altfel, cererea va fi pusă într-olistă de aşteptare până ce resursa vizată va fi eliberată.

Situațiile în care o cerere pentru o anumită resursă poate fi admisă, în condițiile în care există deja o blocare pentru resursa respectivă, sunt ilustrate de matricea compatibilității blocărilor partajabile şi exclusive din fig. 10.3.

T2 / T1 Resursa X	Blocare partajabilă	Blocare exclusivă
Blocare partajabilă	DA	NU
Blocare exclusivă	NU	NU

Fig. 10.3. Matricea compatibilităților

Se observă deci că o blocare pentru citire poate fi acordată altor tranzacții chiar dacă o tranzacție are deja o blocare partajabilă pentru resursa respectivă; blocări exclusive nu pot fi însă acordate până când toate blocările partajabile nu sunt eliberate. Dacă însă o tranzacție a obținut o blocare exclusivă pentru o resursă, nici o altă blocare (partajabilă sau exclusivă) nu va mai putea fi garantată celorlalte tranzacții solicitante.

Accesul concurent la date pe de o parte şi complexitatea mecanismului de blocare al unui SGBD pe de altă parte, sunt influențate de *granularitatea* blocarii. Considerând cele două situații extreme (baza de date şi câmpul), putem sintetiza următoarele aspecte:

- blocarea întregii baze de date la o anumită operație a utilizatorului îi pune pe ceilalți utilizatori în imposibilitatea de a accesa concurent datele, în timp ce gestiunea informațiilor de blocare se simplifică foarte mult;

- blocarea unui singur câmp al unei înregistrări lasă posibilitatea de acces concurent celorlalți utilizatori chiar la câmpuri ale aceleiaşi înregistrări, însă face ca gestionarea informațiilor de blocare să fie foarte complexă.

Cele mai multe SGBD oferă posibilitatea blocării la nivel de înregistrare, la nivelul unui grup de înregistrări pentru care s-a menționat un criteriu de selecție şi la nivel de fişier. Pentru administrarea blocărilor se poate recurge la unul din următoarele moduri:

- setarea unui bit pentru resursa respectivă, indicându-se astfel că aceasta este blocată;

- menținerea unei liste a resurselor blocate ce va trebui consultată ori de câte ori intervine o nouă cerere de blocare;

- menținerea resurselor blocate într-o zonă specială a memoriei.

Interblocarea resurselor

Interblocarea (deadlock) este un impas care rezultă când două tranzacții blochează anumite resurse, apoi solicită fiecare resursele blocate de cealaltă. Această situație este ilustrată în fig. 10.4.

Tranzacția A va aştepta la infinit eliberarea resursei Y, blocată de tranzacția B la pasul 2 şi, la rândul ei, tranzacția B va intra într-o stare de aşteptare infinită pentru resursa X, blocată de A la pasul 1.

La nivelul SGBD trebuie să existe facilități de prevenire sau rezolvare a acestor situații. Astfel, poate fi implementată una din următoarele două strategii:

1.Prevenirea interblocării – presupune ca programele să blocheze toate resursele de care au nevoie încă de la începutul fiecărei tranzacții. În exemplul considerat, tranzacția A ar tebui să blocheze ambele înregistrări încă de la început. Dacă una din resurse ar fi fost deja blocată, ar fi fost necesar să se aştepte până la eliberarea ei.

Această strategie este dificil de implementat şi adesea nu este aplicată, deoarece în cele mai multe cazuri, este imposibil de precizat înainte ce resurse vor fi necesare pentru o tranzacție.

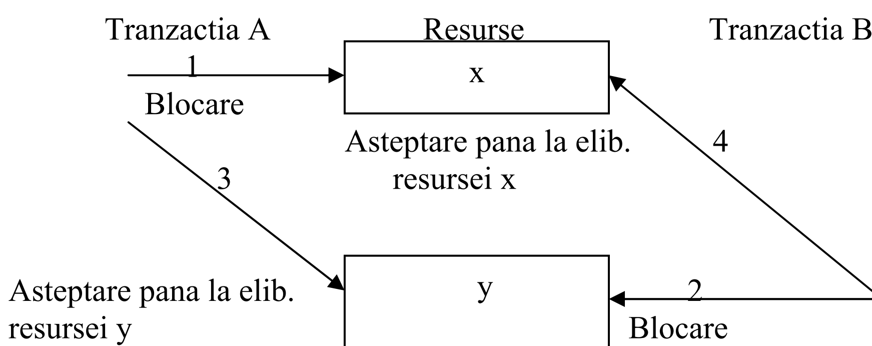
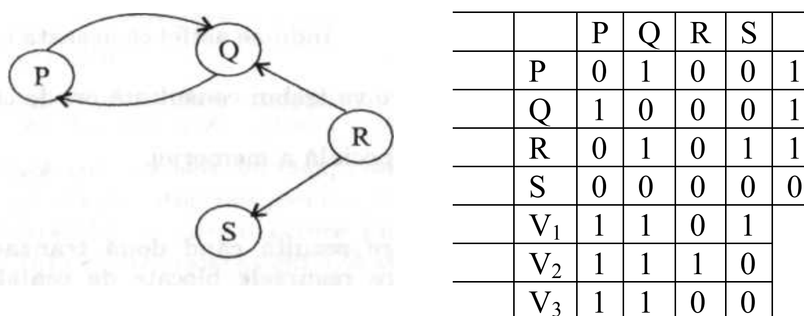


Fig. 10.4 Exemplu de interblocare

2. Soluționarea interblocării - această strategie permite apariția interblocării dar presupune existența unor mecanisme pentru detectarea şi eliminarea acesteia. Intern, sistemul poate utiliza pentru aceasta un graf al precedențelor care reflectă dependențele dintre procese din punctul de vedere al ordinii în care acestea trebuie executate. Într-un astfel de graf, fiecare nod reprezintă un proces activ. Arcele se trasează conform următoarei reguli: dacă procesul P deține resursa A şi procesul Q o solicită, atunci în graf va fi trasat arcul $Q \rightarrow P$, care arată că

execuția procesului Q este condiționată de cea a procesului P. Existența unui interblocaj va fi semnalat prin prezența unui ciclu în graf. Fig. 10.5. prezintă un astfel de graf şi matricea asociată lui.



		P	Q	R	S	
	P	0	1	0	0	1
	Q	1	0	0	0	1
	R	0	1	0	1	1
	S	0	0	0	0	0
	V_1	1	1	0	1	
	V_2	1	1	1	0	
	V_3	1	1	0	0	

Fig.10.5 Graf şi matricea asociată lui

În matrice, cifra 1 arată că procesul de pe linia respectivă se află în aşteptare din cauza procesului de pe coloana corespunzătoare. Algoritmul pentru determinarea interblocajului este următorul:
1. Se efectuează operația logică "SAU" între vectorii asociați liniilor matricei. Rezultatul va fi vectorul V_1.
2. Se efectuează operația logică "SAU" între vectorii asociați coloanelor matricei. Rezultatul va fi vectorul V_2;
3. Se efectuează operația "SI" între V_1 şi V_2. Rezultatul va fi vectorul V_3. Procesele S şi R pentru care s-a obținut rezultatul 0 (vezi figura) vor fi eliminate din matrice. Ele nu pot fi implicate într-un blocaj întrucât S nu aşteaptă după nici un alt proces (0 pe poziția corespunzătoare din V_1) şi după R nu aşteaptă nici un alt proces (0 pe poziția corespunzătoare din V_2);
4. Se elimină liniile şi coloanele corespunzătoare proceselor R şi S şi se reia algoritmul de la pasul 1, până când nici un proces nu va mai fi eliminat. Procesele rămase în matrice vor fi cele între care a intervenit un interblocaj.

În această situație, unul dintre procesele implicate va fi întrerupt, efectele lui de până acum asupra bazei de date vor fi anulate, fiind posibilă apoi continuarea celuilalt. Procesul ce va fi eliminat poate fi:
- procesul cu cele mai puține resurse blocate;
- procesul care nu a făcut încă actualizări asupra bazei de date ;
- procesul cu cea mai mică prioritate;
- procesul cel mai recent început;
- procesul cel mai vechi;
- procesul care a cerut ultimul utilizarea resursei. Procesul abortat urmează să fie reluat ulterior.

10.1.3 Salvarea si restaurarea bazei de date

Salvarea şi restaurarea bazei de date au ca scop readucerea datelor la o formă consistentă în urma unor evenimente care au alterat corectitudinea lor precum:

- funcționarea anormală sau o cădere a SGBD sau SO;
- o defecțiune a suportului fizic pe care este memorată baza de date.

În primul caz, prin întreruperea tranzacțiilor active în momentul respectiv, baza de date va rămâne într-o stare de inconsistență. în cel de-al doilea caz, suportul pe care rezidă baza de date va fi inutilizabil, deci toate datele se vor pierde.

În aceste situații trebuie să fie posibilă refacerea unei stări anterioare consistente a bazei de date. O stare consistentă a bazei de date este starea în care sunt reflectate rezultatele finale ale execuției unor tranzacții, nici o tranzacție nu este în curs de execuție şi sunt satisfăcute restricțiile de integritate semantică.

Îndeplinirea cerinței formulate anterior presupune existența şi utilizarea unor facilități la nivelul SGBD. Acțiunile întreprinse în acest sens se înscriu în procesul de restaurare al bazei de date.

Tehnici de bază folosite în procesul de restaurare

O cădere a sistemului lasă baza de date într-o stare de inconsistență care poate consitui un punct de plecare în procesul de restaurare, în timp ce, în cazul unei defecțiuni a suportului pe care este memorată baza de data, restaurarea va fi posibilă doar dacă există o copie anterioară a bazei de date. Pe aceste baze deci, procesul de restaurare va trebui să asigure o stara consistentă a bazei de date, minimizând totodată volumul prelucrărilor pierdute.

Pentru baza de date inconsistentă, aceasta înseamnă eliminarea efectelor tranzacțiilor active în momentul căderii sistemului şi reflectarea în baza de date a rezultatelor tranzacțiilor terminate, dar care din motive ce vor fi prezentate ulterior nu apar în baza de date. Pentru o copie anterioară a bazei de date, va trebui să existe posibilitatea ca într-un timp cât mai scurt aceasta să fie adusă la o stare consistentă cât mai apropiată de momentul apariției defecțiunii.

În ambele situații este necesară existența unor informații despre derularea tranzacțiilor până în momentul întreruperii lucrului şi aplicarea după caz a uneia din următoarele tehnici de restaurare de bază:

- derularea înapoi a tranzacțiilor necompletate (ROLLBACK) - care presupune anularea modificărilor efectuate de acestea asupra bazei de date;
- derularea înainte a tranzacțiilor completate dar nereflectate în baza de date (ROLLFORWARD) - care presupune efectuarea acelor transformări prin care baza de date restaurată să conțină rezultatele acestora.

Se observă deci că tranzacția poate fi considerată unitatea de restaurare, în sensul că baza de date restaurată trebuie fie să reflecte rezultatele finale ale tranzacțiilor, fie să nu fie afectată de acestea.

Procesul de restaurare utilizează o serie de informații obținute prin aplicarea unei anumite strategii de salvare.

Informații utilizate în procesul de restaurare

Salvarea, în contextul asigurării integrității bazei de date, este procesul de stocare de date în vederea folosirii lor pentru restaurarea bazei de date. Volumul informațiilor ce se salvează, natura lor şi intervalul de timp dintre două operații succesive de salvare, determină strategia de salvare. Aceasta va influența procesul de restaurare a bazei de date. Astfel, stocarea unei cantități mari de date, cu o frecvență mare şi în forme diferite (ceea ce are ca efect întreruperi ale lucrului sau mărirea timpului de răspuns în exploatarea bazei de date) face posibilă restaurarea simplă şi rapidă a bazei de date. în caz contrar, cu cât informațiile de care dispunem sunt mai sărace, cu atât procesul de restaurare va fi mai complicat şi de durată.

