1. GENERALITĂŢI DESPRE BAZE DE DATE

1.1. Introducere

Baza de date este un ansamblu structurat de date coerente, fără redondanţă inutilă, astfel încât acestea pot fi prelucrate eficient de mai mulţi utilizatori într-un mod concurent.

Baza de date este o colecţie de date pe rsistente, care sunt folosite de către sistemele de aplicaţii ale unei anumite „întreprinderi“. Datele din baza de date persistă deoarece, după ce au fost acceptate de către sistemul de gestiune pentru introducerea în baza de date, ele pot fi şterse din bază numai printr-o cerere explicită adresată sistemului de gestiune.

Aici, termenul de „întreprindere“ este un cuvânt generic, utilizat pentru a desemna orice organizaţie independentă, de natură tehnică, comercială, ştiinţifică sau de alt tip. Întreprinderea poate fi, de exemplu, un spital, o bancă, o facultate, o fabrică, un aeroport etc. Fiecare întreprindere are **regulile proprii de funcţionare** şi conţine o **mulţime de date** referitoare la modul său de operare.

Datele din baza de date pot fi atît integrate, cât şi partajate. Noţiunea de **integrat** se referă la faptul că baza de date poate fi considerată ca o unificare a mai multor fişiere, iar prin **partajare** se înţelege că baza de date poate fi partajată concurent între diferiţi utilizatori.

Un **sistem de gestiune a bazelor de date** (SGBD – *Data Base Management System*) este un produs *software* care asigură interacţiunea cu o bază de date, permiţând definirea, consultarea şi actualizarea datelor din baza de date. Toate cererile de acces la baza de date sunt tratate şi controlate de către SGBD.

Organizarea datelor în baze de date constituie o formă de centralizare a acestora. Aceasta implică existenţa unui **administrator al bazei de date** (DBA – *Data Base Administrator*) care este o persoană sau un grup de persoane ce răspund de ansamblul activităţilor (analiză, proiectare, implementare, exploatare, întreţinere etc.) legate de baza de date. Atribuţiile unui administrator pot fi grupate în patru mari categorii: atribuţii de proiectare, atribuţii administrative, atribuţii operative şi atribuţii de coordonare.

Dicţionarul datelor (catalog de sistem), structurat şi administrat ca o bază de date (metabază de date), contine „date despre date“, furnizează descrierea tuturor obiectelor unei baze de date, starea acestor obiecte, diversele constrângeri de securitate şi de integritate etc. Dicţionarul poate fi interogat, la fel, ca orice altă bază de date.

1.2. Gestiunea bazelor de date

Un sistem de baze de date presupune următoarele componente principale, care definesc **arhitectura** acestuia:

- baza de date propriu-zisă în care se memorează datele;
- sistemul de gestiune a bazei de date, care realizează gestionarea şi prelucrarea complexă a datelor;
- un dicţionar al bazei de date (metabaza de date), ce conţine informaţii despre date, structura acestora, statistici, documentaţie;
- mijloace *hardware* (comune sau specializate);
- reglementări administrative destinate bunei funcţionări a sistemului;
- personalul implicat (utilizatori finali, administratorul datelor, administratorul bazei de date, proiectanţi, programatori de aplicaţii).

Se pot identifica patru categorii de persoane implicate în mediul bazelor de date:

- administratorii de date şi baze de date,
- proiectanţii (designeri) de baze de date,
- programatorii de aplicaţii,
- utilizatorii finali.

Administratorul de date (DA) este un manager, nu un tehnician, ce:

- decide care date trebuie stocate în baza de date;
- stabileşte regulile de întreţinere şi de tratare a acestor date după ce sunt stocate. De exemplu, o regulă ar putea fi aceea prin care se stabilesc pentru utilizatori privilegii asupra informaţiilor din baza de date, cu alte cuvinte o anumită politică de securitate a datelor.

Administratorul bazei de date (DBA) este responsabil cu implementarea deciziilor administratorului de date şi cu controlul general al sistemului, la nivel tehnic. El este un profesionist în domeniul IT, care:

- creează baza de date reală;
- implementează elementele tehnice de control;
- este responsabil cu asigurarea funcţionării sistemului la performanţe adecvate, cu monitorizarea performanţelor;
- furnizează diverse servicii tehnice etc.

Proiectanţii de baze de date pot acoperi atât aspectul fizic, cât şi cel logic al concepţiei.

Proiectantul de baze de date care abordează direcţia logică trebuie să posede o cunoaştere completă şi amănunţită a modelului real de proiectat şi a regulilor de funcţionare ale acestuia. Practic, acesta proiectează conceptual baza de date, iar modelul creat este independent de programele de aplicaţii, de limbajele de programare. De asemenea, va proiecta logic baza de date, proiectare care este îndreptată spre un anumit model de date (relaţional, orientat obiect, ierarhic etc.).

Proiectantul de baze de date fizice preia modelul logic de date şi stabileşte cum va fi realizat fizic. Acesta trebuie să cunoască funcţionalităţile SGBD-ului, avantajele şi dezavantajele fiecărei alternative corespunzătoare unei implementări. Practic, se face transpunerea modelului logic într-un set de tabele supuse unor constrângeri, se selectează structuri de stocare şi metode de acces specifice, astfel încât să se asigure performanţe, se iau măsuri privind securitatea datelor.

Utilizatorii finali sunt cei care accesează interactiv baza de date. Aceasta a fost proiectată, implementată, întreţinută pentru a satisface necesităţile informaţionale ale clienţilor. Utilizatorii finali pot fi utilizatori simpli, care nu cunosc nimic despre baza de date, despre SGBD, dar accesează baza prin intermediul unor programe de aplicaţie. În general, această clasă de utilizatori alege anumite opţiuni din meniul aplicaţiei. Există utilizatori finali sofisticaţi, care sunt familiarizaţi cu structura bazei de date. Ei pot utiliza limbaje speciale pentru a exploata posibilităţile oferite de baza de date.

Programatori de aplicaţii sunt responsabili de scrierea programelor aplicaţie ce conferă funcţionalitatea cerută de utilizatorii finali. Programele pot fi scrise în diferite limbaje de programare (*C++*, *PL/SQL*, *Java* etc.).

Cerinţe minimale care se impun unei baze de date:

- asigurarea unei redundanţe minime în date;
- furnizarea în timp util a informaţiilor solicitate (timpul de răspuns la o interogare);
- asigurarea unor costuri minime în prelucrarea şi întreţinerea informaţiei;
- capacitatea de a satisface, cu aceleaşi date, necesităţi informaţionale ale unui număr mare de utilizatori,
- posibilitatea de adaptare la cerinţe noi, răspunsuri la interogări neprevăzute iniţial (flexibilitate);
- exploatarea simultană a datelor de către mai mulţi utilizatori (sincronizare);
- asigurarea securităţii datelor prin mecanisme de protecţie împotriva accesului neautorizat (confidenţialitate);
- înglobarea unor facilităţi destinate validării datelor şi recuperării lor în cazul unor deteriorări accidentale, garantarea (atât cât este posibil) că datele din baza de date sunt corecte (integritate);
- posibilitatea de valorificare a eforturilor anterioare şi anticiparea nevoilor viitoare (compatibilitate şi expandabilitate);
- permisivitatea, prin ierarhizarea datelor după criteriul frecvenţei acceselor, a unor reorganizări (eventual dinamice) care sporesc performanţele bazei.

În cadrul unei baze de date putem vorbi de patru **niveluri de abstractizare şi de percepţie** a datelor: intern, conceptual, logic şi extern. Datele există doar la nivel fizic, iar celelalte trei niveluri reprezintă virtualizări ale acestora.

- Nivelul fizic (intern) este descris de schema fizică a datelor (bit, octet, adresă);
- Nivelul conceptual este descris de schema conceptuală a datelor (articol, înregistrare, zonă) şi reprezintă viziunea programatorilor de sistem asupra datelor;
- Nivelul logic este descris de una din schemele logice posibile ale datelor şi reprezintă viziunea programatorului de aplicaţie asupra datelor;
- Nivelul virtual (extern) reprezintă viziunea utilizatorului final asupra datelor.

Independenţa datelor cuprinde două aspecte fundamentale: o modificare a structurii fizice nu va afecta aplicaţia şi reciproc, modificări ale aplicaţiei vor lăsa nealterată structura fizică de date.

- **Independenţa fizică**: posibilitatea modificării schemei fizice a datelor fără ca aceasta să implice modificarea schemei conceptuale, a schemei logice şi a programelor de aplicaţie. Este vorba despre imunitatea programelor de aplicaţie faţă de modificările modului în care datele sunt stocate fizic şi accesate.
- **Independenţa logică**: posibilitatea modificării schemei conceptuale a datelor fără ca aceasta să implice modificarea schemei logice şi a programelor de aplicaţie. Prin independenţa logică a datelor se urmăreşte a se crea fiecărui utilizator iluzia că este singurul beneficiar al unor date pe care, în realitate, le foloseşte în comun cu alţi utilizatori.

Independenţă faţă de strategiile de acces permite programului să precizeze data pe care doreşte să o acceseze, dar nu modul cum accesează această dată. SGBD-ul va stabili drumul optim de acces la date.

În limbajele de programare uzuale declaraţiile şi instrucţiunile executabile aparţin aceluiaşi limbaj. În lumea bazelor de date, funcţiile de declarare şi de prelucrare a datelor sunt realizate cu ajutorul unor limbaje diferite, numite **limbaje pentru baze de date.**

- Limbaje pentru definirea datelor (LDD – *Data Description Language*). Descrierea concretă a unui LDD este specifică fiecărui sistem de gestiune, dar funcţiile principale sunt aceleaşi. La nivel conceptual, LDD realizează definirea entităţilor şi a atributelor acestora, sunt precizate relaţiile dintre date şi strategiile de acces la ele, sunt stabilite criterii diferenţiate de confidenţialitate şi de validare automată a datelor utilizate.
- Limbaje pentru prelucrarea datelor (LMD – *Data Manipulation Language*). Operaţiile executate în cadrul unei baze de date presupun existenţa unui limbaj specializat, în care comenzile se exprimă prin fraze ce descriu acţiuni asupra bazei. În general, o comandă are următoarea structură: operaţia (calcul aritmetic sau logic, editare, extragere, deschidere-închidere, adăugare, ştergere, căutare, reactualizare etc.), criterii de selecţie, mod de acces (secvenţial, indexat etc.), format de editare. Există limbaje LMD procedurale, care specifică **cum** se obţine rezultatul unei comenzi LMD şi limbaje neprocedurale, care descriu doar datele **ce** vor fi obţinute şi nu modalitatea de obţinere a acestora.

- Limbaje pentru controlul datelor (LCD – *Data Control Language*). Controlul unei baze de date se referă la asigurarea confidenţialităţii şi integrităţii datelor, la salvarea informaţiei în cazul unor defecţiuni, la obţinerea unor performanţe, la rezolvarea unor probleme de concurenţă.

Limbajele universale nu se utilizează frecvent pentru gestionarea unei baze de date, dar există această posibilitate. De exemplu, sistemul Oracle este dotat cu precompilatoare (*C*, *Pascal*, *ADA*, *Cobol*, *PL/1*, *Fortran*) care ajută la incorporarea de instrucţiuni *SQL* sau blocuri *PL/SQL* în programe scrise în alte limbaje, de nivel înalt, numite limbaje gazdă.

Sistemul de gestiune a bazelor de date interacţionează cu programele de aplicaţie ale utilizatorului şi cu baza de date, oferind o mulţime de facilităţi. Realizarea optimă a acestor facilităţi este asigurată de **obiectivele** fundamentale ale unui sistem de gestiune. Câteva dintre aceste obiective vor fi enumerate în continuare.

- Independenţa fizică. Obiectivul esenţial este acela de a permite realizarea independenţei structurilor de stocare în raport cu structurile de date din lumea reală. Se defineşte mulţimea de date indiferent de forma acesteia din lumea reală, ţinând seama doar de a realiza un acces simplu la date şi de a obţine anumite performanţe.
- Independenţa logică. Grupul de lucru care exploatează baza de date poate să utilizeze diferite informaţii de bază (nu aceleaşi), pentru a-şi construi entităţi şi relaţii. Fiecare grup de lucru poate să cunoască doar o parte a semanticii datelor, să vadă doar o submulţime a datelor şi numai sub forma în care le doreşte. Această independenţă asigură imunitatea schemelor externe faţă de modificările făcute în schema conceptuală.
- Prelucrarea datelor de către neinformaticieni. Neinformaticienii văd datele independent de implementarea lor şi pot exploata aceste date prin intermediul unui sistem de meniuri oferit de aplicaţia pe care o exploatează.
- Administrarea centralizată a datelor. Administrarea datelor presupune definirea structurii datelor şi a modului de stocare a acestora. Administrarea este în general centralizată şi permite o organizare coerentă şi eficace a informaţiei.
- Coerenţa datelor. Informaţia trebuie să satisfacă constrângeri statice sau dinamice, locale sau generale.

- Neredundanţa datelor. Fiecare aplicaţie posedă datele sale proprii şi aceasta conduce la numeroase dubluri. De asemenea, organizarea nejudicioasă a relaţiilor poate să genereze redundanţă în date. Administrarea coerentă a datelor trebuie să asigure neduplicarea fizică a datelor. Totuşi, nu sunt excluse nici cazurile în care, pentru a realiza performanţe referitoare la timpul de acces la date şi răspuns la solicitările utilizatorilor, să se accepte o anumită redundanţă a datelor.
- Partajabilitatea datelor. Aceasta permite ca aplicaţiile să partajeze datele din baza de date în timp şi simultan. O aplicaţie poate folosi date ca şi cum ar fi singura care le utilizează, fără a şti că altă aplicaţie, concurent, le poate modifica.
- Securitatea şi confidenţialitatea datelor. Datele trebuie protejate de un acces neautorizat sau rău intenţionat. Există mecanisme care permit identificarea şi autentificarea utilizatorilor şi există proceduri de acces autorizat care depind de date şi de utilizator. Sistemul de gestiune trebuie să asigure securitatea fizică şi logică a informaţiei şi să garanteze că numai utilizatorii autorizaţi pot efectua operaţii corecte asupra bazei de date.

Sistemele de gestiune a bazelor de date au, din nefericire, şi **dezavantaje** dintre care se remarcă:

- complexitatea şi dimensiunea sistemelor pot să crească considerabil, datorită necesităţii extinderii funcţionalităţilor sistemului;
- costul, care variază în funcţie de mediu şi funcţionalitatea oferită, la care se adugă cheltuieli periodice de întreţinere;
- costuri adiţionale pentru elemente de *hardware*;
- costul conversiei aplicaţiilor existente, necesară pentru ca acestea să poată funcţiona în noua configuraţie *hardware* şi *software*;
- impactul unei defecţiuni asupra aplicaţiilor, bazei de date sau sistemului de gestiune.

Structura unui sistem de gestiune a bazelor de date este de complexitate variabilă, iar nivelul real de funcţionalitate diferă de la produs la produs. În orice moment apar noi necesităţi, care cer o nouă funcţionalitate, astfel încât aceasta nu va putea deveni niciodată statică. În general, un SGBD trebuie să includă cel puţin cinci clase de module:

- programe de gestiune a bazei de date (PGBD), care realizează accesul fizic la date ca urmare a unei comenzi;

- module pentru tratarea limbajului de definire a datelor, ce permit traducerea unor informaţii (care realizează descrierea datelor, a legăturilor logice dintre acestea şi a constrângerilor la care sunt supuse), în obiecte ce pot fi apoi exploatate în manieră procedurală sau neprocedurală;
- module pentru tratarea limbajului de prelucrare a datelor (interpretativ, compilativ, generare de programe), care permit utilizatorilor inserarea, ştergerea, reactualizarea sau consultarea informaţiei dintr-o bază de date;
- module utilitare, care asigură întreţinerea, prelucrarea, exploatarea corectă şi uşoară a bazei de date;
- module de control, care permit controlul programelor de aplicaţie, asigurarea confidenţialităţii şi integrităţii datelor, rezolvarea unor probleme de concurenţă, recuperarea informaţiei în cazul unor avarii sau defecţiuni *hardware* sau *software* etc.

Modulele PGBD asigură accesul fizic la date ca urmare a unei comenzi. Cum lucrează aceste module?

- găsesc descrierea datelor implicate în comandă;
- identifică datele şi tipul acestora;
- identifică informaţii ce permit accesul la structurile fizice de stocare (fişiere, volume etc.);
- verifică dacă datele sunt disponibile;
- extrag datele, fac conversiile, plasează datele în spaţiul de memorie al utilizatorului;
- transmit informaţii de control necesare execuţiei comenzii, în spaţiul de memorie al utilizatorului;
- transferă controlul programului de aplicaţie.

Prin urmare, din punct de vedere conceptual:

- utilizatorul lansează o cerere de acces;
- SGBD-ul acceptă cererea şi o analizează;
- SGBD-ul inspectează pe rând, schema internă corespunzatoare utilizatorului, schema conceptuală, definiţia structurii de stocare şi corespondenţele corespunzătoare;
- SGBD-ul execută operaţiile necesare în baza de date stocată.

1.3. Arhitectura sistemelor de gestiune a bazelor de date

Asigurarea independenţei fizice şi logice a datelor impune adoptarea unei arhitecturi de baze de date organizată pe trei niveluri:

- **nivelul intern** (baza de date fizică);
- **nivelul conceptual** (modelul conceptual, schema conceptuală);
- **nivelul extern** (modelul extern, subschema, vizualizarea).

Nivelul central este **nivelul conceptual**. Acesta corespunde structurii canonice a datelor ce caracterizează procesul de modelat, adică structura semantică a datelor fără implementarea pe calculator. Schema conceptuală permite definirea tipurilor de date ce caracterizează proprietăţile elementare ale entităţilor, definirea tipurilor de date compuse care permit regruparea atributelor pentru a descrie entităţile modelului şi legăturile între aceste entităţi, definirea regulilor pe care trebuie să le respecte datele etc.

Nivelul intern corespunde structurii interne de stocare 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, drumurile de acces la aceste articole etc.

La nivel conceptual sau intern, schemele descriu o bază de date. La **nivel extern** schemele descriu doar o parte din date care prezintă interes pentru un utilizator sau un grup de utilizatori. Schema externă reprezintă o descriere a unei părţi a bazei de date ce corespunde viziunii unui program sau unui utilizator. Modelul extern folosit este dependent de limbajul utilizat pentru prelucrarea bazei de date. Schema externă permite asigurarea unei securităţi a datelor. Un grup de lucru va accesa doar datele descrise în schema sa externă, iar restul datelor sunt protejate împotriva accesului neautorizat sau rău intenţionat.

Pentru o bază de date particulară există o singură schemă internă, o singură schemă conceptuală, dar există mai multe scheme externe.

În afară de aceste trei niveluri, arhitectura presupune şi anumite corespondenţe dintre acestea:

- corespondenţa conceptual-intern defineşte relaţia dintre nivelul conceptual şi baza de date stocată, specificând modul în care înregistrările şi câmpurile conceptuale sunt reprezentate la nivel intern;
- corespondenţa extern-conceptual defineşte relaţia dintre o anumită vizualizare externă şi nivelul (vizualizarea) conceptual, reprezentând cheia independenţei logice de date;

- corespondenţa extern-extern permite definirea unor vizualizări externe în funcţie de altele, fără a necesita o definiţie explicită a corespondenţei cu nivelul conceptual.

Arhitectura funcţională de referinţă propusă de grupul de lucru ANSI/X3/SPARC este axată pe dicţionarul datelor şi cuprinde două părţi:

- prima, permite descrierea datelor (compoziţia dicţionarului datelor);
- a doua, permite prelucrarea datelor (interogarea şi reactualizarea bazei de date).

În fiecare parte se regăsesc cele trei niveluri: intern, conceptual şi extern. Acestea nu sunt neapărat distincte pentru orice SGBD.

Interfeţele numerotate din figura 1.1, ce descriu arhitectura de referinţă a unui SGBD, corespund următoarelor transformări:

a) Limbaj de descriere a datelor conceptuale, format sursă – permite administratorului întreprinderii să definească schema conceptuală, format sursă.

b) Limbaj de descriere a datelor conceptuale, format obiect – se obţine din compilarea celui precedent şi permite aranjarea schemei obiect în dicţionarul datelor.

c) Limbaj de descriere a datelor conceptuale, format editare – permite administratorilor aplicaţiilor şi a bazelor să consulte schema conceptuală pentru a defini reguli de corespondenţă.

d) Limbaje de descriere a datelor externe, format sursă – permit administratorilor aplicaţiilor să definească scheme externe corespunzând schemei conceptuale. Deoarece sistemele de gestiune pot suporta mai multe modele externe, pot exista mai multe limbaje de descriere a datelor externe.

e) Limbaje de descriere a datelor externe, format obiect – corespund formelor compilate ale celor precedente şi permit aranjarea schemelor externe (obiect) în dicţionarul datelor.

f) Limbaj de descriere a datelor interne, format sursă – permite administratorului bazei de date să definească schema internă şi regulile de corespondenţă cu schema conceptuală.

g) Limbaj de descriere a datelor interne, format obiect – corespunde formei compilate a celui precedent şi permite aranjarea schemei interne (obiect) în dicţionarul datelor.

h) Limbaje de prelucrare a datelor externe, format sursă – permit programatorilor de aplicaţii sau utilizatorilor neinformaticieni să manipuleze date externe (*view*).

i) Limbaje de prelucrare a datelor externe, format obiect – corespund formelor compilate ale celor precedente.

j) Limbaj de prelucrare a datelor conceptuale, format obiect – produs de procesorul de transformare extern/ conceptual pentru a manipula datele externe.

k) Limbaj de prelucrare a datelor interne, format obiect –produs de procesorul de transformare conceptual/intern pentru a gestiona datele interne.

l) Limbaj de stocare a datelor, format obiect – corespunde interfeţei cu sistemul de stocare a datelor.

m) Interfaţa cu memoria secundară – permite efectuarea de intrări/ieşiri în/din unitatea de memorie secundară.

n) Interfaţa de acces la dicţionarul datelor – permite diverselor procesoare de transformare să acceseze scheme obiect şi reguli de corespondenţă.

Procesoarele din figura 1.1 au următoarele funcţii:

- procesorul schemei conceptuale compilează schema conceptuală şi dacă nu sunt erori depune schema compilată în dicţionarul datelor;
- procesorul schemei externe compilează schemele externe şi regulile de corespondenţă externă şi dacă nu sunt erori aranjează schema compilată şi regulile de corespondenţă în dicţionarul datelor;
- procesorul schemei interne are rol similar pentru schema internă;
- procesorul de transformare extern/conceptual transformă manipulările externe în manipulări conceptuale şi invers, datele conceptuale în date externe;
- procesorul de transformare conceptual/intern transformă manipulările conceptuale în manipulări interne şi invers, datele interne în date conceptuale;
- procesorul de transformare intern/stocare transformă manipulările interne în primitive ale sistemului de stocare şi invers, eliberează datele stocate într-un format corespunzător schemei interne.

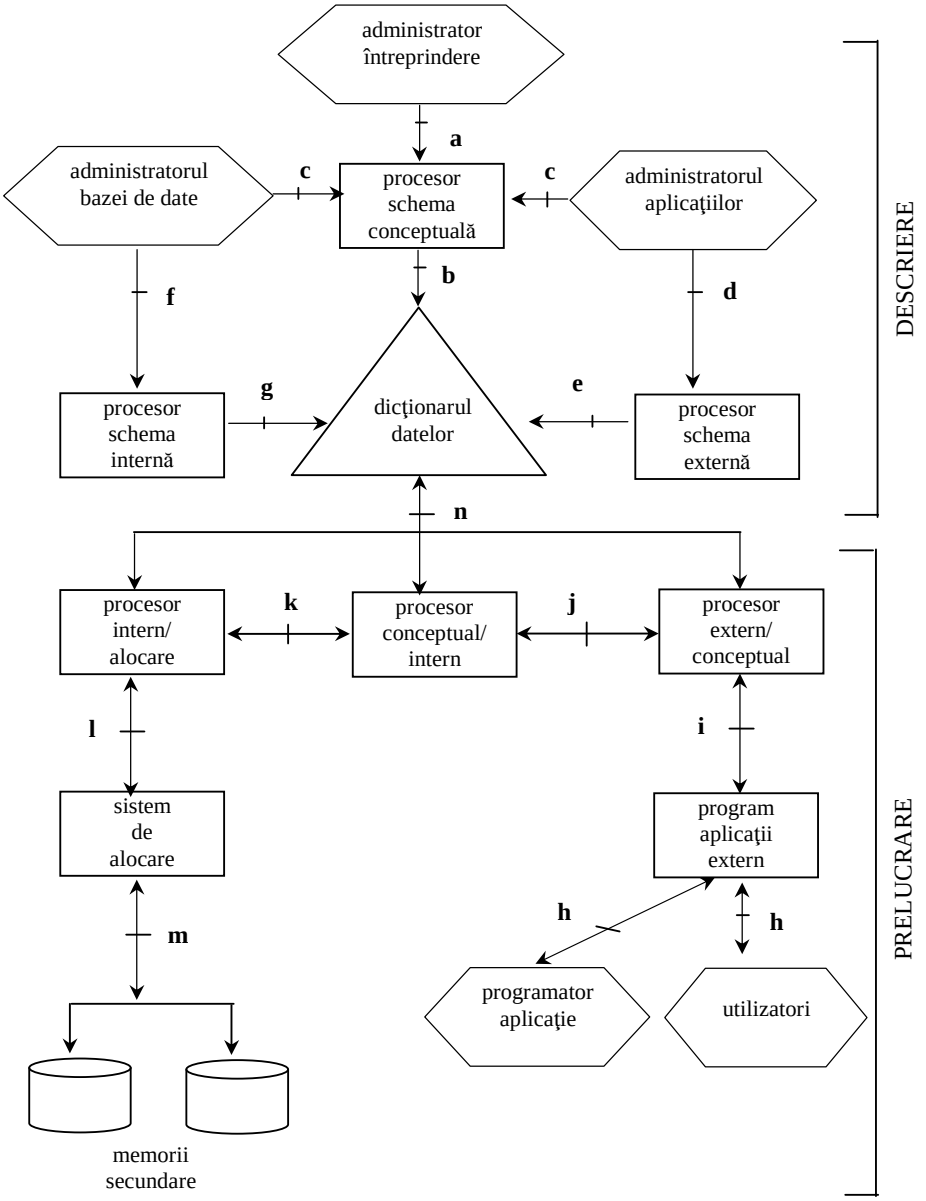


Fig. 1.1. Arhitectura de referinţă a unui SGBD.

