BAZE DE DATE

CAPITOLUL I. PROIECTARE (DESIGN) DE DATE

CURS 1. Preliminarii

Bazele de date reprezintă un instrument indispensabil pentru sistemele informatice. Modelarea bazelor de date constitue un subiect vast care nu poate fi tratat complet într-un singur curs. Baza de date reprezintă o modalitate de stocare pe un suport extern a unei mulţimi de date care modelează un proces (sistem) din lumea reală, cu posibilitatea regăsirii acesteia. De obicei o bază de date este memorată într-unul sau mai multe fişiere. Baza de date însăşi poate fi privită ca un fel de cutie de umplere electronică - adică un container pentru o colecţie de fişiere de date digitale. Bazele de date sunt manipulate cu ajutorul sistemelor de gestiune a bazelor de date.

Acestea, SGBD-urile, sunt responsabile cu crearea, manipularea şi întreţinerea unei baze de date. Principala funcţie a acestuia este de a permite utilizatorilor (prin intermediul programelor) să acceseze date dintr-o bază de date.

Cel mai răspândit model de baze de date este cel relaţional, în care datele sunt memorate în tabele. Pe lângă tabele, o bază de date relaţională mai poate conţine: indecşi, proceduri stocate, trigger-e, utilizatori şi grupuri de utilizatori, tipuri de date, mecanisme de securitate şi de gestiune a tranzacţiilor etc.

Cursul propune trecerea în revistă a principalelor probleme care apar în proiectarea şi implementarea bazelor de date relaţionale. Pentru exemplificarea conceptelor se utilizează sistemul de gestiune Microsoft SQL Server.

1.1. Noţiuni folosite în teoria bazelor de date

1. O **bază de date** :
   - reprezintă un ansamblu structurat de fişiere care grupează datele prelucrate în aplicaţii informatice ale unei persoane, grup de persoane, instituţii etc.
   - este definită ca o colecţie de date aflate în interdependenţă, împreună cu descrierea datelor şi a relaţiilor dintre ele.

2. **Organizarea bazei de date** – se referă la structura bazei de date şi reprezintă un ansamblu de instrumente pentru descrierea datelor, relaţiilor, restricţiilor la care sunt supuse.

3. **Sistemul de gestiune a bazei de date (SGBD)**:
   - este un sistem complex de programe care asigură interfaţa între o bază de date şi utilizatorii acesteia (exemple de programe: ACCESS, Fox Pro, PARADOX, ORACLE, MySQL, SQL Server)
   - este software-ul bazei de date care asigură:
     1) definirea structurii bazei de date;
     2) încărcarea datelor în baza de date;

3) accesul la baza de date (interogare, actualizare);
4) întreţinerea bazei de date (refolosirea spaţiilor goale, refacerea bazei de date în cazul unor incidente);
5) reorganizarea bazei de date (restructurarea şi modificarea strategiei de acces);
6) securitatea datelor.

4. Cele trei concepte de bază utilizate în organizarea bazei de date sunt:
   - entitatea
   - atributul
   - valoarea

   Prin *entitate* se înţelege un obiect concret sau abstract reprezentat prin proprietăţile sale. O proprietate a unui obiect poate fi exprimată prin perechea (ATRIBUT, VALOARE).

   Exemplu**:** În exemplul "Masa x are culoarea alba", atributul este „culoare", iar valoarea este reprezentată de cuvântul „albă". Alte exemple ar putea fi: (Sex, Feminin), (Nume, POP), (Profesie, Medic), (Salariu, 200$). Observaţie: Atributele pot caracteriza o clasă de entităţi, nu doar o entitate.

5. *Data* – este un model de reprezentare a informaţiei, accesibil unui anumit procesor (om, program, calculator) şi se defineşte prin:
- Identificator;
- Tip;
- Valoare.

1.2. Funcţionarea unei baze de date

Exploatarea unei baze de date aflate pe un *suport* specific (magnetic), de către un *utilizator,* prin intermediul unui *sistem de calcul,* având la dispoziţie un *SGBD,* parcurge uzual următoarele etape:

1. Utilizatorul, aflat la un terminal electronic, pune o "întrebare" sau lansează o cerere de date, referitoare la informaţiile din baza de date. Întrebarea se poate pune într-un limbaj de cereri specific SGBD-ului cu care se lucrează (dacă utilizatorul este familiarizat cu acest limbaj - de exemplu, SQL, FoxPro, dBase, Oracle) sau utilizatorul poate fi asistat în adresarea cererii de date de către SGBD (produsul soft pe care îl foloseşte) printr-un sistem de meniuri, butoane sau ferestre de dialog (obiecte de control).
2. Întrebarea este analizată de către calculator, de fapt de SGBD, iar dacă este corectă, se încearcă (SGBD) să i se dea răspuns prin accesarea informaţiilor din baza de date. Răspunsul va fi constituit din mulţimea datelor cerute de utilizator, care verifică criteriile specificate de acesta.

   Acest proces de lansare a unei cereri de date care va fi satisfăcută prin furnizarea datelor care îndeplinesc proprietăţile cerute se numeşte *interogarea* bazei de date.
3. Răspunsul la cererea de date se va afişa pe ecran, se va tipări la imprimantă sau se va memora într-un fişier.

În realizarea unei baze de date se urmăreşte:
- micşorarea timpului de răspuns la o interogare;

- asigurarea costurilor minime de prelucrare şi întreţinere ;
- adaptabilitatea la cerinţe noi (flexibilitate);
- sincronizarea în exploatarea simultană a datelor de către mai mulţi utilizatori(accesul concurent);
- asigurarea protecţiei împotriva accesului neautorizat (confidenţialitate);
- posibilitatea recuperării datelor în cazul deteriorărilor accidentale (integritate) etc.

Exemplu: În figura 1.1 este prezentată o bază de date foarte mică, ce conţine un singur fişier, numit VINOTECA; la rândul său, aceasta cuprinde date despre conţinutul unei anumite vinoteci. În figura 1.2 este prezentat un exemplu de operaţie de consultare din baza de date, împreună cu datele returnate prin această operaţie.

raft#	vin	producator	an	sticle	lansat
2	Cab. Sauvignon	Windsor	1995	12	2004
3	Pinot Noir	Fetzer	1997	3	2004
22	Pinot Noir	Dehlinger	1999	2	2002
50	Merlot	Clos du Bois	1998	9	2004

Fig. 1.1. Baza de date pentru VINOTECA (fişierul VINOTECA)

Consultare:
SELECT vin, raft, producator
FROM VINOTECA
WHERE lansat = 2004;

Rezultat ce apare pe monitorul unui PC:

vin	raft	producator
Cab. Sauvignon	2	Windsor
Pinot Noir	3	Fetzer
Merlot	50	Clos du Bois

Fig. 1.2 Exemplu de consultare

Observaţii: În limbajul SQL, fişierul VINOTECA din figura 1.1 este numit *tabelă*, rândurile unei astfel de tabele pot fi considerate ca *înregistrări* din fişier, iar coloanele pot fi considerate drept *câmpuri*.

1.3 Realizarea unei baze de date

Realizarea unei baze de date presupune parcurgerea etapelor:

1. analiza domeniului (sistemului) pentru care se realizează baza de date;
2. proiectarea structurii bazei de date;
3. încărcarea datelor în baza de date;
4. exploatarea şi întreţinerea bazei de date.

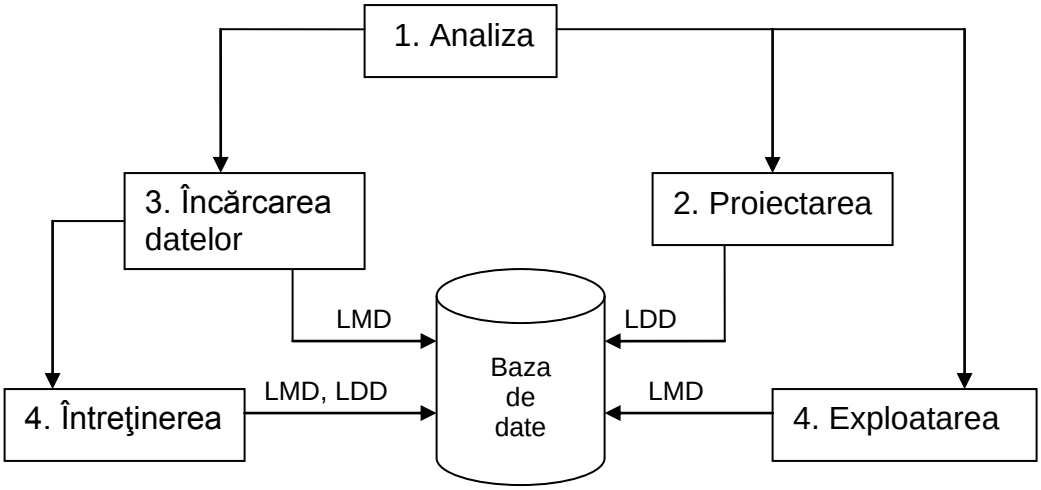


Fig. 1 Realizarea unei baze de date

a) Analiza sistemului

Analiza sistemului presupune stabilirea temei, analiza componentelor sistemului şi analiza legăturilor (asocierilor) dintre aceste componente. Rezultatul analizei formează modelul conceptual al bazei de date.

Cele patru etape necesare realizării unei baze de date vor fi tratate pe parcursul întregului curs urmărind un exemplu concret şi anume o bază de date pentru o agenţie imobiliară din ţară, denumită AGENŢIE IMOBILIARĂ, care facilitează tranzacţii de vânzare/cumpărare între vânzător şi cumpărător, care gestionează documente legate de oferte imobiliare, de întreţinere a nomenclatoarelor specifice domeniului şi care oferă o gamă largă de rapoarte privind situaţia vânzare-cumpărare.

Odată stabilită tema proiectului, se trece la etapa următoare, şi anume la identificarea tuturor tipurilor de informaţii, a legăturilor dintre informaţii şi a operaţiilor necesare pentru gestionarea lor. Această etapă va fi detaliată în cursul următor.

b) Proiectarea structurii bazei de date

Dacă etapa de analiză a modelului conceptual se realizează independent de un SGBD, prin etapa de proiectare a structurii bazei de date se trece la luarea în considerare a SGBD-ului cu ajutorul căruia va fi implementată şi exploatată baza de date.

Proiectarea structurii bazei de date reprezintă transpunerea rezultatelor obţinute în urma analizei modelului conceptual în termenii unui model al datelor suportat de un anumit SGBD. Compilatorul limbajului de descriere a datelor permite

aducerea schemei bazei de date la nivelul la care să poată fi memorată în baza de date.

Astfel, proiectarea presupune o detaliere, de exemplu, de tip pseudocod a modulelor necesare realizării bazei de date: module pentru crearea fişierelor, pentru introducerea datelor, pentru prelucrarea şi extragerea rezultatelor, pentru tratarea erorilor etc.

c) Încărcarea datelor în baza de date

Este etapa în care se realizează popularea masivă cu date a bazei de date, activitate care trebuie să se efectueze cu un minim de efort.

d) Exploatarea şi întreţinerea bazei de date

Exploatarea bazei de date de către diferiţi utilizatori finali este realizată în scopul satisfacerii cerinţelor de informare ale acestora. SGBD sprijină utilizatorii finali în exploatarea bazei de date, oferind o serie de mecanisme şi instrumente cum ar fi limbajele de manipulare a datelor (LMD).

Întreţinerea bazei de date reprezintă o activitate complexă, realizată, în principal, de către administratorul bazei de date şi care se referă la actualizarea datelor din cadrul bazei de date.

CURS 2. Construirea de diagrame entitate-relaţie

Prima etapă pentru realizarea unei baze de date constă în ***analiza sistemului***. Se cunosc mai multe tehnici de analiză, dar cea mai des întâlnită este tehnica entitate-relaţie.

Prin tehnica entiate-relaţie (denumită şi entitate-asociere) se construieşte o diagramă entiate-relaţie (notată E-R) prin parcurgerea următorilor paşi:

a) identificarea entităţilor (componentelor) din sistemul proiectului;
b) identificarea asocierilor (relaţiilor) dintre entităţi şi calificarea lor;
c) identificarea atributelor corespunzătoare entităţilor;
d) stabilirea atributelor de identificare a entităţilor.

a) *Identificarea entităţilor*

Prin *entitate* se înţelege un obiect concret sau abstract reprezentat prin proprietăţile sale. Prin convenţie, entităţile sunt substantive, se scriu cu litere mari şi se reprezintă prin dreptunghiuri. Într-o diagramă nu pot exista două entităţi cu acelaşi nume, sau o aceeaşi entitate cu nume diferite.

Pentru baza de date din domeniul imobiliar considerată anterior, se pot pune în evidenţă următoarele entităţi:

- DATE_PERSOANĂ – entitate care stochează date personale ale ofertantului (vânzătorului) sau ale clientului (cumpărătorului);
- CERERI_ OFERTE – conţine ofertele sau cererile imobiliare propuse de vânzători, respectiv cumpărători;
- DESCRIERE_IMOBIL – stochează informaţiile referitoare la imobile;
- JUDEŢE – entitate ce conţine judeţele în care sunt amplasate imobilele;
- LOCALITĂŢI - entitate ce conţine localităţile în care sunt amplasate imobilele;
- STRĂZI - entitate ce precizează străzile în care sunt amplasate imobilele;
- FACTURI – formularul necesar unei tranzacţii de cumpărare-vânzare.

Figura următoare prezintă o primă formă a diagramei entitate-asociere (E-R).

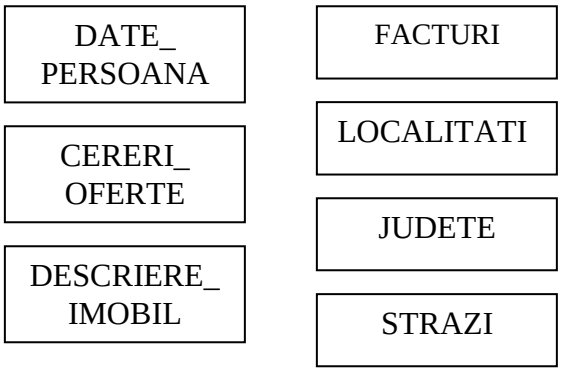


Fig. 2.1. Diagrama E-R pentru domeniul imobiliar (prima formă)

b) *Identificarea asocierilor dintre entităţi şi calificarea lor*

Între majoritatea componentelor (adică a entităţilor) unui sistem economic se stabilesc *legături (asocieri).*

Exemplu: Există o asociere între entităţile CERERI_OFERTE şi FACTURI deoarece facturile reprezintă finalizarea unei cereri/oferte. Această asociere se reprezintă ca în figura de mai jos.

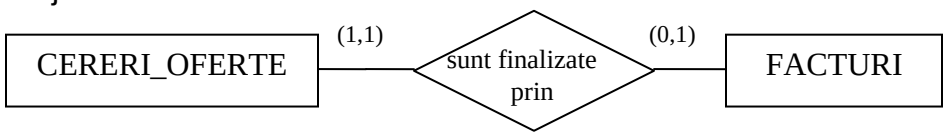


Fig. 2.2. Prezentarea asocierii dintre entităţile CERERI_OFERTE şi FACTURI

Sunt necesare precizarea câtorva notaţii şi noţiuni utilizate în exemplul de mai sus:

- legăturile (asocierile) se reprezintă prin arce neorientate între entităţi;
- fiecărei legături i se acordă un *nume* plasat la mijlocul arcului şi simbolizat printr-un *romb* (semnificaţia legăturii);
- numerele simbolizate deasupra arcelor se numesc *cardinalităţi* şi reprezintă *tipul legăturii*;
- *cardinalitatea* asocierilor exprimă numărul minim şi maxim de realizări ale unei entităţi cu cealaltă entitate asociată.

Exemplu: Cardinalitatea (1,1) ataşată entităţii CERERI_OFERTA înseamnă că o factură poate fi rezultatul tranzacţionării a minim unei cereri/oferte şi a unui număr maxim de tot o cerere/ofertă. Cardinalitatea (0,1) ataşată entităţii FACTURI înseamnă că o cerere se poate finaliza prin maxim o factură sau prin nici una (0 facturi) . Această cardinalitate reiese din analiză:

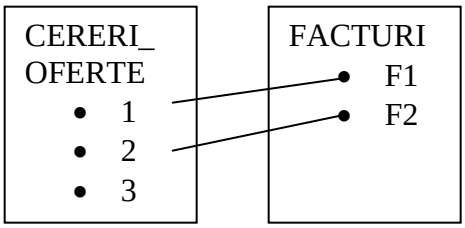


Fig. 2.3. Determinarea cardinalităţii asocierii dintre entităţile CERERI_OFERTE şi FACTURI

Maximele unei cardinalităţi sunt cunoscute şi sub denumirea de *grad de asociere*, iar minimele unei cardinalităţi, obligativitatea participării entităţilor la asociere.