Datele salvate pot fi diferite combinații între:

- copii ale bazei de date şi copii ale jurnalelor acesteia;
- jurnale ale tranzacțiilor;
- jurnale ale imaginii înregistrărilor din baza de date.

Copiile bazei de date pot fi realizate automat de către sistem la anumite intervale de timp, sau la comanda administratorului bazei de date, ori de câte ori este nevoie, de preferat pe suporturi magnetice diferite de cele pe care rezidă baza de date. În cazul unei deteriorări a discului care păstrază baza de date, acestea constituie singura posibilitate de recuperare a

bazei de date. Ele pot fi utilizate însă ca unică sursă de date în procesul de restaurare doar în situația în care prelucrările efectuate între momentul realizării copiilor şi cel al apariției unei defecțiuni pot fi reluate. Acest lucru este posibil dacă prelucrările sunt efectuate într-o secvență cunoscută şi timpul necesar pentru reprocesarea lor nu este foarte mare sau poate fi acceptat în contextul de lucru particular. Această limită, ca şi rata mare a executării unor astfel de copii face ca anumite SGBD să recurgă la efectuarea de copii ale jurnalelor bazelor de date. Volumul datelor ce vor fi copiate este în acest caz mult mai mic, deci durata copierii va fi mai mică, iar procesul de restaurare implică doar într-o mică măsură intervenția umană.

Jurnalul tranzacțiilor este un fişier special întreținut de SGBD**,** în cure sunt memorate informații despre tranzacțiile efectuate asupra bazei de date, cum ar fi:

- identificatorul sau codul tranzacției;
- momentul începerii execuției tranzacției;
- numărul terminalului sau identificatorul utilizatorului care a inițiat tranzacția;
- datele introduse;
- înregistrările modificate şi tipul modificării.

Pe baza lui va putea fi stabilită ulterior succesiunea corectă şi natura prelucrărilor efectuate în intervalul de timp pentru care trebuie să se asigure restaurarea bazei de date.

Jurnalul imaginilor se deosebeşte de jurnalul tranzacțiilor prin aceea că el nu conține descrierea operațiilor efectuate asupra bazei de date ci efectul acestora. Poate îmbrăca una din următoarele forme:

-jurnalul cu imaginea înregistrărilor după modificare (after image) - va conține copia fiecărei înregistrări ce este modificată în forma rezultată după modificare;

- jurnalul cu imaginea înregistrărilor înaintea unei modificări (before image)-va cuprinde copia fiecărei înregistrări ce este modificată în forma inițială, anterioară efectuării modificării;

- jurnal care conține pe cele două de mai sus.

În prima formă, jurnalul imaginilor permite restaurarea bazei de date prin derularea înainte a tranzacțiilor într-un timp mai scurt decât în cazul utilizării jurnalului tranzacțiilor. Nu mai este necesară reprocesarea tranzacțiilor, putând fi utilizat direct rezultatul acestora, reprezentat de ultima imagine modificată a înregistrării respective. În mod similar, cea de-a doua formă a jurnalului imaginilor va putea fi utilizată pentru derularea înapoi a tranzacțiilor, fiind necesară doar regăsirea înregistrării memorate la începutul fiecărei tranzacții. Derularea înapoi a tranzacțiilor este foarte dificilă şi uneori chiar imposibilă dacă se foloseşte doar jurnalul tranzacțiilor. Cea de-a treia formă facilitează mult procesul de restaurare, însă presupune menținerea unui volum mare de informații redundante (pentru o înregistrare ceea ce constituie after image după o modificare va deveni before image la următoarea modificare).

În funcție de defecțiunea care a determinat întreruperea lucrului, restaurarea bazei de date se realizează **automat** de către SGBD, sau **manual,** înțelegând prin aceasta că procesul de restaurare va necesita intervenția umană.

Restaurarea automată a bazei de date

Restaurarea automată este determinată de SGBD după oprirea şi restartarea sistemului în urma unei căderi. Prin acest proces, baza de date este adusă la o stare consistentă, prin derularea înapoi a tranzacțiilor active în momentul defecțiunii şi continuarea tranzacțiilor înregistrate ca finalizate în fişierul jurnal, dar care nu sunt încă reflectate în baza de date.
Situația în care efectele unei tranzacții completată înaintea căderii nu se regăsesc în baza de date ca şi existența informațiilor necesare pentru restaurare în fişierele jurnal sunt explicate de modul în care este gestionată memoria principală.

O cerere de aceea la date primită de SGBD va determina transferul unei pagini de disc în memoria principală. Eventualele modificări ale datelor, aflate acum în memoria principală, nu vor fi urmate imediat de rescrierea paginii respective pe disc. Acest lucru poate fi efectuat

periodic, la un anumit interval de timp, fie la o cerere explicită a sistemului sau pentru a face loc unei alte pagini de disc solicitată.

Inlocuirea cu o altă pagină are la bază un algoritm de tipul LRN (least-recently-used). Astfel, va fi aleasă pentru a fi înlocuită pagina de la a cărei ultimă actualizare s-a scurs cel mai mare interval de timp. Cu excepţia situaţiei în care este necesară înlocuirea unei pagini, rezultă că paginile frecvent utilizate vor fi menţinute în memorie, ceea ce duce la reducerea numărului de operaţii de transfer între memoria principală si suportul extern de memorie, deci la creşterea performanţelor sistemului.

Acelaşi regim de păstrare în memorie până la un transfer ulterior pe disc se aplică şi informaţiilor de jurnalizare a tranzacţiilor. SGBD trebuie însă să asigure înscrierea acestor informaţii în fişierul jurnal înainte de scrierea datelor modificate în baza de date. Dacă această ordine nu ar fi respectată, o cădere a sistemului după scrierea în baza de date dar înainte de înregistrarea transformărilor respective în jurnal, ar face imposibilă derularea înapoi a tranzacţiilor, cu alte cuvinte, chiar procesul de restaurare al bazei de date. De asemenea, paginile de memorie cu informaţiile de jurnalizare a tranzacţiilor, vor fi scrise pe disc automat în momentul execuţiei unei comenzi de genul COMMIT TRANSACTION. Astfel se explică cum o tranzacţie finalizată nu este reflectată de baza de date şi cum jurnalul tranzacţiilor poate oferi informaţiile necesare procesului de restaurare automată a bazei de date.

În afara aspectelor prezentate, sincronizarea memoriei cu baza de date şi fişierul jurnal se realizează şi prin executarea unui punct de verificare (checkpoint). SGBD poate executa puncte de verificare automat, la intervale fixe de sau ca răspuns la o comandă explicită CHECKPOINT. Frecvenţa punctelor de verificare influenţează durata procesului de restaurare automată a bazei de date. Aceasta din urmă este direct proporţională cu activitatea tranzacţională a bazei de date desfăşurată de la cel mai recent punct de verificare până la căderea sistemului. Restaurarea rapidă a bazei de date va necesita deci o frecvenţă mare a punctelor de verificare.

Un punct de verificare presupune executarea următoarelor operaţii:

- îngheţarea proceselor active la momentul respectiv;
- forţarea scrierii paginilor de memorie în jurnale şi apoi în baza de date;
- scrierea unei înregistrări speciale în jurnalul tranzacţiilor, necesară la restaurare şi reluarea prelucrărilor, care indică:

-starea fiecărui proces activ din momentul executării punctului de verificare;
- starea fişierelor temporare de lucru;
- poziţiile în fişierele secvenţiale;
- pointeri la cozile de mesaje;
- continuarea proceselor anterior îngheţate.

La nivelul SGBD pot exista *parametrii de configurare* care să influenţeze procesul de restaurare automată.

Pentru SQL - Server de exemplu, aceştia sunt:

1. Intervalul de restaurare - reprezintă timpul maxim admis pentru efectuarea restaurării automate în momentul încărcării sistemului. Valoarea acestui parametru este folosită de SGBD în calculul frecvenţei de executare a punctelor de verificare, calcul realizat pe baza unui algoritm special. Valoarea implicită a intervalului de restaurare este de 5 minute, ea putând fi modificată în limitele intervalului 1 - 32767 minute.

2. Indicatorul de restaurare - determină ce informaţii va scrie SGBD în fişierul de erori (error log file) în timpul restaurării automate. Valoarea implicită este 0, în acest caz informaţiile rezumându-se la device-ul folosit, baza de date ce este supusă procesului de restaurare, numărul tranzacţiilor derulate înainte şi a celor derulate înapoi. Parametrul poate fi setat pe valoarea 1, şi astfel informaţiile vor include detalii despre modul în care este tratată fiecare tranzacţie.

Ambii parametrii pot fi modificați folosind meniurile unui program utilitar SAF (Server Administration Facility) sau prin apelarea procedurii de sistem SP-configure, ca în exemplul următor:

Sp_configure " recovery interval", 4
Sp_configure " recovery flag", 1
reconfigure

Exemplul realizează setarea intervalului de restaurare la 4 minute şi ba indicatorului de restaurare la valoarea 1. Comanda de reconfigurare este necesară deoarece setarea indicatorului de restaurare nu are efect deâct după reîncărcarea sistemului. Dacă se doresc informații deepre valorile celor *2* parametrii. Pot fi folosite primele după comenzi în forma:

sp_configure "recovery interval"
sp_configure "recovery flag"

Restaurarea manuală a bazei de date

Restaurarea manuală, denumită astfel pentru că implică intervenție umană şi nu pentru că ar fi un proces manual, este necesară în situația distrugerii suportului de memorie externă pe care rezidă baza de date.

În anumite SGBD, acest proces se bazează doar pe efectuarea de copii de siguranță ale bazei de date. Restaurarea va consta în încărcarea celei mai recente copii a bazei de date şi reluarea prelucrărilor efectuate din momentul copierii până la producerea defecțiunii.

Pentru realizarea acestor copii se poate recurge la una din următoarei modalități:

1. Cea mai simplă şi mai utilizată necesită deconectarea tuturor utilizatorilor de la baza de date, efectuarea copiei, după care utilizatorilor le este permisă reconectarea la baza de date şi reluarea lucrului.

2. Mai eficientă, dar necesitând un cost de implementare mai mare, este efectuarea copiilor în mod dinamic, în timp ce utilizatorii accesează baza de date. Această facilitate este utilă în regim de lucru on-line, realizarea copiei fiind transparentă pentru utilizatori. O astfel de copie a bazei de date determina executarea automată a unui punct de verificare. Copia va reflecta starea bazei date din momentul respectiv, inclusiv efectele tranzacțiilor în curs de execuție. SGBD va realiza automat derularea înapoi a acestor tranzacții în procesul încărcării bazei de date, obținându-se astfel o stare consistentă a acesteia.

Timpul consumat de o operație de copiere, dependent de mărimea bazei de date ca şi de metoda de copiere utilizată (la nivelul logic sau la nivel fizic), determină stabilirea frecvenței de realizare a copiilor, care are implicații asupra duratei procesului de restaurare. Astfel, cu cât copia bazei de date de care dispunem va fi mai recentă, cu atât restaurarea va fi mai rapidă.

Restaurarea manuală poate fi facilitată dacă, pe lângă copii de siguranță ale bazei de date, SGBD permite şi efectuarea de copii ale fişierului jurnal, durnta de realizare a acestora fiind redusă. Acestea se efectuează dinamic şi incremental (nu vor conține informații redundante). În intervalul dintre două copieri ale bazei de date se vor realiza mai multe copii ale fişierului jurnal, fiecare dintru aceste reprezentând, din punct de vedere al informațiilor conținute, o continuare a cel anterioare.

Efectuarea unei astfel de copii presupune execuția automată a unui punct de verificare, deci sincronizarea memoriei cu fişierul jurnal şi cu baza de date. Tranzacțiile inactive din jurnal (cele pentru care s-a executat COMMIT TRANSATION înaintea punctului de verificare, deci sunt reflectate în baza de date) vor fi şterse din fişier. Procesul de restaurare va presupune încărcarea celei mai recente copii a bazei de date, urmată de încărcarea copiilor jurnalului în ordinea în care au fost efectuate şi actualizarea bazei de date pe baza acestora. Restaurarea va avea ca efect recrearea bazei de date din momentul reflectat de informațiile din ultima copie a jurnalului tranzacțiilor.