Gardarin a propus o arhitectură funcţională apropiată de arhitectura sistemelor de gestiune actuale care are la bază doar două niveluri:

- schema, care corespunde integrării schemelor interne şi conceptuale;
- vizualizarea, care este o schemă externă.

Sistemul de gestiune gestionează un **dicţionar de date** care este alimentat prin comenzi de definire a schemei şi prin comenzi de definire a vizualizărilor.

Aceste comenzi, precum şi cererile de prelucrare sunt analizate şi tratate de un procesor numit **analizor**. Analizorul realizează analiza sintactică şi semantică a cererii şi o traduce în format intern. O cerere în format intern care face referinţă la o vizualizare este tradusă în una sau mai multe cereri care fac referinţă la obiecte ce există în baza de date (modificarea cererilor).

În cadrul acestei arhitecturi există un procesor, numit **translator**, care realizează modificarea cererilor, asigură controlul drepturilor de acces şi controlul integrităţii în cazul reactualizărilor.

Componenta cheie a sistemului de gestiune este procesorul **optimizor** care elaborează un plan de acces optim pentru a trata cererea. Acest procesor descompune cererea în operaţii de acces elementare şi alege o ordine de execuţie optimală. De asemenea, evaluează costul planului de acces înaintea execuţiei sale.

Planul de acces ales şi elaborat de optimizor este executat de un procesor numit **executor**. La acest nivel este gestionat controlul concurenţei.

1.4. Evoluţia bazelor de date

Istoria bazelor de date şi a sistemelor de gestiune a bazelor de date poate fi rezumată în trei generaţii:

- sisteme ierarhice şi reţea,
- sisteme relaţionale,
- sisteme avansate (orientate obiect, relaţionale orientate obiect, deductive, distribuite, multimedia, multibaze, active, temporale, decizionale, magazii de date etc.).

Baze de date ierarhice şi reţea

Pentru modelele ierarhice şi reţea, datele sunt reprezentate la nivel de articol prin legături ierarhice (arbore) sau de tip graf. Slaba independenţă fizică a datelor complică administrarea şi prelucrarea acestora. Limbajul de prelucrare a datelor impune programatorului să specifice drumurile de acces la date.

Baze de date relaţionale

A doua generaţie de SGBD-uri este legată de apariţia modelelor relaţionale (1970), care tratează entităţile ca nişte relaţii. Piaţa actuală de baze de date este acoperită în majoritate de sisteme **relaţionale**. Bazele de date relaţionale sunt caracterizate de:

- structuri de date simple, intuitive,
- inexistenţa pointerilor vizibili pentru utilizator,
- constrângeri de integritate,
- operatori aplicaţi relaţiilor care permit definirea, căutarea şi reactualizarea datelor.

Dezvoltarea unei aplicaţii riguroase utilizand o bază de date relaţionale necesită cunoaşterea a trei niveluri de instrumente, eterogene din punct de vedere conceptual:

- nivelul instrumentelor grafice (interfaţa);
- nivelul aplicaţie, cu limbajele sale de dezvoltare;
- nivelul SGBD, cu standardul *SQL* (*Structured Query Language*) ce permite definirea, prelucrarea şi controlul bazei de date.

Baze de date orientate obiect

Bazele de date relaţionale nu folosesc însă obiecte complexe şi dinamice, nu realizează gestiunea datelor distribuite şi nici gestiunea cunoştinţelor. A treia generaţie de SGBD-uri, sistemele avansate, încearcă să depăşească aceste limite ale sistemului relaţional.

Suportul obiectelor complexe şi dinamice şi prelucrarea acestora este dificilă pentru sistemele relaţionale, deoarece tipul datelor este limitat la câteva domenii alfanumerice, iar structura datelor este simplă. Sistemele relaţionale nu modelează obiecte complexe ca grafuri, liste etc. Un obiect complex poate să fie descompus în relaţii, dar apar dificultăţi atât la descompunerea, cât şi la refacerea acestuia prin compunere. De asemenea, limbajele modelului relaţional permit prelucrarea cu dificultate a obiectelor complexe.

Un sistem relaţional nu suportă obiecte dinamice care incorporează atât partea de date (informaţii) efective, cât şi o parte relativă la tratarea acestora.

Îmbinarea tehnicii limbajelor orientate obiect cu a bazelor de date a permis realizarea bazelor de date orientate obiect. Acestea permit organizarea coerentă a obiectelor partajate între utilizatori concurenţi. Sistemele de gestiune de baze de date orientate obiect (SGBDOO) prezintă o serie de avantaje:

- realizează o modelare superioară a informaţiei,
- furnizează posibilităţi superioare de deducţie (ierarhie de clase, moştenire),
- permit luarea în considerare a aspectelor dinamice şi integrarea descrierii structurale şi comportamentale,
- asigură îmbunătăţirea interfeţei cu utilizatorul.

Cu toate avantajele incontestabile oferite de SGBDOO-uri, impunerea lor pe piaţa bazelor de date nu a fost uşoară. Câteva motivaţii a acestei situaţii:

- absenţa unei fundamentări teoretice face imposibilă definirea unui SGBDOO de referinţă;
- gestiunea obiectelor complexe este mai dificilă decât accesul la relaţii prin cereri *SQL*;
- utilizatorii au investit sume uriaşe în sistemele relaţionale şi nu le pot abandona cu uşurinţă. Trecerea la noua tehnologie orientată obiect implică investiţii mari şi nu păstrează aproape nimic din vechile soluţii.

Baze de date relaţionale orientate obiect

Simplitatea modelului relaţional, combinată cu puterea tehnologiei orientate obiect a generat un domeniu nou şi promiţător în lumea bazelor de date, şi anume bazele de date relaţionale orientate obiect.

Construcţia unui sistem de gestiune de baze de date relaţionale orientate obiect (SGBDROO) trebuie să pornească de la cele existente. Aceasta se poate realiza în două moduri: dezvoltând un sistem relaţional prin adăugarea caracteristicilor obiectuale necesare sau pornind de la un sistem orientat obiect şi adăugând caracteristicile relaţionale.

Baze de date deductive

O relaţie este o mulţime de înregistrări ce reprezintă fapte. Cunoştinţele sunt aserţiuni generale şi abstracte asupra faptelor. Cunoştinţele permit să raţionezi, ceea ce permite deducerea de noi fapte, plecând de la fapte cunoscute. Un SGBD relaţional suportă o formă limitată de cunoştinţe, şi anume constrângerile de integritate, iar restul trebuie integrate în programele de aplicaţie. Aceasta generează probleme deoarece cunoştinţele trebuie codificate în programe şi apare imposibilitatea de a partaja cunoştinţe între utilizatori. Totul se complică dacă există un volum mare de fapte.

Bazele de date deductive, utilizând programarea logică, gestionează cunoştinţe relativ la baze de date care, în general, sunt relaţionale. Bazele de date deductive permit deducerea de noi informaţii, plecând de la informaţiile stocate în baza de date. Un SGBD deductiv posedă:

- un limbaj de definire a datelor care permite definirea structurii predicatelor sub formă de relaţii şi constrângeri de integritate asociate;
- un limbaj de prelucrare a datelor care permite efectuarea reactualizărilor asupra datelor şi formularea unor cereri;
- un limbaj de reguli de deducţie care permite ca, plecând cu predicatele definite anterior, să se specifice cum pot fi construite predicate derivate.

Baze de date distribuite

Un sistem distribuit este un ansamblu de maşini ce sunt interconectate printr-o reţea de comunicaţie şi utilizate într-un scop global. Administrarea şi prelucrarea datelor distribuite, situate pe diferite calculatoare şi exploatate de sisteme eterogene este obiectivul fundamental al bazelor de date distribuite.

Bazele de date distribuite sunt sisteme de baze de date cooperante care rezidă pe maşini diferite, în locuri diferite. Această mulţime de baze de date este exploatată de utilizator ca şi cum ar fi o singură bază de date. Programul de aplicaţie care exploatează o bază de date distribuite poate avea acces la date rezidente pe mai multe maşini, fără ca programatorul să cunoască localizarea datelor.

Modelul relaţional a rămas instrumentul principal prin care se realizează prelucrarea datelor distribuite.

Câteva dintre argumentele pentru a justifica această afirmaţie sunt:

- bazele relaţionale oferă flexibilitate de descompunere în vederea distribuirii;
- operatorii relaţionali pot fi folosiţi pentru combinaţii dinamice ale informaţiilor descentralizate;
- limbajele sistemelor relaţionale sunt concise şi asigură o economie considerabilă a transmiterii datelor. Ele fac posibil, pentru un nod oarecare al reţelei, să analizeze intenţia unei tranzacţii, să o descompună rapid în componente ce pot fi realizate local şi componente ce pot fi transportate altor noduri.

Calculatoare şi maşini baze de date

Soluţia pentru a descentraliza prelucrarea datelor, în scopul evitării saturării memoriei şi a procesoarelor calculatorului central, a fost apariţia calculatoarelor baze de date şi a maşinilor baze de date. Descentralizarea presupune transferarea unei părţi din funcţiile unui SGBD către un calculator periferic (calculator *backend*) adică deplasarea algoritmilor de căutare şi a celor de actualizare a datelor mai aproape de memoria secundară. Acest calculator periferic permite utilizarea optimă a resurselor şi realizarea paralelismului în tratarea cererilor de informaţii.

Calculatorul periferic poate fi un calculator clasic, dar cu un *software* specific de SGBD (calculator bază de date) sau poate fi o maşină cu *hardware* specializat în gestiunea bazelor de date (maşină bază de date). Maşinile baze de date sunt înzestrate cu arhitecturi paralele special adaptate pentru gestionarea unui volum mare de date. Tratarea paralelă a cererilor permite reducerea timpului total de execuţie a acestora.

O execuţie în paralel solicită, fie descompunerea unui program în module, care pot fi executate în paralel (descompunere funcţională), fie descompunerea datelor în subgrupe, astfel încât toate procesoarele să execute acelaşi lucru, dar pe date diferite. Performanţele tratării paralele depind de modul în care sunt efectuate descompunerile.

Multibaze de date

Diferite departamente ale unei organizaţii mai mari pot folosi diferite sisteme de gestiune. Cu toate că fiecare sistem este dezvoltat pentru a satisface nevoile propriului său departament, informaţiile cu care lucrează pot fi utile şi altor departamente. De aceea, pentru ca organizaţia să funcţioneze bine, trebuie să existe o modalitate globală da a vedea datele din fiecare sistem. Există două caracteristici ale unor astfel de sisteme care fac acceasarea datelor în acest mediu integrat greoaie, uneori chiar imposibilă:

- autonomie – fiecare SGBD are o autonomie completă, ceea ce înseamnă că fiecare *manager* are control deplin asupra sistemului;
- eterogenitate – sistemele pot opera pe diferite platforme, cu diferite modele de date şi limbajele de interogare.

Una dintre soluţiile folosite pentru a depăşi dificultăţile întâmpinate în respectarea autonomiei şi a eterogenităţii este utilizarea sistemelor multibaze de date.

Un sistem multibaze de date (SMB) este alcătuit din mai multe sisteme de baze de date privite integrat, în care se construiesc una sau mai multe scheme globale pe baza schemelor fiecărei baze de date componente, astfel încât să se poată realiza accesul uniform şi integrat la fiecare din bazele de date componente. Fiecare schemă globală este construită pe baza unui model particular de date. De exemplu, se poate construi o schemă globală ce are la bază modelul relaţional pentru utilizatorii care sunt familiarizaţi cu acest model, dar se poate construi o schemă globală bazată pe modelul orientat obiect pentru utilizatorii bazelor de date orientate obiect.

Pentru o schemă globală dată, un sistem multibaze de date constă din sistemele componente împreună cu un sistem *front-end*, care suportă un singur model de date şi un singur limbaj de interogare. Principalele sarcini ale sistemului *front-end* sunt gestionarea schemei globale şi procesarea cererilor globale.

Un avantaj major al acestui model, faţă de altele, este faptul că o singură interogare poate accesa date din mai multe baze de date într-un mod integrat, fără să afecteze nici o aplicaţie care este scrisă utilizând una dintre bazele de date componente.

Baze de date cu suport decizional

Sistemele informatice, în particular bazele de date, au ajuns la maturitate. Marile companii au acumulat o mare cantitate de informaţii din domeniul lor de activitate, pe care le păstrează în tabele istorice şi sunt nefolositoare sistemelor operaţionale ale companiei, care funcţionează cu date curente. Analizate, aceste date ar putea oferi informaţii despre tendinţe şi evoluţii care ar putea interesa compania. Pentru a putea analiza aceste mari cantităţi de date este nevoie de tehnologii şi instrumente speciale.

Ideea de a analiza colecţii de date provenind din sistemele operaţionale ale companiei sau din surse externe pentru a le folosi ca suport în procesul de decizie nu aparţine ultimului deceniu, dar baze de date care să funcţioneze eficient după aceste criterii au fost studiate şi implementate în ultimii ani. Principalul scop al acestor baze de date a fost de a întâmpina nevoile sistemelor operaţionale, a căror natură este inerent tranzacţională.

Sistemele tranzacţionale sunt interesate, în primul rând, să controleze la un moment dat o singură tranzacţie. De exemplu, într-un sistem bancar, atunci când clientul face un depozit, sistemul operaţional bancar este responsabil de a înregistra tranzacţia într-un tabel al tranzacţiilor şi de a creşte nivelul curent al contului clientului, stocat în alt tabel.

Un sistem operaţional tipic operează cu evenimente predefinite şi, datorită naturii lor, necesită acces rapid la date. Uzual, fiecare tranzacţie operează cu cantităţi mici de date.

De-a lungul timpului, nevoile sistemelor operaţionale nu se schimbă mult. Aplicaţia care înregistrează tranzacţia, ca şi cea care controlează accesul utilizatorului la informaţie (partea de raportare a sistemului bancar), nu se modifică prea mult. În acest tip de sistem, informaţia necesară în momentul în care un client iniţiază o tranzacţie trebuie sa fie actuală. Înainte ca o bancă să aprobe un împrumut este nevoie să se asigure de situaţia financiară stabilă a clientului în acel moment, şi nu cu un an înainte.

În ultimii ani s-au pus la punct principii şi tehnologii noi care să servească procesului de analiză şi administrare a datelor. O bază de date optimizată în acest scop defineşte o ***Data Warehouse*** (magazie de date), iar principiul pe care îl urmează este cunoscut sub numele de procesare analitică (OLAP – *On Line Analytical Processing*). Spre deosebire de acesta, principiul pe care se bazează sistemele tranzacţionale a fost numit procesare tranzacţională (OLTP – *On Line Transactional Processing*).

Aplicaţiile unei *Data Warehouse* trebuie să ofere răspunsuri unor întrebări de tipul: „Care zi din săptămână este cea mai aglomerată?“ „Ce clienţi, cu care avem relaţii intense, nu au beneficiat de reduceri de preţuri?“. O caracteristică a bazelor de date analitice este că interogările la care acestea trebuie să răspundă sunt *ad-hoc*, nu sunt predefinite, iar baza de date trebuie optimizată astfel încât să fie capabilă să răspundă la orice fel de întrebare care poate implica mai multe tabele.

În această abordare, organele generale de decizie necesită accesul la toate datele organizaţiei, oriunde s-ar afla acestea. Pentru o analiză corespunzătoare a organizaţiei, afacerilor, cerinţelor, tendinţelor este necesară nu numai accesarea valorilor curente din baza de date, ci şi a datelor istorice. Prin urmare, pentru a facilita acest tip specific de analiză a informaţiei a fost creată magazia de date, care conţine **informaţii extrase din diverse surse, întreţinute de diverse unităţi operative, împreună cu istoricul şi rezumatul tranzacţiilor**.

Sursele de date pentru o magazie cuprind:

- date operaţionale, păstrate în baze de date ierarhice, de prima generaţie;
- date departamentale, păstrate în sisteme de fişiere patentate;

- date cu caracter personal, păstrate pe staţii de lucru şi servere personale;
- sisteme externe (baze de date comerciale, Internet etc.)

Data warehouse este o colecţie de date:

- orientate spre subiect (principalele subiecte ale modelului sunt clienţii, produsele, vânzările, în loc de domeniile de activitate),
- nevolatile (datele nu sunt reactualizate, înlocuite în timp real, ci sunt adăugate ca un supliment al bazei),
- integrate (transpunerea datelor provenite din diverse surse de informaţii se face într-un format consistent),
- variabile în timp (concentrarea depozitului de date se face asupra schimbărilor care au loc în timp).

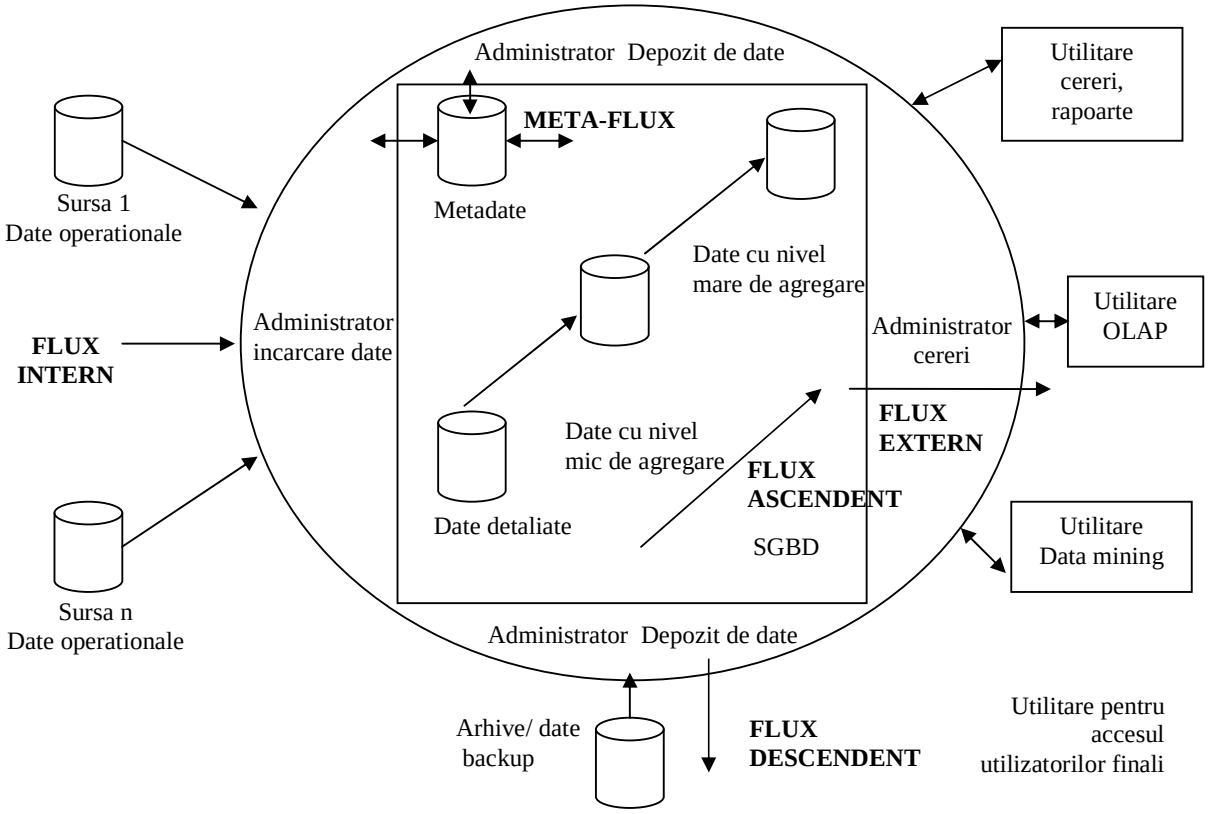


Fig 1.2. Arhitectura unui depozit de date

Înmagazinarea datelor se concentrează asupra gestionării a cinci fluxuri de informaţii:

- fluxul intern, care reprezintă procesele asociate extragerii şi încărcării datelor din fişierele sursă în magazia de date;
- fluxul ascendent, care reprezintă procesele asociate adăugării de valoare datelor din magazie, cu ajutorul împachetării şi distribuirii;
- fluxul descendent, care reprezintă procesele asociate arhivării, salvării, refacerii datelor din magazie;
- fluxul extern, care reprezintă procesele asociate punerii la dispoziţie a datelor pentru utilizatorii finali;
- meta-fluxul, care reprezintă procesele asociate gestionării meta-datelor (date despre date).

În arhitectura depozitului de date intervin câteva componente specifice acestei structuri.

- Administratorul pentru încărcarea datelor (componenta *front-end*) realizează toate operaţiile asociate cu obţinerea (extragerea) şi încărcarea datelor operaţionale într-un depozit de date.
- Administratorul depozitului de date realizează toate operaţiile legate de administrarea datelor din depozit. Operaţiile realizate de componenta de administrare a depozitului de date includ: analiza datelor pentru a asigura consistenţa acestora; transformarea şi mutarea datelor sursă din structurile temporare de stocare în tabelele depozitului de date; crearea de indecşi şi vizualizări asupra tabelelor de bază; generarea denormalizării (dacă este necesar); generarea agregărilor; crearea arhivelor şi a *backup*-urilor.
- Administratorul cererilor (componenta *back-end*) realizează toate operaţiile legate de administrarea cererilor utilizator. Această componentă este construită folosind utilitarele de acces la date disponibile utilizatorilor finali, utilitarele de monitorizare a depozitului de date, facilităţile oferite de sistemul de baze de date şi programele personalizate.
- În zona ce include date agregate sunt stocate toate agregările predefinite de date, pe diferite niveluri. Scopul, menţinerii acestora, este de a mări performanţa cererilor care necesită agregări. Datele agregate sunt actualizate permanent, pe măsură ce sunt încărcate noi informaţii în depozit.

- Scopul principal al depozitelor de date este de a oferi informaţii necesare utilizatorilor pentru luarea deciziilor strategice de marketing. Aceşti utilizatori interacţionează cu depozitul de date prin diferite utilitare de acces (utilitare pentru rapoarte şi cereri, utilitare pentru dezvoltarea aplicaţiilor, utilitare pentru procesarea analitică *on-line* (OLAP), utilitare *data mining*) etc.

Instrumentele de acces pentru utilizatorii finali ai magaziilor de date:

- prelucrarea analitică *on-line*;
- extensiile limbajului *SQL*;
- instrumentele de extragere a datelor.

Prelucrarea analitică *on-line* (OLAP) reprezintă sinteza, analiza şi consolidarea dinamică a unor volume mari de date multidimensionale. Serverele de baze de date OLAP utilizează structuri multidimensionale pentru stocarea datelor şi a relaţiilor dintre date. Aceste structuri pot fi vizualizate prin cuburi de date şi cuburi în cadrul cuburilor etc. Fiecare latură a cubului reprezintă o dimensiune. Serverele de baze de date OLAP multidimensionale acceptă operaţiile analitice uzuale: consolidarea (gruparea), parcurgerea în jos (inversul consolidării), tranşarea, tăierea. OLAP necesită o modalitate de agregare a datelor conform mai multor grupări diferite, în număr foarte mare, iar utilizatorii trebuie să le aibă în vedere pe toate.

Instrumentele OLAP presupun organizarea informaţiei într-un model multidimensional care este susţinut de o bază de date:

- multidimensională (MOLAP), în care datele sunt stocate conceptual în celulele unui tablou multidimensional;
- relaţională (ROLAP), proiectată pentru a permite interogări multidimensionale.

În acest context, a devenit o necesitate extinderea limbajului *SQL* prin operaţii puternice, necesare pentru rezolvarea noului tip de abordare. Au fost introduse noi funcţii numerice (limita inferioară, limita superioară etc.), noi funcţii statistice (distribuţie, distribuţie inversă, corelaţie etc.), noi operatori de agregare, extensii ale clauzei *GROUP BY* etc.

De exemplu, *RISQL* (*Red Brick Intelligent SQL*), proiectat pentru analiştii din domeniul afacerilor, permite: ordonare după rang (pe diferite niveluri - de exemplu, gruparea filialelor în trei categorii pe baza venitului generat în anul precedent), partajarea pieţei, compararea anului curent cu cel precedent etc.

O altă problemă esenţială este extragerea datelor şi utilizarea acestora pentru luarea de decizii cruciale în domeniul afacerilor. Descoperirea unor noi corelaţii, tipare, tendinţe, prin extragerea unor cantităţi mari de date folosind strategia inteligenţei artificiale este una din modalităţile de rezolvare.

Extragerea datelor presupune capacitatea de a construi modele mai degrabă previzibile, decât retrospective. Modelarea predictivă utilizează informaţii pentru a forma un model al caracteristicilor importante ale unui fenomen.

Tehnicile asociate operaţiilor fundamentale de extragere sunt:

- modelarea predictivă (clasificarea cu ajutorul unei reţele neurale sau al unei inducţii de tip arbore şi previziunea valorilor, utilizând tehnici de regresie);
- segmentarea bazei de date (comasarea demografică şi comasarea neurală care se deosebesc prin metodele uilizate pentru a calcula distanţa dintre înregistrări, prin intrările de date permise);
- analiza legăturilor (descoperirea asocierilor, descoperirea tiparelor, descoperirea secvenţelor de timp similare);
- detectarea deviaţiilor (statistici şi vizualizări pentru identificarea împrăştierii datelor, utilizând tehnici moderne de vizualizare grafică). În această clasă pot fi considerate, de exemplu, detectarea fraudelor privind utilizarea cărţilor de credit, pretenţiile de despăgubire ale asiguraţilor etc.

În concluzie, spre deosebire de un sistem OLTP, *Data Warehouse* este o bază de date a cărei structură este proiectată pentru a facilita analiza datelor. Un sistem de suport decizional urmăreşte, în primul rând, obţinerea de informaţii din baza de date, în timp ce unul OLTP urmăreşte introducerea de informaţii în baza de date. Datorită acestor diferenţe, structura optimă a unei *Data Warehouse* este radical diferită de cea a unui sistem OLTP.