Tipuri de asocieri (legături) între entităţi

Asocierile pot fi de mai multe feluri, iar odată cu asocierea, se impune stabilirea calificării acesteia. Asocierea dintre entităţi se face în funcţie de:

i) cardinalitatea asocierii;
ii) numărul de entităţi distincte care participă la asociere.

i. După cardinalitatea asocierii

În funcţie de maxima cardinalităţii (gradul de asociere), se cunosc trei tipuri de asocieri, care, la rândul lor, sunt de două tipuri, în funcţie de minima cardinalităţii (gradul de obligativitate al participării la asociere):

- asocieri de tip „unu la unu”:
  - asocieri parţiale de tip „unu la unu”
  - asocieri totale de tip „unu la unu”

- asocieri de tip „unu la mai mulţi”:
  - asocieri parţiale de tip „unu la mulţi”
  - asocieri totale de tip „unu la mulţi”
- asocieri de tip „mulţi la mulţi”:
  - asocieri parţiale de tip „mulţi la mulţi”
  - asocieri totale de tip „mulţi la mulţi”.

ii. După numărul de entităţi distincte care participă la asociere:

- asocieri binare (între două entităţi distincte);
- asocieri recursive (asocieri ale entităţilor cu ele însele);
- asocieri complexe (între mai mult de două entităţi distincte).

În continuare se descriu asocierile grupate după cardinalitatea lor.

Asocieri în funcţie de cardinalitatea legăturii

1. *Asocieri de tip „unu la unu”* sunt asocieri în care maximele cardinalităţii au valoarea 1.

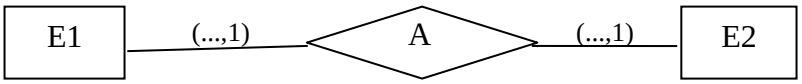


Fig. 2.4. Asociere de tip unu la unu

Exemplu: Asocierea din figura 2.3 este asociere de tip „1 la 1”.

2. *Asocieri de tip „unu la mai mulţi”* sunt asocieri în care maxima cardinalităţii unei entităţi este unu, iar a celeilalte entităţi are valoarea „mulţi”.

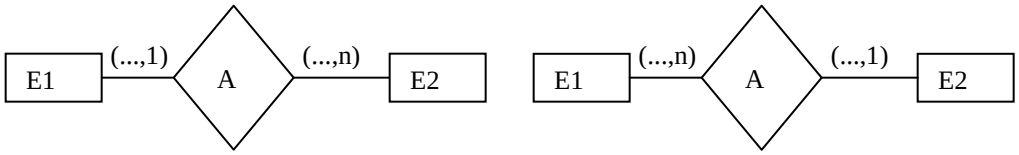


Fig. 2.5. Asociere de tipul unu la mai mulţi

Exemplu:

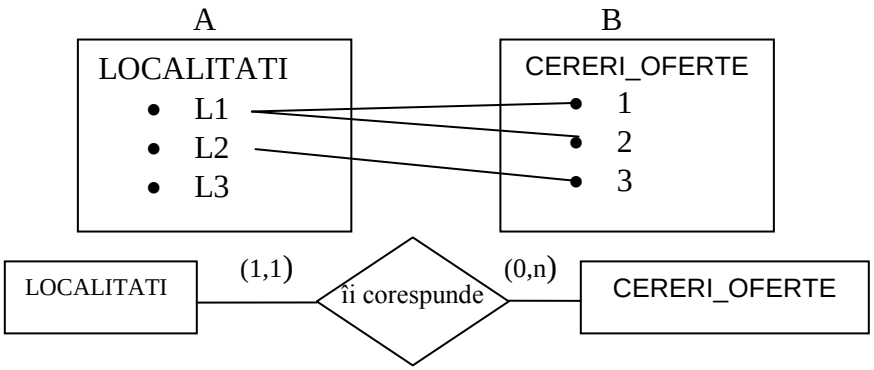


Fig. 2.6. Asociere de unu la mai mulţi între entităţile LOCALITĂŢI şi CERERI_OFERTE

3. Asocieri de tipul „mulţi la mulţi” sunt asocieri în care maximele cardinalităţii au valoarea „mulţi”.

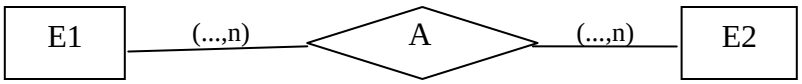


Fig. 2.7. Asociere de tipul mulţi la mulţi

Exemplu:

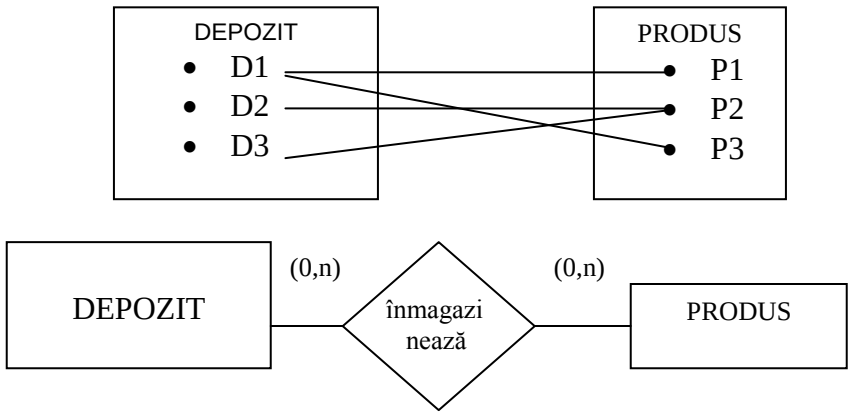


Fig. 2.8. Asociere de tipul mulţi la mulţi între entităţile DEPOZIT şi PRODUS

Observaţie: Uneori (în cazul utilizării unor SGBD), asocierea de tip „mulţi la mulţi” se transformă în două asocieri de tipul „unul la mulţi” fiind, de regulă, mai uşor de implementat şi de utilizat şi anume:

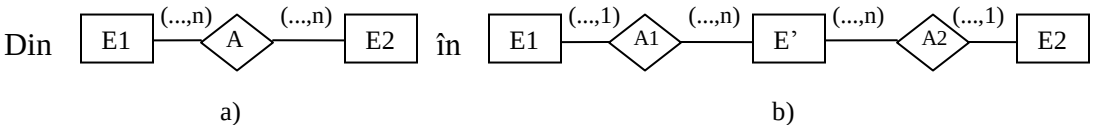


Fig. 2.9. Transformarea unei asocieri de tipul mulţi la mulţi (a) în asocieri de tipul unu la mulţi (b)

Exemplu: În cazul exemplului de mai sus (vezi figura 2.8), transformarea asocierii „mulţi la mulţi” în asocieri de tipul „unu la mulţi” se poate realiza prin construirea unei noi entităţi DEPOZIT_PRODUS astfel:

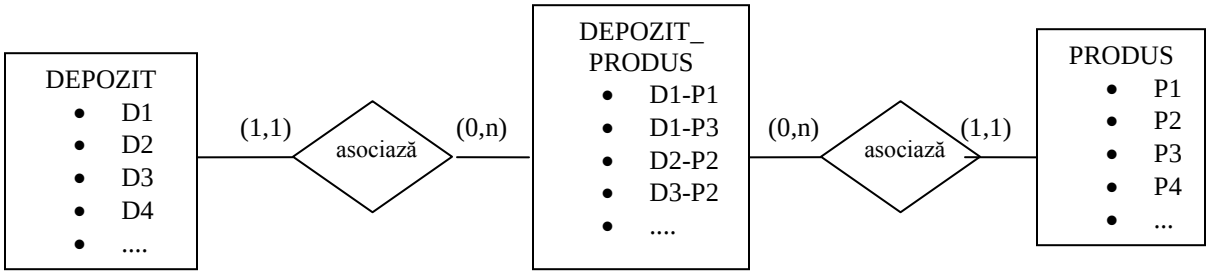


Fig. 2.10. Transformarea asocierii de tipul mulţi la mulţi în asocieri de tipul unu la mulţi

Asocieri parţiale şi totale

Printr-o *asociere parţială* se înţelege o asociere în care nu există obligativitatea participării la această asociere a tuturor entităţilor vizate, ci numai a unora dintre ele sau a nici uneia. Asocierea parţială se caracterizează prin faptul că minima cardinalităţii ataşată unei entităţi este zero.

Observaţii (asupra minimii cardinalităţii)

- minima cardinalităţii zero, are drept rezultat lipsa obligativităţii participării partenerului la această asociere;
- minima cardinalităţii mai mare decât zero, are drept rezultat obligativitatea participării.

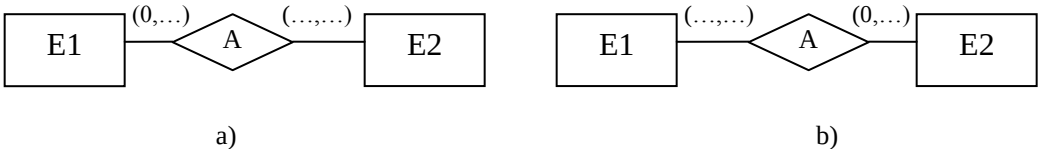


Fig. 2.11 Asocieri parţiale între entităţile E1 şi E2

Exemplu: Asocierea dintre entităţile CERERI_OFERTE şi FACTURI din fig. 2.3 reprezintă o asociere parţială, deoarece participarea entităţii FACTURI **nu** este obligatorie, minima caracteristicii corespunzătoare entităţii FACTURI fiind 0.

O *asociere* este *totală* dacă toate entităţile au obligativitatea să participe la asociere, adică minima cardinalităţii este mai mare decât zero.

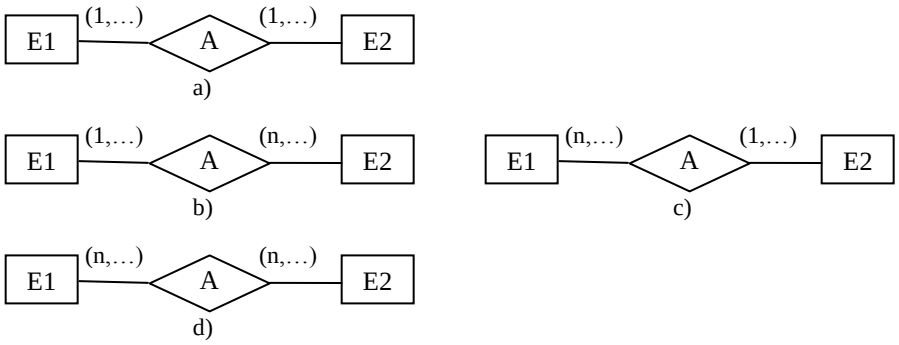
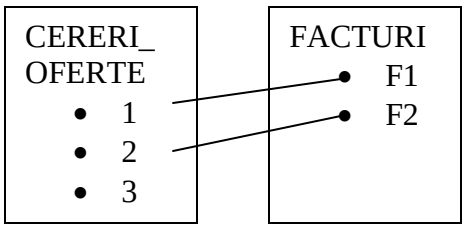


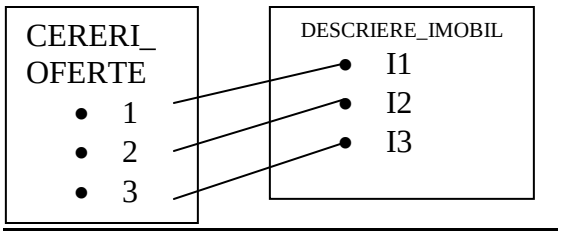
Fig. 2.12 Asocieri totale între entităţile E1 şi E2

În continuare se dau câteva exemple de asocieri totale, respectiv parţiale.

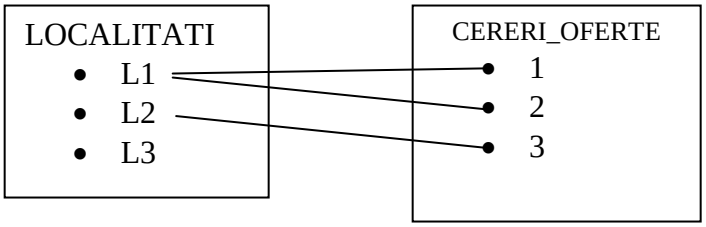
Exemplu: *Asocieri parţiale de tip unu la unu*



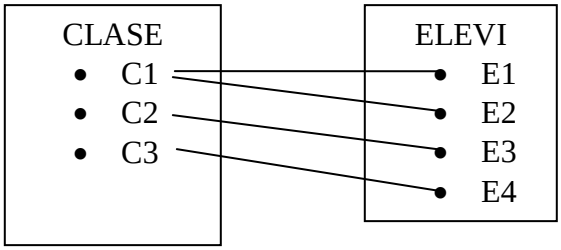
Exemplu: *Asocieri totale de tip unu la unu*



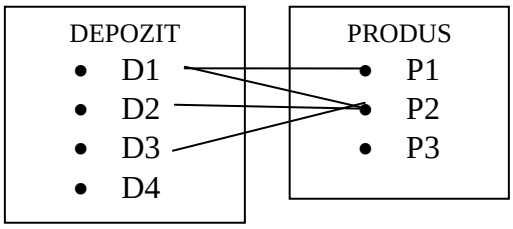
Exemplu: *Asocieri parţiale de tip unu la mulţi*



Exemplu: *Asocieri totale de tip unu la mulţi*



Exemplu: *Asocieri parţiale de tip mulţi la mulţi*



Exemplu: *Asocieri totale de tip mulţi la mulţi*

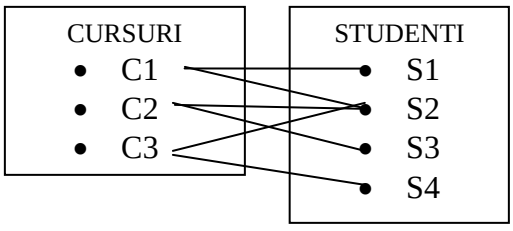


Fig. 2.13 Asocieri după gradul şi obiectivitatea lor

În exemplul bazei de date AGENTIE_IMOBILIARA, tipurile de asocieri dintre entităţi stabilite în funcţie de modul în care se desfăşoară activitatea modelată sunt:

- JUDETE-LOCALITATI 1:n – deoarece unui judeţ îi corespund mai multe localităţi;
- LOCALITATI-STRAZI 1:n - deoarece unei localităţi îi corespund mai multe străzi;
- STRAZI-CERERI_OFERTE 1:n – deoarece unei străzi îi pot corespunde mai multe oferte/cereri;
- FACTURI-CERERI_OFERTE 1:1 – deoarece fiecare factură conţine doar câte o ofertă/cerere;
- CERERI_OFERTE-DESCRIERE_IMOBIL 1:1 – fiecărui imobil i se face o singură descriere;
- FACTURI- DATE_PERSOANA 1:1 – o factură este încheiată de o singură persoană;
- DATE_PERSOANA - CERERI_OFERTE 1:n – o persoană poate lansa mai multe cereri sau oferte de imobil.

c) Identificarea atributelor entităţilor şi a asocierilor dintre entităţi

Atributele unei entităţi reprezintă proprietăţi ale acestora. Atributele sunt substantive, iar pentru fiecare atribut i se va preciza tipul fizic (*integer*, *float*, *char, string* etc.)

Exemplu: Entitatea LOCALITĂŢI are următoarele atribute: codul localităţii, notat „cod_loc”, simbolul de identificare al judeţului „simbol_judeţ” şi denumirea localităţii „nume_loc”.

d) Stabilirea atributelor de identificare a entităţilor

Un *atribut de identificare (*numit *cheie primară)*, reprezintă un atribut care se caracterizează prin unicitatea valorii sale pentru fiecare instanţă a entităţii.

În cadrul diagramei entitate-asociere, un atribut de identificare se marchează prin subliniere sau prin marcarea cu simbolul # plasat la sfârşitul numelui acestuia.

Fig. 2.14. Notaţii uzuale pentru atributele de identificare

Exemplu: Ca atribut de identificare putem considera codul numeric personal „cnp” pentru entitatea DATE_PERSOANĂ.

Pentru ca un atribut să fie atribut de identificare, acesta trebuie să satisfacă unele cerinţe:

- oferă o identificare unică în cadrul entităţii;
- este uşor de utilizat;
- este scurt (de cele mai multe ori, atributul de identificare apare şi în alte entităţi, drept cheie externă).

Pentru o entitate pot exista mai multe atribute de identificare, numite *atribute* (chei) *candidate*. Dacă există mai mulţi candidaţi cheie, se va selecta unul, preferându-se acela cu valori mai scurte şi mai puţin volatile.

Exemplu: În urma analizării celor 4 etape necesare construirii diagramei entitate-asociere:

- identificarea entităţilor domeniului sau a sistemului economic;
- identificarea asocierilor dintre entităţi;
- identificarea atributelor aferente entităţilor şi a asocierilor dintre acestea;
- stabilirea atributelor de identificare a entităţilor,

se poate prezenta forma completă a diagramei asociate domeniului ales în exemplu.