Exemplificând pentru SQL - Server, SGBD cu facilităţi puternice pentru restaurarea bazelor de date, procesul de restaurare manuală va consta în (fig. 10.6.):

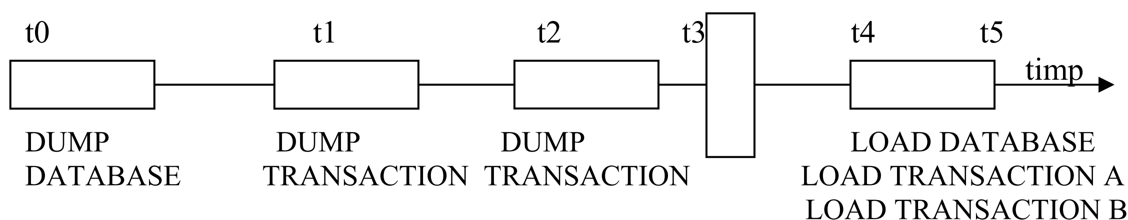


Fig. 10.6 Restaurarea manuală a bazei de date

1. Realizarea de copii de siguranţă
- efectuarea unei copii dinamice a bazei de date, prin comandă-
DUMP DATABASE baza TO dumpdev(momentul t0)
- efectuarea de copii ale jurnalului tranzacţiilor
DUMP TRANSACTION baza TO discdumpa(momentul t1)
DUMP TRANSACTION baza TO discdumpb(momentul t2)

2. Restaurarea bazei de date pe baza copiilor anterioare în urma incidentului de la momentul t3 (presupunem că un nou disc a fost instalat şi s-a alocat spaţiu pentru baza de date)
- încărcarea copiei bazei de date
LOAD DATABASE baza FROM dumpdev
- încărcarea copiilor jurnalelor tranzacţiilor în ordinea în care au fost executate:
LOAD TRANSACTION baza FROM discdumpa
LOAD TRANSACTION baza FROM disdumpb

10.2. Securitatea bazei de date

În general prin *securitate a datelor* se înţelege acea funcţie a unui SGBD prin care se asigură protecţia datelor împotriva unui acces neautorizat. Există două faţete ale conceptului de securitate a datelor: *protecţia datelor* şi *controlul autorizării*.

Protecţia datelor se realizează pentru a nu permite utilizatorilor neautorizaţi să înţeleagă conţinutul fizic al datelor sau pentru a preveni distrugerea voită a datelor. Protecţia fişierelor simple se poate realiza prin facilităţile oferite de sistemul de operare cu care se lucrează sau prin diferite tehnici de protejare a informaţiilor în reţelele de calculatoare. Astfel, în sistemul de operare UNIX administratorul de sistem atribuie *drepturi* asupra fişierelor: *dreptul de citire*(deci de a vedea un fişier), *dreptul de scriere* (deci de a modifica fişierul) şi *dreptul de execuţie.*

Cea mai simplă metodă prin care se poate realiza protecţia datelor este *criptarea* acestora. Această tehnică este utilizată atât pentru fişierele stocate pe disc cât şi pentru transmiterea în reţea a informaţiilor. Decriptarea datelor se poate realiza numai de către utilizatorii autorizaţi, adică acei utilizatori care cunosc codul utilizat. Există două scheme principale pentru criptare: Data Encryption Standard şi schema de criptare cu cheie publică.

Controlul autorizării trebuie să garanteze că numai utilizatorii autorizaţi pot realiza operaţii asupra bazelor de date. Fie că este vorba de un SGBD centralizat sau un SGBDD, acestea trebuie să fie capabile să asigure accesul la o parte a bazelor de date numai pentru o parte a utilizatorilor. Controlul autorizării se poate realiza prin intermediul sistemelor de operare şi astfel obţinem un *control centralizat.* Controlul autorizării în sistemele de baze de date diferă prin câteva aspecte de controlul tradiţional pentru sistemele de fişiere. Astfel, în cazul sistemelor de baze de date trebuie să se asigure drepturi diferite de accesare a aceloraşi obiecte de către diferiţi utilizatori. Este posibil ca pentru anumite obiecte unii utilizatori să aibă anumite drepturi, iar alţi utilizatori să aibă alte drepturi asupra aceloraşi obiecte.

În ceea ce priveşte controlul autorizării trebuie să facem distincţie între controlul asigurat în sistemele centralizate pe de o parte şi pe de altă parte controlul asigurat în sistemele distribuite.

a) **Controlul autorizării centralizate**

Următoarele trei entităţi sunt implicate în controlul autorizării:

- *Utilizatorii*, care sunt interesaţi în execuţia programelor
- *Operaţiile* implicate în programele de aplicaţii
- *Obiectele bazei de date* asupra cărora se execută operaţiile.

Fiind dat un triplet (utilizator, operaţie, obiect), controlul autorizării se poate enunţa ca fiind funcţia sistemului prin care se verifică dacă *utilizatorul* are voie (deci este autorizat) să execute *operaţia* asupra *obiectului.*

Introducerea unui utilizator în sistem se realizează în principal prin specificarea perechii (*nume_utilizator, parolă*). În mod unic *nume_utilizator* identifică un utilizator al sistemului, iar *parola* autentifică utilizatorul. Pe de altă parte *parola*, cunoscută numai de utilizatorul respectiv, împiedică intrarea în sistem a unui intrus.

Granularitatea protecţiei asigurată de un sistem de gestiune a bazelor de date este mai fină decât a sistemelor bazate pe fişiere. La acestea din urmă, granula de protecţie este fişierul. Într-un SGBD mecanismul de vizualizare a obiectelor permite protecţia chiar prin ascunderea unora dintre acestea. Astfel putem avea atribute, tupluri sau relaţii *hiden* (ascunse) şi care nu pot fi vizualizate de utilizatorii neautorizaţi.

Un *drept* exprimă o relaţie dintre un utilizator şi un obiect pentru o mulţime precizată de operaţii. Într-un SGBD relaţional bazat pe un SQL, o operaţie este desemnată printr-o instrucţiune (de exemplu SELECTE, INSERT, UPDATE sau DELETE), iar drepturile sunt *acordate* sau *revocate* prin instrucţiuni de forma:

GRANT < tip operaţie> ON <obiect > TO < utilizator>

REVOKE <tip operaţie> FROM <obiect> TO < utilizator>

sau alte variante în care se pot acorda/revoca mai multe tipuri de operaţii pentru unul sau mai mulţi utilizatori.

Cuvântul cheie *public* se utilizează pentru a desemna toţi utilizatorii. Controlul autorizării ridică următoarea problemă: cine acordă/revocă drepturile utilizatorilor? În controlul centralizat răspunsul este următorul: un singur utilizator, o clasă de utilizatori sau administratorii bazei de date. Ei sunt aceia care au toate privilegiile asupra obiectelor bazei de date şi ei sunt singurii care pot utiliza instrucţiunile GRANT şi REVOKE. Într-un sistem descentralizat problema aceasta este mai complexă. În asemenea sisteme cel care crează un obiect devine *proprietarul* acestuia şi în consecinţă, are toate privilegiile de a acorda sau revoca drepturile asupra obiectului respectiv. Persoana care primeşte dreptul asupra obiectului are la rândul ei posibilitatea de a acorda acest drept altor persoane. Această procedură se generalizează şi prin urmare aceste persoane pot acorda drepturi altor persoane ş.a.m.d. Problema care apare este aceea în cazul revocării drepturilor. Dacă persoana A acordă dreptul persoanei B asupra efectuării unor operaţiuni asupra obiectului O, iar B acordă mai departe aceste drepturi persoanei C şi A revocă dreptul lui B atunci automat şi C pierde dreptul respectiv. De aceea, pentru a executa instrucţiunea REVOKE, sistemul de gestiune va trebui să memoreze ierarhia acordării drepturilor, iar creatorul obiectului se află în rădăcina acestei ierarhii.

Privilegiile acordate utilizatorilor sunt înregistrate într-un *catalog* sau *director al regulilor de autorizare.* Există mai multe moduri de a defini aceste cataloage. Cel mai simplu este să specificăm toate privilegiile într-o *matrice* a *autorizărilor*. O linie a acestei matrice defineşte utilizatorul, o coloană defineşte obiectul, iar elementul matricei de pe linia şi coloana respectivă defineşte operaţia autorizată. Operaţia autorizată se specifică prin *tipul operaţiei* (de exemplu, SELECT, DELETE etc). Uneori alături de tipul operaţiei se precizează

un predicat care aduce anumite restricţii cu privire la accesul la informaţie. Ca exemplu să considerăm următoarea matrice a autorizărilor:

	ANG	ASIG
Petre	UPDATE	NONE
Ana	NONE	SELECT WHERE RESP =”Manager”
George	SELECT	UPDATE

Din această matrice aflăm că Petre are dreptul de a executa operaţiuni de actualizare în relaţia ANG, iar Ana are dreptul de a executa operaţia SELECT în relaţia ASIG numai pentru acele tupluri pentru care atributul RESP are valoarea “Manager”.

Există mai multe puncte de vedere cu privire la memorarea unei matrice a autorizărilor. Considerăm că cel mai natural mod este acela prin care memorarea se face prin intermediul unei relaţii ternare de forma (nume_persoană, obiect, drept).

b) Controlul autorizării distribuite

Problemele legate de controlul autorizării într-un sistem distribuit rezidă din faptul că entităţile (obiectele, relaţiile etc) sunt distribuite. Aceste probleme se referă la autentificarea utilizatorului la distanţă, gestiunea regulilor de autorizare distribuită şi tratarea vizualizărilor şi grupurilor de utilizatori.

Autentificarea utilizatorilor la distanţă este necesară deoarece orice nod al unui sistem de gestiune a bazelor distribuite poate accepta programe iniţiate în alt nod şi autorizate în noduri aflate la distanţă. Pentru a preveni accesul unor utilizatori neautorizaţi, utilizatorul trebuie să fie identificat şi autentificat de asemenea în nodul accesat. Sunt posibile două soluţii:

- Numele utilizatorului şi parola acestuia trebuie să se găsească memorate în toate nodurile din catalogul global
- Toate nodurile se identifică pe ele însele; în acest fel un utilizator autorizat de un nod n_1 identificat de un nod n_2 va fi autorizat de nodul n_2.

Regulile de autorizare distribuită sunt enunţate la fel ca în cazul autorizării centralizate. Aceste reguli se memorează în catalog şi pot fi duplicate pe fiecare nod sau numai pe acel nod care procesează obiectele distribuite solicitate.

O vizualizare a unor obiecte poate fi considerată de asemenea un obiect compus din toate obiectele respective. Prin urmare acordarea drepturilor asupra unei vizualizări se reduce la acordarea drepturilor asupra obiectelor componente. Aici o soluţie poate fi legată de acordarea drepturilor de citire de către proprietarul obiectelor, aşa cum s-a văzut mai înainte.

Administrarea unei baze de date se poate simplifica dacă utilizatorii sunt grupaţi în *grupe de utilizatori*. De obicei toţi utilizatorii unei baze formează o clasă numită *public*. Acest concept se utilizează atât într-un SGBD centralizat cât şi în unul distribuit. Însă în cazul distribuit apare un nivel suplimentar care specifică clasa public pentru un nod particular. Clasa public, chiar şi pentru un nod particular formează un grup de utilizatori. Desigur se pot defini şi alte grupe de utilizatori prin comenzi specifice.