Depozitele de date şi sistemele OLTP sunt supuse unor cerinţe diferite, dintre care cele mai semnificative se referă la operaţii, actualizarea datelor, proiectare, operaţii tipice şi date istorice.

- **Operaţii**. Depozitele sunt create pentru a permite interogări *ad hoc*. Ele trebuie să fie suficient de flexibile pentru a putea răspunde interogărilor spontane ale utilizatorilor. Sistemele OLTP suportă numai operaţii predefinite. Aplicaţiile pot fi optimizate sau create special numai pentru acele operaţii.

- **Actualizarea datelor**. Utilizatorii finali ai unui depozit de date nu fac în mod direct actualizări ale depozitului. În sistemele OLTP, utilizatorii realizează, de obicei, în mod individual procedurile de modificare a bazei de date. În acest fel, baza de date OLTP este întotdeauna la zi şi reflectă starea curentă a fiecărei tranzacţii.
- **Proiectare**. Depozitele de date folosesc, în mod uzual, scheme denormalizate, în timp ce sistemele OLTP folosesc scheme normalizate pentru a optimiza performanţele operaţiilor.
- **Operaţii tipice**. O simplă interogare a depozitului de date poate scana mii sau chiar milioane de înregistrări (de exemplu, cererea „Care sunt vânzările totale către toţi clienţii din luna trecută?“), în timp ce o operaţie tipică OLTP accesează doar o parte mai mică din înregistrări.
- **Date istorice**. Depozitele de date stochează datele pe o perioadă lungă de timp, luni sau ani. Acest lucru oferă suport pentru analiza istorică a informaţiei. Sistemele OLTP reţin date istorice atât timp cât este necesar pentru a îndeplini cu succes cerinţele tranzacţiilor curente.

Sistemele OLTP	*Data Warehouse*
Păstrează date curente	Păstrează date istorice
Stochează date detaliate	Stochează date detaliate, agregate uşor sau puternic
Datele sunt dinamice	Datele sunt în mare măsură statice
Prelucrare repetitivă	Prelucrare *ad-hoc*, nestructurată şi euristică
Nivel înalt de transfer al tranzacţiilor	Nivel mediu sau scăzut de transfer al tranzacţiilor
Tipar de utilizare previzibil	Tipar de utilizare imprevizibil
Conduse prin tranzacţii	Conduse prin analiză
Susţin deciziile de zi cu zi	Susţin deciziile strategice
Deservesc un număr mare de utilizatori	Deservesc un număr relativ redus de utilizatori din administraţie
Orientate spre aplicaţii	Orientate spre subiect

Fig. 1.3. OLTP *versus Data Warehouse*

O bază de date OLAP poate fi relaţională, dar datorită naturii ei orientate spre dimensiuni, au fost dezvoltate pentru gestionarea acestor baze de date construcţii multidimensionale, mai potrivite pentru o raportare flexibilă. Spre deosebire de bazele de date relaţionale, structura unei baze de date multidimensionale nu implică tabele, linii şi coloane, ci obiecte de următoarele tipuri: variabile, dimensiuni, niveluri, ierarhii, atribute.

Data Warehouse, care cuprinde de obicei informaţii despre o întreagă companie, poate fi subîmpărţită în baze de date departamentale, numite rafturi de date (*Data Marts*). De exemplu, poate exista un *Data Mart* al departamentului financiar, un altul al departamentului vânzări şi un altul pentru departamentul marketing.

Construcţia unei astfel de baze de date poate fi abordată în două moduri.

- O primă abordare este de a construi mai întâi un schelet al bazei de date la care se lipesc ulterior rafturile de date. Aceasta necesită o analiză prealabilă a întregului şi o delimitare a blocurilor componente. Ea cere un timp mai lung de dezvoltare, dar rezultatul este o bază de date unitară.
- A doua metodă este de a construi mai întâi rafturi specifice, efortul constând, în acest caz, în asocierea acestora. Această soluţie oferă mai rapid, aplicaţii funcţionale utilizatorilor, dar au de suferit unitatea şi portabilitatea aplicaţiilor finale.

Utilizatorii trebuie să-şi schimbe optica asupra bazelor de date pentru a fi capabili să folosească puterea şi flexibilitatea instrumentelor analitice de care dispun. Instrumentele OLAP au evoluat ca o modalitate de a rezolva interogările complicate necesare procesului de analiză a datelor. Combinaţia între bazele de date multidimensionale şi instrumentele analitice prietenoase face uşoară analiza, sinteza şi consolidarea datelor.

În ultimii ani, marii producători de sisteme de gestiune a bazelor de date relaţionale, precum *Oracle*, au introdus în produsele lor construcţii care să faciliteze accesul la datele din sistemele fundamentale pentru luarea de decizii. Astfel, noile versiuni de SGBD-uri ale firmelor mari prevăd o modalitate mai inteligentă de a realiza operaţia de compunere între două sau mai multe tabele, metode de indexare noi, potrivite pentru marile cantităţi de date statice cu care operează sistemele *Data Warehouse*, capacitatea de a detecta şi optimiza interogări de un tip special, posibilitatea de a folosi mai multe procesoare pentru a rezolva o interogare.

Un sistem *Data Warehouse* are un efect fundamental asupra utilizatorilor. Ei pot manevra mult mai flexibil sistemul, au posibilităţi elevate pentru interogarea datelor, dar ei trebuie să ştie cum să prelucreze şi să vizualizeze datele şi cum să le folosească în procesul de decizie.

Un efort ce trebuie făcut pentru construirea unui sistem de suport pentru decizii (DSS – *Decision Support System*) constă în procesul de descoperire a informaţiilor utile din baza de date. Acest proces, numit ***Data Mining*** sau *Knowledge Discovery in Databases* (KDD), procesează mari cantităţi de date, a căror corelare nu este neapărat evidentă, în vederea descoperirii de tendinţe şi tipare.

1.5. Arhitecturi *multi-user* pentru sisteme de gestiune a bazelor de date

Arhitecturile uzuale care sunt utilizate pentru implementarea sistemelor de gestiune a bazelor de date *multi-user* sunt: teleprocesarea, arhitectura fişier-*server* arhitectura *client-server*.

Teleprocesarea este arhitectura tradiţională, ce cuprinde un calculator cu o singură unitate CPU şi un numar de terminale care sunt incapabile să funcţioneze singure. Terminalele trimit mesaje la programele aplicaţie ale utilizatorilor, care la rândul lor, utilizează serviciile SGBD.

Această arhitectură a plasat o greutate teribilă asupra calculatorului central, care pe lângă rularea programelor de aplicaţii şi ale SGBD-ului, mai trebuie să preia şi din munca terminalelor (de exemplu, formatarea datelor pentru afişarea pe ecran).

Arhitectura fişier-server, presupune deja că procesarea este distribuită în reţea (de obicei o reţea locală LAN). Arhitectura cuprinde fişierele cerute de aplicaţii şi SGBD-ul. Aplicaţiile şi funcţiile SGBD sunt executate pe fiecare staţie de lucru, solicitând când este nevoie fişiere de pe *server*-ul de fişiere. Dintre dezavantaje se remarcă:

- existenţa unui trafic intens pe reţea;
- necesitatea unei copii complete a SGBD-ului pe fiecare staţie de lucru;
- acelaşi fişier poate fi accesat de mai multe SGBD-uri, ceea ce implică un control complex al integrităţii, simultaneităţii, reconstituirii.

Arhitectura *client-server* se referă la modul în care interacţionează componentele *software* pentru a forma un sistem. Există un proces *client*, care necesită resurse şi un proces *server*, care oferă resurse.

În arhitectura *client-server*, clientul (*front end*) emite, prin intermediul reţelei locale, o cerere *SQL* care este executată pe *server* (*back-end*); acesta trimite ca răspuns ansamblul înregistrărilor rezultat. Într-o astfel de interacţiune maşinile sunt eterogene, iar protocoalele de reţea pot fi distincte.

În contextul bazelor de date, ***client***-ul:

- administrează interfaţa cu utilizatorul şi logica aplicaţiei;
- acceptă şi verifică sintaxa intrărilor utilizatorilor;
- procesează aplicaţiile;
- generează cerinţele pentru baza de date şi le trimite *server*-ului;
- transmite răspunsul înapoi la utilizator.

În contextul bazelor de date, ***server***-ul:

- primeşte şi procesează cerinţele clienţilor pentru baza de date;
- verifică autorizarea;
- garantează respectarea constrângerilor de integritate;
- efectuează procesarea interogare-reactualizare şi trimite clientului răspunsul;
- realizează optimizarea interogărilor;
- asigură controlul concurenţei dintre mai multi clienţi care se ignoră (mecanisme de blocare);
- intreţine dictionarul datelor;
- oferă acces simultan la baza de date;
- asigură robusteţea în cazul defecţiunilor;
- oferă controlul reconstituirii etc.

Arhitectura tradiţională *client-server* pe „două etaje (straturi)“ presupune:

- *client*-ul – responsabil, în primul rand, de prezentarea datelor către client;
- *server*-ul – responsabil, în primul rand, de furnizarea serviciilor către client.

Arhitectura *client-server* pe „trei etaje“ presupune trei straturi, fiecare fiind rulat, potenţial, pe o platformă diferită.

- stratul (*client*) format din interfaţa cu utilizatorul, care este rulat pe calculatorul utilizatorului final;

- stratul (*server* de aplicaţie), ce manevrează logica aplicaţiilor şi prelucrării datelor, şi care poate servi mai mulţi clienţi (conectare la celelalte două straturi se face prin reţele locale LAN sau de mare suprafaţă WAN);
- stratul (*server*-ul de baze de date), care se ocupă cu validarea datelor şi accesarea bazei de date (stochează date necesare stratului din mijloc).

Arhitectura se potriveşte natural mediului *Web*. Un *browser Web* acţionând drept *client* şi un *server Web* fiind *server* de aplicaţie.

Middleware este un strat, evident *software*, între aplicaţia postului *client* şi *server*-ul de baze de date, constituit dintr-o interfaţă de programare a aplicaţiilor (API - *Application Programming Interface*) şi un protocol de reţea.

API descrie tipul de interacţiune dintre o aplicaţie *client* şi un *server* la distanţă, *via* un protocol de comunicaţie şi de formatare a datelor. Scopul existenţei interfeţei de programare a aplicaţiilor este de a oferi o interfaţă unică mai multor *server*-e de baze de date.

Este convenabil ca sistemele de baze de date să fie considerate ca fiind formate dintr-un *server* (sistemul SGBD însăşi) şi un set de clienţi (aplicaţiile). Frecvent, clienţii şi *server*-ul pot fi rulate pe calculatoare diferite, realizându-se un tip simplu de procesare distribuită. În general, fiecare *server* poate deservi mai multi clienţi, iar fiecare client poate accesa mai multe *server*-e. Dacă sistemul oferă transparenţă totală (fiecare client se poate comporta ca şi cum ar comunica cu un singur *server*, de pe un singur calculator) atunci este vorba despre un sistem de baze de date distribuite.

1.6. Tehnologia *Web* şi sistemele SGBD

Internet = o colecţie mondială de reţele de calculatoare.

Intranet = un sit *Web* sau un grup de sit-uri care aparţin unei organizaţii, accesibil numai pentru membrii acesteia.

Extranet = o reţea intranet care este parţial accesibilă utilizatorilor externi autorizaţi.

Reţea Web (*World Wide Web*) = un sistem bazat pe hipermedii care pune la dispoziţie un mijloc simplu, de tip „indicare-clic“ de răsfoire a informaţiilor pe Internet, folosind hiperlegăturile.

HTTP = protocolul utilizat pentru a transfera pagini *Web* prin intermediul Internetului.

HTML = limbajul de formatare a documentelor utilizat în proiectarea majorităţii paginilor *Web*.

Adresa *URL* = şir de caractere alfanumerice care reprezintă adresa unei resurse pe Internet şi modul în care trebuie accesată resursa.

Interfaţa de poartă comună (*CGI*) = defineşte modul în care scripturile comunică cu *server*-ul Web. Este tehnica de bază de integrare a bazelor de date în reţeaua *Web*.

În mediul *Web* funcţionează modelul *three tier* format din:

- un strat de interfaţă cu utilizatorul (client),
- un strat de logică a afacerii şi prelucrare a datelor (*server* de aplicaţii),
- un sistem SGBD (*server* de baze de date) distribuit pe calculatoare diferite.

Avantajele reţelei Web ca platformă de baze de date:

- avantagele SGBD;
- simplitate;
- independenţa de platformă;
- interfaţa grafică cu utilizatorul;
- acces transparent în reţea;
- standardizare (HTML standard *de facto*).

Arhitectura de calcul în reţea a sistemului *Oracle* (NCA *Network Computing Architecture*) se axează în principal pe furnizarea extensibilităţii pentru mediile distribuite. Arhitectura este construită pe baza tehnologiei *CORBA* pentru manipularea obiectelor. Este o structură *three tier* care se bazează pe utilizarea de:

- cartuşe de *software* care permit utilizatorilor să adauge funcţionalităţi individuale în aplicaţii (cartuşele pot fi construite în *Java*, *C/C++*, *Visual Basic*, *SQL* şi pot fi conectate la oricare din cele 3 straturi);
- protocoale deschise şi interfeţe standardizate care permit comunicarea între cartuşe (distribuite într-o reţea) prin intermediul unui program magistrală (ICX);
- clienţi extensibili, *server*-e de aplicaţie, *server*-e de baze de date;
- dezvoltarea şi administrarea integrată a cartuşelor.

Un cartus utilizeaza un limbaj de definire a interfetelor (IDL) pentru a putea fi identificat de alte obiecte intr-un sistem distribuit. De exemplu, PL/*SQL* este un astfel de cartus.

Arhitectura *multitier* a sistemului *Oracle9i*

Arhitectura cu mai multe niveluri (*multitier*) conţine următoarele elemente:

- unul sau mai mulţi *client*-i care iniţiază operaţii;
- unul sau mai multe *server*-e de aplicaţii care execută părţi ale operaţiilor;
- un *server* de baze de date care stochează datele folosite de operaţii.

Client-ul, care poate fi un *browser Web* sau un proces *user*, iniţiază o cerere pentru a executa o operaţie referitoare la informaţiile stocate în baza de date. Conectarea la *server*-ul bazei de date se face printr-unul sau mai multe *server*-e de aplicaţii.

Server-ul de aplicaţii constituie interfaţa dintre *client*-i şi *server*-ul bazei de date, asigurând accesul la informaţii. De asemenea, el include un nivel adiţional pentru securitate. *Server*-ul de aplicaţii îşi asumă identitatea *client*-ului, atunci când execută, pe *server*-ul de baze de date, operaţiile solicitate de acesta.

Arhitectura *multitier* permite folosirea unui *server* de aplicaţii pentru acreditarea *client*-ului, conectarea la *server*-ul de baze de date şi execuţia operaţiilor iniţiate de *client*. Privilegiile *server*-ului de aplicaţii sunt limitate pentru a preveni execuţia operaţiilor nedorite sau inutile în timpul unei operaţii *client*.

Server-ul de baze de date pune la dispoziţia *server*-ului de aplicaţii informaţiile necesare pentru soluţionarea operaţiilor lansate de către *client*. De asemenea, acesta face distincţia între operaţiile pe care *server*-ul de aplicaţii le cere în favoarea *client*-ului şi cele pe care le solicită în nume propriu.

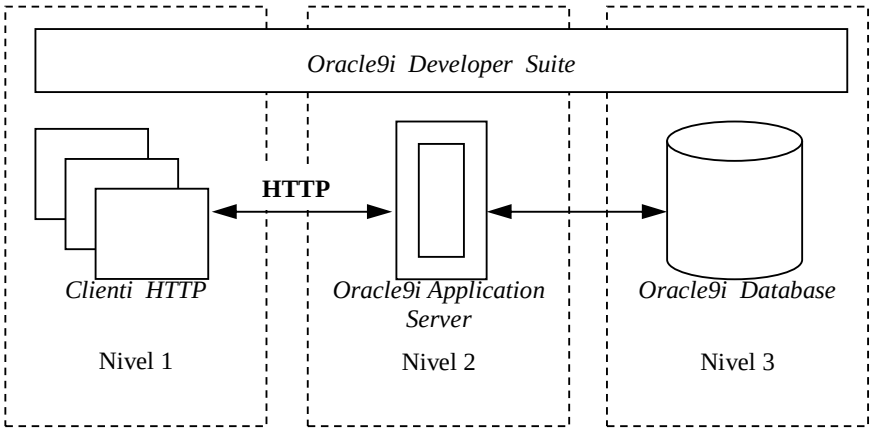


Fig. 1.4. Arhitectura *three-tier* a sistemului *Oracle9i*

Modelarea datelor

Un **model** este o reprezentare a obiectelor şi evenimentelor lumii reale şi a asocierilor dintre ele. De fapt, el reprezintă o abstracţie asupra aspectelor semnificative ale unei „întreprinderi“, ale unui sistem real, ignorând proprietăţile accidentale. Modelul este cel pe care utilizatorii trebuie să-l cunoască; implementarea unui model este cea pe care utilizatorii nu este necesar să o cunoască. Diferenţa dintre model şi implementare este, de fapt, un caz special şi important al deosebirii uzuale dintre logic şi fizic.

Modelele se impun prin sintaxa şi prin semantica lor şi, din acest punct de vedere, există trei tipuri fundamentale de modele:

- modele care descriu aspectele statice ale procesului modelat;
- modele care descriu aspectele dinamice ale procesului modelat;
- modele care descriu aspectele funcţionale ale procesului modelat.

Un **model de date** reprezintă o colecţie integrată de concepte necesare descrierii:

- datelor,
- relaţiilor dintre ele,
- constrângerilor existente asupra datelor sistemului real analizat.

Modelarea unei baze de date permite trecerea de la percepţia unor fapte din lumea reală la reprezentarea lor prin date. Modelul de date trebuie să reflecte fidel fenomene ale lumii reale, să urmărească evoluţia acestei lumi şi comunicarea dintre fenomenele lumii reale.

Modelul trebuie să asigure conceptele de bază care permit proiectantului bazei de date şi utilizatorilor să comunice, fără ambiguităţi, cunoştinţele lor privind funcţionarea şi organizarea modelului real analizat. Prin urmare, un model de date trebuie să reprezinte datele şi să le facă înţelese.

În esenţă, modelul de date are trei componente:

- o mulţime de reguli conform cărora sunt construite bazele de date (partea structurală);
- o mulţime de operaţii permise asupra datelor, care sunt utilizate pentru reactualizarea sau regăsirea datelor (partea de prelucrare);
- o mulţime de reguli de integritate, care asigură coerenţa datelor.

Abordarea generală a problemei modelării semantice a datelor se face în patru etape.

- Se identifică o mulţime de concepte semantice care sunt utile în descrierea lumii reale. Se presupune că lumea reală (modelul real analizat) este formată din entităţi care au anumite proprietăţi, că fiecare entitate are o identitate, că există legături, corelaţii între entităţi. Conceptul de corelaţie, ca şi cel de entitate, este util, în mod intuitiv, la descrierea modelului.
- Se caută o mulţime de obiecte formale, simbolice care sunt utilizate

pentru reprezentarea conceptelor semantice anterioare.

- Se dau reguli de integritate formale şi generale (constrângeri) care să reflecte restricţiile la care este supus modelul.
- Se defineşte o mulţime de operatori formali prin care pot fi prelucrate şi analizate obiectele formale.

Modelul entitate-relaţie

Una dintre cele mai cunoscute abordări ale modelării semantice (cu siguranţă una dintre cele mai utilizate) este cea bazată pe **modelul entitate-relaţie** (E/R). Acesta a fost introdus de către P.P. Chen în 1976 şi rafinat de atunci în diverse moduri de către acesta şi de mulţi alţi cercetători, ca un model de date conceptual, pentru a uşura proiectarea bazelor de date. Pentru reprezentarea grafică a modelului sunt utilizate diagramele E/R, care sunt modele neformalizate pentru reprezentarea unui model, unui sistem din lumea reală.

Diagramele E/R constituie o tehnică de reprezentare a structurii logice a bazei de date, într-o manieră grafică. Aceste diagrame oferă un mijloc simplu şi inteligibil de comunicare a caracteristicilor importante ale designului unei anumite baze de date.

Diagrama E/R este un model de date conceptual de nivel înalt, independent de platforma hardware utilizată şi de tipul SGBD-ului. Modelul este constituit din concepte care descriu structura bazei de date şi tranzacţiile de regăsire sau reactualzare asociate.

Popularitatea modelului E/R ca modalitate de abordare a proiectării bazelor de date poate fi atribuită în principal tehnicii de realizare a diagramelor E/R. Această tehnică, ca şi modelul E/R însuşi, a evoluat de-a lungul timpului datorită noilor problematici care au apărut în proiectarea bazelor de date.

Baza de date poate fi definită ca o mulţime de date ce modelează un sistem real. Acest sistem este format din obiecte legate între ele. Modelul E/R împarte elementele unui sistem real în două categorii: entităţi şi relaţii (legături, asocieri) între aceste entităţi. Entităţiile şi legăturile au anumite caracteristici, numite atribute. Nu trebuie confundat conceptul de relaţie, în sensul de asociere, care intervine în definirea diagramei E/R cu conceptul de relaţie care este specific modelului relaţional.

!!! Studii de caz, exemple – prezentate la curs!

Deficienţe ale modelului E/R

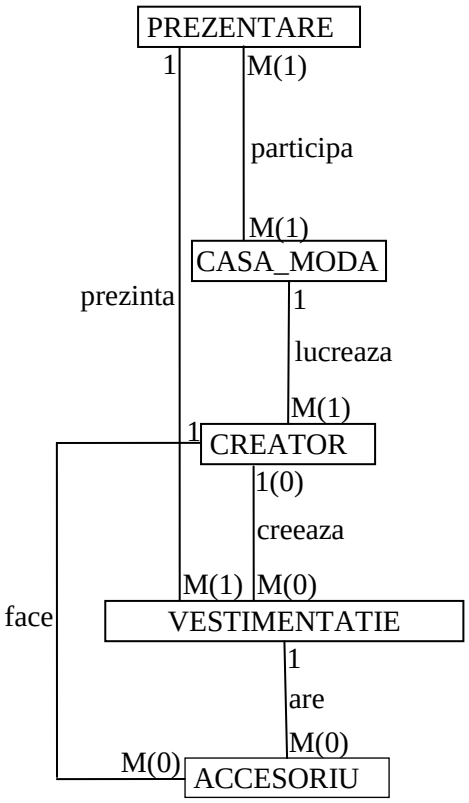
În proiectarea unui model de date pot apărea diverse probleme datorită unei interpretări eronate a sensului unei relaţii. Aceste probleme sunt denumite **capcane de conectare**.

Unele dintre aceste capcane pot să nu fie semnificative pentru modelul particular considerat, în timp ce altele cer restructurarea modelului. Există două clase importante de capcane: de întrerupere şi în evantai.

O capcană de întrerupere poate să apară acolo unde modelul sugerează existenţa unei relaţii între entităţi, dar nu există o cale între anumite apariţii ale entităţilor. Această capcană poate să apară acolo unde există o relaţie cu participare parţială (0 la cardinalitatea minimă), care face parte din calea dintre entităţile ce sunt legate.

O capcană în evantai poate să apară acolo unde modelul ia în considerare o relaţie între entităţi, dar calea dintre anumite apariţii ale entităţilor este ambiguă. Aceste capcane apar când două sau mai multe relaţii one_to_many provin din aceeaşi entitate.

Practic, aceste capcane generează situaţiile în care, aşa cum a fost proiectat modelul de date, el nu poate să răspundă la anumite interogări. De exemplu, pentru a afla pentru ce prezentare de modă a fost creată o anumită vestimentaţie, a fost necesară introducerea unei legături între entităţile PREZENTARE şi VESTIMENTATIE, care însă a generat redundanţă în modelul de date:

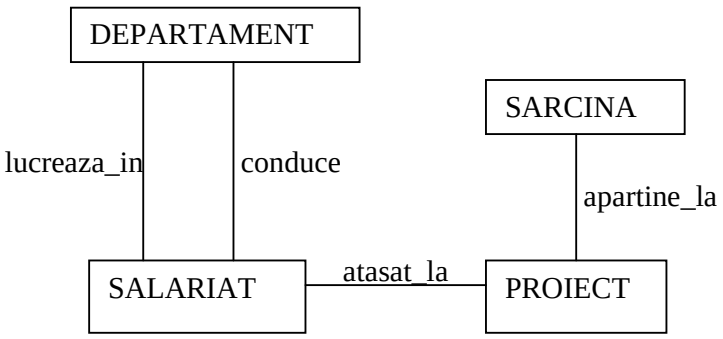

PREZENTARE
1
M(1)
participa
M(1)
CASA_MODA
prezinta
1
lucreaza
M(1)
1
CREATOR
1(0)
creeaza
M(1)
M(0)
face
VESTIMENTATIE
1
are
M(0)
M(0)
ACCESORIU

Material incomplet, perfectibil!!! Vezi curs + bibliografie!!!

Diagrame entitate-relaţie

Diagrama E/R – model neformalizat pentru reprezentarea unui sistem din lumea reală. Este un model de date conceptual de nivel înalt dezvoltat de Chen (1976).

Entitate: persoană, loc, concept, activitate, eveniment care este semnificativ pentru ceea ce modelăm.



Observaţii:

- Entităţile devin tabele în modelele relaţionale.
- În general, entităţile se scriu cu litere mari.
- Entităţile sunt substantive, dar nu orice substantiv este o entitate.
- Pentru fiecare entitate este obligatoriu să se dea o descriere detaliată.
- Nu pot exista, în aceeaşi diagramă, două entităţi cu acelaşi nume, sau o aceeaşi entitate cu nume diferite.

Cheia primară este un identificator unic în cadrul entităţii, făcând distincţie între valori diferite ale acesteia.

Cheia primară:

- trebuie să fie unică şi cunoscută la orice moment;
- trebuie să fie controlată de administratorul bazei;
- trebuie să nu conţină informaţii descriptive, să fie simplă, fără ambiguităţi;
- să fie stabilă;
- să fie familiară utilizatorului.