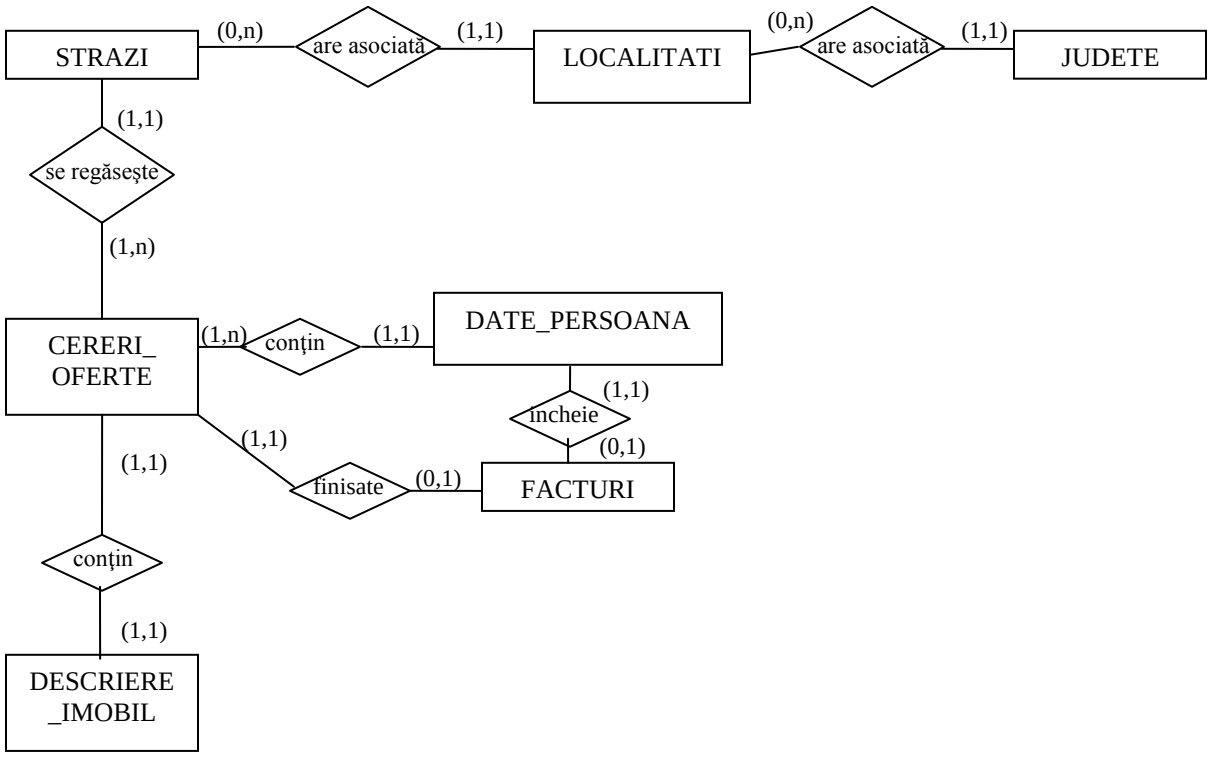


Fig. 2.15. Diagrama E-R pentru domeniul imobiliar (a doua formă)

În cazul în care se doreşte o diagramă care să conţină şi atributele fiecărei entităţi însoţite de precizarea atributelor de identificare (adică a cheilor primare), pentru a nu încărca imaginea, diagrama proiectului se poate fragmenta pe mici domenii, după cum este cazul entităţii CERERI_OFERTE, prezentat în figura 2.16. (S-a considerat un număr relativ mic de atribute).

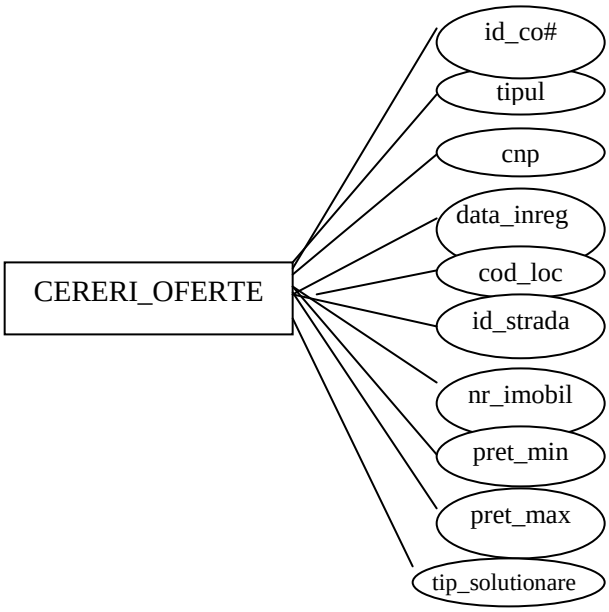


Fig. 2.16. Reprezentarea atributelor aferente entităţii CERERI_OFERTE (detaliu dintr-o diagramă E-R)

În reprezentarea atributelor aferente entităţii CERERI_OFERTE semnificaţia atributelor este următoarea: cheia primară a entităţii „id_co" reprezintă numărul de ordine al cererii sau ofertei de imobil lansată de o anumită persoană, atributul „tipul" specifică dacă este vorba de o cerere sau de o ofertă, prin „cnp" se precizează codul numeric personal al clientului, „data_inreg" reprezintă data la care s-a înregistrat oferta/cererea, apoi urmează câteva date legate de imobil: codul localităţii "cod loc", codul străzii „id_strada", numărul imobilului „nr_imobil", preţul minim, respectiv preţul maxim al imobilului „pret_min", „pret_max". Ultimul atribut, „tip_solutionare" precizează dacă cererea/oferta respectivă a fost soluţionată; pentru o cerere/oferta nou introdusă, acest atribut se va completa cu explicaţia de "nesoluţionat".

Astfel, diagrama bazei de date AGENŢIE IMOBILIARĂ conţine 7 entităţi a căror asociere a fost prezentată în figura 2.15.

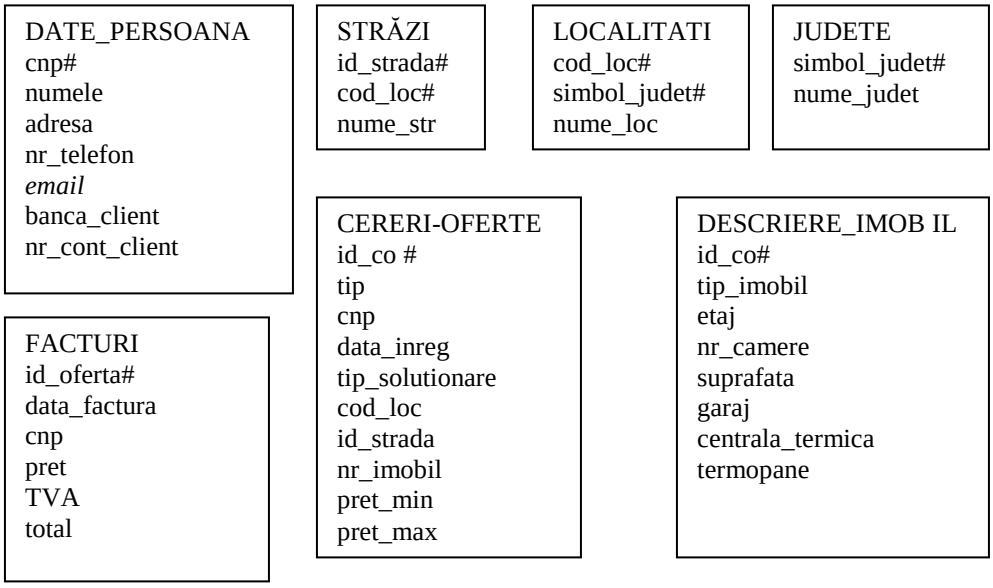


Fig. 2.17. Baza de date AGENŢIE IMOBILIARĂ- entităţi şi atribute

CURS 3. Proiectarea modelului relaţional

Proiectarea corectă a bazelor de date este crucială pentru obţinerea unei aplicaţii de înaltă performanţă.

Modelul relaţional este cel mai utilizat dintre modelele de date existente (modele ierarhice, modele reţea, modele orientate pe obiect). Faţă de modelele ierarhic şi reţea, modelul relaţional prezintă câteva avantaje:

- propune structuri de date uşor de utilizat;
- ameliorează independenţa logică şi fizică;
- pune la dispoziţia utilizatorilor limbaje neprocedurale;
- optimizează accesul la date;
- îmbunătăţeşte confidenţialitatea datelor.

Din punct de vedere istoric, trebuie menţionat că modelul relaţional s-a conturat în două articole publicate de către F.E. Codd în 1969 şi 1970, matematician la centrul de cercetări (California) I.B.M. Codd a propus o structură de date tabelară, independentă de tipul de echipamente şi de software-ul de sistem pe care este implementată. Deşi puternic matematizat, modelul relaţional este relativ uşor de înţeles.

Dacă, teoretic, modelul s-a consacrat în anii 1970, produsele software care să gestioneze baze de date au devenit populare abia în anii 80. Cele mai utilizate sisteme de gestiune a bazelor de date relaţionale (SGBDR) dedicate uzului individual sunt: ACCESS, PARADOX, Visual Fox Pro. Pentru aplicaţiile complexe din bănci şi instituţii de mari dimensiuni se folosesc SGBDR-urile de „categorie grea”, ORACLE, DB2 IBM, Informix IBM, SyBase (SyBase), SQL Server (MicroSoft);sunt mult mai robuste, fiabile, dar şi costisitoare. În ultimul timp şi-au făcut apariţia aşa-zisele SGBD-uri (aproape) gratuite: PostgreSQL, MySQL, mSQL, FireBird etc. (Acestea rulează de obicei pe sisteme de operare Linux).

Modelul relaţional are la bază teoria matematică a relaţiilor şi poate fi privit ca o mulţime de tabele obţinute prin metoda normalizării, eliminându-se astfel anomaliile de actualizare.

Conceptele modelului relaţional sunt:

1. **structura relaţională a datelor;**
2. **operatorii modelului relaţional;**
3. **restricţiile de integritate ale modelului relaţional.**

3.1 Structura relaţională a datelor

O *bază de date relaţională* (*BDR*) reprezintă un ansamblu de relaţii, prin care se reprezintă datele şi legăturile dintre ele.

În cadrul bazei de date relaţionale, datele sunt organizate sub forma unor *tablouri bidimensionale* (tabele) de date, numite *relaţii*. Asocierile dintre relaţii se reprezintă prin atributele de legătură. În cazul legăturilor de tip „unu la mulţi”, aceste atribute figurează într-una dintre relaţiile implicate în asociere. În cazul legăturilor de tip „mulţi la mulţi”, atributele sunt situate într-o relaţie distinctă, construită special pentru explicarea legăturilor între relaţii.

Prezentarea structurii relaţionale a datelor impune definirea noţiunilor de:

- domeniu;
- relaţie;
- atribut;
- schemă a unei relaţii.

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	Camp
Domeniu	Tip de dată	Tip de dată

Fig. 3.1. Concepte uzuale folosite în exprimarea formală, uzuală şi fizică

- *Domeniul*

Domeniul reprezintă o mulţime de valori, notată prin litere mari D_1,D_2 etc., caracterizată printr-un nume.

Modalităţile de definire a unui domeniu sunt:
- explicit: prin enumerarea tuturor valorilor aparţinând domeniului;
- implicit: prin precizarea proprietăţilor pe care le au valorile din cadrul domeniului.

Exemplu: D_1: {“Da”, “Nu”} reprezintă un domeniu definit explicit. D_2: {x| x este de dată calendaristică} sau D_3: {s| s este număr decimal} reprezintă domenii definite implicit, unde prin *număr decimal* se înţelege un număr zecimal pentru care se precizează numărul de cifre componente.

Printr-un *tuplu* se înţelege o succesiune de valori de diferite tipuri. Un tuplu se notează enumerând valorile sale $<V_1,V_2,V_3,...,V_n>$, unde V_1 este o valoare din domeniul D_1, $V_2 \in D_2$ etc.

Exemplu: Considerăm că tuplul referitor la persoana x din entitatea CERERI_OFERTE conţine trei valori diferite ce desemnează:
- codul numeric personal (cnp): 1701205230023;
- data înregistrării ofertei (data_înreg): 2006-07-03;
- tipul soluţionării (tip_soluţionare): „Nu”.

Se formează tuplul <1701205230023, ‘2006-07-03’, ”Nu”>.

- *Relaţia*

Relaţia R este un subansamblu al produsului cartezian dintre mai multe domenii D_1, D_2, ..., D_n, reprezentată sub forma unei tabele de date (tabelul bidimensional) şi deci, o mulţime de tupluri.

Exemplu: Considerăm că:
- D_1 conţine valori care exprimă cnp-ul persoanei.
- D_2 cuprinde valori ale datei calendaristice;
- D_3 cuprinde valori referitoare la tipul soluţionării tranzacţiei: „Da”, dacă tranzacţia a fost soluţionată, „Nu”, în caz contrar.

De asemenea considerăm că se cunosc datele a doi ofertanţi şi că fiecare pune în vânzare doar câte un imobil. Atunci definim relaţia R prin tuplurile care descriu aceste informaţii ale ofertelor celor două persoane:

R: {<1701205230023,'2006-07-03', "Nu">, <2661805270023,'2006-05-27', "Da">}.

sau

R:

D_1	D_2	D_3
2661805270023	2006-05-27	Da
1701205230023	2006-07-03	Nu

Fig. 3.2. Variante de prezentare a unei relaţii R

Observaţia 1. Într-o relaţie, tuplurile trebuie să fie distincte.
Observaţia 2. *Cardinalul* relaţiei este numărul tuplurilor dintr-o relaţie.
Gradul relaţiei este numărul valorilor dintr-un tuplu.

- *Atributul*

Atributul reprezintă coloana unei tabele de date, caracterizată printr-un nume.
Exemplu:
R:

cnp: D_1	**data_înreg:D_2**	**tip_ soluţionare:D_3**
2661805270023	2006-05-27	Da
1701205230023	2006-07-03	Nu

Fig. 3.3. Relaţia R reprezentată cu ajutorul atributelor

Atributele sunt utile atunci când într-o relaţie un domeniu apare de mai multe ori. Prin numele dat fiecărei coloane (atribut), se diferenţiază coloanele care conţin valori ale aceluiaşi domeniu, eliminând dependenţa faţă de ordine.

- *Schema unei relaţii*

Schema unei relaţii este numele relaţiei urmată de lista de atribute, pentru fiecare atribut precizându-se domeniul asociat.

Astfel, pentru o relaţie R cu atributele A_1, A_2, ... , A_n şi domeniile D_1, D_2, ... ,D_m, cu $m \leq n$, schema relaţiei R poate fi prezentată astfel:

$R(A_1: D_1, A_2:D_2, ... , A_n: D_m)$
sau
R:

A_1:D_1	...	A_n:D_m

Fig. 3.4. Reprezentarea schemei relaţiei R

Ca o concluzie, dintre caracteristicile modelului relaţional menţionăm:
- 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.

CURS 4. Operatorii modelului relaţional

3.2 Operatorii modelului relaţional

Limbajele de interogare sunt procedurale sau ne-procedurale. În limbajele procedurale utilizatorul indică sistemului succesiunea de operaţii asupra BD pentru a determina rezultatul dorit. În limbajele ne-procedurale, utilizatorul descrie rezultatul dorit, fără a indica procedura prin care acesta este obţinut

Modelul relaţional oferă două colecţii de operatori pe relaţii:

- algebra relaţională;
- calcul relaţional:
  - calcul relaţional orientat pe tuplu;
  - calcul relaţional orientat pe domeniu.

Algebra relaţională este un limbaj procedural, pe când calculul relaţional pe tupluri şi calculul relaţional pe domenii sunt limbaje ne-procedurale

În acest curs va fi tratat doar cazul algebrei relaţionale.

Algebra relaţională este o colecţie de operaţii pe relaţii, fiecare operaţie având drept operanzi una sau mai multe relaţii, rezultatul fiind o altă relaţie.

Există mai multe criterii de grupare a operaţiilor:

- operaţii de bază:
  - reuniunea;
  - diferenţa;
  - produsul cartezian etc.
- operaţii derivate:
  - intersecţia;
  - diviziunea etc.

sau

- operaţii tradiţionale pe mulţimi (reuniune, intersecţie, diviziune, produs cartezian)
- operaţii relaţionale speciale (selecţia, proiecţia, joncţiunea, etc.)

➢ *Reuniunea*

Reuniunea reprezintă o operaţie a algebrei relaţionale definită pe două relaţii: R_1 şi R_2, ambele cu aceeaşi schemă, în urma căreia se construieşte o nouă relaţie R_3, cu aceeaşi schemă ca şi R_1 şi R_2 şi având drept extensie tuplurile din R_1 şi R_2, luate împreună o singură dată.

Notaţii: $R_1 U R_2$

OR (R_1, R_2)

APPEND (R_1, R_2)

UNION (R_1, R_2)

Reprezentarea grafică

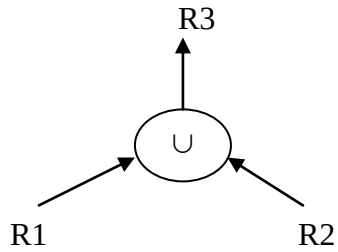
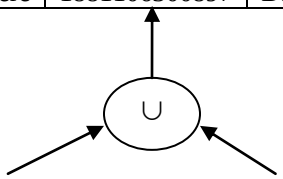


Fig. 4.1. Reprezentarea grafică a operaţiei de reuniune a două relaţii

Exemplu: Deoarece aplicaţia AGENTIE IMOBILIARA luată ca exemplu în acest curs nu conţine două relaţii cu aceeaşi structură, pentru a putea exemplifica operaţia de reuniune se vor construi două relaţii ARHIVA_OFERTE şi ARHIVA_CERERI populate cu informaţiile aferente ofertelor respectiv cererilor soluţionate (s-au ales doar patru atribute: id-ul ofertei sau a cererii, tipul operaţiei(ofertă sau cerere), cnp-ul clientului, tipul soluţionării). Pentru a afla care sunt toate ofertele şi cererile soluţionate, se realizează operaţia de reuniune.