BAZE DE DATE

CURS 11

BAZE DE DATE ORIENTATE PE OBIECTE

11.1 Obiective, domenii de aplicare
11.2 Modelul de date orientat pe obiecte
Concepte de bază
Operatorii modelului de date orientat pe obiecte
11.3 Baze de date orientate pe obiecte
Conceptul de bază de date orientată pe obiecte
Proiectarea bazei de date orientată pe obiecte
11.4 Sisteme de gestiune a bazelor de date orientate pe obiecte
Definiție. Caracteristici.
Arhitecturi si limbaje pentru SGBD-OO

11.1 Obiective, domenii de aplicare.

Aplicațiile din domeniile proiectării asistate de calculator, a sistemelor informatice geografice şi a sistemelor bazate pe cunoştințe, presupun stocarea unor cantități mari de informații cu o structură complexă. Aceste aplicații necesită suport pentru tipurile de date care nu pot fi reprezentate în sistemele clasice. De exemplu, baza de date a unui sistem de proiectare în domeniul ingineriei necesită stocarea unor cantități mari de date sub formă de diagrame şi descrieri ale proiectului tehnic. De asemenea, aplicațiile de proiectare asistată de calculator pot solicita monitorizarea unor desene formate din grupuri de elemente complexe ce trebuie să fie combinate, separate, suprapuse şi modificate astfel încât să permită rularea unor programe variate de proiectare.

Orientarea spre multimedia aduce elemente noi în lumea informaticii. Grafica, imaginea fotografică, video, sunetul nu pot fi tratate în aceeaşi manieră cu structurile tabelare de denumiri şi numere.

Recent, conceptele de obiect au fost înglobate şi în tehnologia SGBD-urilor, rezultând producția de sisteme de gestiune a bazelor de date orientate pe obiecte (SGBD-OO).

Enumerăm doar câteva din *domeniile* care se pretează în mod deosebit la o tratare orientată pe obiecte:

- proiectare CAD (Computer- Aided Design), CAM (Computer-Aided Manufacturing), CAE (Computer-Aided Engineering), CASE (Computer-Aided Software Engineering);
- simulare şi modelare;
- sisteme informaționale spațiale: GIS (Geographic Information System);
- administrarea documentelor: automatizarea muncii de birou, CAP (Computer-Aided Publishig), munca în echipă;
- multimedia: imagine, video, audio;
- ingineria cunoaşterii: baze de cunoştințe, sisteme expert.

Obiectivele principale ale sistemelor de gestiune orientate pe obiecte sunt:

1. Puterea de modelare superioară a datelor. Aceasta se realizează prin:

-deschiderea către noi aplicații: CAO, FAO, cartografie, gestiune de documente, birotică;

-facilitarea sarcinilor de concepție: economie în expresie, posibilitatea de generalizare şi agregare a relațiilor, flexibilitate în modelare;

-evoluție către multimedia (sunet, imagini, texte);

2. Posibilități de deducție superioară (ierarhie de clase, moştenire);

3. Ameliorarea interfeței cu utilizatorul;

4. Luarea în considerare a aspectelor dinamice, integrarea descrierii structurale şi comportamentale.

11.2 Modelul de date orientat pe obiecte

Concepte de bază

Un model de date orientat pe obiecte are la bază noţiunea de entitate conceptuală şi defineşte un obiect ca o colecţie de proprietăţi care descriu entitatea.

Principalele concepte care sta la baza unui model orientat pe obiecte sunt: obiectul, încapsularea, persistenţa, clasa, tipul, moştenirea, polimorfismul, identitatea, domeniul.

Obiectele sunt abstractizări ale entităţilor lumii reale şi se caracterizează prin stare şi comportament. Starea unui obiect este exprimată prin valorile atributelor sale. Colecţia de atribute alesă pentru un obiect trebuie să fie suficientă pentru a descrie entitatea, adică trebuie să includă acele atribute pe care utilizatorii trebuie să le cunoască.

Comportamentul unui obiect reprezintă un set de metode sau operaţii care acţionează asupra atributelor sale. Fiecare obiect are asociat un nume care coincide, de obicei, cu numele entităţii reprezentate.

Exemple de obiecte: un număr întreg, un avion, un angajat, un sindicat.

Exemplul 11.1 Să considerăm obiectul avion. Atributele unui avion pot fi direcţia, altitudinea, viteza la sol (dacă este în mişcare), greutatea şi culoarea. Operaţiile unui avion pot fi abilitatea de mişcare, viteza modificabilă şi riposta la stimuli, cum ar fi vântul sau temperatura exterioară. Obiectul AVION poate avea atribute pentru AVION-ALTITUDINE, AVION-DIRECŢIE, AVION-VITEZĂ, AVION-POZIŢIE, şi AVION-CONSUM DE COMBUSTIBIL. AVION poate avea de asemenea definite proceduri care să ilustreze operaţiile sale: SETARE_ALTITUDINE, PREZENTARE_DIRECŢIE, CALCUL_VITEZĂ, PREZENTARE_VITEZĂ, CALCUL_CONSUM DE COMBUSTIBIL etc.

În general, pot fi identificate trei tipuri de obiecte:

- obiecte elementare: întreg, boolean, şir de caractere, etc;
- obiecte compuse: nume, adresă;
- obiecte complexe: avion, angajat.

Un obiect înglobează următoarele elemente:

- structura de date;
- specificarea operaţiilor;
- implementarea operaţiilor.

Structura unui obiect şi operaţiile (metodele) permise pentru acel obiect sunt definite împreună.

O *metodă* reprezintă un program ce manipulează obiectul sau indică starea sa. Metoda este întotdeauna asociată unei clase. Specificarea metodei poartă numele de "*semnătură*" iar modul de implementare constituie *"corpul"* metodei. Secvenţa de program care implementează metoda poate fi compilată în orice moment al dezvoltării aplicaţiei, fără ca nici o altă recompilare să fie necesară.

Metodele şi atributele nu sunt vizibile din *"exteriorul"* obiectului. Astfel, un obiect are o implementare care este privată şi o interfaţă care este publică şi poate fi *"văzută"* de utilizatori şi de alte obiecte. Implementarea poate fi modificată fără a modifica interfaţa.

Un obiect răspunde la mesaje. *Mesajele* reprezintă cereri adresate obiectului pentru a returna o valoare sau pentru a-şi schimba starea. Ele constituie interfaţa obiectului cu mediul.

Un mesaj indică unul sau mai multe obiecte şi o metodă ce se aplică acestora. Mesajele cuprind: numele mesajului, numele obiectului pentru care au fost transmise şi argumentele necesare, dacă există. Când un obiect primeşte un mesaj, una din procedurile sale

este apelată. Procedura realizează apoi o operație care poate returna un rezultat. Mesajele sunt implementate prin intermediul metodelor.

Pentru a vedea cum operează aceste elemente în Exemplul 11.1, un program simulând zborul avioanelor poate modela efectul vântului asupra avionului prin trimiterea unui mesaj CALCULUL VITEZEI unui obiect AVION. Acest mesaj ar include argumente privind direcția şi viteza vântului. Metoda CALCUL VITEZĂ pentru obiectul AVION este apelată şi o nouă viteză şi direcție sunt calculate. Variabilele corespunzătoare sunt actualizate, aceasta modificând starea internă a obiectului.

Obiectele pot fi compuse din alte obiecte. *Obiecte compuse* sau *obiecte complexe* sunt printre cele mai recente descoperiri în domeniul tehnologiei obiect. Un obiect compus este realizat din alte obiecte. Astfel, atributele unui obiect pot fi ele însele obiecte. Un obiect compus are deci o structură ierarhică.

De exemplu, un obiect AVION poate fi definit ca un obiect compus, conținând părți separate ca MOTOR_JET, FUZELAJ, CABINA_PILOT etc. Fiecare parte componentă este un obiect şi o instanță a unei clase. Procesul poate fi extins pentru a realiza o *ierarhie de părți*, ilustrată în Figura 11.1.

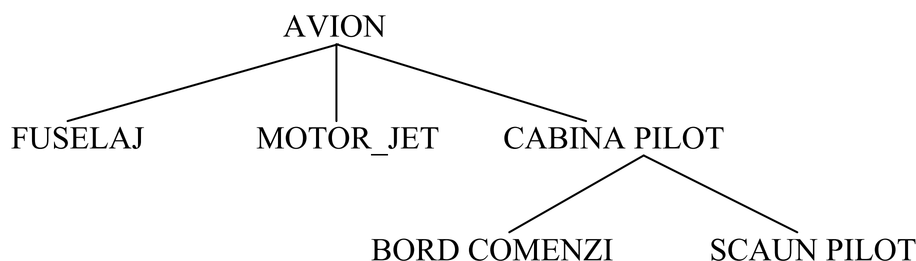


Figura 11.1 Ierarhie de părți

Componentele individuale se consideră a fi într-o relație de *”este parte din”* cu obiectul compus. De exemplu, BORD_COMENZI şi SCAUN_PILOT sunt într-o relație de *“este parte din”* cu CABINA_PILOT.

Structura obiectului şi modul de acțiune al metodelor sale nu pot fi accesate şi actualizate direct de către un agent extern, dar pot fi modificate indirect prin intermediul mesajelor. Această caracteristică ascunsă a stării obiectului este cunoscută sub numele de *încapsulare*. Un obiect este astfel divizat în două părți: o parte de interfața reprezentată de mesaje şi o parte ascunsă de implementare, reprezentată de starea internă şi de metodele obiectului. Interfața permite unui agent din exterior să solicite obiectului executarea unei acțiuni trimițând un mesaj corespunzător metodei asociate acțiunii.

Obiectele pot fi de asemenea considerate ca fiind *date abstracte*. Data abstractă este o tehnică folosită în multe limbaje de programare în care operațiile şi reprezentarea internă a unei entități calculabile sunt parțial ascunse din punct de vedere extern. Abstracția permite vizualizarea rezultatelor entității calculabile dar ascunde modul prin care s-a efectuat calculul. Revenind la Exemplul 11.1, când obiectul AVION primeşte mesajul CALCUL_VITEZĂ, detaliile metodei de realizare a calculului şi actualizările stării interne a obiectului nu sunt vizibile. Oricum, noua viteză şi direcție ale obiectului AVION pot fi obținute folosind protocolul obiectului, prin trimiterea mesajelor PREZENTARE_VITEZĂ şi PREZENTARE_DIRECȚIE care fac ca valorile variabilelor AVION_VITEZĂ şi AVION_DIRECȚIE să fie returnate.

Încapsularea ascunde utilizatorului complexitatea unui obiect, oferindu-i în schimb o imagine funcțională simplificată a acestuia, imagine care-i permite să modeleze şi să rezolve cu mai multă uşurință probleme complexe.

Proprietatea datelor sau a obiectelor de a avea o existenţa mai îndelungată decât procesul care le-a creat, se numeşte *persistenţă*. Este proprietatea prin care starea bazei de date asigură execuţia unui proces pentru a fi refolosit ulterior în alt proces.

Codul aferent metodelor-făcând parte integrantă din obiect- este stocat, ca şi starea obiectului, în baza de date. Aceasta înseamnă că, odată ce a fost descrisă, o metodă devine permanentă simplificând astfel aplicaţia şi asigurându-i independenţa faţă da date. O modificare adusă unei metode devine imediat operantă şi persistă până la o nouă modificare.

Obiectele care au acelaşi fel de atribute şi comportament pot fi clasificate ca făcând parte din acelaşi *tip* sau *clasă*. În raport cu această caracteristică, există două categorii de sisteme orientate pe obiecte:

1. Cele care admit ca noţiune de bază clasa, cum sunt: Smalltalk, Gemstone, Vision, Orion, G-Base;
2. Cele care admit ca noţiune de bază tipul, precum: C_{++}, Simula, Vbase, O2.

Într-un sistem orientat pe obiecte, *tipul* sintetizează elementele comune ale unei mulţimi de obiecte cu aceleaşi caracteristici. Corespunde noţiunii de tip abstract de date şi are două componente: interfaţa şi implementarea.

Pentru utilizator numai partea de interfaţă este vizibilă, cea de implementare fiind obiectul activităţii proiectantului.

Interfaţa constă dintr-o listă de operaţii, în timp ce implementarea presupune două activităţi:

- descrierea structurii interne a datelor obiectului;
- realizarea procedurilor de implementare a operaţiilor interfeţei.

Fie A un univers al atributelor şi T o mulţime de tipuri predefinite: integer, real, boolean, char, string etc.