Relaţie (asociere): o comunicare între două sau mai multe entităţi. Existenţa unei relaţii este subordonată existenţei entităţilor pe care le leagă.

Observaţii:

- În modelul relaţional, relaţiile devin tabele speciale sau coloane speciale care referă chei primare.
- Relaţiile sunt verbe, dar nu orice verb este o relaţie.
- Pentru fiecare relaţie este important să se dea o descriere detaliată.
- În aceeaşi diagramă pot exista relaţii diferite cu acelaşi nume. În acest caz, le diferenţiază entităţile care sunt asociate prin relaţia respectivă.
- Pentru fiecare relaţie trebuie stabilită cardinalitatea (maximă şi minimă) relaţiei, adică numărul de tupluri ce aparţin relaţiei.

poate (cardinalitate maximă) → trebuie (cardinalitate minima)

Exemplu:

Câţi salariaţi **pot** lucra într-un departament? Mulţi!
În câte departamente **poate** lucra un salariat? In cel mult unul!

→

Relaţia SALARIAT_lucreaza_in_DEPARTAMENT are cardinalitatea maximă ***many-one*** (n:1).

Exemplu:

Câţi salariaţi **trebuie** să conducă un departament? Cel puţin unul!
Câte departamente **trebuie** să conducă un salariat? Zero!

→

Relaţia SALARIAT_conduce_DEPARTAMENT are cardinalitatea minimă ***one-zero*** (1:0).

Atribut: proprietate descriptivă a unei entităţi sau a unei relaţii.

Observaţii:

- Trebuie făcută distincţia între atribut (devine coloană în modelele relaţionale) şi valoarea acestuia (devine valoare în coloane).
- Atributele sunt substantive, dar nu orice substantiv este atribut.
- Fiecărui atribut trebuie să i se dea o descriere completă (exemple, contraexemple, caracteristici).
- Pentru fiecare atribut trebuie specificat numele, tipul fizic (*integer*, *float*, *char* etc.), valori posibile, valori implicite, reguli de validare, tipuri compuse.

Pentru proiectarea **diagramei entitate-relaţie** au fost stabilite anumite reguli (nu sunt unice):

1. entităţile sunt reprezentate prin dreptunghiuri;
2. relaţiile dintre entităţi sunt reprezentate prin arce neorientate;
3. atributele care reprezintă chei primare trebuie subliniate sau marcate prin simbolul „#“, plasat la sfârşitul numelui acestor atribute;
4. cardinalitatea minimă este indicată în paranteze, iar cardinalitatea maximă se scrie fără paranteze;
5. nu trebuie specificate toate atributele.

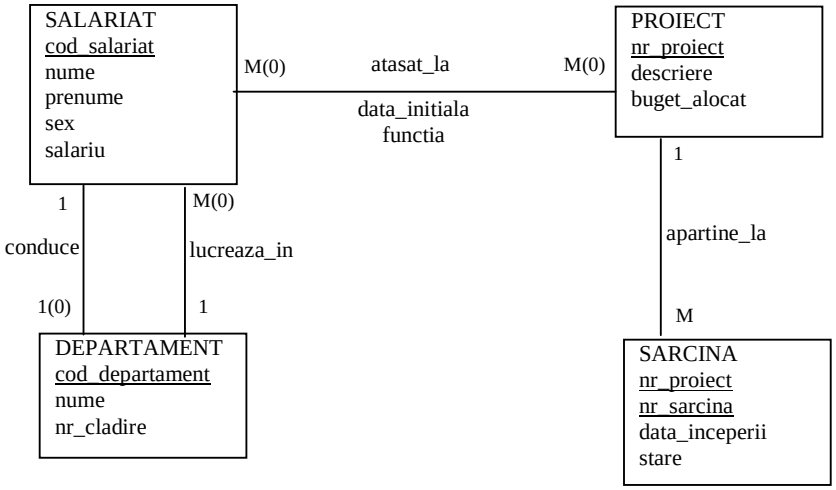


Diagrama E/R.

Cazuri speciale de entităţi, relaţii, atribute şi modul lor de reprezentare în cadrul diagramei entitate-relaţie.

1. **Entitate dependentă** – nu poate exista în mod independent (SARCINA depinde de PROIECT). Cheia primară a unei entităţi dependente include cheia primară a sursei (*nr_proiect*) şi cel puţin o descriere a entităţii (*nr_sarcina*). Entitatea dependentă se desenează prin dreptunghiuri cu linii mai subţiri.
2. Moştenirea atributelor. **Subentitate** (subclasă) – submulţime a unei alte entităţi, numită **superentitate** (superclasă) (SALARIAT < — > PROGRAMATOR). Subentitatea se desenează prin dreptunghiuri incluse în superentitate. Există o relaţie între o subentitate şi o

superentitate, numită ISA, care are cardinalitatea maximă 1:1 şi minimă 1:0. Cheile primare, atributele şi relaţiile unei superentităţi sunt valabile pentru orice subentitate. Afirmaţia reciprocă este falsă.

3. Generalizare. Din entităţi similare care au mai multe atribute comune se pot crea superentităţi. Aceste superentităţi conţin atributele comune, iar atributele speciale sunt asignate la subentităţi. Pentru noile superentităţi se introduc chei primare artificiale.

4. Specializare. După valorile unor atribute clasificatoare se pot determina **clase**. Un grup de subentităţi reciproc exclusive defineşte o clasă. Clasele se aliniază în desen vertical.

5. Într-o diagramă E/R se pot defini relaţii recursive.

6. Unele relaţii sunt relative la două entităţi şi le numim de tip 2, iar dacă relaţiile implică mai mult de două entităţi, le vom numi de tip 3. Trei relaţii de tip 2 sunt diferite de o relaţie de tip 3. Rupând o relaţie de tip 3 în trei relaţii de tip 2, pot apărea informaţii incorecte.

7. Trebuie excluse din model relaţiile indirecte deoarece ele pot conduce la redundanţă în baza de date.

8. Atributele derivabile trebuie eliminate şi introduse expresii prin care aceste atribute pot fi calculate.

9. Relaţie sau atribut? Dacă un atribut al unei entităţi reprezintă cheia primară a unei alte entităţi, atunci el referă o relaţie (*cod_departament* în tabelul SALARIAT).

10. Entitate sau relaţie? Se cercetează cheia primară. Dacă aceasta combină cheile primare a două entităţi, atunci este vorba de o relaţie. (cheia primară a relaţiei *asociat_la* combină *cod_salariat* cu *nr_proiect*, prin urmare, *SALARIAT_asociat la_PROIECT* va defini o relaţie şi nu o entitate).

11. Un atribut indirect este inoportun. El nu descrie real relaţia sau entitatea. Prin urmare, atributele indirecte trebuie reasignate. De fapt, un atribut indirect este un caz special de relaţie indirectă care trebuie eliminată pentru că introduce redundanţă în date (numărul clădirii în care lucrează un salariat este un atribut al entităţii DEPARTAMENT şi nu este o caracteristică a entităţii SALARIAT).

12. Există atribute opţionale, a căror valoare este uneori necunoscută, alteori neaplicabilă. Aceste atribute trebuie introduse la subentităţi (comisionul pentru deplasare şi zona de lucru sunt atribute specifice unui agent teritorial şi trebuie introduse la subentitatea AGENT_TERITORIAL).

Algoritmul pentru proiectarea diagramei entitate-relaţie:

1. identificarea entităţilor din cadrul sistemului analizat;
2. identificarea relaţiilor dintre entităţi şi stabilirea cardinalităţii;
3. identificarea atributelor aferente entităţilor şi asocierilor dintre entităţi;
4. stabilirea atributelor de identificare a entităţilor (stabilirea cheilor).

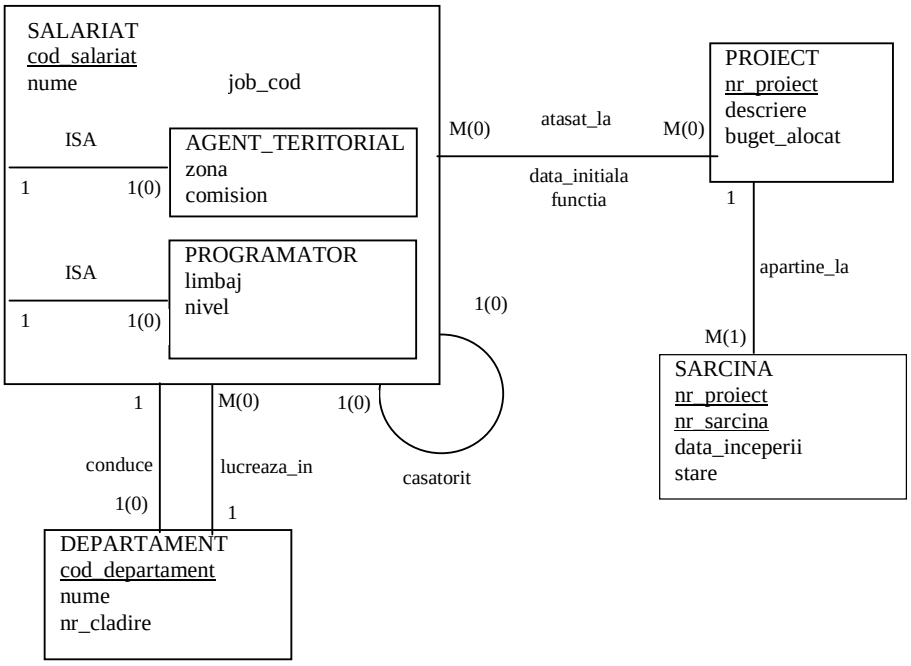
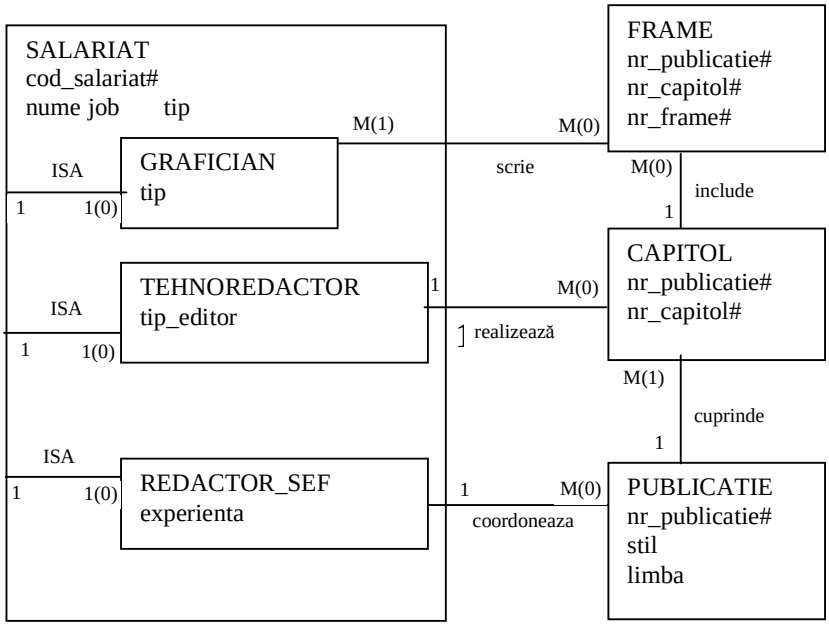


Diagrama E/R.

Modelul EER (modelul E/R extins) = Diagrama E/R + concepte aditionale (subclasă, superclasă, moştenire, specializare, generalizare).

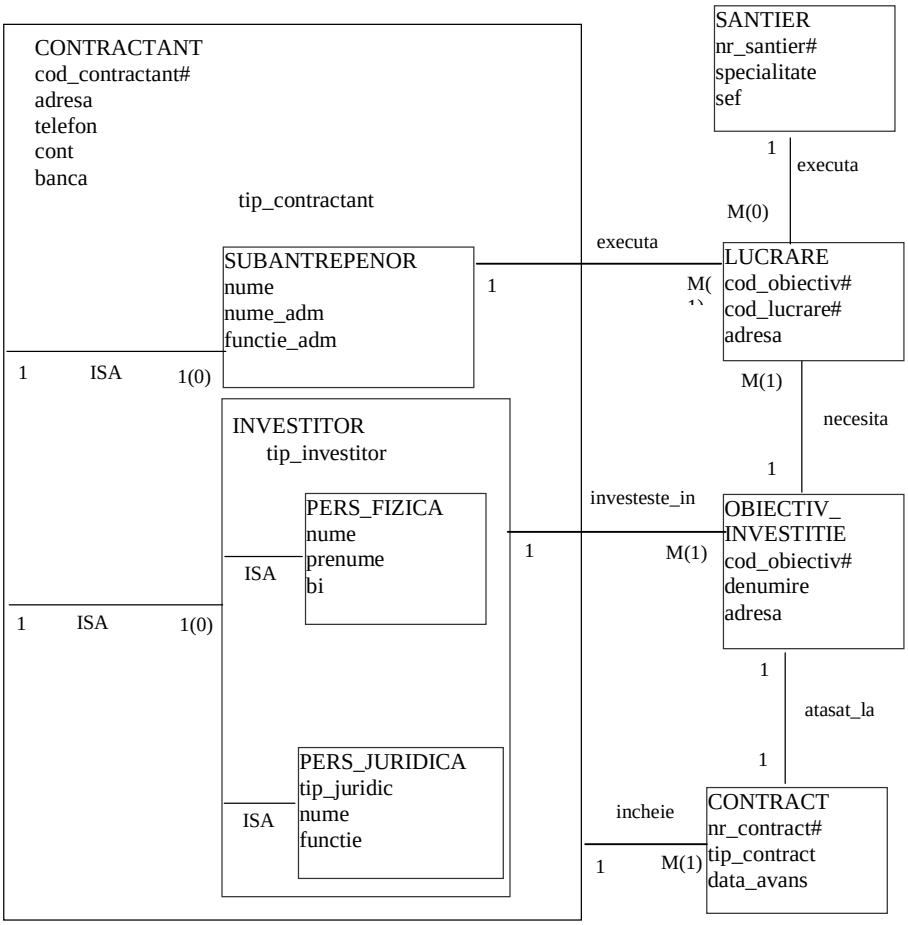
Gestiunea activităţilor de editare dintr-o editură

Se analizeaza activitatea dintr-o editură referitoare la culegerea textelor, realizarea elementelor grafice, machetarea unor publicaţii.



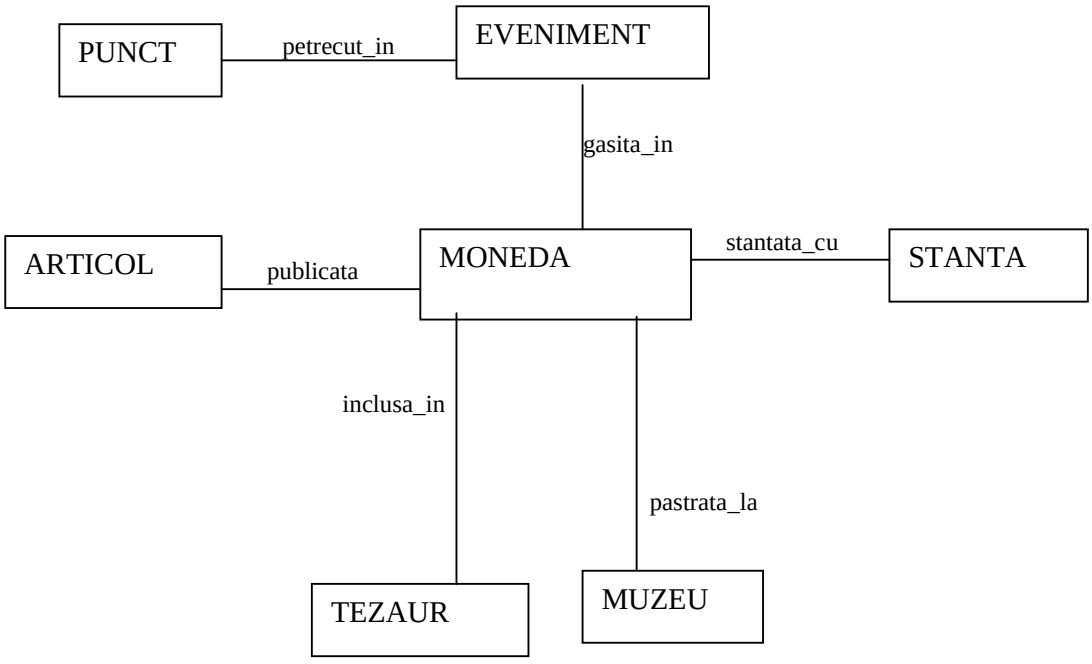
Gestiunea activităţilor unei firme de construcţii

Baza de date construită prin acest model, furnizează informaţii legate de obiective de execuţie, investitori, executanţi, şantiere, contracte etc. necesare unui manager al unei firme de construcţii



Vezi erori!

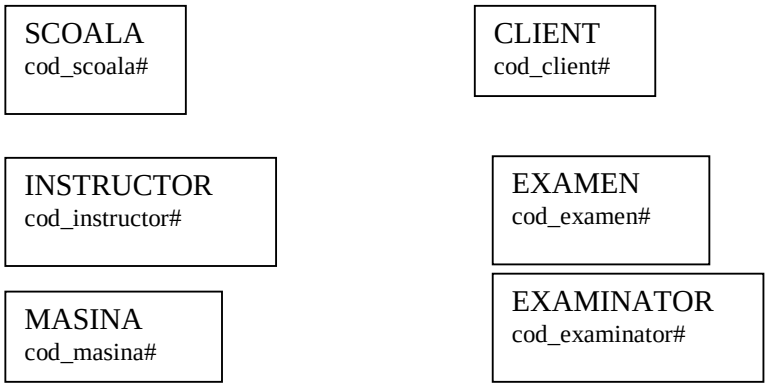
Descoperiri de monede antice din Romania



STANŢA (nr_stanţă, împărat emitent, valoare nominală, an emitere, monetăria, legenda de pe avers, legenda de pe revers) == > atribute ale entităţii **STANTA**

Completaţi cardinalitatea!

Evidenţa şcolilor de şoferi din Romania



Completaţi relaţiile (*lucreaza_la, conduce, sustine, asista, instruieste*) dintre entităţi şi specificaţi cardinalitatea!

Campionatele de fotbal ale diferitelor ţări

Modelul relaţional

Modelul relaţional a fost conceput şi dezvoltat de E.F. **Codd**. El este un model formal de organizare conceptuală a datelor, destinat reprezentării legăturilor dintre date, bazat pe teoria matematică a relaţiilor. Modelul relaţional este alcătuit numai din relaţii şi prin urmare, orice interogare asupra bazei de date este tot o relaţie. Cercetarea în domeniu → 3 mari proiecte (*System R*, *INGRES*, *PRTV*)

Calităţi:

- este simplu;
- riguros din punct de vedere matematic;
- nu este orientat spre sistemul de calcul.

Modalităţi pentru **definirea unui SGBD relaţional**:

- prezentarea datelor în tabele supuse anumitor operaţii de tip proiecţie, selecţie, reuniune, compunere, intersecţie etc.
- un sistem de baze de date ce suportă un limbaj de tip SQL – *Structured Query Language*;
- un sistem de baze de date care respectă principiile modelului relaţional introdus de E.F. Codd.

Caracteristicile unui model relaţional:

- structura relaţională a datelor;
- operatorii modelului relaţional;
- regulile de integritate care guvernează folosirea cheilor în model.

Aceste trei elemente corespund celor trei componente ale ingineriei *software*: informaţie, proces, integritate.

Structura datelor

Definirea noţiunilor de domeniu, relaţie, schemă relaţională, valoare *null* şi tabel vizualizare (*view*).

Conceptele utilizate pentru a descrie formal, uzual sau fizic elementele de bază ale organizării datelor sunt date în următorul tabel:

Formal	Uzual	Fizic
relaţie	tablou	fişier
tuplu	linie	înregistrare
atribut	coloană	câmp
domeniu	tip de dată	tip de dată

Domeniu – mulţime de valori care poate fi definită fie enumerând elementele componente, fie definind o proprietate distinctivă a domeniului valorilor.

Fie D_1, D_2, ..., D_n domenii finite, nu neapărat disjuncte. **Produsul cartezian** $D_1 \times D_2 \times ... \times D_n$ al domeniilor D_1, D_2, ..., D_n este definit de mulţimea tuplurilor (V_1, V_2, ..., V_n), unde $V_1 \in D_1$, $V_2 \in D_2$, ..., $V_n \in D_n$. Numărul *n* defineşte **aritatea tuplului**.

O **relaţie** *R* pe mulţimile D_1, D_2, ..., D_n este o submulţime a produsului cartezian $D_1 \times D_2 \times ... \times D_n$, deci este o mulţime de tupluri. Caracteristicile unei relaţii → comentat curs!

Definirea unei relaţii se referă la mulţimi care variază în timp. Pentru a caracteriza o relaţie este necesară existenţa un element invariant în timp: structura relaţiei (schema relaţională). Mulţimea numelor atributelor corespunzătoare unei relaţii *R* defineşte **schema relaţională** a relaţiei respective. Vom nota schema relaţională prin $R(A_1, A_2, ..., A_n)$. Exemplu!

Putem reprezenta o relaţie printr-un tabel bidimensional în care fiecare linie corespunde unui tuplu şi fiecare coloană corespunde unui domeniu din produsul cartezian. O coloană corespunde de fapt unui atribut. Numărul atributelor defineşte **gradul** relaţiei, iar numărul de tupluri din relaţie defineşte **cardinalitatea** relaţiei.

Exemplu (crearea unui tabel în *SQL*):

```
CREATE TABLE salariat (
    cod_salariat      SMALLINT,
    nume              VARCHAR(25),
    prenume           VARCHAR(20),
    sex               CHAR(1),
    salariu           INTEGER,
    sot               SMALLINT,
    job_cod           VARCHAR(6),
    cod_departament   SMALLINT );
```

Când se inserează tupluri într-o relaţie, de multe ori un atribut este necunoscut sau neaplicabil. Pentru a reprezenta acest atribut a fost introdusă o valoare convenţională în relaţie, şi anume valoarea ***null***.

Este necesară o aritmetică şi o logică nouă care să cuprindă acest element. Rezultatul operatorilor aritmetici sau logici este *null* când unul din argumente este *null*. Comentat excepţii! Prin urmare, „*null = null*“ are valoarea *null*, iar ¬ *null* este *null*.

AND	T	F	Null
T	T	F	Null
F	F	F	F
Null	Null	F	Null

OR	T	F	Null
T	T	T	T
F	T	F	Null
Null	T	Null	Null

Tabele de adevăr pentru operatorii AND şi OR.

Tabelul **vizualizare** (*view*, filtru, relaţie virtuală, vedere) constituie un filtru relativ la unul sau mai multe tabele, care conţine numai informaţia necesară unei anumite abordări sau aplicaţii. Securitate, reactualizări → comentat la curs!

Vizualizarea este virtuală deoarece datele pe care le conţine nu sunt în realitate memorate într-o bază de date. Este memorată numai definiţia vizualizării. Vizualizarea nu este definită explicit, ca relaţiile de bază, prin mulţimea tuplurilor componente, ci implicit, pe baza altor relaţii prin intermediul unor expresii relaţionale. Stabilirea efectivă a tuplurilor care compun vizualizarea se realizează prin evaluarea expresiei atunci când utilizatorul se referă la acest tabel.

Exemplu (crearea unei vizualizări în *SQL*):

```
CREATE VIEW programator(nume,departament)
 AS SELECT  nume,cod_departament
    FROM    salariat
    WHERE   job_cod='programator';
```

Reguli de integritate → aserţiuni pe care datele conţinute în baza de date trebuie să le satisfacă.

Trebuie făcută distincţia între:

- regulile structurale inerente modelării datelor;
- regulile de funcţionare specifice unei aplicaţii particulare.

Există trei tipuri de constrângeri structurale (de cheie, de referinţă, de entitate) ce constituie mulţimea minimală de reguli de integritate pe care **trebuie** să le respecte un SGBD relaţional. Restricţiile de integritate minimale sunt definite în raport cu noţiunea de cheie a unei relaţii.

O mulţime minimală de atribute ale căror valori identifică unic un tuplu într-o relaţie reprezintă o **cheie** pentru relaţia respectivă.

Fiecare relaţie are cel puţin o cheie. Una dintre cheile candidat va fi aleasă pentru a identifica efectiv tupluri şi ea va primi numele de **cheie primară**. Cheia primară nu poate fi reactualizată. Atributele care reprezintă cheia primară sunt fie subliniate, fie urmate de semnul #.

O cheie identifică linii şi este diferită de un index care localizează liniile. O **cheie secundară** este folosită ca index pentru a accesa tupluri. Un grup de atribute din cadrul unei relaţii care conţine o cheie a relaţiei poartă numele de **supercheie**.

Fie schemele relaţionale *R*1(*P*1, *S*1) şi *R*2(*S*1, *S*2), unde *P*1 este cheie primară pentru *R*1, *S*1 este cheie secundară pentru *R*1, iar *S*1 este cheie primară pentru *R*2. În acest caz, vom spune că *S*1 este **cheie externă** (cheie străină) pentru *R*1.

Modelul relaţional respectă trei reguli de integritate structurală.

- **Regula 1** – unicitatea cheii. Cheia primară trebuie să fie unică şi minimală.
- **Regula 2** – integritatea entităţii. Atributele cheii primare trebuie să fie diferite de valoarea *null*.
- **Regula 3 –** integritatea referirii. O cheie externă trebuie să fie ori *null* în întregime, ori să corespundă unei valori a cheii primare asociate.

Proiectarea modelului relaţional (exemple → curs!)

Transformarea entităţilor

- Entităţile independente devin **tabele independente**. Cheia primară nu conţine chei externe.
- Entităţile dependente devin **tabele dependente**. Cheia primară a entităţilor dependente conţine cheia primară a entităţii de care depinde (cheie externă) plus unul sau mai multe atribute adiţionale.
- Subentităţile devin **subtabele**. Cheia externă se referă la supertabel, iar cheia primară este această cheie externă (cheia primară a subentităţii PROGRAMATOR este *cod_salariat* care este o cheie externă).