REZ:

id	**tipul**	**cnp**	**tip_solutionare**
1066	oferta	2660805270023	Da
1210	oferta	1881106300897	Da
220	cerere	2820506300898	Da
1316	cerere	1881106300897	Da



ARHIVA_OFERTE:

id	**tipul**	**cnp**	**tip_solutionare**
1066	oferta	2660805270023	Da
1210	oferta	1881106300897	Da

ARHIVA_CERERI:

id	**tipul**	**cnp**	**tip_solutionare**
220	cerere	2820506300898	Da
1316	cerere	1881106300897	Da

Fig. 4.2. Reuniunea relaţiilor ARHIVA_OFERTE şi ARHIVA_CERERI

- *Diferenţa*

Diferenţa reprezintă o operaţie a algebrei relaţionale definită pe două relaţii R_1 şi R_2, ambele cu o aceeaşi schemă, în urma căreia se construieşte o nouă relaţie R_3, cu schema identică cu R_1 şi R_2, având drept extensie acele tupluri ale relaţiei R_1 care nu se regăsesc în relaţia R_2.

Notaţii: $R_1 - R_2$

REMOVE (R_1, R_2)

MINUS (R_1, R_2)

Reprezentarea grafică:

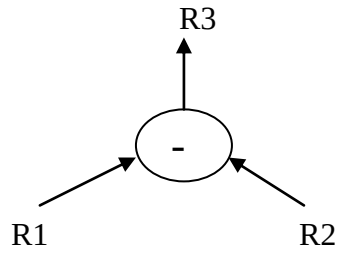
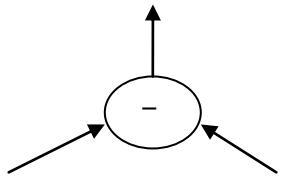


Fig. 4.3. Reprezentarea grafică a operaţiei de diferenţă a două relaţii

Exemplu: Presupunând că există clienţi care au înregistrări în ambele tabele (adică au oferit imobil spre vânzare, dar şi au achiziţionat un alt imobil în acelaşi timp), pentru a afla care au fost *doar* ofertanţii de imobile, se aplică diferenţa dintre relaţiile ARHIVA_OFERTE şi ARHIVA_CERERI.

REZ:

id	**tipul**	**cnp**	**tip_solutionare**
2066	oferta	2660805270023	Da



ARHIVA_OFERTE:

id	**tipul**	**cnp**	**tip_solutionare**
1210	oferta	1881106300897	Da
2066	oferta	2660805270023	Da

ARHIVA_CERERI:

id	**tipul**	**cnp**	**tip_solutionare**
0221	cerere	2820506300898	Da
1210	cerere	1881106300897	Da
3161	cerere	2690125270032	Da

Fig. 4.4. Diferenţa relaţiilor ARHIVA_OFERTE şi ARHIVA_CERERI

- *Produsul cartezian*

Produsul cartezian reprezintă o operaţie a algebrei relaţionale definită pe două relaţii R_1 şi R_2, în urma căreia se construieşte o nouă relaţie R_3, a cărei schemă se obţine prin concatenarea schemelor relaţiilor R_1 şi R_2, având ca extensie toate combinaţiile tuplurilor din R_1 cu cele din R_2 (operaţie laborioasă).

Notaţie: $R_1 x R_2$

PRODUCT (R_1, R_2)

TIMES (R_1, R_2)

Reprezentarea grafică:

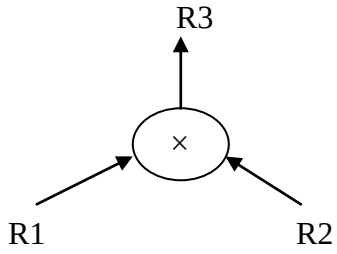
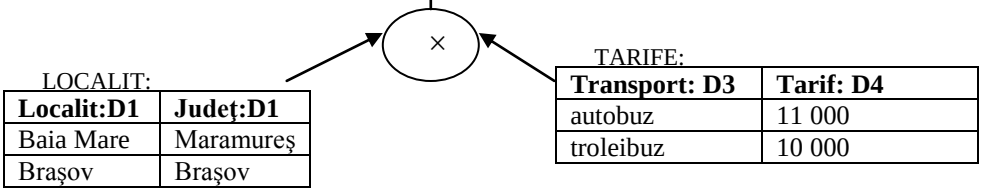


Fig. 4.5. Reprezentarea grafică a produsului cartezian

Exemplu: Operaţia de produs cartezian va fi exemplificată pe un exemplu independent de aplicaţia AGENŢIA IMOBILIARĂ considerată. Astfel:

TRANSPORT:

Localit:D1	**Judeţ:D1**	**Transport:D3**	**Tarif: D4**
Baia Mare	Maramureş	autobuz	11 000
Braşov	Braşov	autobuz	11 000
Baia Mare	Maramureş	troleibuz	10 000
Braşov	Braşov	troleibuz	10 000



LOCALIT:

Localit:D1	**Judeţ:D1**
Baia Mare	Maramureş
Braşov	Braşov

TARIFE:

Transport: D3	**Tarif: D4**
autobuz	11 000
troleibuz	10 000

Fig. 4.6. Produsul cartezian dintre relaţiile LOCALIT şi TARIFE

- *Proiecţia*

Proiecţia reprezintă o operaţie a algebrei relaţionale definită asupra unei relaţii R, în urma căreia se construieşte o nouă relaţie P, în care se găsesc acele atribute din R specificate explicit în cadrul operaţiei.

Prin operaţie de proiecţie se trece de la o relaţie de grad n (are n coloane) la o relaţie de grad mai mic, p (p<n).

Notaţie: $\Pi_{A_i,A_j,...,A_m}(R)$

$R[A_i, A_j, ..., A_m]$

$PROJECT(R, A_i, A_j, ..., A_m)$

Reprezentarea grafică:

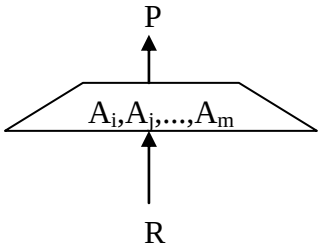
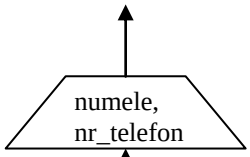


Fig. 4.7. Reprezentarea grafică a operaţiei de proiecţie

Exemplu: Pentru a obţine o listă cu numele şi numerele de telefon ale ofertanţilor/cumpărătorilor, se poate aplica operaţia de proiecţie a relaţiei DATE_PERSOANE asupra atributelor „numele”, „nr_telefon”

REZ:

numele	nr_telefon
Pop Ana	-
Sas Ioan	0362/409209



DATE_PERSOANA:

cnp	numele	adresa	nr_telefon	email
1701205230032	Pop Ana	Str. Viilor, nr.55/4, Oradea, Bihor	-	pa@yahoo.it
2660805270023	Sas Ioan	Str. Victoriei, nr.22/12, Baia Mare, Maramures	0362/409209	-

Fig. 4.8. Proiecţia relaţiei DATE_PERSOANA pe atributele „numele”, „nr_telefon”

- *Selecţia*

Selecţia reprezintă o operaţie din algebra relaţională definită asupra unei relaţii R, în urma căreia se construieşte o nouă relaţie S, cu aceeaşi schema ca R, având extensia construită din acele tupluri din R care satisfac o condiţie menţionată explicit în cadrul operaţiei (se poate interpreta ca „*tăiere orizontală*”: nu toate tuplurile din R satisfac această condiţie sau filtru).

Condiţia precizată în cadrul operaţiei de selecţie se reprezintă sub forma:

atribut, operator de comparaţie, valoare

unde “operator de comparaţie” poate fi unul din semnele <, <=, >=, > sau ≠.

Notaţie: $\sigma_{\text{condiţie}}$ (R)

R [condiţie]

RESTRICT (R, condiţie)

Reprezentarea grafică:

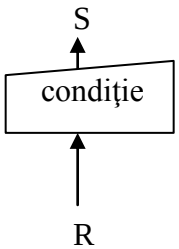
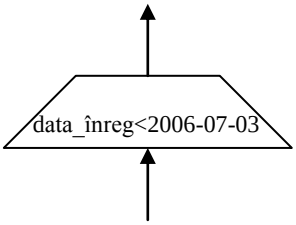


Fig. 4.9. Reprezentarea grafică a operaţiei de selecţie

Exemplu: În cazul în care se doreşte afişarea ofertelor/cererilor anterioare datei de 2006-07-03, se poate aplica operaţia de selecţie asupra relaţiei CERERI_OFERTE, după cum se va vedea în figura 4.10.

OFERTE VECHI:

id_ co#	tipul	cnp	data_ inreg	Cod_loc	Id_ strada	Nr_ imobil	etaj	Pret_ min	Pret_ max
12	oferta	2660805270023	2006-05-27	CJ147	120	22	P	45	47



OFERTE:

id_ co#	tipul	cnp	data_ inreg	Cod_loc	Id_ strada	Nr_ imobil	etaj	Pret_ min	Pret_ max
12	oferta	2660805270023	2006-05-27	CJ147	120	22	P	45	47
13	oferta	1701205230023	2006-07-03	BV230	120	52	2	30	35

Fig. 4.10. Selecţia efectuată asupra relaţiei CERERI_OFERTE

- *Joncţiunea*

Joncţiunea (joinul) reprezintă o operaţie a algebrei relaţionale definită pe două relaţii: R_1 şi R_2, în urma căreia se construieşte o altă relaţie R_3, prin concatenarea unor tupluri din R_1 cu tupluri din R_2 care îndeplinesc o anumită condiţie specificată explicit în cadrul operaţiei.

Notaţie: $R_1 \bowtie R_2$;

JOIN(R1,R2,condiţie)

Reprezentarea grafică:

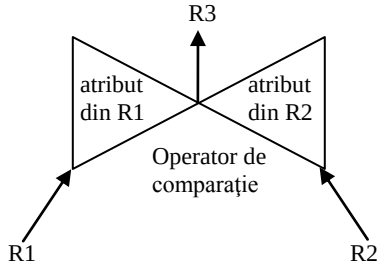


Fig. 4.11. Reprezentarea grafică a operaţiei de joncţiune

Condiţia de concatenare din cadrul operaţiei de joncţiune este de forma:

atribut din R_1 operator de comparaţie atribut din R_2

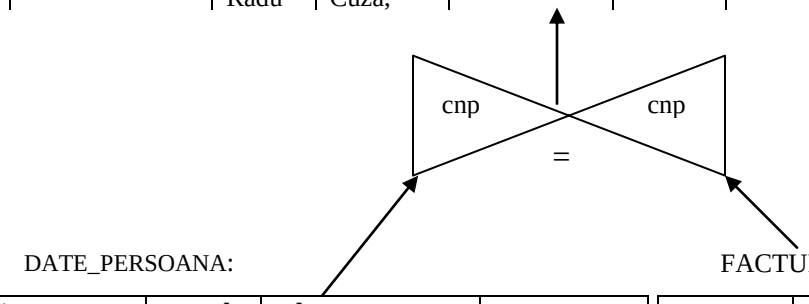
În funcţie de operatorul de comparaţie din condiţia de concatenare, joinul poate fi de mai multe feluri, însă cel mai important este *equijoinul:*

atribut din R_1 = atribut din R_2

Exemplu: Aplicând operaţia de equijoin relaţiilor DATE_PERSOANE şi FACTURI pentru atributul „cnp”, se obţin informaţii referitoare la clienţii care au încheiat facturi. Pentru a nu încărca figura, pentru cele două relaţii s-au ales doar câteva atribute.

REZ

cnp	**numele**	**adresa**	**nr_telefon**	**nr_ factura**	**id_co**	**cnp**
1551212245038	Pop Radu	Str. Al. Cuza,	0744304505	22	43	1551212245038



DATE_PERSOANA:

cnp#	**numele**	**adresa**	**nr_ telefon**
1551212245038	Pop Radu	Str. Al. Cuza, nr.4/34, Ploiesti	0744304505
2560405570053	Chis Alina	Str. Luminii, 76, Buzau	0721435622

FACTURI:

nr_ factura#	**id_co**	**cnp**
22	43	1551212245038

Fig. 4.12. Operaţia de equijoin a relaţiilor DATE_PERSOANA şi FACTURI

Observaţie: Operaţia de joncţiune se poate exprima cu ajutorul operaţiilor de produs cartezian şi selecţie, rezultatul unui join fiind asemenea cu cel al operaţiei de selecţie asupra unui produs cartezian:

JOIN (R_1, R_2, condiţie) = RESTRICT (PRODUCT (R_1, R_2), condiţie).

Este indicată utilizarea joinului în locul produsului cartezian, de câte ori este posibil.

Tipuri de joncţiuni

În funcţie de
- tipul condiţiilor de conectare
- modul de definire a schemei
- extensia relaţiei rezultate prin joncţiune,

vom studia:
- joncţiunea naturală
- joncţiunea externă
- semijoncţiunea.

➢ *Joncţiunea naturală*

Joncţiunea naturală este o operaţie definită pe două relaţii R_1 şi R_2, în urma căreia se construieşte o nouă relaţie R_3, a cărei schemă este obţinută prin reuniunea atributelor din relaţiile R_1 şi R_2 (atributele cu aceleaşi nume se iau o singură dată) şi a cărei extensie conţine tuplurile obţinute prin concatenarea tuplurilor din R_1 cu cele din R_2 care prezintă aceleaşi valori pentru atributele cu aceleaşi nume.

Joncţiunea naturală elimină inconvenientul ce apare în cazul equijoinului şi anume: schema relaţiei în cazul equijoinului conţine toate atributele celor două relaţii. Astfel, în relaţia R_3 a joncţiunii naturale, atributele cu acelaşi nume vor apărea o singură dată.

Reprezentarea grafică:

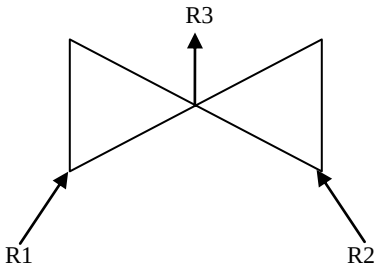
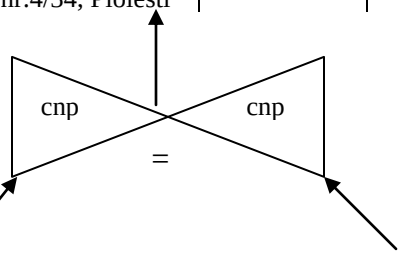


Fig. 4.13. Reprezentarea grafică a operaţiei de joncţiune naturală

Exemplul 1: Reluând exemplul anterior, prin joncţiunea naturală se elimină atributul repetitiv „cnp”.

REZ

cnp	numele	adresa	nr_telefon	nr_ factura	id_co
1551212245038	Pop Radu	Str. Al. I. Cuza, nr.4/34, Ploiesti	0744304505	22	43



DATE_PERSOANA:

cnp#	numele	adresa	nr_ telefon
1551212245038	Pop Radu	Str. Al. Cuza, nr.4/34, Ploiesti	0744304505
2560405570053	Chis Alina	Str. Luminii, 76, Buzau	0721435622

FACTURI:

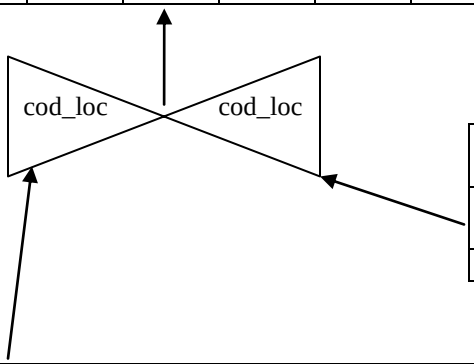
nr_ factura#	id_co	cnp
22	43	1551212245038

Fig. 4.14. Operaţia de joncţiune naturală a relaţiilor DATE_PERSOANA şi FACTURI

Exemplul 2: Dacă se doreşte aflarea denumirilor localităţilor în care sunt oferte sau cereri, cum în relaţia CERERI_OFERTE se află doar codul localităţii respective iar în relaţia LOCALITATI este asociat fiecărui cod de localitate denumirea localităţii, trebuie să se realizeze o joncţiune naturală între aceste două relaţii. Astfel rezultatul joncţiunii va fi cel prezentat în figura 4.15.

REZ:

id_ co#	tipul	cnp	data_ inreg	cod_ loc	id_ strada	nr_ imobil	pret_ min	pret_ max	tip_ solutionare	nume _loc	simbol_ judet
12	oferta	1701205230023	2006-07-03	BV230	120	52	30	35	da	Brasov	BV
234	cerere	2760805270024	2006-05-27	CJ400	120	22	45	47	da	Cluj-Napoca	CJ



LOCALITATI:

cod_loc#	nume_loc	simbol _judet
CJ400	Cluj-Napoca	CJ
BV230	Brasov	BV

CERERI_OFERTE:

id_ co#	tipul	cnp	data_ inreg	cod_loc	id_ strada	nr_ imobil	pret_ min	pret_ max	tip_ solutionare
12	oferta	1701205230023	2006-07-03	BV230	120	52	30	35	da
234	cerere	2760805270024	2006-09-21	CJ400	120	22	45	47	da
44	oferta	2661111246642	2006-09-17	MM430	133	4	50		nu

Fig. 4.15. Joncţiunea naturală a relaţiilor CERERI_OFERTE şi LOCALITATI

- *Joncţiunea externă*

Joncţiunea externă este operaţia definită pe două relaţii: R_1 şi R_2, în urma căreia se obţine o nouă relaţie R_3 prin joncţionarea relaţiilor R_1 şi R_2. În relaţia R_3 apar şi tuplurile din R_1 şi R_2 care nu au participat la join (atributul de joncţiune – cel care are acelaşi nume şi în relaţia R_1 şi în relaţia R_2 – nu prezintă aceleaşi valori). Aceste tupluri sunt completate cu valoarea „NULL”.