Un tip este construit recursiv, astfel:

- orice element al mulţimii T este un tip *atomic* (sau *elementar*);
- dacă $t_1, t_2, ..., t_n$ sunt tipuri de dată şi $a_1, a_2, ..., a_n$ o mulţime de atribute din A, atunci $t=[a_1:t_1, a_2:t_2, ..., a_n:t_n]$ este un tip de dată *tuplu*;
- dacă t_1 este un tip de dată, atunci: $t=\{t_1\}$este un tip de dată *mulţime*;
- dacă t_1 este un tip de dată, atunci: $t = (t_1)$ este tip de dată *listă*;
- orice tip de date se poate obţine prin aplicarea repetată a regulilor de mai sus.

Vom exemplifica modul de declarare şi implementare a operaţiilor pentru aceste tipuri prin intermediul produsului O2 implementat în C_{++}.

Declararea unui tuplu:

```
type persoana:
tuple (vâarsta: integer,
        nume    : string,
        co_leg  : persoana,
        copii   : set (Persoana),
        oras    : tuple (nume: string,
                         cod: integer))
```

Declararea unei clase:

```
class Populaţie type
        list (tuple (nume   : string,
         familie:  set (Persoana)))
```

Noţiunea de *clasă* deşi are aceeaşi specificaţie cu cea de tip, este diferită de aceasta, fiind legată mai mult de faza de execuţie. Presupune două aspecte:

- generarea de obiecte (prin operaţia “new” aplicată unei clase);
- stocarea setului de obiecte care reprezintă instanţele clasei.

O clasă are o descriere ce constă dintr-un set de structuri de date comune, cunoscute ca variabile de instanță, un protocol comun ce constă dintr-un set de mesaje, la care instanțele clasei vor răspunde şi un set de metode pentru implementarea de operații comune.

Clasele sunt de asemenea referite uneori ca tip de dată abstractă. Descrierea clasei serveşte ca şablon după care vor fi create noile obiecte. O clasă este deci un tip abstract de date care defineşte atât structura obiectelor din clasa respectivă, cât şi mulțimea metodelor pentru aceste obiecte. Ca urmare, obiectele din aceeaşi clasă au aceleaşi atribute şi aceleaşi metode şi răspund la acelaşi mesaj. Din această cauză putem spune că metodele aparțin claselor şi nu obiectelor în sine.

Din punct de vedere al bazelor de date, o clasă poate fi considerată ca un tip înregistrare constând din *metadata* care asigură întreaga informație necesară pentru a construi şi a folosi un anume obiect. Similar, e posibil de a considera instanțele unei clase ca fiind înregistrări stocate într-un fişier. Noi înregistrări având diferite valori pot fi adăugate în fişier.

Într-o bază da date orientată pe obiecte, clasele sunt aranjate într-o ierarhie în care fiecare clasă moşteneşte toate atributele şi metodele superclasei din care face parte. *Moştenirea* este un concept puternic care conduce la posibilitatea de reutilizare a unor secvențe de program, şi deci la creşterea semnificativă a productivității în proiectare.

Deoarece unele obiecte sunt tipuri specifice, particulare, a altor obiecte, se pot realiza definiri de clasă în care o clasă specifică poate împărți sau împrumuta o parte din descrierea unei clase mai generale.

Mecanismul de realizare a definirii unei clase în care derivă variabile de instanță şi metode din altă definire de clasă se numeşte *moştenire*. Când o clasă moşteneşte variabile de instanță şi metode din altă clasă, este considerată a fi o subclasă. Clasa de la care variabilele de instanță şi metodele sunt moştenite este supraclasă. Conceptele de subclasă şi supraclasă sunt analoge conceptelor de *generalizare* şi *specializare* familiare metodologiilor de modelare a datelor.

În Exemplul 11.1, AVION_PASAGERI şi AVION_DE_MARFĂ pot fi tipuri specializate ale AVION. Un AVION_DE_PASAGERI poate avea toate atributele şi mare parte din comportamentul său derivate din AVION. Dar poate avea şi atribute în plus, cum ar fi numărul de pasageri. AVION_DE_PASAGERI poate avea de asemenea un comportament specific diferit de AVION, determinat de restricții cum ar fi rata maximă de aterizare.

AVION_DE_PASAGERI poate fi definit astfel, ca o subclasă a obiectului AVION, moştenind toate varibilele de instanță din definirea clasei AVION, dar are şi una proprie: NUMĂR_DE_PASAGERI. Toate instanțele obiectului AVION_DE_PASAGERI vor avea variabilele de instanță ale obiectului AVION dar vor avea, de asemenea, şi NUMĂR_DE_PASAGERI.

Exemplu de implementare a moştenirii în sistemul O2:

```
class Avion_de_pasageri
   type tuple ( număr   : întreg,
                plecare  : Data,
                sosire   : Data )
   method public calc_tarif :  real
   end;

class Avion_tip_linie inherit Avion_tip_navetă
   type tuple ( coef  :  real )
   method public calc_tarif  :  real,
          public afis_coef
   end;
```

Pentru clasa *Avion_tip_linie* metoda *calc_tarif* va fi redefinită deoarece tariful va include în acest caz o suprataxă.

Formal noțiunea de moştenire se poate defini astfel.

Considerăm o mulțime T de tipuri de dată şi o mulțime A de atribute. Vom defini în mod recursiv relația de ordine (≤) pe mulțimea tipurilor de dată, numită *relație de moştenire* sau relație de subtipaj:

- dacă t_1, t_2 sunt tipuri de dată atomice, iar a_1 şi a_2 sunt atribute din A, atunci:
  - $[a_1: t_1, a_2: t_2] \leq [a_1: t_1]$;
  - $\{t_1, t_2\} \leq \{t_1\}$;
  - $(t_1, t_2) \leq (t_1)$.
- dacă t_1, t_2 sunt tipuri de dată şi a un atribut, atunci: $[a : t_1] \leq [a : t_2]$, dacă $t_1 \leq t_2$;
- dacă t_1 şi t_2 sunt tipuri de dată, atunci: $\{t_1\} \leq \{t_2\}$, dacă $t_1 \leq t_2$;
- dacă t_1 şi t_2 sunt tipuri de dată, atunci $(t_1) \leq (t_2)$, dacă $t_1 \leq t_2$.

Pe baza moştenirii, pot fi create *ierarhii de clase* care reflectă relațiile naturale regăsite în domeniile lumii reale (Figura 11.2).

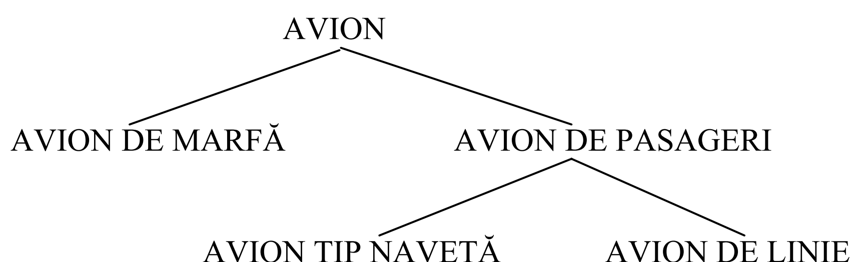


Figura 11.2 Ierarhie de clasă

Este de asemenea posibil ca o clasă să aibă mai mult decât o supraclasă. Acesta este cunoscut ca *moştenire multiplă.*

Pentru SGBD-OO noțiunea de moştenire este esențială. Ea poate fi simplă sau multiplă. Dacă avem în vedere doar relația de moştenire simplă, modelarea complexă nu este posibilă. De exemplu, o Persoană poate fi Student sau Salariat dar întâlânim şi cazul Student-Salariat.

Polimorfismul se referă la faptul că, la primirea unui mesaj, stabilirea metodei care se aplică se face în mod dinamic, în funcție de clasa obiectului în cauză. Astfel, instanțe ale unor clase diferite pot fi adresate uniform (primesc aceleaşi mesaje), dar manifestă comportamente diferite. Acest fapt asigură manipularea simplă şi coerentă a seturilor eterogene de obiecte.

Un alt tip de comportament polimorfic este asociat cu moştenirea. Răspunsul unui obiect la un mesaj poate fi determinat de metodele moştenite de la supraclasă. Moştenirea multiplă permite definirea unor forme complexe de comportament polimorfic care pot antrena uneori combinarea metodelor de la două sau mai multe supraclase.

Identitatea este un mijloc de a distinge un obiect de altul. Prin identitate se asigură şi persistența datelor.

Oricare din obiectele unei baze de date orientate pe obiecte are identitate care este independentă de valorile atributelor sale. Spre deosebire de modelul relațional, care utilizează unicitatea cheii primare pentru a identifica "obiectul", tehnologia orientată pe obiecte permite modificarea valorilor oricărui atribut fără a-i afecta identitatea. Mai mult chiar, obiectele au "conştiința de sine", şi anume pointerul Self, prin intermediul căruia obiectele se pot referi la ele însele.

Fiecare instanță sau realizare a obiectului are un identificator de obiect intern, repartizat lui şi cunoscut ca un *ID obiect* sau *pointer*. Acesta este independent de valorile atributelor sale. Fiind generat de sistem, identificatorul este unic şi nu este accesibil utilizatorului. Deci, pot exista multe obiecte tip AVION, fiecare cu un ID obiect unic.

Operatorii modelului de date orientat pe obiecte

Operațiile acestui model pot fi grupate astfel:
- obiectele comunică între ele prin mesaje. Transmiterea şi respectiv primirea de mesaje stă la baza operațiilor modelului orientat pe obiecte;
- un mesaj poate fi trimis instanțelor mai multor clase. Comportamentul polimorfic presupune selectarea metodei adecvate definită pentru clasa obiectului respectiv sau pentru o superclasă;
- metodele pot fi definite, şterse sau modificate;
- clasele pot fi definite şi actualizate prin operații de creare, ştergere şi modificare;
- instanța unei clase poate fi actualizată prin metode ce modifică valorile variabilelor propriei instanțe, aceasta modificând starea internă a obiectului;

Într-o serie de implementări, definirile de clasă sunt ele însele obiecte, numite obiecte clasă. Obiectele clasă sunt instanțe ale unei clase generice sau ale unei supraclase. Deci, operațiile de creare, modificare şi ştergere a definirilor de clasă pot fi implementate ca mesaje.

Definiția 11.1 *O bază de date este o mulțime finită de clase* $B = \{C_1, C_2, ..., C_n\}$, *unde* C_i *este o mulțime finită de obiecte* $O_i = \{O_{i1}, O_{i2}, ..., O_{im}\}$ *de tip* t_I *ce are o mulțime* M_i *de metode,* $\forall i \in \{1, 2, ..., n\}$.

Invocarea unei medode a clasei $C_i \in B$ se realizează printr-o funcție m_i: $C_i \rightarrow C_i$, numită *mesaj*. Mulțimea mesajelor $m = \{m_1, m_2, . . . , m_p\}$ care pot fi adresate clasei C_i se va numi *protocol de mesaje*. Clasele de obiecte sunt deci colecții de obiecte care respectă un protocol comun de mesaje.

Pentru exemplificarea operațiilor care vor fi definite, vom considera trei clase ale unei baze de date şi anume:

a) clasa **Profesor**, conține informații despre profesori:

curs_predat	{**Curs**}
nume	string
prenume	string
matricol	integer

b) clasa **Student**, conține informații despre studenți:

curs_luat	{**Curs**}
nume	string
prenume	string
matricol	integer

c) clasa **Curs**, conține informații despre cursurile predate/luate:

nume_curs	string
studenți	{**Student**}
profesori	{**Profesor**}

cu următoarele referințe:

studenți	Referință către clasa **Student.**
profesori	Referință către clasa **Profesor.**
Curs_luat	Referință către clasa **Curs.**

Într-o bază de date orientată obiect se pot crea clase noi de obiecte care vor face referi la obiectele originale, şi care vor filtra mesajele către obiectelor originale.