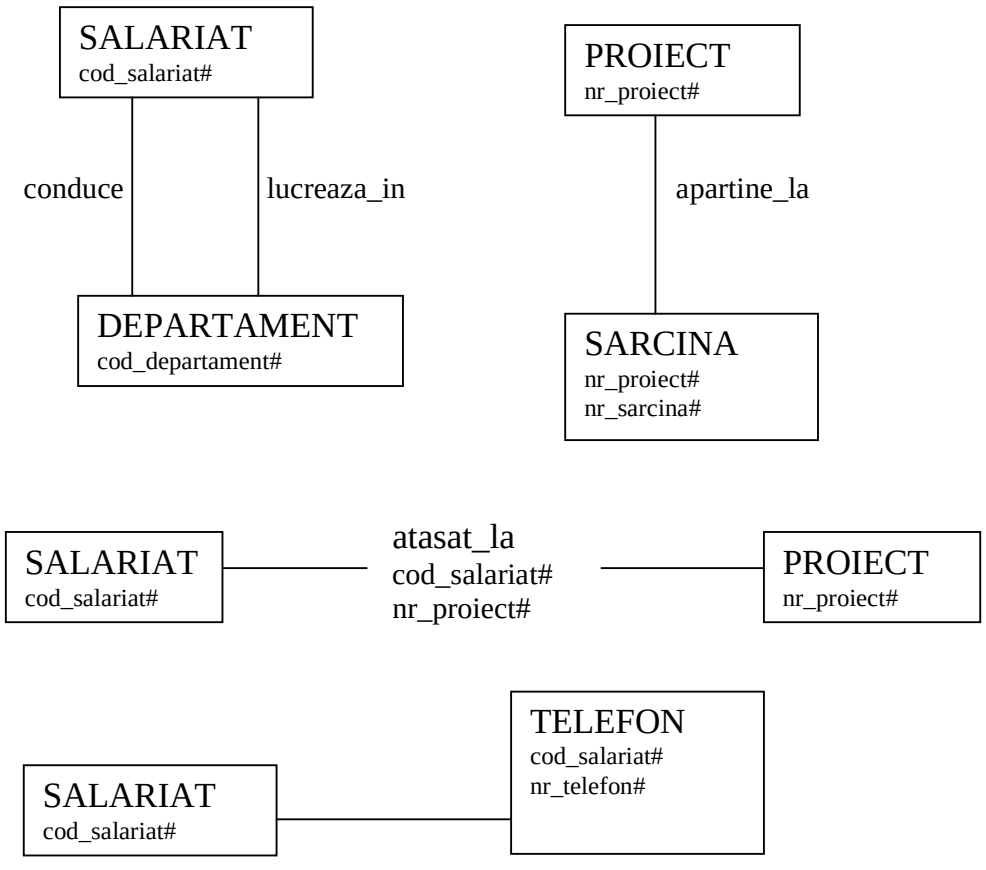
Transformarea relaţiilor

- Relaţiile 1:1 şi 1:*n* devin chei externe. Relaţia *conduce* devine coloană în tabelul DEPARTAMENT, iar relaţia *lucreaza_in* devine coloană în tabelul SALARIAT. Simbolul „×“ indică plasamentul cheii externe, iar simbolul „$\underline{\times}$“ exprimă faptul că această cheie externă este conţinută în cheia primară. Relaţia 1:1 plasează cheia externă în tabelul cu mai puţine linii.

- Relaţia *m*:*n* devine un tabel special, numit tabel **asociativ**, care are două chei externe pentru cele două tabele asociate. Cheia primară este compunerea acestor două chei externe plus eventuale coloane adiţionale. Tabelul se desenează punctat.
- Relaţiile de tip trei devin tabele asociative. Cheia primară este compunerea a trei chei externe plus eventuale coloane adiţionale.

Transformarea atributelor

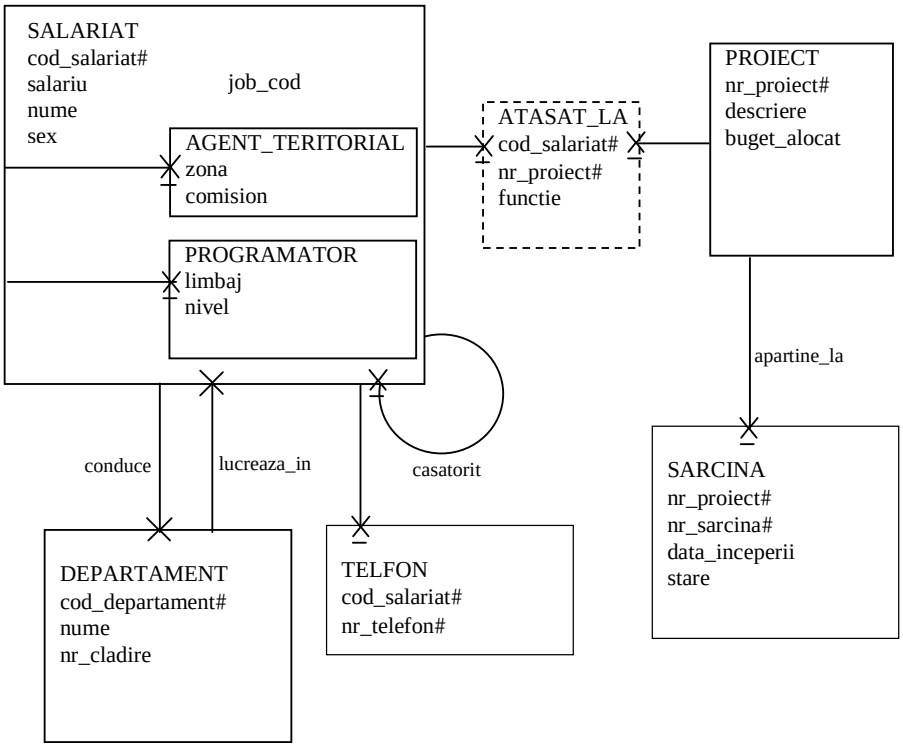
- Un atribut singular devine o coloană.
- Atributele multiple devin tabele dependendente ce conţin cheia primară a entităţii şi atributul multiplu. Cheia primară este o cheie externă, plus una sau mai multe coloane adiţionale.
- Entităţile devin tabele, iar atributele lor devin coloane în aceste tabele. Ce devin atributele relaţiilor? Pentru relaţii 1:1 şi 1:*n*, atributele relaţiilor vor aparţine tabelului care conţine cheia externă, iar pentru relaţii *m*:*n* şi de tipul trei, atributele vor fi plasate în tabelele asociative.



Cele patru tipuri de tabele (independente, dependente, subtabele şi asociative) se deosebesc prin structura cheii primare.

Tabel	Reprezintă	Cheie primară
Independent	entitate independentă	nu conţine chei externe
Subtabel	Subentitate	o cheie externă
Dependent	entitate dependentă	o cheie externă şi una sau mai multe coloane adiţionale
	atribute multiple	
Asociativ	relaţie m:n	două sau mai multe chei externe şi (opţional) coloane adiţionale
	relaţii de tip 3	

Diagrama conceptuală pentru proiectarea modelului relaţional comentat a fost construită din diagrama E/R prin adăugarea tabelelor asociative şi prin marcarea cheilor externe.



Schemele relaţionale corespunzătoare acestei diagrame conceptuale sunt următoarele:

– SALARIAT(cod_salariat#, nume, prenume, sex, job_cod, cod_sot, forma_plata, nr_depart);

- DEPARTAMENT(cod_departament#, nume, numar_cladire, cod_sal);
- ATASAT_LA(cod_salariat#, nr_proiect#, functia);
- PROIECT(nr_proiect#, descriere, buget_alocat);
- SARCINA(nr_proiect#, nr_sarcina, data_inceperii, stare);
- AGENT_TERITORIAL(cod_salariat#, zona, comision);
- PROGRAMATOR(cod_salariat#, limbaj, nivel);
- TELEFON(cod_salariat#, nr_telefon#).

Gestiunea activităţilor unei firme de construcţii

CONTRACTANT(cod_contractant#, adresa, telefon, cont, banca, tip_contractant);

SUBANTREPRENOR(cod_contractant#, nume, nr_reg_comert, nume_adm, functie_adm);

INVESTITOR(cod_contractant#, tip_investitor);

PERS_FIZICA(cod_contractant#, nume, prenume, bi);

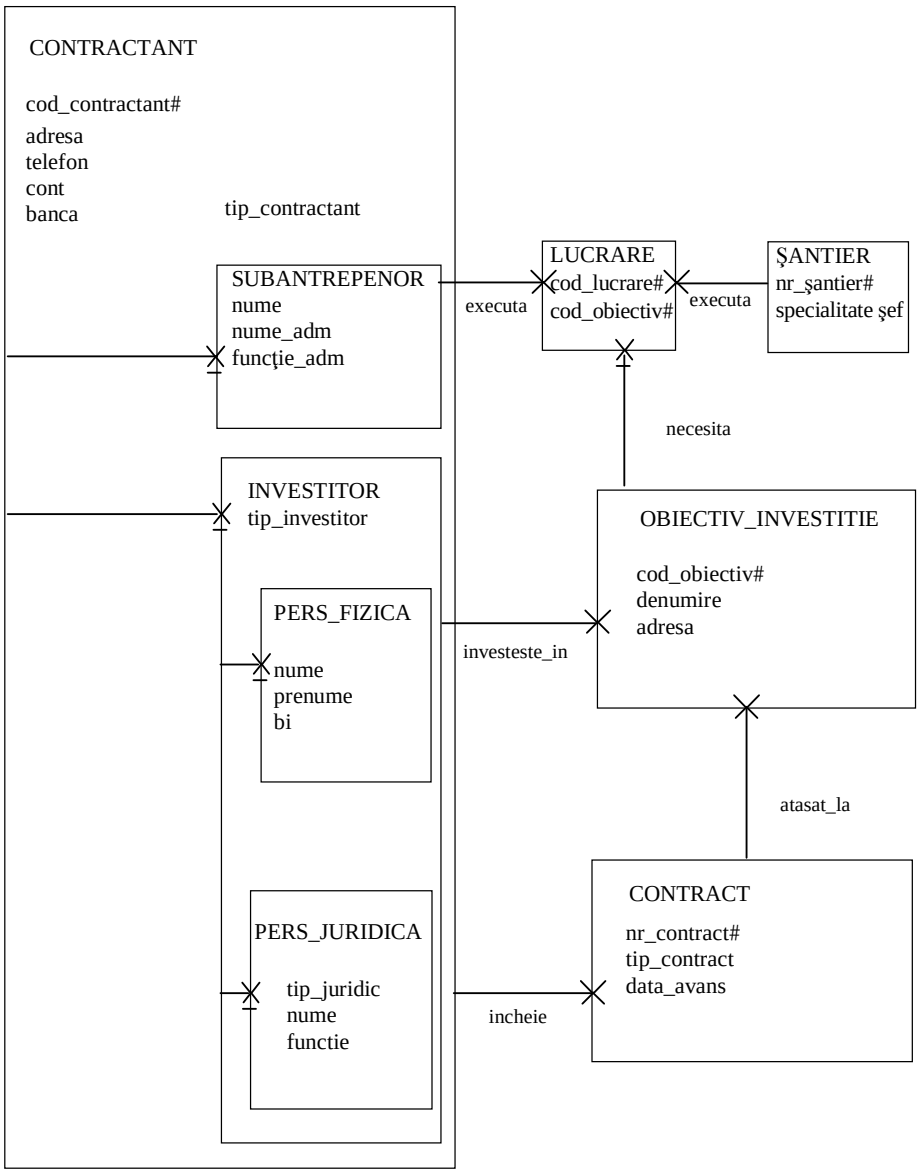
PERS_JURIDICA(cod_contractant#, tip_juridic, nume, reprez_legal, functie);

CONTRACT(nr_contract#, tip_contract, data_incheiere, garantie, val_investitie, durate_executie, cont, banca, perioada, avans, data_avans, cod_contractant);

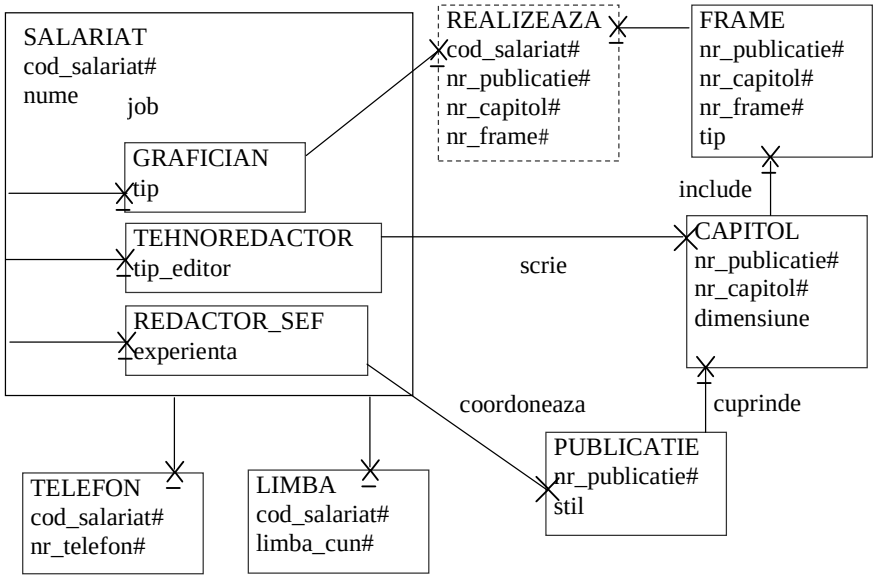
SANTIER(nr_santier#, specialitate, sef);

OBIECTIV_INVESTITIE(cod_obiectiv#, denumire, adresa, adc, nr_cert_urb, nr_aut_constr, nr_contract, cod_contractant);

LUCRARE(cod_lucrare#, cod_obiectiv#, tip_lucrare, nume, data_inc, data_sf, nr_santier, cod_contractant);


CONTRACTANT
cod_contractant#
adresa
telefon
cont
banca
tip_contractant
SUBANTREPENOR
nume
nume_adm
funcţie_adm
executa
LUCRARE
cod_lucrare#
cod_obiectiv#
executa
ŞANTIER
nr_şantier#
specialitate şef
necesita
INVESTITOR
tip_investitor
OBIECTIV_INVESTITIE
cod_obiectiv#
denumire
adresa
PERS_FIZICA
nume
prenume
bi
investeste_in
atasat_la
PERS_JURIDICA
tip_juridic
nume
functie
incheie
CONTRACT
nr_contract#
tip_contract
data_avans

Gestiunea activităţilor de editare dintr-o editură



SALARIAT(cod_salariat#, nume, prenume, vechime, salariu, job);

GRAFICIAN(cod_salariat#, tip);

TEHNOREDACTOR(cod_salariat#, tip_platforma, tip_editor, viteza);

REDACTOR_SEF(cod_salariat#, experienta);

LIMBA(cod_salariat#, limba_cunos#);

TELEFON(cod_salariat#, nr_telefon#);

REALIZEAZA(cod_salariat#, nr_frame#, nr_publicatie#, nr_capitol#, data_inc, data_lim);

FRAME(nr_frame#, nr_publicatie#, nr_capitol#, tip, dim, format);

CAPITOL(nr_publicatie#, nr_capitol#, dimensiune, cod_salariat);

PUBLICATIE(nr_publicatie#, stil, beneficiar, autor, cod_salariat, cost, titlu, limba).

Exemple → curs!

Algebra relaţională

Limbajul de definire a datelor (LDD) precizează entităţile, relaţiile dintre ele, atributele, structura atributelor, cheile, constrângerile, prin urmare defineşte structura obiectelor bazei de date (schema bazei).

Limbajul de prelucrare a datelor (LMD) dintr-o bază de date relaţionale cuprinde aspecte referitoare la introducerea, eliminarea, modificarea şi căutarea datelor.

- **Introducerea** datelor – permite adăugarea de tupluri la o relaţie. Tuplurile pot fi introduse de utilizator sau pot fi obţinute din alte relaţii existente în baza de date.
- **Eliminarea** datelor – permite ştergerea tuplurilor ce satisfac condiţii date.
- **Modificarea** datelor – permite reactualizarea tuplurilor ce satisfac condiţii date cu noi valori ale atributelor sau cu rezultate ale unor operaţii aritmetice efectuate asupra unor valori existente.
- **Căutarea** datelor – permite găsirea tuplurilor sau a unor părţi ale tuplurilor ce satisfac condiţii date.

Modelul relaţional oferă două mulţimi de operatori pe relaţii:

- algebra relaţională (filtrele se obţin aplicând operatori specializaţi asupra uneia sau mai multor relaţii din cadrul bazei relaţionale);
- calculul relaţional (filtrele se obţin cu ajutorul unor formule logice pe care tuplurile rezultatului trebuie să le satisfacă).

Algebra relaţională a fost introdusă de E.F. Codd ca o mulţime de operaţii formale acţionând asupra unor relaţii şi având ca rezultat alte relaţii. Baza teoretică pentru limbajele de interogare relaţionale o constituie operatorii introduşi de Codd pentru prelucrarea relaţiilor.

Operatorii modelului relaţional definesc operaţiile care se pot efectua asupra relaţiilor în scopul realizării funcţiilor de prelucrare asupra BD.

Operatorii sunt numai pentru citire (nu actualizeaza operanzi)!!!

Scopul fundamental al algebrei relationale este de a permite scrierea expresiilor relationale. Expresiile servesc ca o reprezentare de nivel superior, simbolică, a intenţiilor utilizatorului şi pot fi supuse unei diversităţi de reguli de transformare (optimizare).

Relaţiile sunt închise faţă de algebra relaţională (operanzii şi rezultatele sunt relaţii → ieşirea unei operaţii poate deveni intrare pentru alta) → posibilitatea imbricării rxpresiilor în algebra relaţională).

Operatorii algebrei relaţionale sunt:

- operatori tradiţionali pe mulţimi (UNION, INTERSECT, PRODUCT, DIFFERENCE);
- operatori relaţionali speciali (PROJECT, SELECT, JOIN, DIVISION).

Calculul relaţional reprezintă o adaptare a calculului predicatelor la domeniul bazelor de date relaţionale. Ideea de bază este de a identifica o relaţie cu un predicat. Pe baza unor predicate iniţiale, prin aplicarea unor operatori ai calculului cu predicate (conjuncţia, disjuncţia, negaţia, cuantificatorul existenţial şi cel universal) se pot defini noi relaţii.

Calculul relaţional poate să fie orientat pe tupluri sau orientat pe domenii.

Echivalenţa dintre algebra relaţională şi calculul relaţional a fost demonstrată de J.D.Ullman. Această echivalenţă arată că orice relaţie posibil de definit în algebra relaţională poate fi definită şi în cadrul calcului relaţional, şi reciproc.

Operatorii (unari sau binari) algebrei relaţionale realizează următoarele funcţii:

- SELECT (selecţie) – extrage tupluri ce satisfac o condiţie specificată;
- PROJECT (proiecţie) – extrage atributele specificate;
- DIFFERENCE (diferenţă) – extrage tupluri care apar într-o relaţie, dar nu apar în cealaltă;
- PRODUCT (produs cartezian) – generează toate perechile posibile de tupluri, primul element al perechii fiind luat din prima relaţie, iar cel de-al doilea element din cealaltă relaţie;
- UNION (reuniune) – reuneşte două relaţii;
- INTERSECT (intersecţie) – extrage tupluri care apar în ambele relaţii;
- DIVISION (diviziune) – extrage valorile atributelor dintr-o relaţie, care apar în toate valorile atributelor din cealaltă relaţie;
- JOIN (compunere) – extrage tupluri din mai multe relaţii corelate:
- NATURAL JOIN (compunere naturală) – combină tupluri din două relaţii, cu condiţia ca atributele comune să aibă valori identice;

- SEMI-JOIN (semi-compunere) – selectează tupluri ce aparţin unei singure relaţii, care sunt corelate cu tupluri din cea de a doua relaţie;
- Θ-JOIN (Θ-compunere) – combină tupluri din două relaţii (nu neaparat corelate), cu condiţia ca valorile atributelor specificate să satisfacă o anumită condiţie;
- OUTER JOIN (compunere externă) – combină tupluri din două relaţii, astfel încât condiţiile de corelare să fie satisfăcute. Tuplurile din orice relaţie care nu satisfac aceste condiţii sunt completate cu valori *null*.

Pentru operatorii UNION, INTERSECT şi DIFFERENCE, se presupune că sunt aplicaţi numai la relaţii având aceeaşi aritate, iar ordinea (nu numele) atributelor este aceeaşi.

Operatorul PROJECT

Proiecţia este o operaţie unară care elimină anumite atribute ale unei relaţii producând o submulţime „pe verticală“ a acesteia. Suprimarea unor atribute poate avea ca efect apariţia unor tupluri duplicate, care trebuie eliminate.

Prin proiecţie se construieşte dintr-o relaţie *R*, o nouă relaţie:

a) ştergând din *R* atributele care nu sunt menţionate în parametrii proiecţiei;

b) eliminând dublurile care apar după ştergere.

Pentru a reprezenta operatorul proiecţie sunt utilizate diferite **notaţii**:

$\Pi_{A1, ..., Am}$ (R) PROJECT (R, A_1, ..., A_m)

$R[A_1, ..., A_m]$

unde A_1, A_2, ..., A_m sunt parametrii proiecţiei relativ la relaţia *R*.

***Exemplu*.** Să se obţină o listă ce conţine numele, prenumele şi sexul angajaţilor.

1. Proiecţie în algebra relaţională:

Rezultat = PROJECT(SALARIAT, nume, prenume, sex)

2. Proiecţie cu dubluri în SQL:

```
SELECT   nume, prenume, sex
FROM     salariat;
```

3. Proiecţie fără dubluri în SQL:

```
SELECT   DISTINCT nume, prenume, sex
FROM     salariat;
```

Operatorul SELECT

Selecţia (restrictia) este o operaţie unară care produce o submulţime pe „orizontală“ a unei relaţii *R*. Această submulţime se obţine prin extragerea tuplurilor din *R* care satisfac o condiţie specificată.

Sunt utilizate diferite notaţii:

$\sigma_{condiţie}(R)$ *R*[condiţie]

SELECT(*R*, condiţie) RESTRICT(*R*, condiţie).

***Exemplu*.** Să se obţină informaţii complete despre angajaţii de sex masculin.

1. Selecţie în algebra relaţională:

Rezultat = SELECT(SALARIAT, sex = ‘m’)

2. Selecţie în SQL:

```
SELECT   *
FROM     salariat
WHERE    sex = 'm';
```

Operatorul UNION

Reuniunea a două relaţii *R* şi *S* este mulţimea tuplurilor aparţinând fie lui *R*, fie lui *S*, fie ambelor relaţii.

Sunt utilizate notaţiile:

$R \cup S$

UNION(*R*, *S*)

OR(*R*, *S*)

APPEND(*R*, *S*).

***Exemplu*.** Să se obţină lista cu numele persoanelor fizice şi a subantreprenorilor.

```
SELECT   nume
FROM     subantreprenor
  UNION
SELECT   nume
FROM     pers_fizica;
```

Operatorul DIFFERENCE

Diferenţa a două relaţii *R* şi *S* este mulţimea tuplurilor care aparţin lui *R*, dar nu aparţin lui *S*. Diferenţa este o operaţie binară necomutativă care permite obţinerea tuplurilor ce apar numai într-o relaţie.

Sunt utilizate diferite notaţii:

R – S

DIFFERENCE(*R*, *S*)

REMOVE(*R*, *S*)

MINUS(*R*, *S*).

***Exemplu*.** Să se obţină lista cu numărul contractului, tipul contractului, valoarea investiţiei şi durata lucrării pentru contractele de subantrepriză pentru care valoarea investiţiei nu depăşeşte 60000$.

1. Diferenţă în algebra relaţională:

R=PROJECT(SELECT(CONTRACT, tip_contract=T),
nr_contract, tip_contract, val_investitie, durata_lucrare);
S=PROJECT(SELECT(CONTRACT, val_investitie > 60000),
nr_contract, tip_contract, val_investitie, durata_lucrare);
Rezultat = DIFFERENCE(*R*, *S*)

2. Diferenţa în SQL:

```
SELECT   nr_contract,tip_contract,
         val_investitie,durata_lucrare
FROM     contract
WHERE    tip_contract
   MINUS
SELECT   nr_contract,tip_contract,
         val_investitie,durata_lucrare
FROM     contract
WHERE    val_investitie>60000;
```

Evident diferenţa se poate referi la tabele diferite! Implementaţi cererea prin care se listează toate oraşele în care se află o filială, dar nici o proprietate.

Operatorul INTERSECT

Intersecţia a două relaţii *R* şi *S* este mulţimea tuplurilor care aparţin şi lui *R* şi lui *S*. Operatorul INTERSECT este un operator binar, comutativ, derivat:

$$R \cap S = R - (R - S)$$

$$R \cap S = S - (S - R).$$

Sunt utilizate diferite notaţii:

INTERSECT(*R*, *S*)

$R \cap S$

AND(*R*, *S*).

În anumite dialecte SQL există operator special (INTERSECT), care realizează această operaţie. Operatorii INTERSECT şi DIFFERENCE pot fi simulaţi în SQL (în cadrul comenzii SELECT) cu ajutorul opţiunilor EXISTS, NOT EXISTS, IN, != ANY.

Exemplu. Utilizând tabelele *agent_teritorial* şi *programator* să se obţină lista codurilor salariaţilor care sunt programatori, dar care lucrează şi ca agenţi teritoriali.

1. Intersecţie în algebra relaţională:

R = PROJECT(AGENT_TERITORIAL, cod_salariat);
S = PROJECT(PROGRAMATOR, cod_salariat),
Rezultat = INTERSECT(*R*, *S*).

2. Intersecţie în SQL:

```
SELECT   cod_salariat
FROM     agent_teritorial
  INTERSECT
SELECT   cod_salariat
FROM     programator;
```

3. Simularea intersecţiei în SQL:

```
SELECT   cod_salariat
FROM     programator p
WHERE EXISTS
     (SELECT  cod_salariat
      FROM    agent_teritorial a
      WHERE   p.cod_salariat=a.cod_salariat);
```

Operatorul PRODUCT

Fie *R* şi *S* relaţii de aritate *m*, respectiv *n*. Produsul cartezian al lui *R* cu *S* este mulţimea tuplurilor de aritate $m + n$ unde primele *m* componente formează un tuplu în *R*, iar ultimele *n* componente formează un tuplu în *S*.