Joncţiunea externă elimină inconvenientul cauzat de joncţiunea internă şi anume pierderea de tupluri (vezi figura 4.15; tuplul <44,”oferta”, 2661111246642, 2006-09-17, MM430, 133, 4, 50, “nu”> nu mai apare în relaţia REZ, deoarece simbolul „MM430” corespunzătoare atributului cod_loc din relaţia CERERI_OFERTA nu figurează printre valorile atributului cu acelaşi nume din relaţia LOCALITATI.

Reprezentarea grafică:

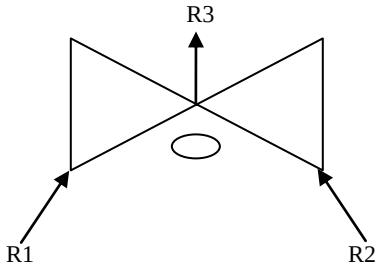
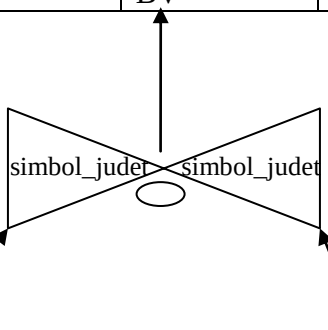


Fig. 4.16. Reprezentarea grafică a operaţiei de join extern

Exemplu: Joncţiunea externă este o operaţie care din punct de vedere al programării prezintă inconvenientul manipulării valorilor nule. În relaţia REZ, judeţului Braşov nu i s-a asignat o localitate, deci nici codul localităţii.

REZ:

cod_loc#	nume_loc	simbol_judet	nume_judet
430	Baia Mare	MM	Maramures
435	Borsa	MM	Maramures
400	Cluj-Napoca	CJ	Cluj
710	Botosani	BT	-
-	-	BV	Brasov



LOCALITATI:

cod_loc#	nume_loc	simbol_judet
430	Baia Mare	MM
435	Borsa	MM
400	Cluj-Napoca	CJ
710	Botosani	BT

JUDETE:

simbol_judet#	nume_judet
MM	Maramures
CJ	Cluj
BV	Brasov

Fig. 4.17. Operaţia de joncţiune externă a relaţiilor LOCALITĂŢI şi JUDEŢE

- Semijoncţiunea

Semijoncţiunea este o operaţie definită pe două relaţii R_1 şi R_2, în urma căreia se construieşte o nouă relaţie R_3, a cărei extensie conţine tuplurile relaţiei R_1 care participă la joncţiunea celor două relaţii, conservând atributele relaţiei R_1.

Notaţie: $R_1 \ltimes R_2$;

SEMIJOIN(R_1, R_2).

Reprezentarea grafică:

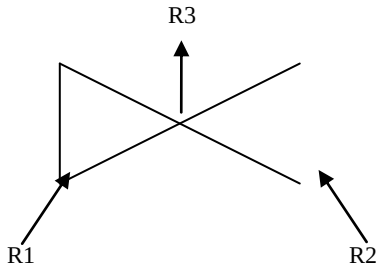
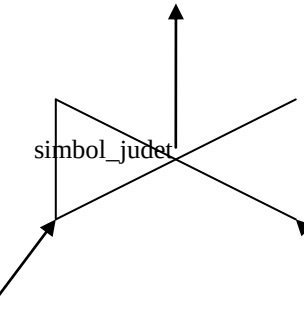


Fig. 4.18. Reprezentarea grafică a operaţiei de semijoncţiune

Exemplu: Semijoncţiunea următoare realizează lista localităţilor care au referinţă în relaţia JUDETE.

REZ:

cod_loc#	nume_loc	**simbol_judet**
430	Baia Mare	MM
435	Borsa	MM
400	Cluj-Napoca	CJ



LOCALITATI:

cod_loc#	**nume_loc**	**simbol_judet**
430	Baia Mare	MM
435	Borsa	MM
400	Cluj-Napoca	CJ
710	Botosani	BT

JUDETE:

simbol_judet#	**nume_judet**
MM	Maramures
CJ	Cluj
BV	Brasov

Fig. 4.19. Operaţia de semijoncţiune a relaţiilor LOCALITATI şi JUDETE

Observaţia 1. Această operaţie a fost introdusă de P.A. Bernstein, fiind necesară la optimizarea cererilor de date.

Observaţia 2. Semijoncţiunea produce acelaşi rezultat ca operaţia de proiecţie pe atributele din relaţia R_1 efectuată asupra joncţiunii dintre R_1 şi R_2
PROJECT (JOIN (R_1, R_2, condiţia), A_1, A_2, A_3)=SEMIJOIN (R_1, R_2).

- *Intersecţia*

Intersecţia reprezintă o operaţie a algebrei relaţionale definită pe două relaţii, R_1 şi R_2, ambele cu aceeaşi schemă, în urma căreia se construieşte o nouă relaţie R_3, cu schema identică cu a operanzilor şi cu extensia formată din tuplurile comune lui R_1 şi R_2.

Notaţie: $R_1 \cap R_2$
INTERSECT (R_1, R_2)
AND (R_1, R_2)

Reprezentarea grafică:

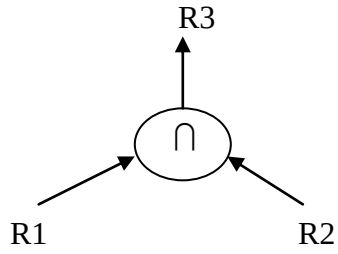
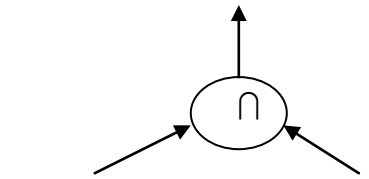


Fig. 4.20. Reprezentarea grafică a operaţiei de intersecţie

Exemplu:

REZ:

localitate	judete	populatie
Brasov	Brasov	350 000



ORASE:

localitate	judete	populatie
Borsa	Maramures	27 000
Brasov	Brasov	350 000

MUNICIPII:

localitate	judete	populatie
Baia Mare	Maramures	148 270
Brasov	Brasov	350 000

Fig. 4.21. Intersecţia relaţiilor ORASE şi MUNICIPII

Observaţie: Intersecţia se poate exprima prin intermediul unor operaţii de bază. De aceea intersecţia este o operaţie derivată.

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

- *Diviziunea*

Diviziunea reprezintă o operaţie a algebrei relaţionale definită asupra unei relaţii R cu schema $R(A_1:D_1, \dots, A_p:D_k, \dots, A_{p+1}:D_i, \dots, A_n:D_m)$, în urma căreia se construieşte o nouă relaţie Q cu ajutorul unei relaţii r cu schema r $(A_{p+1}:D_l, \dots, A_n:D_m)$, relaţia Q având schema: $Q(A_1:D_1, \dots, A_p:D_k)$.

Tuplurile relaţiei Q concatenate cu tuplurile relaţiei r permit obţinerea tuplurilor relaţiei R.

Notaţie: R÷r

Division (R, r).

Reprezentarea grafică:

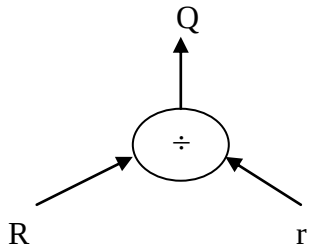


Fig. 4.22. Reprezentarea grafică a operaţiei de diviziune

Exemplul se lasa ca exerciţiu suplimentar.

- *Complementarea*

Complementarea reprezintă o operaţie (adiţională) a algebrei relaţionale definită asupra unei relaţii R, în urma căreia se construieşte o nouă relaţie C, numită complementarea relaţiei R. Extensia relaţiei C va conţine ansamblul tuplurilor din

produsul cartezian al domeniilor asociate atributelor relaţiei, care nu figurează în extensia relaţiei considerate.

Notaţii: ┐R
NOT (R)
COMP(R)

Exemplu: Fie relaţia: R(A_1:D_1, A_2:D_2), unde
A_1 = culoare;
A_2 = număr;
D_1 = {“Roşu”, „Galben”, „Albastru”}
D_2 = {1, 2, 3}
reprezentată prin tabelul:
R:

A_1:D_1	A_2.D_2
Roşu	1
Roşu	2
Galben	3

a) relaţia R

Complementarea relaţiei R va fi relaţia NOT (R) repezintată prin tabelul:
NOT (R):

A_1:D_1	A_2:D_2
Roşu	3
Galben	1
Galben	2
Albastru	1
Albastru	2
Albastru	3

b) relaţia not R

Fig. 4.24. Complementarea relaţiei R

Observaţie: Complementaritatea este puţin utilizată, datorită rezultatului foarte mare de tupluri.

- *Splitarea*

Splitarea (*spargerea*) reprezintă o operaţie (adiţională) a algebrei relaţionale definită asupra unei relaţii R, în urma căreia se construiesc două relaţii R_1 şi R_2 cu aceeaşi schemă cu R, relaţii obţinute pe baza unei condiţii definite asupra atributelor din R.

Extensia lui R_1 conţine tuplurile din R care verifică condiţia specificată, iar R_2 conţine tuplurile din R care nu verifică această condiţie.

Exemplu: Considerând relaţia R din figura 4.24 (a) şi condiţia A_2>2, operaţia de splitare a relaţiei R produce relaţiile R_1 şi R_2 reprezentate prin tabelele:

R_1

A_1:D_1	A_2: D_2
Galben	3

R_2

A_1:D_1	A_2:D_2
Roşu	1
Roşu	2

Figura 4.25. Rezultatul operaţiei de splitare a relaţiei R din figura 4.24 (a) pe baza condiţiei A_2>2

- *Închiderea tranzitivă*

Închiderea tranzitivă este o operaţie (adiţională) a algebrei relaţionale, definită asupra unei relaţii R, a cărei schemă conţine două atribute A_1 şi A_2 cu acelaşi domeniu asociat, operaţie care constă în adăugarea la relaţia R a tuplurilor care se obţin succesiv prin tranzitivitate: dacă în R există tuplurile: <a,b> şi <b,c> se va adăuga la R tuplul <a,c>.

Notaţie: $\tau(R)$
R^+
CLOSE(R)

Exemplu:

R:

Persoana: D	Urmaş: D
Ana	Maria
Ana	Ion
Ion	Vasile
Ion	Nicoleta
Maria	Oana

a)

$\tau(R)$:

Persoana: D	Urmaş: D
Ana	Maria
Ana	Ion
Ion	Vasile
Ion	Nicoleta
Maria	Oana
Ana	Oana
Ana	Vasile
Ana	Nicoleta

b)

Fig. 4.26. Închiderea tranzitivă a relaţiei R

CURS 5. Restricţii de integritate ale modelului relaţional

3.3 Restricţii de integritate ale modelului relaţional

Restricţiile de integritate ale modelului relaţional reprezintă cerinţe pe care trebuie să le îndeplinească datele din cadrul bazei de date pentru a putea fi considerate corecte şi coerente în raport cu lumea reală pe care o reflectă. Dacă o bază de date nu respectă aceste cerinţe, ea nu poate fi utilizată cu un maxim de eficienţă.

Restricţiile sunt de două tipuri:

- restricţii de integritate **structurale**, care se definesc prin egalitatea sau inegalitatea unor valori din cadrul relaţiilor:
  - restricţia de unicitate a cheilor;
  - restricţia entităţii;
  - dependenţele între ele;
- restricţii de integritate de **comportament** care ţin cont de semnificaţia valorilor din cadrul bazei de date.

Utilizarea modelului relaţional nu impune definirea şi verificarea tuturor acestor tipuri de restricţii de integritate. Din acest punct de vedere există restricţii de integritate **minimale**. Acestea sunt obligatoriu de definit şi de respectat când se lucrează cu modelul relaţional. Dintre restricţiile minimale fac parte:

- restricţia de unicitate a cheii;
- restricţia referenţială;
- restricţia entităţii.

Alte restricţiii de integritate ar fi

- dependenţele;
- restricţii de comportament.

Restricţii de integritate minimale

Restricţiile de integritate minimale sunt definite în raport cu noţiunea de **cheie** a unei relaţii. Cheia identifică un tuplu în cadrul unei relaţii fără a face apel la toate valorile din tuplu.

Cheia unei relaţii reprezintă ansamblul minimal de atribute prin care se poate identifica în mod unic orice tuplu al relaţiei.

Oricare relaţie posedă cel puţin o cheie:

- cheie *simplă*, când cheia este construită dintr-un singur atribut;
- cheie *compusă*, când cheia este construită din mai multe atribute.

Exemplul 1:

R1:

$\underline{A}$:D_A	B:D_B
a_1	b_1
a_2	b_3
a_3	b_2

a) cheie simplă

R2:

$\underline{A}\cdot D_A$	$\underline{B}$:D_B
a_1	b_1
a_1	b_2
a_2	b_3
a_3	b_2

b) cheie compusă

Fig. 5.1. Chei simple şi chei compuse

Determinarea cheii unei relaţii necesită cunoaşterea tuturor extensiilor posibile, nu numai a aceleia din momentul în care se stabileşte cheia. Astfel, presupunând că R_1 şi R_2 sunt versiuni ale aceleiaşi relaţii R la momente de timp diferite: t_1, respectiv t_2, alegerea la momentul t_1 drept cheie atributul A a relaţiei R se dovedeşte a fi greşită, întrucât atributul A nu face posibilă identificarea unică a tuplurilor şi la momentul t_2. Cheia relaţiei R este reprezentată, prin urmare, de perechea de atribute (A,B).

Exemplul 2:

JUDETE:

simbol_judet#	**nume_jud**
MM	Maramures
AB	Alba

STRAZI:

simbol_judeţ	**cod_loc#**	**id_stradă#**	**nume_str**
BV	BV230	120	Independenţei
BV	BV230	078	Gării
CJ	CJ147	120	Cireşilor

Fig. 5.2. Chei simple şi chei compuse în cadrul relaţiilor JUDEŢE, respectiv STRĂZI

Observaţie: Cheia relaţiei JUDEŢE este „simbol_judeţ”, deoarece fiecare judeţ are o codificare unică a denumirii sale, deci fiecărui judeţ îi corespunde un singur simbol. Cheia relaţiei STRĂZI este compusă din atributele „cod_loc” şi „id_strada”. Dacă s-ar alege drept cheie primară doar atributul „id_strada”, acesta nu ar identifica în mod unic numele unei străzi dintr-o localitate, id-ul străzii respective putându-se regăsi şi în alte localităţi ale aceluiaşi judeţ sau a altui judeţ. La fel, dacă s-ar alege drept cheie primară atributul „cod_loc”, acesta nu ar mai identifica în mod unic un tuplu, deoarece într-o localitate există mai multe străzi.

O relaţie poate avea mai multe combinaţii de atribute, cu proprietatea de identificare unică a tuplurilor. Se spune în acest caz că relaţia posedă mai mulţi *candidaţi cheie* (chei candidate).

Definiţia 1. Se numeşte *cheie primară*, cheia aleasă dintre cheile candidate care să servească în mod efectiv la identificarea tuplurilor.

Cheia primară nu poate fi reactualizată. Cheia primară a unei relaţii nu este altceva decât atributul de identificare a unei entităţi, prin urmare se reprezintă fie prin subliniere, fie urmate de semnul #.

Definiţia 2. Se numeşte *cheie externă* atributul/grupul de atribute dintr-o relaţie R_1 a cărui/căror valori sunt definite pe acelaşi domeniu/aceleaşi domenii ca şi cheia primară a unei alte relaţii R_2 şi care are rolul de a modela asocierea între entităţile

reprezentate prin relaţiile R_1 şi R_2. În acest caz, R_1 se numeşte *relaţie care referă*, iar R_2 se numeşte *relaţie referită*.

Exemplul 1:

DATE_PERSOANA:

cnp	numele	simbol_ judet
1701205230023	Sas Ioan	BV
2660805270023	Pop Ana	CJ

JUDETE:

simbol_judet	descriere
BV	Brasov
CJ	Cluj
MM	Maramures

Fig. 5.3. Reprezentarea legăturii dintre relaţiile DATE_PERSOANA şi JUDETE cu ajutorul cheii externe „simbol_judet" din cadrul relaţiei DATE_PERSOANA

Observaţia1: În exemplul de mai sus, relaţia DATE_PERSOANA are drept cheie primară atributul „cnp", iar ca şi cheie externă atributul „simbol_judet", iar relaţia JUDETE are drept cheie primară cheia „simbol_judet". Relaţia DATE_PERSOANA este relaţia care referă, iar JUDETE este relaţia referită.
Observaţia2: Între cele două relaţii (entităţi) avem următoarea asociere:

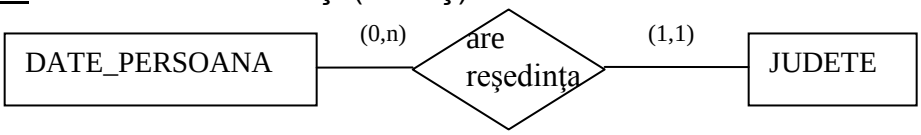


Fig. 5.4. Asocierea dintre entităţile DATE_PERSOANA şi JUDETE

Exemplul 2:

JUDETE:

simbol_judet#	**nume_jud**
BV	Braşov
CJ	Cluj

LOCALITATI:

simbol_judeţ	**cod_loc#**	**nume_loc**
BV	BV230	Braşov
CJ	CJ147	Dej

STRĂZI:

simbol_judet	**cod_loc#**	**id_strada#**	**nume_str**
BV	BV230	001	Independenţei
BV	BV230	002	Gării
CJ	CJ147	003	Cireşilor

Fig. 5.5. Reprezentarea legăturii între relaţiile JUDETE şi LOCALITATI cu ajutorul cheilor externe „simbol_judet" şi „cod_localitate"

- *Restricţia de unicitate a cheii*

Restricţia de unicitate a cheii impune ca într-o relaţie să nu existe două linii identice (linii care să nu conţină aceleaşi valori pentru toate atributele). Altfel spus, restricţia de unicitate a cheii impune ca într-o relaţie să nu existe două tupluri cu o aceeaşi valoare pentru atributul cheie.