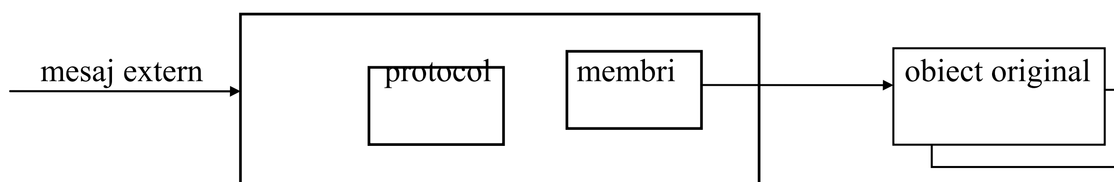


Figura 11.3 O instanță a unei CV

Prin aplicarea operațiilor algebrice se vor obține *clase virtuale* (*CV*) de obiecte, ale cărei componente sunt obiecte ale claselor existente care sunt văzute împreună deoarece posedă aceleaşi caracteristici şi recunosc aceleaşi mulțimi de mesaje.

O instanță a clasei CV va conține două variabile, indicate în Figura 11.3.

Dacă prin aplicarea unei operații algebrice se crează o clasă CV**,** variabila *membri* reprezintă lista obiectelor referite, iar variabila *protocol* reprezintă mulțimea mesajelor.

Tabelul următor prezintă notațiile utilizate în definirea formală a operațiilor.

s, r	Reprezintă identificatori de obiecte
S, R	Reprezintă mulțimi de obiecte din aceeaşi clasă
M_S, M_R	Protocoale de mesaje partajate de mulțimile R şi S
M	o mulțime a mesajelor
CV (s)	o reprezentare pentru s, adică CV.membri conține s
CV (s, r)	o reprezentare pentru s şi r, adică CV.membri conține s şi r
c	o condiție
CO	clasă de obiecte

În continuare vom defini principalele operații ale algebrei relaționale, importante şi pentru o bază de date orientată obiect:.

1. **Selecția**: $\sigma_{(condiție)}$ *CO.*

Rezultatul acestei operații este o submulțime de obiecte din *CO* care satisfac condiția dată. Dacă operația se termină cu succes, mesajele din condiție trebuie să fie înțelese de obiectele din *CO*.

O definiție formală a selecției este următoarea:

$$\sigma(R, p) = \{s \mid s \in S \wedge p(s)\}$$

De exemplu, $\sigma_{(nume\ =\ "Popa"\ \wedge\ prenume\ =\ "Ion")}$ *studenți* are ca rezultat o mulțime de obiecte din clasa **Student** pentru care **r**ăspunsul la mesajul *nume* este ″Popa″ şi răspunsul la mesajul *prenume* este ″Ion″. Această operație returnează mulțimea studenților cu numele ″Popa″ şi prenumele ″Ion″.

2. Proiecția: $\Pi_{(listă_de_mesaje)}$ *CO.*

Rezultatul acestei operații este o submulțime a obiectelor clasei CV, câte un obiect pentru fiecare membru al mulțimii *CO***.** Obiectele din clasa CV înțeleg mesajele conținute în *listă_de_mesaje* care trebuie să fie o submulțime a mesajelor înțelese de obiectele din *CO*.

O definiție formală: $\Pi(S, M_S) = \{CV(s) \mid s \in S \wedge CV(s).protocol = M_S\}$.

De exemplu, $\Pi_{(nume)}$ *studenți* are ca rezultat o mulțime a clasei de obiecte CV, unul pentru fiecare membru din clasa **Student**. Obiectele clasei CV înțeleg mesajul *nume*, care trebuie să fie înțeles de toate obiectele clasei **Student**.

3. Produs cartezian (×): $CO_1 \times CO_2$.

Rezultatul acestei operații este o mulțime a clasei de obiecte CV, câte unul pentru fiecare combinație posibilă de obiecte din CO_1 şi CO_2.

O definiție formală a acestei operații este următoarea**:**

$$S \times R = \{CV(s, r) \mid r \in R \wedge s \in S \wedge CV(s, r).protocol = M_S \cup M_R\}$$

De exemplu, *cursuri* × *profesori* are ca rezultat o mulțime a clasei de obiecte CV, în toate combinațiile posibile între obiectele clasei **Curs** şi ale clasei **Profesor.**

4. Reuniunea (∪): $CO_1 \cup CO_2$.

O definiție formală a acestei operații este:

$$S \cup R = \{t \mid t \in S \vee t \in \text{i}\}$$

De exemplu, ($\sigma_{(matricol\ >\ 100)}$ *studenți*) $\cup$ ($\sigma_{(nume\ =\ "Ionescu")}$ *studenți*) are ca rezultat mulțimea studenților pentru care numărul matricol este mai mare decât 100 sau au numele ″Ionescu″.

5. Diferența (–): $CO_1 - CO_2$.

Rezultatul acestei operații este mulțimea obiectelor care sunt în CO_1 dar nu şi în CO_2.

O definiție formală a acestei operații este:

$$S\text{-}R=\{t \mid t \in S \wedge \neg (t \in R)\}$$

De exemplu, ($\sigma_{(matricol\ >\ 100)}$ *studenți*) – ($\sigma_{(nume\ =\ "Ionescu")}$ *studenți*) are ca rezultat mulțimea obiectelor din **Student** care furnizează un răspuns mai mare ca 100 pentru mesajul *matricol* şi nu dau răspunsul egal cu ″Ionescu″ la mesajul *nume*.

În continuare vom prezenta câteva operații care derivă din operațiile de bază şi deci se pot exprima cu ajutorul acestora.

6. Intersecția ($\cap$): $CO_1 \cap CO_2$.

Rezultatul acestei operații este mulțimea obiectelor care sunt atât în CO_1 cât şi în CO_2. Pentru testarea apartenenței obiectelor la o anumită clasă se utilizează identitatea obiect. Mulțimile de obiecte trebuie să fie compatibile, iar această operație este comutativă.

O definiție formală este: $S \cap R = S\text{-}(S\text{-}R)$.

De exemplu, ($\sigma_{(matricol\ >\ 100)}$ *studenți*) $\cap$ ($\sigma_{(nume\ =\ "Ionescu")}$ *studenți*) are ca rezultat mulțimea obiectelor din **Student** care furnizează un răspuns mai mare ca 100 pentru mesajul *matricol* şi răspunsul egal cu ″Ionescu″ la mesajul *nume*.

7. Join: CO_1 **Join**$_{\langle conditie \rangle}$ CO_2

unde *condiție* are forma: < listă_de_mesaje1 operator_de_comparare listă_de_mesaje2>.

Rezultatul acestei operații este o mulțime de obiecte din clasa CV, fiecare conținând un obiect din CO_1 şi un obiect din CO_2, unde obiectele dau răspunsuri corespunzătoare mesajelor din *listă_de_mesaje*$_1$ şi *listă_de_mesaje*$_2$ care îndeplinesc condițiat. Obiectele din clasa CV vor înțelege toate mesajele care pot fi înțelese de fiecare dintre obiectele din CO_1 şi CO_2. Dacă operatorul din condiție este "=", spunem că avem un **Join natural**.

O definiție formală a acestei operații este:

$$\textbf{Join}\ (S, R, c) = \sigma\ (\times\ (S, R), c)$$

De exemplu, *studenți* **Join**$_{\langle cursuri_luate=cursuri_predate \rangle}$ *profesori* are ca rezultat o mulțime a clasei de obiecte CV, fiecare conținând un obiect al clasei **Student** şi un obiect al clasei **Profesor**, şi unde ambele obiecte furnizează acelaşi răspuns la mesajele *cursuri_luate* şi respectiv *cursuri_predate*.

8. Semi-Join (SJ) : CO_1 **SJ**$_{\langle conditie \rangle}$ CO_2

Rezultatul acestei operații este o mulțime de obiecte din clasa CO_1 pentru care mesajele din *listă_de_mesaje*$_1$ furnizează valori care se compară cu răspunsurile corespunzătoare mesajelor din *listă_de_mesaje*$_2$ care vizează un obiect din CO_2**. SJ** generează o clasă de obiecte CV, dar protocolul clasei CV este acelaşi cu protocolul de mesaje al obiectelor din CO_1.

O definiție formală este: $\textbf{SJ}(S, R, M_S, c) = \Pi\ (\textbf{Join}\ (S, R, c), M_S)$.

De exemplu, *studenți* **SJ**$_{\langle nume=nume \rangle}$ *profesori* are ca rezultat o mulțime de obiecte din clasa **Profesor** pentru care mesajul *nume* returnează aceeaşi valoare cu mesajul *nume* al unui obiect din **Student.**

9. Diviziune obiect(Odiv): CO_1 $_{(Mesaj)}$ **Odiv** CO_2

Vom da o definiție formală pentru operatorul de diviziune Odiv, care utilizează operația DIVISION preluată din algebra relațională:

$$\textbf{Odiv}\ (S, R, M_S, M_R) = \textbf{SJ}(S, \textbf{DIVISION}\ (S, R, M_S, M_R), M_S, c)$$

De exemplu, *Studenți* $_{(cursuri_luate)}$ Odiv *Cursuri.* are ca rezultat o submulțime a clasei de obiecte **Student**, şi anume studenții care iau toate cursurile.

11. 3. Baze de date orientate pe obiecte

Conceptul de bază de date orientată pe obiecte

Baza de date orientată pe obiecte poate fi definită ca rezultatul aplicării tehnologiei orientate pe obiecte în domeniul stocării şi regăsirii informațiiloră6ş. Ea oferă posibilitatea de a reprezenta structuri de date foarte complexe cu ajutorul obiectelor.

Definirea clasei este mecanismul de specificare a schemei bazei de date. Schema bazei de date constă din toate clasele care au fost definite pentru o aplicație particulară. Definițiile de clasă includ moştenirea, relațiile de înrudire (superclasa, subclasa) şi relațiile structurale dintre clase.

O schemă completă de bază de date poate consta din una sau mai multe ierarhii de clasă împreună cu relațiile structurale. Descrierile individuale ale schemei se referă la variabilele de instanță ale claselor individuale.

Schema bazei de date poate fi modificată dinamic, în funcție de necesitățile utilizatorilor. Modificarea schemei presupune:

- Definirea unei taxonomii şi a unui model al modificărilor. Taxonomia defineşte un set de modificări semnificative ale schemei, iar modelul furnizează o bază pentru specificarea semanticilor acestei modificări;
- Implementarea modificărilor schemei. Pot fi identificate două tipuri de modificare a schemei unei baze de date orientate pe obiecte:

1. Modificări referitoare la modul de definire al unei clase. Acestea includ schimbările atributelor şi metodelor definite pentru o clasă, cum ar fi schimbarea numelui sau domeniului unui atribut, adăugarea, ştergerea unui atribut sau metode;
2. Modificări referitoare la structura ierarhiei de clase care include adăugarea sau ştergerea unei clase şi schimbarea relațiilor superclasa/subclasa dintre o pereche de clase.

Proiectarea bazei de date orientată pe obiecte

Modul clasic de proiectare se bazează pe tehnica top-down. Se identifică mai întâi componentele majore, se stabilesc corelațiile între ele, iar apoi se trece la rafinări succesive, *"în cascadă"*, a componentelor.

Proiectarea orientată pe obiecte se bazează mai mult pe tehnica bottom-up. Se stabilesc mai întâi componentele funcționale pe baza cărora se va construi apoi întregul sistem. Se identifică în colecțiile existente, obiectele care pot fi reutilizate pentru noul proiect. Acestea vor fi preluate ca atare sau, dacă este cazul, vor fi ajustate. Cele ce nu există vor fi create, dar de cele mai multe ori ca subclase ale unor clase existente. Odată creată ierarhia de clase potrivită, se testează componentele specifice, se pune la punct documentația şi se poate începe acțiunea de implementare sau comercializare.

Această metodologie modifică în mod substanțial planificarea lucrărilor şi etapelor. Partea cea mai minuțioasă a proiectării se află la începutul proiectului. Dacă există deja biblioteci de obiecte utilizate în alte aplicații, realizarea unui prototip se poate face într-un timp foarte scurt. Pe baza lui se stabilesc aspectele funcționale şi de interfață ale aplicației, după care se trece la etapa de identificare a obiectelor, a claselor, a ierarhiei, a modului de comunicare între obiecte. Etapa finală a proiectului constă în asamblarea acestor elemente.