Sunt utilizate diferite notaţii:

$R \times S$

PRODUCT(R, S)

TIMES(R, S).

Exemplu. Să se obţină lista tuturor posibilităţilor de investiţie în diverse obiective de către o firmă care este persoană juridică.

1. Produs cartezian în algebra relaţională:

R = PROJECT(PERS_JURIDICA, nume, cod_contractant);
S = PROJECT(OBIECTIV_INVESTITIE, denumire);
Rezultat = PRODUCT(R, S).

2. Produs cartezian în SQL:

```
SELECT   cod_contractant, nume, denumire
FROM     obiectiv_investitie, pers_juridica;
```

Operatorul DIVISION

Diviziunea este o operaţie binară care defineşte o relaţie ce conţine valorile atributelor dintr-o relaţie care apar **în toate** valorile atributelor din cealaltă relaţie.

Sunt utilizate diferite notaţii:

DIVIDE(R, S)

DIVISION(R, S)

$R \div S$.

Diviziunea conţine acele tupluri de dimensiune $n - m$ la care, adăugând orice tuplu din S, se obţine un tuplu din R.

Operatorul diviziune poate fi exprimat formal astfel:

$$R^{(n)} \div S^{(m)} = \{t^{(n-m)} \mid \forall\, s \in S, (t, s) \in R\} \text{ unde } n > m \text{ şi } S \neq \varnothing.$$

Operatorul DIVISION este legat de cuantificatorul universal ($\forall$) care nu există în SQL. Cuantificatorul universal poate fi însă simulat cu ajutorul cuantificatorului existenţial ($\exists$) utilizând relaţia:

$$\forall x\, P(x) \equiv \neg\, \exists\, x \neg\, P(x).$$

Prin urmare, operatorul DIVISION poate fi exprimat în SQL prin succesiunea a doi operatori NOT EXISTS.

Exemplu. Să se obţină codurile salariaţilor ataşaţi tuturor proiectelor pentru care s-a alocat un buget egal cu 1000.

1. Diviziune în algebra relaţională:

R = PROJECT(ATASAT_LA, cod_salariat, nr_proiect);
S = PROJECT(SELECT(PROIECT, buget = 1000), nr_proiect);
Rezultat = DIVISION(*R*, *S*).

2. Diviziune în SQL:

```
SELECT   UNIQUE cod_salariat
FROM     atasat_la sx
WHERE NOT EXISTS
      (SELECT *
      FROM    proiect pp
      WHERE   proiect.buget='1000'
      AND NOT EXISTS
         (SELECT  *
         FROM     atasat_la bb
         WHERE         pp.nr_proiect=bb.nr_proiect
         AND   bb.cod_salariat=sx.cod_salariat));
```

3. Simularea diviziunii cu ajutorul funcţiei COUNT:

```
SELECT   cod_salariat
FROM     atasat_la
WHERE    nr_proiect IN
     (SELECT  nr_proiect
      FROM    proiect
      WHERE   buget=1000)
GROUP BY cod_salariat
HAVING   COUNT(nr_proiect)=
              (SELECT  COUNT(*)
               FROM    proiect
               WHERE   buget=1000);
```

Operatorul JOIN

Operatorul de compunere (uniune) permite regăsirea informaţiei din mai multe relaţii corelate. Operatorul combină produsul cartezian, selecţia şi proiecţia.

Operatorul NATURAL JOIN

Operatorul de compunere naturală (NATURAL JOIN) combină tupluri din două relaţii *R* şi *S*, cu condiţia ca atributele comune să aibă valori identice.

Algoritmul care realizează compunerea naturală este următorul:

1. se calculează produsul cartezian $R \times S$;

2. pentru fiecare atribut comun A care defineşte o coloană în R şi o coloană în S, se selectează din $R \times S$ tuplurile ale căror valori coincid în coloanele $R.A$ şi $S.A$ (atributul $R.A$ reprezintă numele coloanei din $R \times S$ corespunzătoare coloanei A din R);
3. pentru fiecare astfel de atribut A se proiectează coloana $S.A$, iar coloana $R.A$ se va numi A.

Operatorul NATURAL JOIN poate fi exprimat formal astfel:

$$\text{JOIN}(R, S) = \Pi_{i1,\ldots,im}\, \sigma_{(R.A1 = S.A1) \wedge \ldots \wedge (R.Ak = S.Ak)}(R \times S),$$

unde A_1, ..., A_k sunt atributele comune lui R şi S, iar i_1, ..., i_m reprezintă lista componentelor din $R \times S$ (păstrând ordinea iniţială) din care au fost eliminate componentele $S.A_1$, ..., $S.A_k$.

Exemplu. Să se obţină informaţii complete despre angajaţi şi departamentele în care lucrează.

1. Operatorul de compunere naturală în algebra relaţională:

Rezultat = JOIN(SALARIAT, DEPARTAMENT).

2. Operatorul de compunere naturală în SQL:

```
SELECT   *
FROM     salariat, departament
WHERE    nr_depart = cod_departament;
```

Operatorul θ-JOIN

Operatorul **θ**-JOIN combină tupluri din două relaţii (nu neapărat corelate) cu condiţia ca valorile atributelor specificate să satisfacă o anumită condiţie specificată explicit în cadrul operaţiei.

Operatorul **θ**-JOIN este un operator derivat, fiind o combinaţie de produs scalar şi selecţie:

$$\text{JOIN}(R, S, \text{condiţie}) = \sigma_{\text{condiţie}}\,(R \times S)$$

Exemplu. Să se afişeze pentru fiecare salariat, codul acestuia şi grila sa de salarizare.

```
SELECT   empno, level
FROM     salgrade, emp
WHERE    sal BETWEEN losal AND hisal;
```

Exemplu. Să se obţină informaţii despre contractanţi (codul şi banca) şi obiectivele de investiţie asociate acestora (denumire, număr certificat de urbanizare) cu condiţia ca obiectivele să nu fie la aceeaşi adresă ca şi contractanţii.

1. Operatorul **θ**-JOIN în algebra relaţională:

R = PROJECT(CONTRACTANT, cod_contractant, banca);
S = PROJECT(OBIECTIV_INVESTITIE, denumire, nr_cert_urb);
Rezultat = JOIN(*R*, *S*, OBIECTIV_INVESTITIE.adresa <> CONTRACTANT.adresa).

2. Opratorul **θ**-JOIN în SQL:

```
SELECT   cod_contractant,banca, nr_cert_urb,
         denumire
FROM     contractant a,obiectiv_investitie b
WHERE    b.adresa <> a.adresa;
```

Operatorul SEMI-JOIN

Operatorul SEMI-JOIN conservă atributele unei singure relaţii participante la compunere şi este utilizat când nu sunt necesare toate atributele compunerii. Operatorul este asimetric.

Tupluri ale relaţiei R care participă în compunerea (naturală sau θ-JOIN) dintre relaţiile R şi S.

SEMI-JOIN este un operator derivat, fiind o combinaţie de proiecţie şi compunere naturală sau proiecţie şi θ-JOIN:

$$\text{SEMIJOIN}(R, S) = \Pi_M (\text{JOIN}(R, S))$$

$$\text{SEMIJOIN}(R, S, \text{condiţie}) = \Pi_M (\text{JOIN}(R, S, \text{condiţie})),$$

unde am notat prin *M* atributele relaţiei *R*.

Exemplu. Să se obţină informaţii referitoare la persoanele fizice (nume, buletin) care investesc în obiective cu caracter recreativ.

1. Operatorul SEMI-JOIN în algebra relaţională:

R = SELECT(OBIECTIV_INVESTITIE, denumire = ’cabana’ OR denumire = ’casa de vacanta’)
S = JOIN(PERS_FIZICA, *R*)
Rezultat = PROJECT(*S*, nume, buletin).

2. Operatorul SEMI-JOIN în SQL:

```
SELECT   nume,bi
FROM     pers_fizica a,obiectiv_investitie b
WHERE    a.cod_contractant = b.cod_contractant
AND      (denumire=’cabana’)OR (denumire= ’casa
          de vacanta’);
```

Operatorul OUTER JOIN

Operaţia de compunere externă combină tupluri din două relaţii pentru care sunt satisfăcute condiţiile de corelare. În cazul aplicării operatorului JOIN se pot pierde tupluri, atunci când există un tuplu în una din relaţii pentru care nu există nici un tuplu în cealaltă relaţie, astfel încât să fie satisfăcută relaţia de corelare.

Operatorul elimină acest inconvenient prin atribuirea valorii *null* valorilor atributelor care există într-un tuplu din una dintre relaţiile de intrare, dar care nu există şi în cea de-a doua relaţie.

Practic, se realizează compunerea a două relaţii *R* şi *S* la care se adaugă tupluri din *R* şi *S*, care nu sunt conţinute în compunere, completate cu valori *null* pentru atributele care lipsesc.

Compunerea externă poate fi: LEFT, RIGHT, FULL. De exemplu, OUTER JOIN LEFT reprezintă compunerea în care tuplurile din R, care nu au valori similare în coloanele comune cu relaţia S, sunt de asemenea incluse în relaţia rezultat.

Exemplu. Să se obţină informaţii referitoare la persoanele fizice care sunt investitori (chiar dacă nu au investit în obiective industriale) şi la obiectivele de investiţie industriale (chiar şi cele care nu sunt construite de persoane fizice).

R = SELECT(OBIECTIV_INVESTITIE, denumire = 'industrial')
Rezultat = OUTERJOIN(PERS_FIZICA, *R*).

Operatorii algebrei relaţionale pot fi reprezentaţi grafic cu ajutorul unor simboluri speciale. → curs!

Operaţii adiţionale: complement, despicare, închidere tranzitivă. Funcţii asociate: MIN, MAX, COUNT, AVG, SUM, VAR, STD etc.

Evaluarea şi optimizarea interogărilor

Procesarea interogărilor

O **expresie** a algebrei relaţionale este constituită din relaţii legate prin operaţii din algebra relaţională. O expresie se poate reprezenta grafic cu ajutorul unui arbore, numit **arbore algebric**, în care nodurile corespund operatorilor din cadrul expresiei respective.

Evaluarea unei expresii presupune efectuarea prelucrărilor indicate de operatorii din expresie în ordinea apariţiilor sau în ordinea fixată prin paranteze. Rezultatul evaluării unei expresii este o relaţie derivată din relaţiile menţionate ca operanzi în cadrul expresiei.

Două expresii sunt **echivalente**, dacă în urma evaluării lor se obţine ca rezultat aceeaşi relaţie.

Exemple referitoare la moduri echivalente de exprimare a unei cereri (vor fi rezolvate la curs!).

1. Informaţii despre salariaţii care nu contribuie la machetarea nici unei publicaţii, dar au retribuţia mai mare decât o valoare dată.

2. Codul şi numele subantreprenorilor care au realizat lucrări specializate la obiective case de vacanţă sau cabane.

3. Codurile şi telefoanele investitorilor, valoarea si durata de execuţie a investitiilor a caror valoare este între două limite specificate.

4. Perioada de desfăşurare şi preţul ofertelor care încep după 1 ianuarie 2003 şi sunt:

- sejururi la munte;
- excursii în care autocarele sunt conduse de şoferi angajaţi după 1 mai 1987 şi supravegheate de ghizi ce cunosc limba engleză care au făcut specializare în Suedia.

În majoritatea sistemelor de gestiune, şi în special în cele relaţionale, interfaţa cu utilizatorul este de tip **neprocedural**. Utilizatorul defineşte datele pe care doreşte să le vizualizeze fără a da algoritmii de acces. Sistemul trebuie să convertească cererea utilizatorului:

- într-o cerere optimală;
- în proceduri de acces optimal la datele fizice.

Garantarea absolută a performanţelor optime pentru procesorul limbajului relaţional este imposibilă. Corectă ar fi utilizarea cuvântului „ameliorare“ în locul cuvântului „optimizare“.

Evaluarea unei interogări se efectuează în trei etape.

- ❒ Analiza cererii presupune studierea sintactică şi semantică a cererii pentru a verifica corectitudinea sa şi a simplifica criteriul de căutare.
- ❒ Ordonanţarea presupune descompunerea cererii într-o mulţime de operaţii elementare şi determinarea unei ordini optimale a acestor operaţii. Operaţiile sunt, în general, cele ale algebrei relaţionale. La sfârşitul etapei se obţine un plan de execuţie pentru cerere.
- ❒ Execuţia presupune efectuarea (paralel şi/sau secvenţială) operaţiilor elementare furnizate de planul de execuţie pentru a obţine rezultatul cererii.

Presupunem că utilizatorul transmite sistemului de gestiune o cerere exprimată prin ordine SQL. Pentru a răspunde cererii, SGBD-ul trebuie să înţeleagă cererea utilizatorului. Cererea trebuie să fie corectă sintactic, datele trebuie să fie disponibile utilizatorului şi trebuie localizate analizând diferite drumuri de acces la ele. Aceste funcţii sunt realizate de SGBD cu ajutorul a două module funcţionale care comunică permanent:

- **analizorul cererilor**, care asigură verificarea sintactică şi semantică a cererii, localizarea datelor implicate în cerere (găsirea adresei blocurilor ce conţin datele), furnizarea planului de execuţie.
- **administratorul datelor** (executorul), care execută efectiv cererea (primeşte planurile de execuţie furnizate de modulul de optimizare şi le execută). Execuţia presupune căutarea blocurilor ce conţin datele şi transferul blocurilor în memoria *cache.*

Ideea generală:

cerere→ arbore algebric (nu este unic) → plan de executie → optimizare

Un plan de execuţie implică o secvenţă de paşi pentru evaluarea cererii (în mod obişnuit, fiecare pas din planul de execuţie corespunde unei operaţii relaţionale) precum şi metoda care va fi folosită pentru evaluarea operaţiei. De obicei, pentru o operaţie relaţională dată, există mai multe metode ce pot fi folosite pentru evaluarea acesteia.

Două planuri de execuţie diferite care au întotdeauna acelaşi rezultat se numesc echivalente. **Planuri de execuţie echivalente pot avea diferite costuri**. Scopul optimizării cererilor este de a găsi, printre diversele planuri de execuţie echivalente, pe acela de cost minim. Într-un sistem centralizat, costul evaluării unei cereri este suma a două componente, costul I/O (transferuri de date) şi costul CPU (verificare de condiţii, operaţii *join* etc.).

Ordinea de execuţie a operaţiilor

O interogare constă dintr-un număr de operaţii. Ordinea în care se efectuează operaţiile are un rol important în evaluarea costului necesar realizării interogării.

Există două modalităţi de abordare pentru a determina ordinea de execuţie a operaţiilor:

- algebric;
- bazat pe estimarea costului.

Ambele folosesc o mulţime de reguli care permit transformarea unui plan de execuţie (reprezentat ca o expresie scrisă în termenii algebrei relaţionale) în altul, echivalent.

Optimizarea cererilor bazată pe algebra relaţională se realizează în două etape:

- se exprimă cererile sub forma unor expresii algebrice relaţionale;
- se aplică acestor expresii transformări algebrice care conduc la expresii echivalente, dar care vor fi executate mai eficient.

Procesul de transformare a cererilor se realizează conform unei strategii de optimizare care poate să fie:

- independentă de modul de memorare a datelor (strategie generală);
- dependentă de modul de memorare (strategie specifică unui anumit SGBD).

Proprietăţile operatorilor algebrei relaţionale

Proprietatea 1. Comutativitatea operaţiilor de *join* şi produs cartezian:

$$\text{JOIN}(R1, R2) = \text{JOIN}(R2, R1)$$
$$R1 \times R2 = R2 \times R1$$

Proprietatea 2. Asociativitatea operaţiilor de *join* şi produs cartezian:

$$\text{JOIN}(\text{JOIN}(R1, R2), R3) = \text{JOIN}(R1, \text{JOIN}(R2, R3))$$
$$(R1 \times R2) \times R3 = R1 \times (R2 \times R3)$$

Proprietatea 3. Compunerea proiecţiilor:

$$\Pi_{A1,\ldots,Am}(\Pi_{B1,\ldots,Bn}(R)) = \Pi_{A1,\ldots,Am}(R),$$

unde $\{A_1, A_2,\ldots,A_m\} \subseteq \{B_1, B_2,\ldots,B_n\}$.

Proprietatea 4. Compunerea selecţiilor:

$$\sigma_{cond1}(\sigma_{cond2}(R)) = \sigma_{cond1 \wedge cond2}(R) = \sigma_{cond2}(\sigma_{cond1}(R)),$$

unde am notat prin *cond* condiţia după care se face selecţia.

Proprietatea 5. Comutarea selecţiei cu proiecţia:

$$\Pi_{A1,...,Am}(\sigma_{cond}(R)) = \sigma_{cond}(\Pi_{A1,...,Am}(R)),$$

unde condiţia după care se face selecţia implică numai atributele $A_1,...,A_m$.

Dacă condiţia implică şi atributele $B_1,...,B_n$, care nu aparţin mulţimii $\{A_1,...,A_m\}$, atunci:

$$\Pi_{A1,...,Am}(\sigma_{cond}(R)) = \Pi_{A1,...,Am}(\sigma_{cond}(\Pi_{A1,...,Am,B1,...,Bn}(R)))$$

Proprietatea 6. Comutarea selecţiei cu produsul cartezian:

Dacă toate atributele menţionate în condiţia după care se face selecţia sunt atribute ale relaţiei $R1$, atunci:

$$\sigma_{cond}(R1 \times R2) = \sigma_{cond}(R1) \times R2$$

Dacă condiţia este de forma $cond1 \wedge cond2$ şi dacă $cond1$ implică numai atribute din $R1$, iar $cond2$ implică numai atribute din $R2$, atunci

$$\sigma_{cond}(R1 \times R2) = \sigma_{cond1}(R1) \times \sigma_{cond2}(R2)$$

Dacă $cond1$ implică numai atribute din $R1$, iar $cond2$ implică atribute atât din $R1$ cât şi din $R2$, atunci:

$$\sigma_{cond}(R1 \times R2) = \sigma_{cond2}(\sigma_{cond1}(R1) \times R2)$$

Proprietatea 7. Comutarea selecţiei cu reuniunea:

$$\sigma_{cond}(R1 \cup R2) = \sigma_{cond}(\mathrm{R1}) \cup \sigma_{cond}(R2)$$

Proprietatea 8. Comutarea selecţiei cu diferenţa:

$$\sigma_{cond}(R1 - R2) = \sigma_{cond}(R1) - \sigma_{cond}(R2)$$

Proprietatea 9. Comutarea proiecţiei cu reuniunea:

$$\Pi_{A1,...,Am}(R1 \cup R2) = \Pi_{A1,...,Am}(R1) \cup \Pi_{A1,...,Am}(R2)$$

Proprietatea 10. Comutarea proiecţiei cu produsul cartezian:

Dacă $A_1,...,A_m$ este o listă de atribute ce apar în schemele relaţionale $R1$ şi $R2$ şi dacă lista este formată din atribute aparţinând lui $R1$ (notate prin $B_1,...,B_n$) şi din atribute aparţinând lui $R2$ (notate prin $C_1,...,C_k$) atunci:

$$\Pi_{A1,...,Am}(R1 \times R2) = \Pi_{B1,...,Bn}(R1) \times \Pi_{C1,...,Ck}(R2)$$

Proprietatea 11. Compunerea proiecţiei cu operaţia *join*:

Dacă $A_1,...,A_m$ este o listă de atribute ce apar în schemele relaţionale $R1$ şi $R2$ şi dacă lista este formată din atribute aparţinând lui $R1$ (notate prin $B_1,...,B_n$) şi din atribute aparţinând lui $R2$ (notate prin $C_1,...,C_k$) atunci:

$$\Pi_{A1,...,Am}(\mathrm{JOIN}(R1,R2,D)) = \Pi_{A1,...,Am}(\mathrm{JOIN}(\Pi_{D,B1,...,Bn}(R1), \Pi_{D,C1,...,Ck}(R2),D),$$

unde am notat prin JOIN($R1$, $R2$, D) operaţia de compunere naturală între $R1$ şi $R2$ după atributul comun D.

Proprietatea 12. Compunerea selecţiei cu operaţia *join*:

$$\sigma_{cond}\,(\text{JOIN}\,(R1, R2, D)) = \sigma_{cond}\,(\text{JOIN}\,(\Pi_{D,A}\,(R1), \Pi_{D,A}\,(R2), D)),$$

unde *A* reprezintă atributele care apar în condiţia după care se face selecţia.

Reguli de optimizare frecvent folosite:

Regula de optimizare 1. Selecţiile se execută cât mai devreme posibil. Motivaţia acestei reguli este că selecţiile reduc substanţial dimensiunea relaţiilor. Regula de transformare 4 poate fi folosită pentru a separa două sau mai multe selecţii în selecţii individuale care pot fi distribuite *join*-ului sau produsului cartezian folosind comutarea selecţiei cu *join*-ul.

Regula de optimizare 2. Produsurile carteziene se înlocuiesc cu *join*-uri, ori de câte ori este posibil. Un produs cartezian între două relaţii este de obicei mult mai scump (ca şi cost) decât un *join* între cele două relaţii, deoarece primul generează concatenarea tuplurilor în mod exhaustiv şi poate genera un rezultat foarte mare. Această transformare se poate realiza folosind legătura dintre produs cartezian, *join* şi selecţie.

Regula de optimizare 3. Dacă sunt mai multe *join*-uri atunci cel care se execută primul este cel mai restrictiv. Un *join* este mai restrictiv decât altul dacă produce o relaţie mai mică. Se poate determina care *join* este mai restrictiv pe baza factorului de selectivitate sau cu ajutorul informaţiilor statistice. Algebric, acest lucru se poate realiza folosind regula de transformare 2.

Regula de optimizare 4. Proiecţiile se execută la început pentru a îndepărta atributele nefolositoare. Dacă un atribut al unei relaţii nu este folosit în operaţiile ulterioare atunci trebuie îndepărtat. În felul acesta se va folosi o relaţie mai mică în operaţiile ulterioare. Aceasta se poate realiza folosind comutarea proiecţiei cu *join*-ul.

Exemple curs!!!

Regulile lui Codd

Caracteristici ale modelului relaţional:

- nu există tupluri identice;
- ordinea liniilor şi a coloanelor este arbitrară;
- articolele unui domeniu sunt omogene;
- fiecare coloană defineşte un domeniu distinct şi nu se poate repeta în cadrul aceleiaşi relaţii;

- toate valorile unui domeniu corespunzătoare tuturor cazurilor nu mai pot fi descompuse în alte valori (sunt atomice).

Avantajele modelului relaţional:

- fundamentare matematică riguroasă;
- independenţă fizică a datelor;
- posibilitatea filtrărilor;
- existenţa unor structuri de date simple;
- realizarea unei redundanţe minime;
- supleţe în comunicarea cu utilizatorul neinformatician.

Ca **limite** ale modelului relaţional putem menţiona:

- rămâne totuşi redundanţă,
- ocupă spaţiu,
- apar fenomene de inconsistenţă,
- nu există mecanisme pentru tratarea optimă a cererilor recursive,
- nu lucrează cu obiecte complexe,
- nu există mijloace perfecţionate pentru exprimarea constrângerilor de integritate,
- nu realizează gestiunea totala a datelor distribuite,
- nu realizează gestiunea cunoştinţelor.

În anul 1985, E.F. Codd a publicat un set de 13 reguli în raport cu care un sistem de gestiune a bazelor de date poate fi apreciat ca relaţional. Nici un sistem de gestiune a bazelor de date pus în vânzare pe piaţa comercială nu respectă absolut toate regulile definite de Codd, dar acest lucru nu împiedică etichetarea acestor sisteme drept relaţionale.

Nu trebuie apreciat un SGBD ca fiind relaţional sau nu, ci măsura în care acesta este relaţional, deci numărul regulilor lui Codd pe care le respectă.

Regula 1 – regula gestionării datelor. *Un SGBD relaţional trebuie să fie capabil să gestioneze o bază de date numai prin posibilităţile sale relaţionale.*

Regula 2 – regula reprezentării informaţiei. *Într-o bază de date relaţională, informaţia este reprezentată la nivel logic sub forma unor tabele ce poartă numele de relaţii*.

Regula 3 – regula accesului garantat la date. *Fiecare valoare dintr-o bază de date relaţională trebuie să poată fi adresată în mod logic printr-o combinaţie formată din numele relaţiei, valoarea cheii primare şi numele atributului.*

Regula 4 – regula reprezentării informaţiei necunoscute. *Un sistem relaţional trebuie să permită utilizatorului definirea unui tip de date numit „null“ pentru reprezentarea unei informaţii necunoscute la momentul respectiv.*

Regula 5 – regula dicţionarelor de date. *Asupra descrierii bazelor de date* (*informaţii relative la relaţii, vizualizări, indecşi etc.*) *trebuie să se poată aplica aceleaşi operaţii ca şi asupra datelor din baza de date.*

Regula 6 – regula limbajului de interogare. *Trebuie să existe cel puţin un limbaj pentru prelucrarea bazei de date*.

Regula 7 – regula de actualizare a vizualizării. *Un SGBD trebuie să poată determina dacă o vizualizare poate fi actualizată şi să stocheze rezultatul interogării într-un dicţionar de tipul unui catalog de sistem.*

Regula 8 – regula limbajului de nivel înalt. *Regulile de prelucrare asupra unei relaţii luată ca întreg sunt valabile atât pentru operaţiile de regăsire a datelor, cât şi asupra operaţiilor de inserare, actualizare şi ştergere a datelor*.

Regula 9 – regula independenţei fizice a datelor: *Programele de aplicaţie şi activităţile utilizatorilor nu depind de modul de depunere a datelor sau de modul de acces la date.*

Regula 10 – regula independenţei logice a datelor. *Programele de aplicaţie trebuie să fie transparente la modificările de orice tip efectuate asupra datelor*.

Regula 11 – regula independenţei datelor din punct de vedere al integrităţii. *Regulile de integritate trebuie să fie definite într-un sublimbaj relaţional, nu în programul de aplicaţie*.