Exemplele 1 şi 2 respectă restricţia de unicitate a cheii.

➢ *Restricţia referenţială*

Restricţia referenţială impune ca într-o relaţie R_1 care referă o relaţie R_2, valorile cheii externe să figureze printre valorile cheii primare din R_2 sau să fie valori „null" (nedefinite). R_1 şi R_2 nu trebuie să fie neapărat distincte.

Exemplele 1 şi 2 prezintă un mecanism de legare a relaţiilor şi respectă restricţia referenţială a cheii.

➢ *Restricţia entităţii*

Restricţia entităţii impune ca într-o relaţie atributele cheii primare să fie nenule. (Atributele cheie să nu conţină valori nule). Dacă există valori „null", cheia îşi poate pierde rolul de identificator de tuplu. Astfel, la încărcarea unui tuplu, valoarea cheii trebuie să fie cunoscută, pentru a se putea verifica faptul că această valoare nu există deja încărcată.

Exemplele 1 şi 2 respectă restricţia entităţii.

Relaţia REZ din figura 4.17 încalcă restricţia entităţii deoarece există chei primare ce conţin valori nule, după cum este cazul judeţului Braşov.

Alte restricţii de integritate

În categoria restricţii de integritate intră următoarele tipuri de restricţii:

a) restricţii referitoare la dependenţa datelor:
- dependenţă funcţională;
- dependenţă multivaloare;

b) restricţii de comportament:
- restricţii de domeniu;
- restricţii temporale.

Restricţiile referitoare la dependenţa datelor, reprezintă modul în care datele depind unele de altele.

➢ *Dependenţele funcţionale*

Dependenţele funcţionale reprezintă dependenţa între date prin care se poate identifica un atribut/grup de atribute prin intermediul altui atribut/grup de atribute.

Dacă X şi Y sunt două subansamble de atribute ale atributelor relaţiei R, spunem că între X şi Y există o *dependenţă funcţională*, notată $X \to Y$, dacă şi numai dacă:

(i) fiecare valoare a lui X poate fi asociată unei singure valori din Y, şi

(ii) două valori distincte ale lui X nu pot fi asociate decât aceleiaşi valori ale lui Y.

X se numeşte *determinantul* (*sursa*) dependenţei, iar Y se numeşte *determinatul* (*destinaţia*) dependenţei.

Exemple: Următoarele atribute se află în dependenţă funcţională:

- cod_poştal→localitate, deoarece unui cod poştal îi corespunde o singură localitate (sensul dependenţei este foarte important, deoarece, viceversa ar însemna: o localitate are un singur cod_poştal);

- nr_factură→data_factură, deoarece cunoaşterea numărului facturii determină cu exactitate data facturii.

Două atribute *nu sunt în dependenţă funcţională*, notată X $\not\rightarrow$ Y, atunci când cunoaşterea unei valori a primului atribut fie nu permite cunoaşterea nici uneia dintre valorile celui de al doilea atribut, fie permite cunoaşterea mai multor valori ale celui de al doilea atribut.

Exemplu: Următorul atribut nu se află în dependenţă funcţională: **c**np $\not\rightarrow$ nr_factură, deoarece pentru o persoană se pot întocmi mai multe facturi (aferente fiecărei vânzări).

În cadrul modelului relaţional, o dependenţă funcţională este reprezentată printr-o săgeată ce porneşte din sursă şi se termină în destinaţie.

FACTURI:

nr_factura#	**data_facturii**	**cnp**
1	2006-07-05	1701205230023
2	2006-06-28	2581023457723

Fig. 5.6. Reprezentarea dependenţei funcţionale între atributele „nr_factura" şi „data_facturii"

- *Dependenţele multivaloare*

Dependenţele multivaloare reprezintă dependenţa în care un atribut/ grup de atribute poate reprezenta/ identifica mai multe valori pentru o singură valoare a unui alt atribut/ grup de atribute.

Dacă X,Y şi Z sunt trei subansambluri de atribute ale atributelor relaţiei R, spunem că între X şi Y există o *dependenţă multivaloare*, notată $X \rightarrow\rightarrow Y$ sau $X \rightarrow\rightarrow Y \mid Z$, dacă şi numai dacă:

(i) la fiecare valoare a lui X poate fi asociată una sau mai multe valori ale lui Y, şi
(ii) această asociere nu depinde de apariţiile lui Z.

Altfel spus, dacă $X \rightarrow\rightarrow Y$ şi (x,y,z), (x',y',z') sunt două tupluri din R, atunci şi (x,y',z), (x,y,z') sunt tupluri din R.

Exemplu: În relaţia OFERTE (alcătuită din atributele: „id_tip_oferte", „cnp" şi „simbol_judet") valorile atributului „id_tip_oferte" au următoarea semnificaţie: 01 desemnează imobil de tip apartament, iar 02, imobil de tip casă. Astfel, următoarea relaţie conţine dependenţe multivaloare:

OFERTE:

id_tip_oferte	cnp	simbol_judet
01	1701205230023	MM
01	2581023457723	MM
02	1701205230023	SM
01	2581023457723	SM
01	2581023457723	SM
02	2581023457723	CJ

deoarece

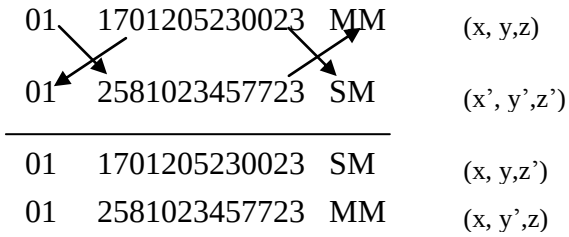


Fig. 5.7. Relaţia OFERTE în care există dependenţă multivaloare

CURS 6. Prelucrarea/evaluarea şi optimizarea cerinţelor

Regulile lui Codd

Prin *sistem de gestiune a bazelor de date relaţionale* (SGBDR) se înţelege un SGBD care utilizează drept concepţie de organizare a datelor modelul relaţional.

Definirea unui SGBDR impune o detaliere a caracteristicilor pe care trebuie să le prezinte un SGBD pentru a putea fi considerat relaţional. În acest sens, Codd a formulat (în 1985) 13 reguli, care exprimă cerinţele pe care trebuie să le satisfacă un SGBD. Aceste reguli sunt deosebit de utile în evaluarea unui SGBDR.

R_0: Regula privind gestionarea datelor la nivel de relaţie.
- sistemul trebuie să gestioneze BD numai prin mecanisme relaţionale.

R_1: Regula privind reprezentarea logică a datelor.
- într-o bază de date relaţionată, informaţia este reprezentată la nivel logic sub forma unor tabele (relaţii);
- acest lucru înseamnă că toate datele trebuie să fie memorate şi prelucrate în acelaşi mod.

R_2: Regula privind reprezentarea fizică a datelor
- orice data din baza de date relaţionată trebuie să poată fi accesată prin specificarea:
  - numelui relaţiei;
  - valorii cheii primare;
  - numele atributului.

R_3: Regula privind valorile nule
- sistemul trebuie să permită declararea şi manipularea sistematică a valorilor NULL (semnifică lipsa unor date);
- valorile NULL diferă de şirurile de caractere „spaţiu”, şirurile vide de caractere.
- valorile NULL sunt deosebit de importante în implementarea restricţiilor de integritate:
  - integritatea entităţilor;
  - integritatea referenţială.

R_4: Regula privind metadatele
- utilizatorii autorizaţi trebuie să poată aplica asupra descrierii bazei de date aceleaşi operaţii ca şi asupra datelor obişnuite.

R_5: Regula privind facilităţile limbajelor de utilizare:
- trebuie să existe cel puţin un limbaj care să exprime oricare din următoarele operaţii:
  - definirea relaţiilor;
  - sa vizualizeze datele;
  - să regăsească informaţia;
  - să poată reactualiza informaţia;
  - să verifice şi să corecteze datele de intrare, etc.
- în general, toate implementările SQL respectă această regulă.

R_6: Regula privind actualizarea tabelelor virtuale:
- toate tabelele/relaţiile virtuale trebuie să poată fi actualizate.
- nu toate tabelele virtuale sunt teoretic actualizate.

Exemplu: Fie tabela de bază PROD, cu următoarea schemă PROD(Denp:D_1, Cant:D_2, Pret:D_3), cu ajutorul tabelei PROD este definită o tabelă virtuală DISP, cu schema: DISP (Denp:D_1, Cant:D_2, Pret:D_3, Val:D_4). Valorile atributului „Val" se calculează astfel:

Val=Cant*Pret.

Presupunem că se doreşte schimbarea preţului unitar la un anumit produs, această schimbare trebuie efectuată în tabela de bază PROD, atributul „Pret" din tabela virtuală DISP, fiind actualizabil, întrucât actualizarea se poate propaga spre tabela de bază.

Presupunem că se doreşte schimbarea valorii „Val" la un anumit produs:

- modificarea de la tabela virtuală spre tabela de bază nu mai este posibilă, atributul „Val" nu este actualizabil, deoarece schimbarea valorii „Val" se poate datora schimbării cantităţii „Cant" şi/sau a preţului unitar „Pret".
- astfel trebuie să existe un mecanism prin care să se poată determina dacă anumite vizualizări pot fi modificate sau nu.

- majoritatea implementărilor SQL îndeplinesc această cerinţă.

R_7: Regula privind inserările, modificările şi ştergerile din baza de date.

- un SGBDR nu trebuie să oblige utilizatorul să caute într-o relaţie, tuplu cu tuplu, pentru a regăsi informaţia dorită;
- această regulă exprimă cerinţa ca în operaţiile prin care se schimbă conţinutul bazei de date să se lucreze la un moment dat pe o întreagă relaţie.

R_8: Regula privind independenţa fizică a datelor

- o schimbare a structurii fizice a datelor nu trebuie să blocheze funcţionarea programelor de aplicaţii;
- într-un SGBDR trebuie să se separe aspectul fizic al datelor (stocare sau acces la date) de aspectul logic al datelor.

R_9: Regula privind independenţa logică a datelor.

- o schimbare a relaţiilor bazei de date nu trebuie să afecteze programele de aplicaţie.

R_{10}: Regula privind restricţiile de integritate

- restricţiile de integritate trebuie să fie definite într-un limbaj relaţional, nu în programul de aplicaţie.

R_{11}: Regula privind distribuirea geografică a datelor

- distribuirea datelor pe mai multe calculatoare dintr-o reţea de comunicaţii de date, nu trebuie să afecteze programele de aplicaţie.

R_{12}: Regula privind prelucrarea datelor la nivelul de bază

- dacă sistemul posedă un limbaj de bază orientat pe prelucrarea de tupluri şi nu pe prelucrarea relaţiilor, acest limbaj nu trebuie să fie utilizat pentru a evita restricţiile de integritate (se introduc inconsistenţe).

Clasificarea regulilor lui Codd

În funcţie de tipul de cerinţe pe care le exprimă, regulile sunt grupate în 5 categorii:

1. Reguli de bază: R_0 şi R_{12};

2. Reguli structurale: R_1 şi R_6;
3. Reguli privind integritatea datelor: R_3 şi R_{10};
4. Reguli privind manipularea datelor: R_2, R_4, R_5, R_7;
5. Reguli privind independenţa datelor: R_8, R_9, R_{11}.

Evaluarea/prelucrarea cerinţelor şi optimizarea

În SGBDR interfaţa cu utilizatorul este de tip neprocedural. Utilizatorul defineşte datele pe care doreşte să le vizualizeze fără a da algoritmi de acces. Sistemul trebuie să convertească cererea utilizatorului într-o cerere optimală.

Evaluarea unei cereri se efectuează în trei etape:

1. Analiza cererii ce constă în studierea sintactică şi semantică a cererii pentru a verifica corectitudinea sa şi a simplifica criteriul de căutare.
2. Ordonarea presupune:
   - descompunerea cererii într-o mulţime de operaţii elementare şi
   - determinarea ordinii optimale a acestor operaţii.
3. Execuţia în paralel şi/sau secvenţială a operaţiilor elementare 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.

Ideea generală este concretizată în schema de mai jos:

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

adică optimizarea cererilor de date se realizează prin parcurgerea următoarelor etape:

1. exprimarea cererilor sub forma unei expresii algebrice relaţionale;
2. aplicarea unor transformări algebrice asupra expresiilor obţinute în etapa precedentă, în scopul executării mai eficiente a lor;
3. planul de execuţie;
4. optimizarea.

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.).

Strategiile de optimizare pot fi de două tipuri:

1. Strategii generale de optimizare (independente de modul de memorare al datelor);

2. Strategii specifice anumitor SGBDR (ţin cont de modul de memorare al datelor).

Strategiile generale de optimizare a cererilor de date sunt:

Regula de optimizare 1. Selecţiile se execută cât mai devreme posibil. Motivaţia acestei reguli este că selecţiile reduc substanţial dimensiunea relaţiilor.

Regula de optimizare 2. Produsele 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.

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.

CURS 7. Tehnica normalizării relaţiilor

La proiectarea structurii unei baze de date relaţionale trebuie stabilite (după cum s-a văzut în cursurile anterioare) în primul rând tabelele în care vor fi memorate datele şi asocierile dintre tabele. Acestea sunt stabilite într-o formă iniţială, după care, prin rafinare succesivă se ajunge la forma definitivă. **Acestei structuri iniţiale îi sunt aplicate un set de reguli care reprezintă paşii de obţinere a unei *baze de date normalizate***. Dacă o bază de date nu este normalizată ea nu poate fi utilizată cu un maxim de eficienţă. Algoritmul de normalizare a bazelor de date relaţionale precum şi paşii acestuia au fost descrişi de către E. F. Codd în 1972.

***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). Scopul normalizării este de a suprima *redundanţele* logice şi de a evita *anomaliile* la reactualizare.

Exemplu: Pentru a evidenţia câteva exemple de redundanţe şi anomalii, se va considera cazul relaţiei iniţiale OFERTANTI. Pentru a nu încărca relaţia, se vor considera valori ale atributelor prescurtate.

OFERTANTI:

cnp#	numele	adresa_ client	nr_ telefon	oferta	adresa_ imobil
Cnp1	**N1**	**Str. Victoriei, nr.22/12, Baia Mare, Maramures**	**Nr1**	casa	A_imobil1
Cnp1	**N1**	**Str. Victoriei, nr.22/12, Baia Mare, Maramures**	**Nr1**	hala	A_imobil2
Cnp2	N2	Str. Viilor, nr.55/4, Oradea, Bihor	Nr2	casa	A_imobil3

Fig.7.1. Relaţia OFERTANTI

- **Redundanţa logică**: Tripletul („N1”, “Str. Victoriei, nr.22/12, Baia Mare, Maramures”, ‘Nr1’) apare de două ori.
- **Anomalii la inserare**: Dacă o persoană oferă spre vânzare mai multe imobile, pentru înregistrarea ofertei trebuie rescris codul numeric personal încă o dată, deci cheia devine duplicat.
- **Anomalii de ştergere**: Ştergerea unei persoane din baza de date atrage după sine pierderea informaţiilor despre oferta respectivă.
- **Anomalii la modificare**: Dacă se modifică numele străzii Victoriei din localitatea Baia Mare în strada Independenţei, modificarea trebuie efectuată pentru fiecare ofertă din Baia Mare amplasată pe strada Victoriei. Dacă ar exista 25 de oferte în această localitate pe strada Victoriei, costul modificării ar fi mare pentru a modifica toate înregistrările. Această redundanţă este eliminată dacă atributul „adresa” este împărţit în alte trei atribute: „simbol_judet”, „cod_loc”, „id_strada”.

Valorile acestea vor fi codul judeţului, localităţii, respectiv a străzii preluate din relaţiile deja existente JUDETE, LOCALITATI, respectiv STRAZI. În acest caz, modificarea se face doar o singură dată, în tabela STRAZI.

Normalizarea

Codd a definit iniţial 3 forme normale, notate prin FN_1, FN_2 şi FN_3. Întrucât într-o primă formulare, definiţia FN_3 ridică ceva probleme, Codd şi Boyce au elaborat o nouă variantă, cunoscută sub numele de Boyce-Codd Normal Form (BCNF). Astfel BCNF este reprezentată separat în majoritatea lucrărilor. R. Fagin a tratat cazul FN_4 şi FN_5.

O relaţie este într-o *formă normală* dacă satisface o mulţime de constrângeri specificată în figura 7.2. De exemplu, se spune că o relaţie se află în a doua formă normală FN2 dacă şi numai dacă se află în FN1.