Refacerea unor aplicații "clasice" în noua tehnologie implică, de cele mai multe ori, refacerea aproape integrală a aplicațiilor.

Pe de altă patre, metodologia orientată pe obiecte poate fi aplicată cu succes în proiectare chiar dacă nu se urmăreşte utilizarea unor tehnici de realizare orientate pe obiecte. Avantajul constă în faptul că aceasta obligă la o analiză mai atentă a arhitecturii sistemului şi, mai departe, la o proiectare modulară, exprimată prin componentele aflate în interacțiune. Adăugarea sau înlocuirea ulteroiară a unor astfel de componente este mult mai uşoară.

11. 4. Sisteme de gestiune a bazelor de date orientate pe obiecte

11.4.1 Definiție. Caracteristici.

SGBD-OO conțin în plus, față de facilitățile oferite de sistemele tradiționale, structuri şi reguli orientate către lucrul cu obiecte. Ele includ:
- un sistem de date abstracte pentru construirea de noi tipuri de date;
- constructori de tip şir, secvență, înregistrare, set, reuniune;
- funcții, ca tip distinct;
- o compunere recursivă a elementelor de mai sus.

Principiile ce guvernează un SGBD-OO sunt următoareleă7ş:

1. Într-un SGBD-OO se utilizează funcții ce conțin metode şi proceduri ale bazei de date cu restricția ca acestea să fie cât mai compacte, încapsulate, ermetizate.

Încapsularea funcțiilor îl ajută pe programatorul de aplicație să asocieze funcțiile pe care şi le crează cu colecțiile utilizate. De exemplu: ANGAJARE (SALARIAT), SPOR_SAL (SALARIAT). Acest fapt oferă avantaje din punct de vedere al reducerii numărului de accese la informația stocată în colecții precum şi în activitatea de rescriere a procedurilor.

A apărut astfel noțiunea de *"opacitate"* susținută de adepții orientării pe obiecte şi ai folosirii în exclusivitate a funcțiilor încapsulate, în opoziție cu cea de *"transparență"* promovată de adepții facilităților limbajelor de interogare.

2. SGBD-OO şi în general SGBD-urile din generația a treia vor subsuma avantajele SGBD-lor din a doua generație.

Generația a doua de SGBD-uri a avut o contribuție pozitivă în două direcții:
- Accesul nonprocedural. Existența unui limbaj de interogare prin intermediul căruia să putem interoga baza de date asupra unor atribute ale înregistrării;
- Independența datelor. SGBD-urile din generația a doua mențin consistența tuturor căilor de acces la date prin intermediul unui optimizator. În plus există posibilitatea definirii colecțiilor virtuale. Cele două caracteristici trebuiesc păstrate şi în SGBD-urile din generația a treia.

Adepții SGBD-OO propun navigarea către informația dorită prin intermediul unor proceduri de interfață de nivel scăzut. De fapt se caută o modalitate de acces la o înregistrare poziționată într-o colecție oarecare, problema reducându-se la un sistem de pointeri către identificatorii de obiecte.

Practica a demonstrat că acest stil de navigare nu este indicat deoarece înlocuieşte funcția de optimizare a evaluării cererilor, cu subrutine scrise de programator.

Pot exista două modalități de a specifica o colecție: prin enumerarea membrilor săi sau utilizând limbajul de interogare pentru a stabili calitatea de membru.

Prima formă de stabilire a calității de membru al unei colecții (metoda extinsă) are dezavantajul unui efort mare de programare. Avantajul îl constituie posibilitatea de aplicare în cazul unor sisteme nestructurate şi fără conexiuni între seturile de înregistrări.

Cea de a doua formă (metoda intensivă) este specifică pentru seturi şi prezintă avantajul introducerii automate a unei noi înregistrări în setul corespunzător. Transformările semantice pot fi executate de optimizator căruia i se lasă libertatea de decizie asupra căii de acces la informație, fără a mai fi limitat de structura de pointeri.

Modelul orientat pe obiecte utilizează prima metodă cu toate că a doua este mai performantă.

3. Un SGBD-OO trebuie să fie deschis către alte sisteme.
Acest principiu se referă la posibilitatea conexiunii cu limbajele din generația a patra, cu diferite instrumente de luare a deciziilor, cu sistemele de largă circulație.
Accesul la informația stocată în bază se va face prin intermediul limbajelor de nivel înalt.

Modalitatea universal valabilă de interogare va fi un limbaj de tip SQL, iar dialogurile sub forma întrebare/răspuns între utilizator şi calculator vor constitui nivelul cel mai scăzut de comunicare admis.

Un SGBD orientat pe obiecte trebuie, în primul rând, să satisfacă două criterii:

- să fie un sistem orientat pe obiecte, deci bazat pe paradigma modelării orientate pe obiecte;
- să îndeplinească cerințele unui sistem de gestiune a bazelor de date.

Cele două criterii generează un set de caracteristici sau reguli fundamentale ale modelului orientat pe obiecte. Aceste caracteristici pot fi grupate în trei categorii: obligatorii, opționale şi deschise.

Manipularea obiectelor complexe. Noțiunea de obiect complex a apărut prin aplicarea de constructori asupra obiectelor simple. Este un proces asemănător celui de creare de structuri complexe din entități simple cum sunt: întregi, şiruri de caractere de orice lungime, variabile de tip boolean.

Mulțimea minimală de constructori pe care sistemul trebuie să îi conțină cuprinde: setul, lista, tuplu (înregistrare).

Identitatea obiectelor. Este o caracteristică a limbajelor de programare preluată ulterior şi de proiectanții de SGBD-uri. Astfel, orice obiect există independent de valorile atributelor sale, ceea ce conduce la două relații posibile:

- identitatea a două obiecte (ele sunt unul şi acelaşi obiect);
- egalitatea a două obiecte (ele au aceeaşi valoare).

Din cele două relații derivă noțiunile de partajare şi modificare a obiectelor. Partajarea obiectelor afirmă faptul că două obiecte pot împărți aceleaşi componente. În funcție de tipul sistemului căruia i se aplică (cu sau fără identitate) pot apărea interpretări diferite.

Considerăm de exemplu, înregistrarea de tip persoană cu atributele: nume, vârsta, copii de forma:

(Petre, 40, {(Ion, 15 {})})
(Ana, 41, {(Ion, 15, {})})

În realitate pot exista două situații concrete descrise de aceste înregistrări: cea în care Petre şi Ana sunt părinții aceluiaşi copil, sau cea în care este vorba de doi copii. Într-un model ce acceptă identitatea obiectelor cele două structuri pot avea sau nu în comun componenta (Ion, 15, {}) în timp ce un model fără identitate a obiectelor elimină alte forme de interpretare, conducând către o structură de tip arborescent.

Modificarea obiectelor. Pe exemplul de mai sus, acceptând varianta în care Petre şi Ana sunt părinții lui Ion, orice modificare asupra fiului Anei sau asupra fiului lui Petre se va aplica asupra lui Ion.

Într-un sistem care acceptă identitatea obiectelor, dacă modificarea s-a efectuat asupra fiului Anei, automat ea va fi extinsă şi asupra fiului lui Petre.

Încapsularea. A apărut din două motive:

1. Necesitatea de a diferenția specificarea unei operații de implementarea acesteia;
2. Modularizarea ca obiectiv.

La origine încapsularea derivă din tipurile de date abstracte, presupunând existența unei interfețe şi a implementării corespunzătoare.

Partea de interfață reprezintă specificarea unui set de operații ce se pot aplica asupra unui obiect, fiind singura zonă vizibilă a obiectului.

Implementarea se referă atât la componenta de date cât şi la cea de procedură. Partea de date este reprezentarea obiectului , în timp ce partea de procedură descrie, cu ajutorul limbajelor de programare, implementarea fiecărei operații.

Considerăm încapsularea adusă la o formă finală în momentul în care numai operațiile sunt vizibile, iar datele şi implementarea operațiilor sunt mascate în obiect.
Ierarhii sau clase de tipuri (moştenire). Moştenirea reduce efortul de programare. Există patru modalităţi de a moşteni:
1. Prin substituție; dacă tipul *t* moşteneşte de la tipul *t~,* atunci asupra obiectului de tip *t* se vor putea executa mai multe operații decât asupra celui de tip t~. Acest tip de moştenire se bazează pe comportare, nu pe valori.
2. Prin incluziune; corespunde noțiunii de clasificare. Un tip *t* este un subtip al lui *t~*, dacă orice obiect de tip *t* este un obiect de tip *t~*. Această moştenire se bazează pe structură, nu pe operații.
3. Prin restricție; este un caz particular al moştenirii prin incluziune. Un tip *t* este un subtip al tipului *t~* dacă, conține toate obiectele de tip *t~* ce satisfac o restricție specificată.
4. Prin specializare; tipul *t* este un subtip al lui *t~* dacă obiectele din *t*, aparțin şi lui *t~*, dar conțin un surplus de informație specifică.
Extensibilitate. SGBD-OO trebuie să includă pe lângă clasele sau tipurile predefinite (clasa obiect, clasa dată etc.) şi instrumentele care să permită utilizatorului definirea de noi clase sau tipuri.
Completitudinea. Limbajul de interogare pentru baza de date orientată pe obiecte trebuie să permită exprimarea tuturor programelor posibile. Din acest punct de vedere putem spune că SQL nu este un limbaj complet.
Persistența. Limbajele de programare orientate pe obiecte utilizează conceptul de obiecte. Totuşi, aceste obiecte există numai în timp ce programul este executat. O bază de date orientată pe obiecte conține obiecte persistente care există permanent în baza de date.
Concurența în exploatare. Se referă la posibilitatea ca mai mulți utilizatori să manipuleze datele concurent.

11.4.2. Arhitecturi si limbaje pentru SGBD-OO

Termenul de arhitectură se referă la o descriere abstractă a organizării unui sistem în scopul de a prezenta componentele funcționale şi interfețele dintre ele. Nu există o părere comună privind arhitectura SGBD-OO. Mai mult, arhitectura poate fi o problemă de perspectivă, fiind dependentă de modul cum este văzut SGBD-OO de către cercetător, ca utilizator final sau ca administrator de sistem.

Arhitectura SGBD-OO cuprinde trei componente majore:
1.*Gestionarul de obiecte* (Obiect Manager) care asigură interfața dintre procesele externe şi SGBD-OO;
2. *Server-ul de obiecte* care este responsabil cu asigurarea serviciilor de bază ale SGBD-urilor, cum ar fi: gestiunea tranzacției şi gestiunea stocului de obiecte;
3. Stocul rezident de obiecte sau ODB-ul însăşi.

În Figura 11.4 se prezintă componentele de bază ale arhitecturii SGBD-OO.

Utilizatorii finali externi şi cei care dezvoltă aplicații pot folosi instrumente soft, cum ar fi: editoare de texte, editoare grafice, browseri de obiecte şi clase, accesorii de proiectare automată de baze de date şi interfețe pentru sisteme de proiectare CAD/CAM. Aceste sisteme pot servi ca instrumente front_end ce realizează interfață cu Gestionarul de obiecte.

Gestionarul de obiecte asigură implementarea completă a modelului de date obiecte pentru utilizatorul extern. Aceasta ar include posibilitatea de a defini structurile şi de a executa operațiile specificate prin model.

Gestionarul de obiecte primeşte cereri de creare de definiri de clase, de modificare a definirilor de clase deja existente, de manipulare a mesajelor generate de un program de aplicație în execuție şi de prelucrare a cererilor ad-hoc folosind translatorul de cereri. De

asemenea realizează legăturile dinamice şi operaţiile de verificare a sintaxei şi tipului (datei) necesare. Cerinţele sunt apoi transmise Server-ului de obiecte ca tranzacţii.

Funcţiile Gestionarului de obiecte, aşa cum reies din Figura 11.4, constau în:
- funcţiile de prelucrare a mesajelor, incluzşnd legătura la momentul execuţiei şi verificarea tipului, ca şi translatarea cererilor;
- facilităţi de definire şi modificare a schemei bazei de date, incluzând definiri de clasă noi sau corectate în ierarhii sau reţele de clasă existente.