Regula 12 – regula independenţei datelor din punct de vedere al distribuirii. *Distribuirea datelor pe mai multe calculatoare dintr-o reţea de comunicaţii de date, nu trebuie să afecteze programele de aplicaţie*.

Regula 13 – regula versiunii procedurale a unui SGBD. *Orice componentă procedurală a unui SGBD trebuie să respecte aceleaşi restricţii de integritate ca şi componenta relaţională.*

Deoarece regulile lui Codd sunt prea severe pentru a fi respectate de un SGBD operaţional, s-au formulat criterii minimale de definire a unui sistem de gestiune relaţional.

Un SGBD este **minimal relaţional** dacă:

- toate datele din cadrul bazei sunt reprezentate prin valori în tabele;
- nu există pointeri observabili de către utilizator;
- sistemul suportă operatorii relaţionali de proiecţie, selecţie şi compunere naturală, fără limitări impuse din considerente interne.

Un SGBD este **complet relaţional** dacă este minimal relaţional şi satisface în plus condiţiile:

- sistemul suportă restricţiile de integritate de bază (unicitatea cheii primare, constrângerile referenţiale, integritatea entităţii).
- sistemul suportă toate operaţiile de bază ale algebrei relaţionale.

NORMALIZAREA RELAŢIILOR

În procesul modelării unei baze de date relaţionale, o etapă importantă o reprezintă **normalizarea relaţiilor conceptuale** (Codd, 1972), adică obţinerea de relaţii „moleculare“fără a pierde nimic din informaţie pentru a elimina:

- redundanţa;
- anomaliile reactualizării informaţiilor.

Tehnica normalizării permite obţinerea unei scheme conceptuale rafinate printr-un proces de ameliorare progresivă a unei scheme conceptuale iniţiale a bazei de date relaţionale. După fiecare etapă de ameliorare, relaţiile bazei de date ating un anumit grad de perfecţiune, deci se află într-o anumită formă normală. Trecerea unei relaţii dintr-o formă normală în alta, presupune eliminarea unei anumit tip de dependenţe nedorite, care sunt transformate în dependenţe admisibile, adică dependenţe care nu provoacă anomalii.

Procesul de ameliorare a schemei conceptuale **trebuie**:

- să garanteze **conservarea datelor**, adică în schema conceptuală finală trebuie să figureze toate datele din cadrul schemei iniţiale;
- să garanteze **conservarea dependenţelor** dintre date, adică în schema finală fiecare dependenţă trebuie să aibă determinantul şi determinatul în schema aceleiaşi relaţii;
- să reprezinte o **descompunere minimală** a relaţiilor iniţiale, adică nici una din relaţiile care compun schema finală nu trebuie să fie conţinută într-o altă relaţie din această schemă.

Există două metode pentru a modela baze de date relaţionale fără anomalii sau pierderi de informaţie.

- **Schema descompunerii** pleacă de la o schemă relaţională universală ce conţine toate atributele BD. Schema se descompune prin proiecţii succesive în subrelaţii. Descompunerea se opreşte când continuarea ei ar duce la pierderi de informaţie. Algoritmii de descompunere se bazează, în general, pe descrierea formală a dependenţei dintre atribute.
- **Schema sintezei** pleacă de la o mulţime de atribute independente. Utilizând proprietăţi de semantică şi legături între atribute se pot compune noi relaţii, astfel încât, acestea să nu sufere de anumite anomalii. Algoritmii se bazează, în general, pe teoria grafurilor pentru a reprezenta legătura între atribute.

Dependenţe funcţionale

O relaţie universală este o relaţie ce grupează toate atributele care modelează sistemul real cercetat. Fie *E*, mulţimea dependenţelor considerate de proiectantul bazei pentru o schemă relaţională sau pentru o relaţie universală. Plecând de la o mulţime de proprietăţi formale ale dependenţelor, proprietăţi considerate drept reguli de deducţie (axiome), poate fi obţinută mulţimea maximală de dependenţe asociate lui *E*. Această mulţime defineşte **închiderea** lui *E*.

Fie *E* mulţimea dependenţelor unei relaţii şi p_1, p_2, ..., p_r, $r \geq 1$, proprietăţi formale ale acestor dependenţe. Dacă există o mulţime E', astfel încât orice dependenţă a mulţimii *E* este derivabilă din E' prin aplicarea proprietăţilor p_1, p_2, ..., p_r, atunci mulţimea E' defineşte **acoperirea** lui *E* pentru proprietăţile p_1, p_2, ..., p_r.

E' este o **acoperire minimală** pentru *E*, dacă nu există nici o submulţime proprie, nevidă a lui E' care să fie o acoperire pentru *E*. **Evident, *E* şi E' au închideri identice, deci dispun de acelaşi potenţial informaţional!**

Fie $R(A_1, A_2, ..., A_n)$ o schemă relaţională şi fie *X*, *Y* submulţimi de atribute ale lui *R*. *X* **determină funcţional** *Y* sau *Y* depinde funcţional (FD) de *X*, dacă pentru orice relaţie *r* (valoare curentă a lui *R*) nu există două tupluri care să aibă aceleaşi valori pentru atributele lui *X* şi să aibă valori diferite pentru cel puţin un atribut din *Y*. Cu alte cuvinte, o valoare a lui *X*, determină unic o valoare a lui *Y*.

Notaţia utilizată pentru desemnarea dependenţei funcţionale este $\overline{X \rightarrow Y}$. X este numit **determinant,** iar Y este numit **determinat** (sau dependent). Dependenţa funcţională $X \rightarrow Y$ este **trivială** dacă $Y \subseteq X$.

Comparând toate submulţimile de atribute ale unei relaţii şi determinând legăturile dintre ele, se pot obţine toate dependenţele funcţionale pe care o relaţie le satisface. Această abordare **nu** este eficientă, consumând mult timp.

Există posibilitatea ca, ştiind anumite dependenţe funcţionale şi utilizând reguli de deducţie, să fie obţinute **toate** dependenţele funcţionale.

Fie X, Y, Z, W mulţimi de atribute ale unei scheme relaţionale R şi fie următoarele axiome:

Ax1 – reflexivitate. $X \rightarrow X$. Mai general, dacă $Y \subseteq X$, atunci $X \rightarrow Y$.

Ax2 – creşterea determinantului. Pot fi considerate următoarele formulări echivalente pentru această axiomă.

1. Dacă $X \rightarrow Y$ şi $X \subseteq Z$, atunci $Z \rightarrow Y$.
2. Dacă $X \rightarrow Y$ şi $W \subseteq Z$, atunci $X \cup Z \rightarrow Y \cup W$.
3. Dacă $X \rightarrow Y$ atunci $X \cup Z \rightarrow Y \cup Z$.
4. Dacă $X \rightarrow Y$ atunci $X \cup Z \rightarrow Y$.

Ax3 – tranzitivitate. Dacă $X \rightarrow Y$ şi $Y \rightarrow Z$, atunci $X \rightarrow Z$.

O mulţime de axiome este **completă** dacă şi numai dacă plecând de la o mulţime de dependenţe E se pot obţine toate dependenţele închiderii lui E, utilizând axiomele mulţimii.

O mulţime de axiome este **închisă** dacă şi numai dacă plecând de la o mulţime de dependenţe E, nu poate fi dedusă cu ajutorul axiomelor o dependenţă care nu aparţine închiderii lui E. (nu obţin altele!)

Ullman a demonstrat că axiomele Ax1 – Ax3, numite axiomele lui Amstrong, reprezintă o mulţime închisă şi completă de axiome. Consecinţa acestui rezultat este că **închiderea lui *E* reprezintă mulţimea dependenţelor deduse din *E*, prin aplicarea axiomelor lui Amstrong!!!**

Nu toate dependenţele funcţionale sunt folositoare pentru modelarea relaţională. O dependenţă funcţională $X \rightarrow Y$ se numeşte **dependenţă funcţională totală** (FT), dacă şi numai dacă nu există nici o submulţime proprie $X' \subset X$, astfel încât $X' \rightarrow Y$. Dacă există o submulţime proprie $X' \subset X$, astfel încât $X' \rightarrow Y$, atunci dependenţa funcţională $X \rightarrow Y$ este **parţială**. În axioma Ax2, dependenţa $Z \rightarrow Y$ este o dependenţă funcţională parţială.

În cazul dependenţei funcţionale totale, axiomele lui Amstrong se reduc la o axiomă unică şi anume pseudo-tranzitivitatea:

dacă $X \rightarrow Y$ şi $W \cup Y \rightarrow Z$, atunci $W \cup X \rightarrow Z$.

Această axiomă este o regulă de deducţie completă pentru total dependenţe:

- pseudo-tranzitivitatea implică tranzitivitatea ($W = \varnothing$);
- reflexivitatea nu poate fi utilizată pentru a obţine dependenţe totale;
- reflexivitatea şi pseudo-tranzitivitatea implică creşterea.

Dacă F este o mulţime de dependenţe funcţionale totale, atunci **închiderea pseudo-tranzitivă** F^+ a acestei mulţimi este reuniunea mulţimilor dependenţelor funcţionale totale care pot fi obţinute din F folosind axioma de pseudo-tranzitivitate.

Două mulţimi de dependenţe funcţionale totale sunt **echivalente** dacă au închideri pseudo-tranzitive identice. Pentru a modela scheme relaţionale se consideră mulţimi minimale de dependenţe funcţionale totale, capabile să genereze toate închiderile pseudo-tranzitive. Aceste mulţimi definesc acoperiri minimale.

O mulţime de dependenţe funcţionale totale F^* asociată unei mulţimi de atribute A defineşte o **acoperire minimală** dacă satisface următoarele proprietăţi:

- nici o dependenţă funcţională din F^* nu este redundantă;
- toate dependenţele funcţionale totale între submulţimi ale lui A sunt în închiderea pseudo-tranzitivă a lui F^*.

Orice mulţime de dependenţe totale are cel puţin o acoperire minimală. Alegerea acoperirii minimale este punctul de start în modelarea schemelor relaţionale.

Dependenţele funcţionale între atributele bazei pot fi reprezentate grafic. Fie $A = \{A_1, A_2, ..., A_n\}$ o mulţime de atribute şi fie o mulţime de dependenţe funcţionale $\{X_i \rightarrow A_j\}$, unde X_i este o submulţime a lui A.

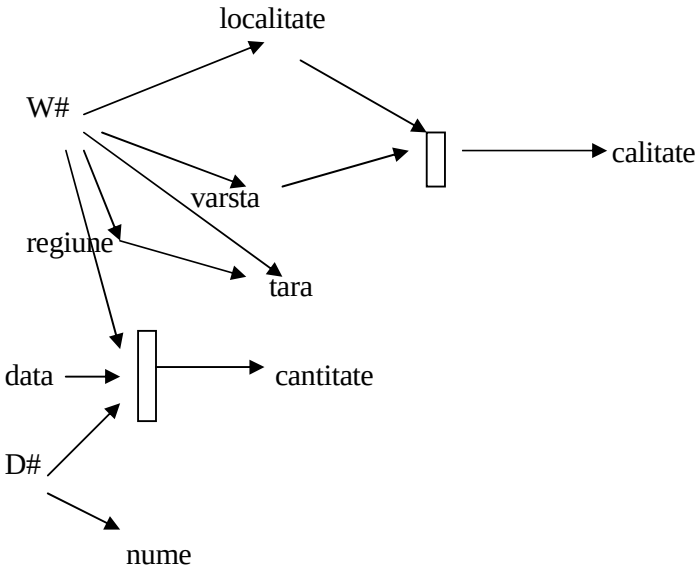
Graful dependenţelor funcţionale este un graf direcţionat bipartit, definit astfel:

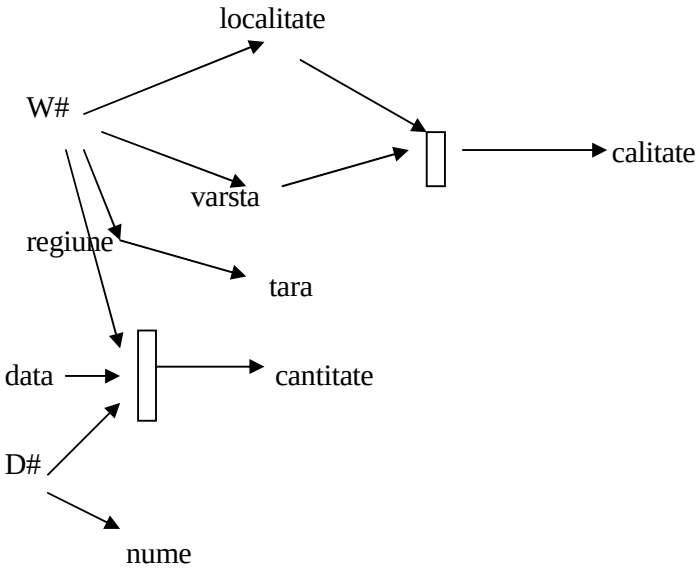
1. pentru fiecare atribut A_j există un singur nod având eticheta A_j;
2. pentru fiecare dependenţă funcţională de forma $A_i \rightarrow A_j$, există un arc de la A_i la A_j;

3. pentru fiecare dependenţă funcţională de forma $X_i \rightarrow A_j$, unde mulţimea X_i este definită de $X_i = \{Ai_1, ..., Ai_p\}$ cu $p > 1$, există un nod auxiliar etichetat prin X_i şi o mulţime de arce plecând de la $Ai_1, ..., Ai_p$ pentru a obţine pe X_i şi printr-un arc adiţional de la X_i la A_j. Nodurile X_i se reprezintă prin dreptunghiuri.

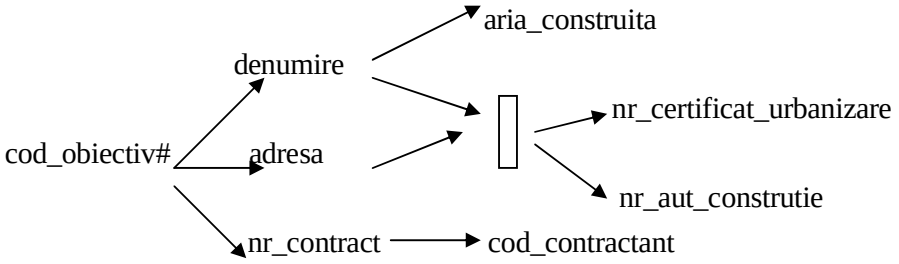
Exemplu:

1. Graful dependenţelor funcţionale pentru schema relaţională CONSUMATOR_DE_VIN(W#, localitate, varsta, calitate, regiune, tara, D#, nume, data, cantitate) şi acoperirea minimală.





2. Graful dependenţelor funcţionale pentru schema relaţională OBIECTIV_INVESTITIE. Dependentele sunt deduse din regulile impuse de beneficiar!



Necesitatea normalizării

Anomaliile care apar în lucrul cu baza de date se produc datorită dependenţelor care există între datele din cadrul relaţiilor bazei de date. Dependenţele sunt plasate greşit în tabele!!!

Avion

A#	nume	capacitate	localitate
1	AIRBUS	250	PARIS
2	AIRBUS	250	PARIS
3	AIRBUS	250	LONDRA
4	CAR	100	PARIS
5	B707	150	LONDRA
6	B707	150	LONDRA

Constrângere:
toate avioanele cu acelaşi nume au aceeaşi capacitate.

Datorită dependenţei introduse pot exista: anomalii la inserare, modificare sau ştergere, redundanţă în date, probleme de reconexiune.

1. **Redundanţă logică**. Cuplul (AIRBUS, 250) apare de trei ori.
2. **Anomalie la inserţie**. S-a cumpărat un B727 cu 150 locuri. El poate fi inserat în relaţia AVION doar dacă se defineşte o nouă valoare pentru cheia primară.
3. **Anomalie la ştergere**. Dacă este ştearsă înregistrarea pentru care A# este 4, atunci se pierde informaţia că un avion CAR are capacitatea 100.

4. **Anomalie la modificare**. Dacă se modifică capacitatea lui B$\overline{707}$ de la 150 la 170, atunci costul modificării este mare pentru a modifica toate înregistrările, iar dacă se modifică doar o înregistrare atunci constrângerea nu va mai fi verificată.
5. **Problema reconexiunii**. Considerăm schemele relaţionale:
   AVION1 = PROJECT(AVION, A#, nume)
   AVION22 = PROJECT(AVION, nume, capacitate, localitate)
   AVION3 = JOIN(AVION1, AVION2).
   Se observă că schema AVION3 este diferită de AVION.
   Apare un tuplu nou: (3, AIRBUS, 250, PARIS).

Anomaliile au apărut datorită dependenţei funcţionale (constrângerii) introduse anterior!!!

Normalizarea are drept scop:

- suprimarea redundanţei logice,
- evitarea anomaliilor la reactualizare,
- rezolvarea problemei reconexiunii.

Există o teorie matematică a normalizării al cărei autor este E.F. Codd. Soluţia: construirea unor tabele standard (forme normale).

Normalizarea este procesul reversibil de transformare a unei relaţii, în relaţii de structură mai simplă. Procesul este reversibil în sensul că nici o informaţie nu este pierdută în timpul transformării. O relaţie este într-o formă normală particulară dacă ea satisface o mulţime specificată de constrângeri.

Procesul normalizării se realizează plecând de la o relaţie universală ce conţine toate atributele sistemului de modelat, plus o mulţime de anomalii. Orice formă normală se obţine aplicând o schemă de descompunere. Există două tipuri de descompuneri.

- **Descompuneri ce conservă dependenţele**. Această descompunere presupune desfacerea relaţiei R în proiecţiile R_1, R_2, ..., R_k, astfel încât dependenţele lui R sunt echivalente (au închideri pseudo-tranzitive identice) cu reuniunea dependenţelor lui R_1, R_2, ..., R_k.
- **Descompuneri fără pierderi de informaţie** (*L-join*). Această descompunere presupune desfacerea relaţiei R într-o mulţime de proiecţii R_1, R_2, ..., R_j, astfel încât pentru orice realizare a lui R este adevărată relaţia:

$$R = \text{JOIN}(\Pi_{B1}(R), \Pi_{B2}(R), ..., \Pi_{Bj}(R))$$

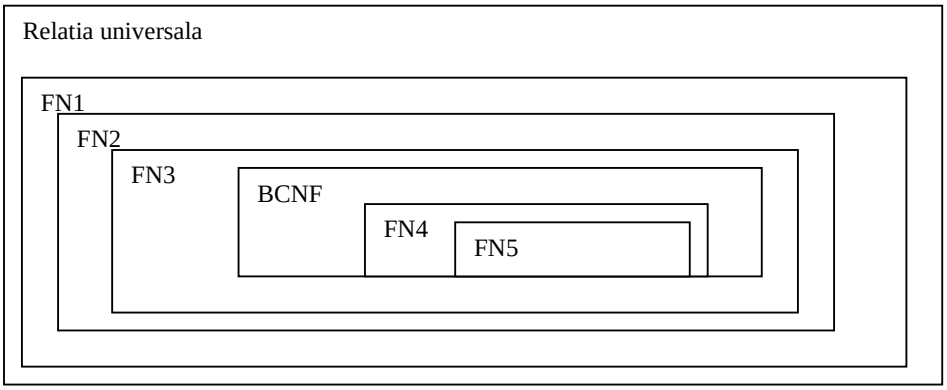
unde, pentru $1 \le k \le j$, B_k reprezintă mulţimea atributelor corespunzătoare proiecţiei R_k ($R_k = \Pi_{Bk}(R)$). Prin urmare, relaţia iniţială poate fi reconstruită considerând compunerea naturală a relaţiilor obţinute prin descompunere. Formele normale sunt obţinute prin descompuneri fără pierderi de informaţie.

O descompunere fără pierdere de informaţie, utilizată în procesul normalizării, este dată de **regula Casey-Delobel**:

Fie $R(A)$ o schemă relaţională şi fie α, β, γ o partiţie a lui A. Presupunem că α determină funcţional pe β. Atunci:

$$R(A) = \text{JOIN}(\Pi_{\alpha\cup\beta}(R), \Pi_{\alpha\cup\gamma}(R)).$$

$\alpha\cup\beta \rightarrow$ mulţimea atributelor care intervin în dependenţele funcţionale;
$\alpha\cup\gamma \rightarrow$ reprezintă reuniunea determinantului cu restul atributelor lui A.



Pentru exemplul analizat anterior:
α = {nume},
β = {capacitate},
γ = {A#, localitate}.

Aplicând Casey-Delobel se obţin schemele:

AVION1(nume#, capacitate)
AVION2)A#, nume, localitate).

Se observă că anomaliile comentate au dispărut!

AVION1

Nume	Capacitate
AIRBUS	150
CAR	100
B707	150

AVION2

A#	Nume	Localitate
1	AIRBUS	PARIS
2	AIRBUS	PARIS
3	AIRBUS	LONDRA
4	CAR	PARIS
5	B707	LONDRA
6	B707	LONDRA

Forma normală (FN1)

O relaţie este în prima formă normală dacă fiecărui atribut care o compune îi corespunde o valoare indivizibilă (atomică).

Exemplu: variante pentru a implementa FN1 pentru tabelul MASINA:

Persoana	Vehicul
Eu	R25 - W14 - R21
Tu	205
El	R5 - 305
noi	BX - 305 - R12 - R25

Varianta 1

Persoana	Vehicul
Eu	R25
Eu	W14
Eu	R21
Tu	205
El	R5
El	305
Noi	BX
Noi	305
Noi	R12
Noi	R25

Varianta 2

Persoana	Prima	Doi	Trei	Patru
Eu	R25	W14	R21	
Tu	205			
El	R5	305		
Noi	BX	305	R12	R25

Varianta 3 (4 tabele)

Masina 31 (similar se definesc Masina_32, Masina_33, Masina_34)..

Persoana	Vehicul
Eu	R25
Tu	205
El	R5
Noi	BX

Masina_34

Persoana	Vehicul
Noi	R25

Forma normală 2 (FN2)

O relaţie *R* este în a doua formă normală dacă şi numai dacă:

- relaţia *R* este în FN1;
- fiecare atribut care nu este cheie (nu participă la cheia primară) este dependent de întreaga cheie primară.

atasat_la

Cod_salariat#	Job_cod	Nr_proiect#	Functia	Suma
S1	Programator	P1	Supervizor	60
S1	Programator	P2	Cercetator	25
S1	Programator	P3	Auxiliar	10
S3	Vanzator	P3	Supervizor	60
S5	Inginer	P3	Supervizor	60

atasat_2a

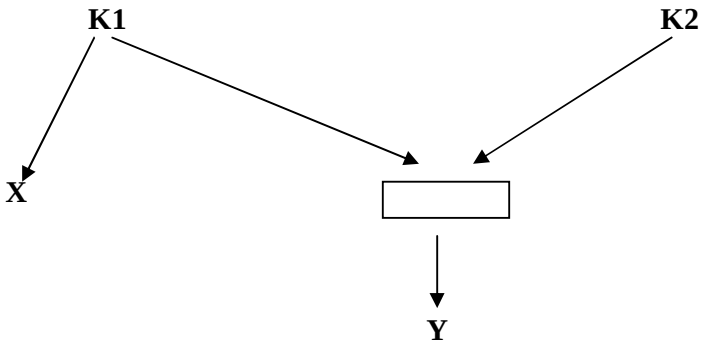
Cod_salariat#	Nr_proiect#	Functia	Suma
S1	P1	Supervizor	60
S1	P2	Cercetator	25
S1	P3	Auxiliar	10
S3	P3	Supervizor	60
S5	P3	Supervizor	60

atasat_2b

Cod_salariat#	Job_cod
S1	Programator
S3	Vanzator
S5	Inginer

A doua condiţie exprimă necesitatea total dependenţei de cheia primară. Această formă normală interzice manifestarea unor dependenţe funcţionale parţiale în cadrul relaţiei *R*!!!

Pentru a obţine o relaţie FN2 se poate aplica regula Casey-Delobel. Fie relaţia *R*(*K*1, *K*2, *X*, *Y*), unde *K*1 şi *K*2 definesc cheia primară, iar *X* şi *Y* sunt mulţimi de atribute, astfel încât $K1 \rightarrow X$. Din cauza dependenţei funcţionale $K1 \rightarrow X$ care arată că *R* nu este în FN2, se înlocuieşte *R* (fără pierdere de informaţie) prin două proiecţii *R*1(*K*1, *K*2, *Y*) şi *R*2(*K*1, *X*).



***Exemplu*.** Presupunem că un şantier poate executa mai multe lucrări de bază şi că o lucrare poate fi executată de mai multe şantiere.

LUCRARE(cod_obiectiv#, cod_lucrare#, nume);

SANTIER(nr_santier#, specialitate, sef);

EXECUTA(cod_obiectiv#, cod_lucrare#, nr_santier#, descriere, functie, conducator, data_inceput, data_sfarsit).

Pentru relaţia EXECUTA sunt evidente dependenţele:

{cod_obiectiv#, cod_lucrare#} $\rightarrow$ {data_inceput, data_sfarsit},

{cod_obiectiv#, cod_lucrare#, nr_santier#} $\rightarrow$ {descriere, functie, conducator}.

Relaţia EXECUTA este în FN1, dar nu este în FN2 deoarece atributele *data_inceput* şi *data_sfarsit* nu depind de numărul şantierului, deci nu depind de întreaga cheie primară. Pentru a obţine o relaţie în FN2 se aplică regula Casey Delobel şi relaţia EXECUTA se desface în:

EXECUTA_1(cod_obiectiv#, cod_lucrare#, nr_santier#, descriere, functie, conducator)

EXECUTA_2(cod_obiectiv#, cod_lucrare#, data_inceput, data_sfarsit).

Forma normală 3 (FN3)

Intuitiv, o relaţie *R* este în a treia formă normală dacă şi numai dacă:

- relaţia *R* este în FN2;
- fiecare atribut care nu este cheie (nu participă la o cheie) depinde direct de cheia primară.

Fie *R* o relaţie, *X* o submulţime de atribute ale lui *R* şi *A* un atribut al relaţiei *R*. *A* este **dependent tranzitiv** de *X* dacă există *Y* astfel încât $X \rightarrow Y$ şi $Y \rightarrow A$ (*A* nu aparţine lui *Y* şi *Y* nu determină pe *X*). *X* nu este dependent funcţional de *Y* sau *A*!