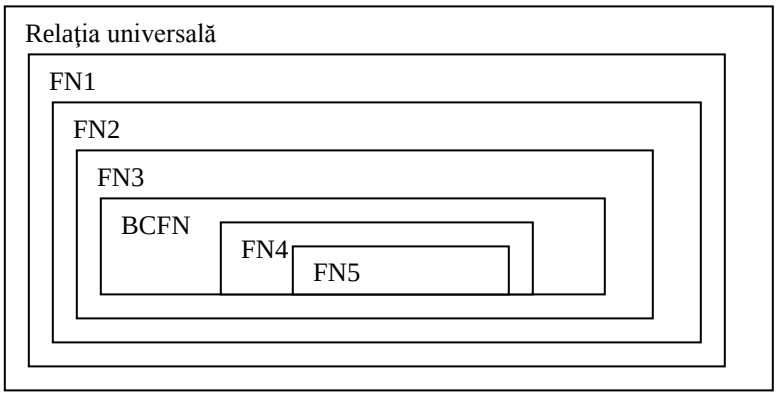


Fig.7.2. Formele normale ale relaţiilor dintr-o BDR

Normalizarea bazei de date relaţionale poate fi imaginată ca un proces prin care pornindu-se de la relaţia iniţială/universală R se realizează descompunerea succesivă a acesteia în subrelaţii, aplicând operatorul de proiecţie. Relaţia R poate fi ulterior reconstruită din cele n relaţii obţinute în urma normalizării, prin operaţii de joncţiune.

7.1 Prima formă normală (FN_1)

FN_1 este strâns legată de noţiunea de *atomicitate* a atributelor unei relaţii. Astfel, aducerea unei relaţii în FN_1 presupune introducerea noţiunilor de:

- atribut simplu;
- atribut compus;
- grupuri repetitive de atribute.

➢ *Atributul simplu- Atribut compus*

Prin *atribut simplu* (*atribut atomic*) se înţelege un atribut care nu mai poate fi descompus în alte atribute, în caz contrar, atributul este *compus* (*atribut neatomic*).

Exemplu: Următoarele exemple de atribute pot fi considerate simple sau compuse în funcţie de circumstanţe şi de obiectivele bazei de date.

- Data calendaristică – este un atribut în care apar câmpurile: zi, lună, an;

- Adresa – este un atribut în care apar câmpurile: strada, nr, bloc, scara, etaj, apartament, localitate, judeţ;
- Data operaţiunii bancare – este un atribut în care apar câmpurile data, ora;
- Buletin/carte identitate – este un atribut în care apar câmpurile: seria, nr.

Aceste atribute pot fi atomice sau neatomice. Astfel adresa clienţilor agenţiei imobiliare interesează la nivel global, pe când pentru adresa ofertei sau a cererii de imobile este vitală prelucrarea separată a fiecărui câmp considerat.

Analog, atributul „nume" reprezintă un atribut simplu al acestei baze de date, deoarece numele clientului interesează la nivel global.

➢ *Grupuri repetitive de atribute*

Un *grup repetitiv* este un atribut (grup de atribute) dintr-o relaţie care apare cu valori multiple pentru o singură apariţie a cheii primare a relaţiei nenormalizate.

Exemplu: Fie relaţia nenormalizată (primară) FACTURI. Dorim să stabilim o structură de tabele care să permită stocarea informaţiilor conţinute în document (factură) şi obţinerea unor situaţii sintetice privind evidenţa sumelor facturate pe produse, pe clienţi, pe anumite perioade de timp.

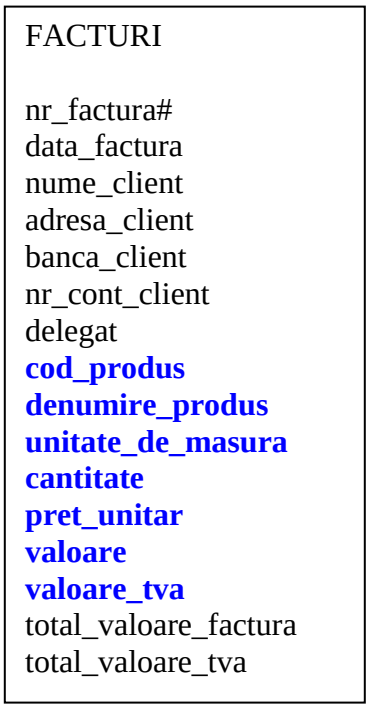


Fig. 7.3. Relaţia FACTURI nenormalizată

În cazul în care o factură conţine mai multe produse, relaţia de mai sus va avea grupurile repetitive: „cod_produs", „denumire_produs", „cantitate", „pret_unitar", „valoare", „valoare_tva".

Aducerea unei relaţii universale la FN_1

FN_1 este tratată în general cu superficialitate, deoarece principala cerinţă – atomicitatea valorilor – este uşor de îndeplinit (cel puţin la prima vedere).

Dintre toate formele normale, doar FN_1 are caracter de obligativitate. Se spune că o bază de date este normalizată daca toate relaţiile se află măcar în FN_1.

***O relaţie este în FN_1* dacă domeniile pe care sunt definite atributele relaţiei sunt constituite numai din valori atomice**. Un tuplu nu trebuie să conţină atribute sau grupuri de atribute repetitive.

Aducerea relaţiilor în FN_1 presupune eliminarea atributelor compuse şi a celor repetitive.

Se cunosc trei soluţii pentru determinarea grupurilor repetitive:

- eliminarea grupurilor repetitive pe orizontală (în relaţia R iniţială, în locul atributelor compuse se trec componentele acestora, ca atribute simple);
- eliminarea grupurilor repetitive prin adăugarea de tupluri;
- **eliminarea grupurilor repetitive prin construirea de noi relaţii**.

Primele două metode generează relaţii stufoase prin duplicarea forţată a unor atribute, respectiv tupluri, creându-se astfel redundanţe masive cu multiple anomalii de actualizare.

Metoda a treia presupune eliminarea grupurilor repetitive prin construirea de noi relaţii, ceea ce generează o structură ce oferă cel mai mic volum de redundanţă.

Etapele de aducere a unei relaţii în FN_1 sunt:

I. se construieşte câte o relaţie pentru fiecare grup repetitiv;
II. în schema fiecărei noi relaţii obţinute la pasul 1 se introduce şi cheia primară a relaţiei R nenormalizate;
III. cheia primară a fiecărei noi relaţii va fi compusă din atributele chei ale relaţiei R, plus unul sau mai multe atribute proprii.

Exemplu: Deoarece o factură poate avea unul sau mai multe produse înscrise pe aceasta, informaţiile legate de produse vor fi separate într-o altă tabelă. Aplicând etapele de aducere la FN1, se obţin două relaţii:

FACTURI
nr_factura#
data_factura
nume_client
adresa_client
banca_client
nr_cont_client
delegat
toal_valoare_factura
toal_valoare_tva

LINII_FACTURI
nr_factura#
cod_produs#
denumire_produs
unitate_de_masura
cantitate
pret_unitar
valoare
valoare_tva

Fig. 7.4. Relaţia FACTURI adusă în forma normală FN1

Observaţia1: Câmpul „adresa_client” cuprinde informaţii despre judeţul, localitatea, strada şi numărul domicililului clientului. Dacă se consideră că este de interes o evidenţă a sumelor factorizate pe judeţe sau localităţi, se vor pune în locul câmpului „adresa_client” trei câmpuri distincte: „judet_client”, „localitate_client”, „adresa_client”, uşurând în acest fel interogările.

Observaţia2: Între tabela FACTURI şi tabela LINII_FACTURI există o relaţie de „unu la mulţi”, adică unui număr unic de factură îi pot corespunde unul sau mai multe produse care sunt memorate ca înregistrări în tabele LINII_FACTURI. Cheia primară în această tabelă este o cheie compusă, formată din două câmpuri: „nr_factura” şi „cod_produs”.

Însă eliminarea grupurilor repetitive, adică aducerea unei relaţii la FN_1, nu rezolvă complet problema normalizării.

7.2. A doua formă normală (FN_2)

FN_2 este strâns legată de noţiunea de *dependenţă funcţională*. Noţiunea de dependenţă funcţională a fost prezentată în cursul 5: „Restricţii de integritate ale modelului relaţional”.

O relaţie se află în a doua formă normală FN2 dacă:

1. se află în forma normală FN1 şi
2. fiecare atribut care nu este cheie este dependent de întreaga cheie primară.

Etapele de aducere a unei relaţii de la FN_1 la FN_2 sunt:

I. Se identifică posibila cheie primară a relaţiei universale aflată în FN1;
II. Se identifică toate dependenţele dintre atributele relaţiei, cu excepţia acelora în care destinaţia este un atribut component al cheii primare;
III. Se identifică toate dependenţele care au ca sursă un atribut sau subansamblu de atribute din cheia primară;
IV. Pentru fiecare atribut (sau subansamblu) al cheii de la pasul III se creează o relaţie care va avea cheia primară atributul (subansamblul) respectiv, iar celelalte atribute vor fi cele care apar ca destinaţie în dependenţele de la etapa III.
V. Din relaţia iniţială sunt eliminate toate atributele destinaţie (noncheie) ale DF găsite la pasul III.

Exemplu: Relaţia care conţine date redundante (de exemplu, modificarea denumirii unui produs atrage după sine modificarea în fiecare tuplu în care apare acest produs) este relaţia LINII_FACTURI. Se observă ca nu există nici o dependenţă funcţională între atributele necomponente ale cheii. În schimb, toate atributele care nu intră în alcătuirea cheii compuse sunt dependente de aceasta, iar DF dintre atributul component al cheii primare sunt: cod_produs --> denumire_produs, cod_produs --> unitate_de_masura. Ca urmare se formează încă două relaţii.

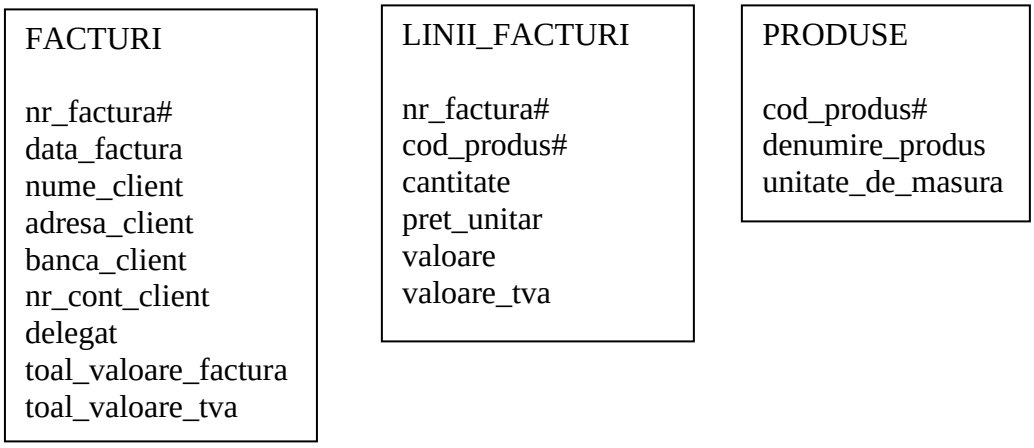


Fig. 7.5. Relaţia FACTURI în a doua forma normală FN2

Chiar dacă au fost eliminate o parte din redundanţe, mai rămân şi alte redundanţe ce se vor elimina aplicând alte forme normale.

CURS 8. A treia formă normală

7.3. A treia formă normală (FN_3)

O relaţie este în *forma normală trei FN3* dacă:
1. se găseşte în FN2 şi
2. fiecare atribut care nu este cheie (nu participă la o cheie) depinde direct de cheia primară.

A treia regulă de normalizare cere ca toate câmpurile din tabele să fie independente între ele.

Etapele de aducere a unei relaţii de la FN_2 la FN_3 sunt:

I. Se identifică toate atributele ce nu fac parte din cheia primara şi sunt surse ale unor dependenţe funcţionale;
II. Pentru aceste atribute, se construieşte câte o relaţie în care cheia primară va fi atributul respectiv, iar celelalte atribute, destinaţiile din DF considerate;
III. Din relaţia de la care s-a pornit se elimină atributele destinaţie din DF identificată la pasul I, păstrându-se atributele surse.

Exemplu: În relaţia FACTURI se observă că atributul „nume_client” determină în mod unic atributele „adresa_client”, „banca_client” şi „nr_cont_client”. Deci pentru atributul „nume_client” se construieşte o relaţie CLIENTI în care cheia primară va fi acest atribut, iar celelalte atribute vor fi „adresa_client”, „banca_client” şi „nr_cont_client”. Câmpurile „valoare” şi „valoare_tva” depind de câmpurile „cantitate”, „pret_unitar”, şi de un procent fix de TVA. Fiind câmpuri ce se pot calcula în orice moment, ele vor fi eliminate din tabelă LINII FACTURI, deoarece constituie informaţie memorată redundant.

FACTURI
nr_factura#
data_factura
nume_client
delegat
toal_valoare_factura
toal_valoare_tva

LINII_FACTURI
nr_factura#
cod_produs#
cantitate
pret_unitar

PRODUSE
cod_produs#
denumire_produs
unitate_de_masura

CLIENTI
nume_client#
adresa_client
banca_client
nr_cont_client

Fig. 8.1. Relaţia FACTURI în a treia forma normală FN3

Observaţia 1: Această a treia formă normală mai poate suferi o serie de rafinări pentru a putea obţine o structură performantă de tabele ale bazei de date. De exemplu se observă că „nume_client” este un câmp în care este înscris un text destul de lung format dintr-o succesiune de litere, semne speciale (punct, virgulă, cratimă), spaţii, numere. Ordonarea şi regăsirea informaţiilor după astfel de câmpuri este lentă şi mai greoaie decât după câmpuri numerice. Din acest motiv se poate introduce un

nou atribut „cod_client” care să fie numeric şi care să fie cheia primară de identificare pentru fiecare client.

Observaţia 2: O altă observaţie care poate fi făcută în legătură cu tabelele aflate în cea de a treia formă normală este aceea că „total_valoare_factura” este un câmp care ar trebui să conţină informaţii sintetice obţinute prin însumarea valorii tuturor ofertelor aflate pe o factură. Este de preferat ca astfel de câmpuri să fie calculate în rapoarte sau interogări şi să nu fie memorate în tabelele bazei de date.

FACTURI	LINII_FACTURI	PRODUSE	CLIENTI
nr_factura# data_factura cod_client delegat	nr_factura# cod_produs# cantitate pret_unitar	cod_produs# denumire_produs unitate_de_masura	cod_client# nume_client adresa_client banca_client nr_cont_client

Verificarea aplicării corecte a procesului de normalizare se realizează astfel încât uniunea acestor relaţii să producă relaţia iniţială, cu alte cuvinte, descompunerea este fără pierderi.

Celelalte forme normale se întâlnesc mai rar în practică. Aceste forme nu sunt respectate, în general, pentru că beneficiile de eficienţă pe care le aduc nu compensează costul şi munca de care este nevoie pentru a le respecta.

CAPITOLUL II. SQL

CURS 9. Limbajul SQL

Noţiuni introductive

Limbajul SQL (**S**tructured **Q**uery **L**anguage) este limbajul utilizat de majoritatea sistemelor de baze de date relaţionale (SGBDR) pentru definirea şi manipularea datelor.

Din punct de vedere istoric ar trebui menţionat faptul că limbajul SQL a fost dezvoltat într-un prototip de sistem de gestiune a bazelor de date relaţionale la IBM, în 1970. În 1979 corporaţia Oracle a introdus prima implementare a limbajului SQL în varianta comercială. În 1987 Institutul Naţional de Standarde (ANSI) a elaborat standardul limbajului SQL. Ulterior au avut loc mai multe revizii ale acestui standard.

Majoritatea limbajelor posedă un set de instrucţiuni comun unanim acceptat de toate marile companii producătoare de soft, precum MICROSOFT sau ORACLE.

Termenii utilizaţi de limbajul SQL sunt

- tabel (Table) utilizat pentru a desemna o relaţie;
- linie (row) utilizat pentru a desemna un tuplu;
- coloană (column) utilizat pentru a desemna un atribut.

Componentele pe care le cuprinde limbajul SQL sunt următoarele:

1. *componenta de descriere* a datelor relaţionale (limbajul de descriere a datelor - LDD),
2. *componenta de manipulare* a datelor relaţionale (limbajul de manipulare a datelor - LMD),ambele fiind absolut necesare în gestiunea BD.

Pe lângă aceste componente principale, standardul SQL2 mai prevede şi alte componente ale limbajului:

3. *controlul tranzacţiilor*;
4. *controlul securităţii şi refacerea datelor*.

Controlul tranzacţiilor conţine comenzi pentru specificarea tranzacţiilor. Unele implementări adaugă comenzilor prevăzute în standard şi alte comenzi suplimentare de *control al concurenţei şi refacerea datelor.*

Controlul securităţii şi refacerea datelor conţine comenzi de administrare a bazei de date pentru definirea utilizatorilor şi a drepturilor acestora de acces la tabele. Această componentă este dependentă de SGBD, iar pentru sisteme performante, administratorul BD este obiectul activităţii unei categorii speciale de utilizatori ai BD – administratori ai BD.