Serverul de obiecte realizează recuperarea (refacerea), adăugarea, ştergerea şi actualizarea obiectelor stocate în stocul rezident în obiecte. Un singur server poate manipula tranzacţii transmise de la mai mulţi gestionari de obiecte. Funcţiile serverului de obiecte constau din:
- gestiunea tranzacţiilor, incluzând controlul concurenţei, gestiunea buffer-ului şi servicii de refacere în caz de incident;
- gestiunea fizică a stocului, incluzând plasarea obiectelor şi implementarea metodelor de acces:
- serviciile de arhivare şi de asigurare a copiilor de rezervă, pot fi , de asemenea, asigurate prin serverul de obiecte.

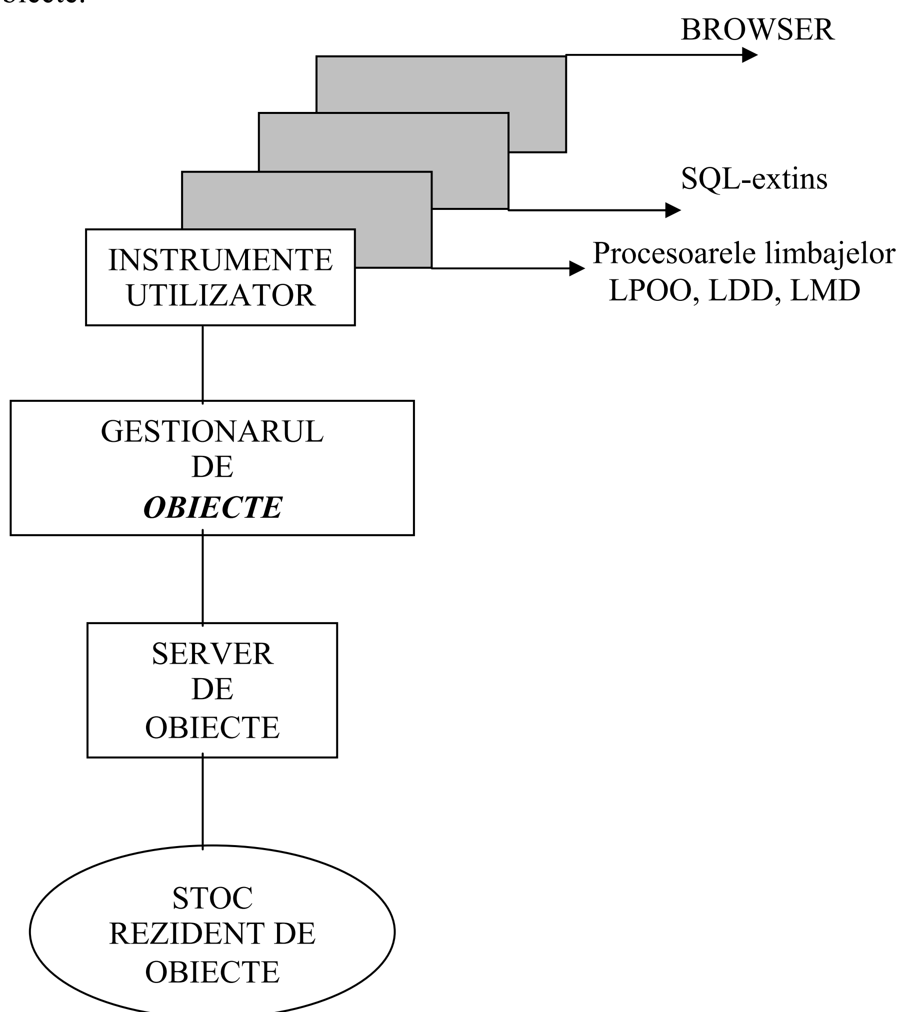


Figura 11.4 Arhitectura generală a unui SGBD-OO

Cerinţele aplicaţiilor de proiectare pot pretinde ca SGBD-ul să existe pe mai multe platforme hard care pot comunica printr-o reţea de calculatoare. Într-un sistem de proiectare CAD, proiectanţii folosind instrumente soft dezvoltate şi lucrând pe diferite platforme, pot accesa date stocate în SGBD-OO. Folosind un instrument CAD, fiecare proiectant poate dirija o sesiune interactivă pentru care este creată o copie individuală a gestionarului de obiecte.

Mai mult, un gestionar de obiecte poate transmite tranzacţii, în mod curent, server-lui de obiecte.

Serverul de obiecte cu care comunică gestionarul de obiecte poate exista de asemenea pe aceeaşi platformă ca şi gestionarul de obiecte sau gestionarul şi server-ul pot exista pe platforme diferite. Similar, o organizare cu un set de activităţi de proiectare intercorelate poate solicita mai mult decât un server de obiecte, fiecare dintre acestea gestionând separat un stoc rezident de obiecte. Pentru a facilita comunicarea între platforme hard separate, pot fi solicitate softuri de comunicaţii reţea. Figura 11.5 ilustrează un sistem reţea cu mai mulţi gestionari şi servere de obiecte.

În vederea asigurării prelucrării distribuite, SGBD-OO trebuie să gestioneze automat accesul la obiecte stocate în platforme hard separate. Legătura dintre gestionarul de obiecte şi server-ul de obiecte trebuie să fie iniţiată explicit de utilizator.

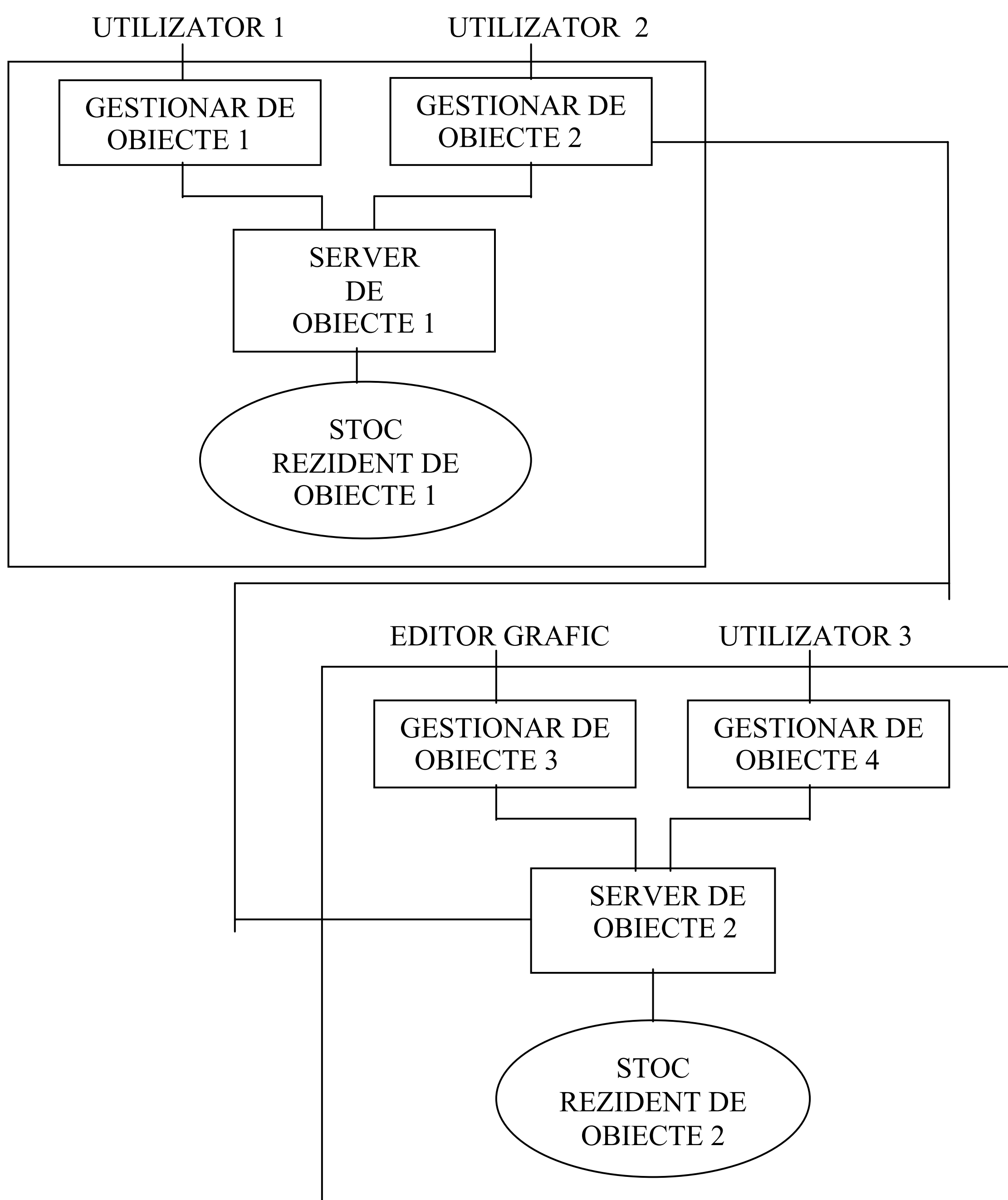


Figura 11.5 Arhitectură reţea de SGBD-OO.

În prezent, SGBD-OO comerciale sunt accesate în primul rând prin limbajele de programare orientate pe obiect, cum ar fi: Smaltalk, Common Lisp şi C++. Interfaţa dintre LP-OO (limbaje de programare orientate pe obiecte) şi SGBD-OO o reprezintă limbajul pentru baze de date. Un SGBD trebuie să asigure un limbaj pentru baze de date pentru a permite definirea şi manipularea schemei bazei de date şi a datelor. Un SGBD-OO trebuie să posede un limbaj pentru baze de date pentru a permite accesul şi manipularea modelului de date obiect şi regăsirea şi actualizarea obiectelor.

Spre deosebire de SGBD-urile convenţionale, limbajul pentru baze de date al SGBD-OO este parte integrantă a LP-OO.

Deoarece LPT-OO au existat înaintea SGBD-OO, numeroase declaraţii ale LDD şi LMD sunt adaptări ale declaraţiilor LP-OO deja existente.

Limbajul pentru baze de date al SGBD-OO constă din următoarele:

- *Limbaj de definire a datelor (LDD).* SGBD-OO trebuie să asigure un LDD pentru definirea schemei. LDD-ul trebuie să permită definirea claselor inclusiv a legăturilor de moştenire şi definirea metodelor care specifică comportamentul obiectului. *Limbaj de manipulare a datelor (LMD)*
- *Limbaj de manipulare a datelor(LMD).* Un SGBD-OO trebuie să asigure un LMD pentru regăsirea, crearea, ştergerea şi actualizarea obiectelor individuale. În cadrul SGBD-OO, acestea se realizează prin mecanismul de transmitere a mesajelor.
- *Limbaj de interogare.* Aproape orice SGBD asigură regăsirea subseturilor unei baze de date prin specificarea condiţiilor logice bazate pe valori, folosind un limbaj de interogare. Modelul de date obiect, permite regăsirea obiectelor individuale prin referirea ID-ului obiectului. Pentru a asigura regăsirea subsetului printre grupul de obiecte, unele SGBDOO-uri (şi unele implementări al LP-OO) posedă un limbaj de interogare. În numeroase implementări acest limbaj se bazează pe transmiterea unui mesaj pentru selectarea şi regăsirea obiectelor.

SGBD-OO pot avea interfeţe cu unul sau mai multe limbaje de programare orientate pe obiect. Definirile de clasă (inclusiv metode) sunt implementate ca obiecte clasă. Declaraţiile LDD de creare a definirilor de clasă sunt mesaje către o clasă generică, sau metaclasă, pentru a crea o instanţă a ei însăşi, de exemplu un obiect clasă. Deci, în legătură cu paradigma obiect, toate operaţiile specificate de limbajul pentru baze de date ale unui SGBD-OO pot fi implementate prin transmiterea de mesaje.