De exemplu, dacă $K_1, K_2, K_3 \rightarrow A_1$ şi dacă $K_1, K_2, A_1 \rightarrow A_2$, atunci $K_1, K_2, K_3 \rightarrow K_1, K_2, A_1$ şi $K_1, K_2, A_1 \rightarrow A_2$. Prin urmare, A_2 este dependent tranzitiv de K_1, K_2, K_3.

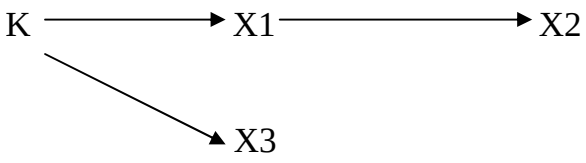
Formal, o relaţie *R* este în a treia formă normală dacă şi numai dacă:

- relaţia *R* este în FN2;
- fiecare atribut care nu este cheie (nu participă la o cheie) nu este dependent tranzitiv de nici o cheie a lui *R*.

O relaţie este în FN3 dacă şi numai dacă fiecare atribut (coloană) care nu este cheie, depinde de cheie, de întreaga cheie şi numai de cheie.

Pentru a obţine o relaţie FN3 se poate aplica regula Casey-Delobel.

Fie relaţia $R(K, X_1, X_2, X_3)$, unde atributul X_2 depinde tranzitiv de *K*, iar *K* este cheia primară a lui *R*. Presupunem că $K \rightarrow X_1 \rightarrow X_2$. Din cauza dependenţei funcţionale $X_1 \rightarrow X_2$ care arată că *R* nu este în FN3, se înlocuieşte *R* (fără pierdere de informaţie) prin două proiecţii $R1(K, X_1, X_3)$ şi $R2(X_1, X_2)$.



Exemplu: Tabelul *atasat_2a* nu este in FN3. De ce?

atasat_3a

Cod_salariat#	Nr_proiect#	Functia
S1	P1	Supervizor
S1	P2	Cercetator
S1	P3	Auxiliar
S3	P3	Supervizor
S5	P3	Supervizor

atasat_3b

Functia	Suma
Supervizor	60
Cercetator	25
Auxiliar	10

Exemplu. În tabelul EXECUTA1(cod_obiectiv#, cod_lucrare#, nr_santier#, descriere, functie, conducator) continuă să existe redundanţă în date.

Atributul *conducator* depinde indirect de cheia primară prin intermediul atributului *functie*. Între atributele relaţiei există dependenţele:

{cod_obiectiv#, cod_lucrare#, nr_santier#} → {descriere},

{cod_obiectiv#, cod_lucrare#, nr_santier#} → {functie} → {conducator}.

Pentru a aduce relaţia EXECUTA_1 în FN3 se aplică regula Casey-Delobel. Relaţia se desface, prin eliminarea dependenţelor funcţionale tranzitive, în proiecţiile:

EXECUTA11(cod_obiectiv#, cod_lucrare#, nr_santier#, descriere, functie)

EXECUTA12(functie, conducator).

Schema de sinteză pentru obţinerea lui FN3

Algoritmul de sinteză construieşte o acoperire minimală *F* a dependenţelor funcţionale totale. Se elimină atributele şi dependenţele funcţionale redundante. Mulţimea *F* este partiţionată în grupuri F_i, astfel încât în fiecare grup F_i sunt dependenţe funcţionale care au acelaşi membru stâng şi nu există două grupuri având acelaşi membru stâng. Fiecare grup F_i produce o schemă FN3. Algoritmul realizează o descompunere ce conservă dependenţele.

Algoritm SNF3 (aducerea unei relaţii în FN3 prin utilizarea unei scheme de sinteză):

1. Se determină *F* o acoperire minimală a lui *E* (mulţimea dependenţelor funcţionale).
2. Se descompune mulţimea *F* în grupuri notate F_i, astfel încât în cadrul fiecărui grup să existe dependenţe funcţionale având aceeaşi parte stângă.
3. Se determină perechile de chei echivalente (*X*, *Y*) în raport cu *F* (două mulţimi de atribute X, Y sunt chei echivalente dacă în mulţimea de dependenţe E există atât dependenţa X →Y, cât şi dependenta Y → X).

4. Pentru fiecare pereche de chei echivalente: se identifică grupurile F_i şi F_j care conţin dependenţele funcţionale cu partea stângă X şi respectiv Y; se formează un nou grup de dependenţe F_{ij}, care va conţine dependenţele funcţionale având 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 dependenţele $X \rightarrow Y$, unde X şi Y sunt chei echivalente (celelalte dependenţe sunt redundante).
6. Se construiesc relaţii FN3 (câte o relaţie pentru fiecare grup de dependenţe funcţionale).

Se observă că algoritmul solicită

- determinarea unei acoperiri minimale (algoritmii EAR şi EDF);
- determinarea închiderii (A^+) unei mulţimi de atribute A în raport cu mulţimea de dependenţe funcţionale E (algoritm AIDF).

Determinarea acoperirii minimale presupune eliminarea atributelor şi dependenţelor redundante. Acoperirea minimală nu este unică şi depinde de ordinea în care sunt eliminate aceste atribute şi dependenţe redundante.

Algoritm EAR (elimină atributele redundante din determinantul dependenţelor funcţionale)

Pentru fiecare dependenţă funcţională din E şi pentru fiecare atribut din partea stângă a unei dependenţe funcţionale:

Pas1. Se elimină atributul considerat.

Pas2. Se calculează închiderea părţii stângi reduse.

Pas3. Dacă închiderea conţine toate atributele din determinantul dependenţei, 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.

Algoritm EDF (elimină dependenţele funcţionale redundante din E)

Pentru fiecare dependenţă funcţională X →Y din E:

Pas1. Se elimină dependenţa din E.

Pas2. Se calculează închiderea X^+, în raport cu mulţimea redusă de dependenţe.

Pas3. Dacă Y este inclus în X^+, atunci dependenţa X →Y este redundantă şi rămâne eliminată. În caz contrar, se reintroduce în E.

Algoritm AIDF (determină închiderea lui A)

Pas1. Se caută dacă există în E dependenţe X →Y pentru care determinantul X este o submulţime a lui A, iar determinatul Y nu este inclus în A.

Pas2. Pentru fiecare astfel de dependenţă funcţională se adaugă mulţimii A atributele care constituie determinatul dependenţei.

Pas3. Dacă nu mai există dependenţe funcţionale de tipul de la pasul 1, atunci $A^+ = A$.

Exemplu:

Fie dependenţele funcţionale:

f1: F → N; f2: F → P; f3: P, F, N → U;

F4: P → C; f5: P → T; f6: C → T; f7: N → F.

Pas1 (suprimarea atributelor redundante). Atributul A_i este redundant în dependenţa funcţională $A_1, A_2, \ldots A_i, \ldots A_n \rightarrow Z$, dacă dependenţa funcţională $A_1, A_2, \ldots A_{i-1}, A_{i+1} \ldots A_n \rightarrow Z$ poate fi generată plecând de la mulţimea iniţială de dependenţe (E) şi de la axiomele considerate.

f1: F → N; f3: P, F, N → U sau N, P, F → U;

Aplicând axioma de pseudotranzitivitate se obţine:

F, P, F → U ➔ P, F → U

Pas2 (suprimarea dependenţelor funcţionale redundante). Dependenţa funcţională f este redundantă în E dacă $E^+ = (E - f)^+$, unde E^+ reprezintă închiderea lui E.

Se observă că f5 este redundantă (poate fi obţinută din f4 şi f6). La sfârşitul etapei se obţine:

f1: F → N; f2: F → P; f3: P, F → U; P este redundant =>F-->U

f4: P → C; f6: C → T; f7: N → F.

Pas3 (gruparea dependenţelor având acelaşi membru stâng).

F1 = {f1, f2, f3}; F2 = {f4}; F3 = {f6}; F4 = {f7}

Pas4 (regruparea mulţimilor F_i si F_j dacă există dependenţe de forma X → Y şi Y → X, unde X reprezinta partea stanga a dependenţei lui F_i şi Y este partea stângă a dependenţei lui F_j.

Din F1 şi F4 se obţine:

G1 = {f1, f2, f3, f7}; G2 = {f4}; G3 = {f6}.

Pas5 (generarea relaţiilor FN3).

R1(F#, N, P, U); R2(P#, C); R3(C#, T).

De remarcat: R1 nu este in BCNF! De ce? Exista dependenta N -->F.

Exerciţiu:

Presupunem că o parte din activităţiile de pe un aeroport sunt caracterizate de atributele:

A -- număr zbor;
B -- tip avion;
C -- aeroport plecare;
D -- aeroport sosire;
E -- linia aeriană;
F -- ora plecării;
G -- capacitate;
H -- număr locuri rezervate;
I -- data;
J -- tarif;
K -- prestaţii la bord.

Între aceste atribute există legături exprimate prin dependenţele:

A → BEFG
ACDI → H
CD → J
CDF → K
B → G
CF → ABE
AC → D
ABD → C
EG → B

Aplicând algoritmul de sinteză se obţin relaţiile:

R1(B#, G)
R2(E#, G#, B)
R3(A#, B, E, F)
R4(C#, D#, J)
R5(A#, C#, I#, H)
R6(A, C, D, F, K) în care cheile primare pot să fie oricare dintre:
AD sau AC sau CF.

Încercaţi să justificaţi acest rezultat!!!

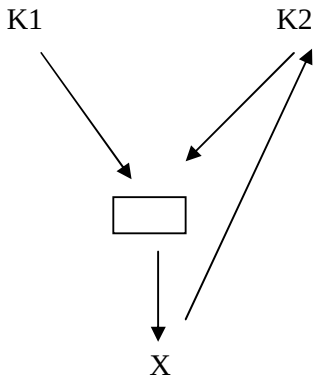
Forma normală Boyce-Codd (BCNF)

Determinantul este un atribut sau o mulţime de atribute neredundante, care constituie un identificator unic pentru alt atribut sau altă mulţime de atribute ale unei relaţii date.

Intuitiv, o relaţie *R* este în forma normală Boyce-Codd dacă şi numai dacă fiecare determinant este o cheie candidat.

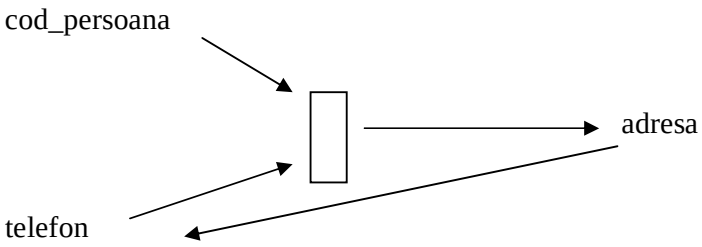
Formal, o relaţie *R* este în forma normală Boyce-Codd dacă şi numai dacă pentru orice dependenţă funcţională totală $X \rightarrow A$, *X* este o cheie (candidat) a lui *R*.

Regula Casey Delobel pentru R(K1#, K2#, X) presupunând că există dependenţa: X → K2. ➔ R1(K1#, X) şi R2(X#, K2)



Exemplu:

ADRESA(cod_parsoana#, telefon#, adresa)



În dependenţa *adresa → telefon* se observă că determinantul nu este o cheie candidat. Relaţia ADRESA se desface în:

ADRESA_1(cod_persoana#, adresa);

ADRESA_2(adresa#, telefon).

Relaţiile sunt în BCNF, se conservă datele, dar nu se conservă dependenţele (s-a pierdut *cod_persoana, telefon → adresa*).

Exemplu:

Relaţia INVESTESTE_IN leagă entităţile INVESTITOR şi OBIECTIV_INVESTITIE. Ea are schema relaţională:

INVESTESTE_IN(cod_contractant#, cod_obiectiv#, nr_contract, cota_parte). Între atributele relaţiei există dependenţele:

{cod_contractant#, cod_obiectiv#} $\rightarrow$ {nr_contract, cota_parte},
{nr_contract} $\rightarrow$ {cod_obiectiv}.

Se aplică regula Casey-Delobel şi se aduce relaţia în BCNF.

INVESTESTE_IN_1(cod_obiectiv, nr_contract#);
INVESTESTE_IN_2(cod_contractant#, nr_contract, cota_parte).

Pentru ca o relaţie să fie adusă în BCNF nu trebuie în mod obligatoriu să fie în FN3. Se pot aduce în BCNF şi relaţii aflate în FN1 sau FN2. Acest lucru este posibil întrucât dependenţele funcţionale parţiale şi cele tranzitive sunt tot dependenţe noncheie, adică dependenţe ai căror determinanţi nu sunt chei candidat. Presupunem că *R* este o relaţie ce conţine atributele *A*.

Algoritm TFBCNF (aducerea unei relaţii *R* din FN1 în BCNF)

1. Dacă relaţia conţine cel mult două atribute, atunci *R* este în BCNF şi algoritmul s-a terminat.
2. Dacă relaţia conţine mai mult de două atribute, se consideră toate perechile (*X*, *Y*) de atribute distincte din *A*.
3. Se determină A_1^+, închiderea mulţimii $A_1 = A - \{X, Y\}$.
4. Dacă pentru orice pereche (*X*, *Y*), $X \notin A_1^+$ atunci relaţia *R* este în BCNF şi algoritmul s-a terminat.
5. În caz contrar (pentru cel puţin o pereche (*X*, *Y*), *X* aparţine lui A_1^+), relaţia *R* nu este în BCNF.
6. Se reduce progresiv schema relaţiei şi se reia algoritmul, exploatând relaţia redusă. Orice relaţie obţinută prin reducerea lui *R* şi care este în BCNF se consideră ca făcând parte din descompunerea lui *R* în procesul aducerii sale în BCNF.

Forma normală 4 (FN4)

FN4 elimină redundanţele datorate relaţiilor *m*:*n*, adică datorate dependenţei multiple.

Intuitiv, o relaţie *R* este în a patra formă normală dacă şi numai dacă relaţia este în BCNF şi nu conţine relaţii *m*:*n* independente.

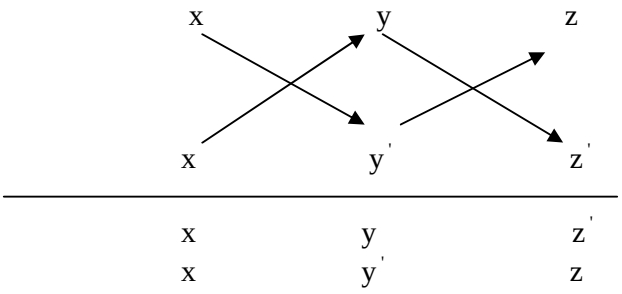
Fie R o relaţie definită pe o mulţime de atribute $A = \{A_1, A_2, ..., A_n\}$ şi fie $X, Y, Z \subset A$. Se spune că X **multidetermină** pe Z sau că Z este multidependent de X :

- dacă pentru fiecare valoare a lui Z în R există numai o valoare pentru perechea (X, Y);
- dacă valoarea lui Z depinde numai de valoarea lui X.

Acest tip de dependenţă, numită şi multivaloare sau multidependenţă (MVD) se notează prin $X \rightarrow\rightarrow Z$.

Intuitiv, multidependenţa reprezintă situaţia în care valoarea unui atribut (sau a unei mulţimi de atribute) determină o mulţime de valori a altui atribut (sau mulţimi de atribute)!!!

Multidependenţa $X \rightarrow\rightarrow Y$ poate fi gândită ca o regulă de deducţie: dacă tuplurile $<x, y, z>$ şi $<x, y', z'>$ sunt în relaţie la un moment r, atunci la momentul r sunt în relaţie şi tuplurile $<x, y, z'>$ şi $<x, y', z>$.



Orice dependenţă funcţională este o multidependenţă. Afirmaţia inversă nu este adevărată. Dacă $X \rightarrow Y$ (FD), atunci pentru oricare două tupluri $<x, y, z>$ şi $<x, y', z'>$, se obţine $y = y'$. Prin urmare în relaţie apar tuplurile $<x, y', z>$ şi $<x, y, z'>$ şi deci $X \rightarrow\rightarrow Y$ (MVD).

Fie W, V, X, Y şi Z submulţimi de atribute ale unei scheme relaţionale R. Fiind dată o mulţime T de multidependenţe există o mulţime completă de axiome (Ax1–Ax8) care permit obţinerea tuturor multidependenţelor ce se pot deduce din mulţimea T:

Ax1. Dacă $Y \subset X$, atunci $X \rightarrow Y$.

Ax2. Dacă $X \rightarrow Y$, atunci $X \cup Z \rightarrow Y \cup Z$.

Ax3. Dacă $X \rightarrow Y$ şi $Y \rightarrow Z$, atunci $X \rightarrow Z$.

Ax4. Dacă $X \rightarrow\rightarrow Y$, atunci $X \rightarrow\rightarrow R - \{X \cup Y\}$.

Ax5. Dacă $X \rightarrow\rightarrow Y$ şi $V \subset W$, atunci $W \cup X \rightarrow\rightarrow V \cup Y$.

Ax6. Dacă $X \rightarrow\rightarrow Y$ şi $Y \rightarrow\rightarrow Z$, atunci $X \rightarrow\rightarrow (Z - Y)$.

Ax7. Dacă $X \rightarrow Y$, atunci $X \rightarrow\rightarrow Y$.

Ax8. Dacă $X \rightarrow\rightarrow Y$, $Z \rightarrow W$, $W \subset Y$ şi $Y \cap Z = \varnothing$, atunci $X \rightarrow W$.

O **multidependenţă elementară** este o multidependenţă care are părţi stângi şi drepte minimale (nu există $X' \subset X$ şi $Y' \subset Y$ a.i. $X' \rightarrow\rightarrow Y'$).

Formal, relaţia R este în a patra formă normală dacă şi numai dacă:

- R este în BCNF;
- orice dependenţă multivaloare este o dependenţă funcţională.

O relaţie BCNF este în FN4 dacă pentru orice multidependenţă elementară de forma $X \rightarrow\rightarrow Y$, X este o supercheie a lui R. Aducerea relaţiilor în FN4 presupune eliminarea dependenţelor multivaloare atunci când sunt mai mult de una în cadrul unei relaţii.

Regula de descompunere în relaţii FN4. Fie $R(X, Y, Z)$ o schemă relaţională care nu este în FN4 şi fie $X \rightarrow\rightarrow Y$ o multidependenţă elementară care nu este de forma „CHEIE $\rightarrow\rightarrow$ atribut“. Această relaţie este descompusă prin proiecţie în două relaţii:

$$R = \text{JOIN}(\Pi_{X \cup Y}(R), \Pi_{X \cup Z}(R)).$$

Aplicând recursiv această regulă, se obţin relaţii FN4.

***Exemplu*.** Fie relaţia INVESTITIE(cod_contractant#, denumire, telefon) şi presupunem că un investitor poate avea mai multe numere de telefon şi că poate investi în mai multe obiective. Între atributele relaţiei există multidependenţele:

cod_contractant# $\rightarrow\rightarrow$ denumire;

cod_contractant# $\rightarrow\rightarrow$ telefon.

Relaţia INVESTITIE este în BCNF. Pentru a aduce relaţia în FN4 o vom descompune prin proiecţie în două relaţii:

INVESTITIE_1(cod_contractant#, denumire),

INVESTITIE_2(cod_contractant#, telefon).

INVESTITIE = JOIN(INVESTITIE_1, INVESTITIE_2).

Forma normală 5 (FN5)

FN5 îşi propune eliminarea redundanţelor care apar în relaţii *m:n* dependente. În general, aceste relaţii nu pot fi descompuse. S-a arătat că o relaţie de tip 3 este diferită de trei relaţii de tip 2. Există totuşi o excepţie, şi anume, dacă relaţia este ciclică

Intuitiv, o relaţie *R* este în forma normală 5 dacă şi numai dacă:

1. relaţia este în FN4;
2. nu conţine dependenţe ciclice.

Dependenţa funcţională şi multidependenţa permit descompunerea prin proiecţie, fără pierdere de informaţie, a unei relaţii în două relaţii. Regulile de descompunere (FN1 – FN4) nu dau toate descompunerile posibile prin proiecţie ale unei relaţii. Există relaţii care nu pot fi descompuse în două relaţii dar pot fi descompuse în trei, patru sau mai multe relaţii fără a pierde informaţii. Pentru a obţine descompuneri L-*join* în trei sau mai multe relaţii, s-a introdus conceptul de *join*-dependenţă sau dependenţă la compunere (JD).

Fie $\{R_1, R_2, ..., R_p\}$ o mulţime de scheme relaţionale care nu sunt disjuncte şi a căror reuniune este *R*.

R satisface ***join*-dependenţa** $*\{R_1, R_2, ..., R_p\}$ dacă la fiecare moment al lui *R* are loc egalitatea:

$$R = \text{JOIN}(\Pi_{\alpha 1}(R), \Pi_{\alpha 2}(R), ..., \Pi_{\alpha p}(R))$$

unde α_k reprezintă mulţimea atributelor corespunzătoare lui $R_k (1 \le k \le p)$.

Join-dependenţa $*\{R_1, R_2, ..., R_p\}$ are loc în *R*, dacă $R_1, R_2, ..., R_p$ este o descompunere L-*join* a lui *R*. Pentru $p = 2$ se regăseşte multidependenţa. O *join*-dependenţă $*\{R_1, R_2, ..., R_p\}$ în care una dintre R_i este chiar *R*, defineşte o *join*-dependenţă trivială.

Join-dependenţa generalizează multidependenţa. Într-adevăr, multidependenţa $X \rightarrow\rightarrow Y$ în relaţia $R(X, Y, Z)$ (deci şi $X \rightarrow\rightarrow Z$), corespunde *join*-dependenţei $*\{X \cup Y, X \cup Z\}$. Invers, *join*-dependenţa $*\{R_1, R_2\}$ corespunde multidependenţei $R_1 \cap R_2 \rightarrow\rightarrow R_1 - (R_1 \cap R_2)$.

Formal, o relaţie *R* este în FN5 dacă şi numai dacă orice *join*-dependenţă $*\{R_1, R_2, ..., R_p\}$ care are loc în *R* fie este trivială, fie conţine o supercheie a lui *R* (adică, o anumită componentă R_i este o supercheie a lui *R*). Cu alte cuvinte, o relaţie *R* este în FN5 dacă orice *join*-dependenţă definită pe *R* este implicată de cheile candidat ale lui *R*.

Între mulţimile de atribute *X*, *Y* şi *Z* din cadrul relaţiei *R* există o *join*-dependenţă dacă există multidependenţe între fiecare dintre perechile de mulţimi (*X*, *Y*), (*Y*, *Z*) şi (*X*, *Z*).

Aducerea în FN5 prin eliminarea join dependenţelor!

Exemplu.

Fie schema R(furnizor, cod_consumabil, cantitate, pret).

Reguli:

- un furnizor produce mai multe consumabile;
- nu toţi furnizorii produc aceleaşi consumabile;
- preţul unui consumabil de la un furnizor este variabil şi nu depinde de cantitate.

Furnizor	Cod_consumabil	Cantitate	Pret
F1	1	500	100
F2	1	100	80
F2	1	500	100
F2	2	500	100

Relaţia este în FN4, dar există redundanţă în date. Relaţia se descompune prin proiecţie în:

R1(furnizor#, cod_consumabil#)

R2(furnizor#, cantitate, pret)

R3(cod_consumabil#, cantitate, pret).

S-au eliminat redundanţele:

(F2,1) pentru R1;

(F2, 500, 100) pentru R2;

(1, 500, 100) pentru R3.

Se observă că:
JOIN(R1, R2) ≠ R;
JOIN(R1, R3) ≠ R;
JOIN(R3, R2) ≠ R;
JOIN(R1, R2, R3) = JOIN(R1, JOIN(R2, R3)) = R

Există join dependenţa:

*{R1(furnizor, cod_consumabil), R2(furnizor, cantitate, pret), R3(cod_consumabil, cantitate, pret)}

Exemplu.

Fie schema relaţională:

EXECUTANT(nr_santier#, cod_obiectiv#, cod_lucrare#, data_inceput, data_sfarsit). Un şantier poate executa mai multe lucrări referitoare la acelaşi obiectiv sau poate executa o lucrare pentru un obiectiv în intervale de timp distincte. Se presupune că mai multe şantiere pot executa aceeaşi lucrare, în acelaşi interval de timp sau în intervale de timp distincte.

Relaţia, datorită dependenţelor formulate anterior, nu este în FN5. Ea se poate desface prin proiecţie în trei relaţii:

EX1(nr_santier#, cod_obiectiv#, cod_lucrare#);

EX2(nr_santier#, data_inceput, data_sfarsit);

EX3(cod_obiectiv#, cod_lucrare#, data_inceput, data_sfarsit).

Sunt evidente relaţiile:

EXECUTANT ≠ JOIN(EX1, EX2),

EXECUTANT ≠ JOIN(EX1, EX3),

EXECUTANT ≠ JOIN(EX2, EX3),

EXECUTANT = JOIN(JOIN(EX1, EX2), EX3).

Concluzii:

1. FN1 → FN2 elimină redundanţele datorate dependenţei netotale a atributelor care nu participă la o cheie, faţă de cheile lui *R*. Se suprimă dependenţele funcţionale care nu sunt totale.
2. FN2 → FN3 elimină redundanţele datorate dependenţei tranzitive. Se suprimă dependenţele funcţionale tranzitive.
3. FN3 → BCNF elimină redundanţele datorate dependenţei funcţionale. Se suprimă dependenţele în care partea stângă nu este o supercheie.
4. BCNF → FN4 elimină redundanţele datorate multidependenţei. Se suprimă toate multidependenţele care nu sunt şi dependenţe funcţionale.
5. FN4 → FN5 elimină redundanţele datorate dependentei. ciclice. Se suprimă toate *join*-dependenţele care nu sunt implicate de o cheie.
6. BCNF, FN4 şi FN5 corespund la regula că orice determinant este o cheie, dar de fiecare dată dependenţa cu care se defineşte determinantul este alta şi anume dependenţa funcţională, multidependenţa sau *join*-dependenţa).
7. Descompunerea unei relaţii FN2 în FN3 conservă datele şi dependenţele, pe când descompunerea unei relaţii FN3 în BCNF şi, respectiv, a unei relaţii BCNF în FN4 conservă doar datele.