Structura lexicală a limbajului SQL

Elementele unei instrucţiuni (statement) sunt:

- *cuvintele cheie* (key words), dintre care fac parte comenzile (SELECT, UPDATE, INSERT etc), operatorii (AND, OR, NOT, LIKE), clauzele (WHERE, SET, VALUES etc);
- *identificatorii* (identifier) sunt elementele care denumesc tabela, coloana sau alt obiect BD; SQL nu face diferenţa între literele mari şi mici, deci nu este „case-sensitive”; identificatorul care conţine ghilimele se numeşte *identificator delimitat*;
- *constantele* (literal) reprezintă şiruri de caractere (‘ ‘), numere întregi, numere reale (ex. 3.5; 4. ; .001; 5e2), constanta NULL care simbolizează lipsa de informaţie, constante de tip logic (1 pentru TRUE şi 0 pentru FALSE);
- *caracterele speciale*, cum ar fi : care semnifică terminarea comenzilor ; ,care semnifică virgula zecimală, sau * care simbolizează operatorul de înmulţire.

Operatori SQL

SQL are următorii operatori:

- *operatori aritmetici binari:*

  \+
  \-
  *
  % modulo
  ^ ridicarea la putere
  & AND orientat pe biţi
  | OR orientat pe biţi
  # XOR orientat pe biţi
  << deplasare la stânga
  >> deplasare la dreapta

- *operatori binari de comparaţie*

  <
  >
  <=
  >=
  =
  <> sau != diferit

- *operatori aritmetici mari*

@ valoarea absolută
! factorial
!! factorial, operator postfix
~ NOT orientat pe biţi

- *operatori de comparaţie*

A BETWEEN min AND max (A între două valori: min şi max, inclusiv)
A IN (v_1,...,v_n) compară A cu o listă de valori
A IS NULL
A IS NOT NULL
A LIKE model_şir

- *operatori logici*

Operatorii logici sunt legaţi prin cuvintele cheie AND, OR, NOT şi returnează o valoare logică TRUE, FALSE sau NULL.

- *operatori relaţionali*

UNION (reuniune)
INTERSECT (intersecţie)
MINUS (diferenţa).

Funcţii definite în SQL

- *Funcţii agregat*

Funcţiile agregat calculează un rezultat din mai multe linii ale unui tabel (funcţii de totalizare):
COUNT (furnizează numărul de linii ale unui rezultat);
SUM (execută suma tuturor valorilor dintr-o coloană);
MAX (returnează valoarea cea mai mare dintr-o coloană);
MIN (returnează valoarea cea mai mică dintr-o coloană);
AVG (calculează media valorilor dintr-o coloană).
Aceste funcţii vor fi folosite în instrucţiunea SELECT.

- *Funcţii scalare*

Funcţiile scalare primesc unul sau mai multe argumente şi returnează valoarea calculată sau NULL în caz de eroare. Argumentele funcţiilor pot fi constante sau valori ale atributelor specificate prin numele coloanelor corespunzătoare. Dintre funcţiile scalare amintim:

- *funcţii numerice*
  - de calcul trigonometric: sin, cos, tg, ctg etc.
  - de calcul al logaritmului: ln, log, lg
  - de calcul al puterilor: pow
  - de rotunjire: floor, ceil etc.
- *funcţii pentru manipularea şirurilor de caractere*
- *funcţii pentru data calendaristică*
- *funcţii de conversie*

Tipuri de date

În limbajul SQL sunt definite mai multe tipuri de date: numeric, şir de caractere, şir de biţi, data (calendaristică), timp.

Denumirile tipurilor de date, precum şi limitele acestora diferă de la un SGBD la altul, dar în general, sunt destul de asemănătoare.

➢ *Tipul numeric include*

- numere întregi: INTEGER sau INT reprezentat pe 4 octeţi;
  SMALLINT reprezentat pe 2 octeţi;
- numere reale reprezentate în virgulă flotantă, cu diferite precizii:
  FLOAT reprezentat pe 4 octeţi;
  REAL reprezentat pe 8 octeţi;
  DOUBLE [PRECISION] reprezentat pe 8 octeţi;
- numere zecimale reprezentate cu precizia dorită:
  tipul NUMERIC sau DECIMAL, cu forma *numeric[(p,s)]*, unde *p* este numărul total de cifre afişate, iar *s* este numărul de cifre după punctul zecimal.

➢ *Tipul şir de caractere*

CHARACTER (n) sau CHAR (n) definesc şiruri de caractere cu lungimea fixă.
CHARACTER VARYING sau VARCHAR (n) defineşte şirul de caractere cu lungimea variabilă.

Asemănarea dintre cele două tipuri prezentate mai sus este aceea că ambele reprezintă şiruri de maxim *n* caractere, iar deosebirea este aceea că pentru şiruri cu număr de caractere mai mic ca *n*, CHAR (n) completează şirul cu spaţii albe până la *n* caractere, iar VARCHAR (n) memorează numai atâtea caractere câte are şirul dat.

➢ *Tipul şiruri de biţi*

BIT(n) defineşte secvenţe de cifre binare (care pot lua valoarea 0 sau 1) de lungime finită n;
BIT VARYING (n) defineşte secvenţe de lungime variabilă, cu limita maximă n.

`

- *Tipuri pentru data calendaristică şi timp*

DATE permite memorarea datelor calendaristice în formatul yyyy-mm-dd;
TIME permite memorarea timpului, folosind trei câmpuri hh:mm:ss;
TIMESTAMP(p) permite memorarea combinată a datei calendaristice şi a timpului, cu precizia *p* pentru câmpul SECOND (al secundelor); valoarea implicită a lui *p* este 6;
INTERVAL este utilizat pentru memorarea intervalelor de timp.

Tipurile de date sunt „case-insensitive”, deci nu ţin cont de caracterele mari sau mici.

CURS 10. Limbaje relaţionale de definire a datelor (LDD)

Limbajul de definire a datelor (a schemei unei BD) include instrucţiuni ce permit:

- crearea schemei bazei de date;
- adăugarea relaţiilor la schema bazei;
- ştergerea unor relaţii existente;
- adăugarea de noi atribute relaţiilor existente;
- optimizarea bazei de date (index, grup, declanşator);
- definirea structurii fizice şi logice a unei BD;
- restricţii cu privire la utilizarea structurii de mai sus.

➢ *Comenzi pentru crearea unei baze de date*

Comanda pentru crearea unei baze de date este

```
CREATE DATABASE  nume_baza;
```

Exemplu: Să se creeze baza de date AGENTIA_IMOBILIARA.

CREATE DATABASE AGENTIA_IMOBILIARA;

Această comandă creează o BD cu numele nume_baza. Nu toate SGBDR suportă noţiunea explicită de BD, deşi utilizarea unei asemenea noţiuni poate facilita controlul drepturilor de acces la relaţiile BD. Sisteme precum DB2 nu posedă noţiunea explicită de BD, în timp ce sistemul dBASE o suportă.

Creatorul bazei de date devine automat administratorul BD.

➢ *Comenzi pentru suprimarea unei baze de date*

Comanda pentru suprimarea unei baze de date este

```
DROP DATABASE  nume_baza;
```

Această comandă distruge BD cu numele nume_baza.

➢ *Comenzi pentru crearea relaţiilor de bază*

În cadrul acestor comenzi se precizează numele relaţiei precum şi numele şi tipul atributelor.

În SQL, cele mai frecvente tipuri de date sunt:
CHAR pentru şir de caractere de lungime fixă;
VARCHAR2 pentru şir de caractere de lungime variabilă;
NUMBER pentru numere întregi sau reale de lungime variabilă;
DATE pentru date calendaristice;
LONG pentru texte de lungime variabilă
RAW pentru informaţie binară de lungime variabilă.

Comanda de creare a unei relaţii este

```
CREATE TABLE nume_tabela (atribute);
```

- *Crearea unei relaţii indicând cheia la nivel de coloană*

Exemplu: Să se creeze relaţia JUDETE (simbol_judet, nume_judet).

```
CREATE TABLE JUDETE
        (simbol_judet CHAR(2) PRIMARY KEY,
        nume_judet VARCHAR(30));
```

- *Crearea unei relaţii indicând cheile la nivel de tabel*

Exemplu: Să se creeze relaţia LOCALITATI (cod_loc, simbol_judet, nume_loc).

```
CREATE TABLE LOCALITATI
        (cod_loc VARCHAR(7),
        simbol_judet CHAR (2),
        nume_loc VARCHAR (50),
        PRIMARY KEY (cod_loc, simbol_judet),
        FOREIGN KEY (simbol_judet)
        REFERENCES JUDETE(simbol_judet));
```

Dacă cheia primară are mai mult de o coloană atunci cheile trebuie indicate la nivel de tabel.

- *Crearea unui tabel prin copiere*

Exemplu: Să se creeze relaţia LOCALITATI_CLUJ (cod_loc, simbol_judet, nume_loc) utilizând copierea datelor din relaţia LOCALITATI.

```
CREATE TABLE LOCALITATI_CLUJ
SELECT
        cod_loc ,
        simbol_judet ,
        nume_loc
FROM LOCALITATI
WHERE simbol_judet LIKE 'CJ';
```

➢ *Comenzi pentru suprimarea unei relaţii de bază*

Comanda de suprimarea unei relaţii este

```
DROP TABLE nume_tabela;
```

Comanda SQL distruge relaţia nume_tabela.

➢ *Comenzi pentru schimbarea numelui unei relaţii*

Comanda SQL pentru schimbarea numelui unei relaţii este

```
RENAME nume_tabela TO nume_tabela_nou;
```

Exemplu: Să se modifice numele relaţiei LOCALITATI_CLUJ în LOC_CJ, apoi să se suprime relaţia LOC_CJ.

RENAME TABLE LOCALITATI_CLUJ **TO** LOC_CJ;
DROP TABLE LOC_CJ;

- *Comenzi pentru modificarea structurii unei relaţii*

Prin *modificarea structurii unei relaţii* se înţelege:
- extinderea schemei relaţiei prin adăugarea de noi atribute;
- restrângerea schemei unei relaţii prin suprimarea unor atribute;
- modificarea numelui şi/sau tipului unui atribut din cadrul relaţiei.

Unele limbaje relaţionale (QBE) admit toate aceste tipuri de modificări în schema unei relaţii, iar altele (SQL sau QUEL) numai o parte.

Comanda de modificare a unei relaţii este

ALTER TABLE nume_tabel ...

- *Adăugarea unui atribut cu ajutorul opţiunii ADD*

Exemplu: Să se adauge atributul „regiunea" la relaţia LOCALITATI. (Valorile acestui atribut vor fi: Transilvania, Banat, Oltenia, Muntenia, Moldova, Dobrogea.)

ALTER TABLE LOCALITATI **ADD**
(regiunea VARCHAR(10));

- *Modificarea unui atribut cu ajutorul opţiunii MODIFY*

Exemplu: Presupunând că relaţia FACTURI a fost adăugată bazei de date AGENTIA_IMOBILIARA şi că atributul „pret_unitar" este de tip INTEGER, să se modifice forma preţului unitar.

ALTER TABLE FACTURI **MODIFY**
pret_unitar DECIMAL (10,2) ;

- *Comenzi pentru declararea restricţiilor de integritate (a constrângerilor)*

Constrângere este un mecanism care asigură că valorile unei coloane sau a unei mulţimi de coloane satisfac o condiţie declarată. Unei constrângeri i se poate da un nume unic. Dacă nu se specifică un nume explicit atunci sistemul automat îi atribuie un nume de forma SYS_Cn, unde n reprezintă numărul constrângerii. Constrângerile pot fi şterse, pot fi adăugate, pot fi activate sau dezactivate, dar nu pot fi modificate.

Prin comanda CREATE TABLE pot fi specificate anumite restricţii (constrângeri) prin care se exprimă o condiţie care trebuie respectată de toate tuplurile uneia sau mai multor relaţii. Acestea pot fi definite cu ajutorul comenzii

ALTER TABLE

Constrângerile declarative pot fi:
- constrângeri de domeniu, care definesc valorile luate de un atribut:

DEFAULT
NOT NULL
UNIQUE
CHECK

- constrângeri de integritate a entităţii care precizează cheia primară
  PRIMARY KEY
- constrângeri de integritate referenţială care asigură corespondenţa între cheile primare şi cheile externe corespunzătoare
  FOREIGN KEY

Fiecărei restricţii i se poate da un nume, lucru util atunci când, la un moment dat (salvări, restaurări, încărcarea BD) se doreşte dezactivarea uneia sau mai multora dintre acestea. Astfel se prefigurează numele fiecărei restricţii cu tipul său:
pk_(PRIMARY KEY) pentru cheile primare
un_(UNIQUE) pentru cheile alternative
nn_(NOT NULL) pentru atributele obligatorii
ck_(CHECK) pentru reguli de validare la nivel de atribut
fk_(FOREIGN KEY) pentru cheile străine.

Exemplu: Să se realizeze constrângerea de cheie primară, de cheie externă şi constrângerea de domeniu pentru relaţia DESCRIERE_IMOBIL.

```
CREATE TABLE DESCRIERE_IMOBIL
  (id_co SMALLINT (7)  NOT NULL,
     CONSTRAINT FOREIGN KEY fk_co(id_co)
     REFERENCES CERERI_OFERTE(id_co),
     CONSTRAINT pk_co PRIMARY KEY (id_co),
  tip_imobil VARCHAR(10),
  etaj VARCHAR(10),
  nr_camere SMALLINT (6),
  suprafata DECIMAL (12,4),
  garaj TINYINT (4),
  centrala_termica TINYINT (4),
  termopane TINYINT (4));
```

Observaţia 1: Liniile ce nu respectă constrângerea sunt depuse automat într-un tabel special.

Observaţia 2: Constrângerile previn ştergerea unui tabel dacă există dependenţe. (vezi cursul „Ştergerea datelor”)

Observaţia 3: Constrângerile pot fi activate sau dezactivate în funcţie de necesităţi.

Observaţia 4: Constrângerile pot fi create o dată cu tabelul sau după ce acesta a fost creat.

- *Modificarea unei restricţii de integritate*

Comanda de modificare a unei restricţii este

```
ALTER TABLE nume_tabela MODIFY (nume_atribut TIP_CONSTRÂNGERE);
```

Exemplu: Să se modifice una din constrângerile din exemplul de mai sus.

```
ALTER TABLE DESCRIERE_IMOBIL MODIFY
tip_imobil VARCHAR (10) NOT NULL ;
```

➢ *Activarea şi/sau dezactivarea unei constrângeri*

Activarea sau dezactivarea unei constrângeri se realizează cu ajutorul opţiunilor ENABLE sau DISABLE.

Exemplu: Să se dezactiveze, apoi să se activeze constrângerea de cheie primară din relaţia DESCRIERE_OFERTA.

```
ALTER TABLE DESCRIERE_IMOBIL ADD
        (CONSTRAINT id_co PRIMARY KEY (id_co)
        DISABLE);
ALTER TABLE DESCRIERE_IMOBIL
        ENABLE (CONSTRAINT id_co);
```

➢ *Suprimarea unei constrângeri cu ajutorul opţiunii DROP*

Comanda pentru suprimarea unei restricţii este

```
ALTER TABLE nume_tabela DROP PRIMARY KEY;
```

Exemplu: Să se suprime restricţia de cheie primară pentru atributul „nr_factura” din tabela FACTURI.

```
ALTER TABLE FACTURI DROP PRIMARY KEY;
```

➢ *Adăugarea unei constrângeri cu ajutorul opţiunii ADD*

Comanda pentru adăugarea unei restricţii este

```
ALTER TABLE nume_tabela ADD CONSTRAINT ...;
```

Exemplu: Să se adauge restricţia de cheie primară „nr_factura” pentru relaţia FACTURI.

```
ALTER TABLE FACTURI ADD CONSTRAINT
pk_nr_factura PRIMARY KEY (nr_factura);
```

Observaţia 1: Astfel, comanda ALTER TABLE realizează modificarea structurii tabelului (la nivel de coloană sau la nivel de tabel), dar nu modificarea conţinutului acestuia.

Observaţia 2: Constrângerile pot fi adăugate (ADD CONSTRAINT), şterse (DROP CONSTRAINT), activate (ENABLE) sau dezactivate (DISABLE), dar nu pot fi modificate.

Observaţia 3: Dacă există o cheie externă care referă o cheie primară şi dacă se încearcă ştergerea cheii primare, această ştergere nu se poate realiza (tabelele sunt legate prin declaraţia de cheie externă). Ştergerea este totuşi permisă dacă în comanda ALTER apare opţiunea CASCADE, care determină şi ştergerea cheilor externe ce referă cheia primară urmărind sintaxa

```
ALTER TABLE Nume_tabela
DROP PRIMARY KEY CASACDE;
```

Ex

➢ *Comenzi pentru acordarea drepturilor de acces la baza de date*

La nivel logic, limbajele relaţionale oferă comenzi pentru acordarea drepturilor de acces la baza de date. Accesul unor persoane la BD se poate realiza doar în condiţiile recunoaşterii acestora de către sistem drept utilizatori autorizaţi.

Creatorul unei relaţii primeşte în mod automat toate privilegiile de operare asupra acestei relaţii:

- căutări de date în relaţie
- actualizări ale datelor în relaţie
- actualizări ale schemei relaţiei
- ataşarea unor restricţii de integritate
- suprimarea relaţiei.

Poate acorda, la rândul său, privilegii asupra relaţiei şi altor utilizatori în funcţie de sistem:

- sistem centralizat (INGRES) în cadrul căruia singurul care poate acorda drepturi de acces la BD este administratorul bazei de date, funcţia de administrator neputând fi delegată altor persoane;
- sistem descentralizat (DB2, SABRINA, ORACLE) în cadrul căruia administratorul poate da drepturi de acces la BD, dar, în acelaşi timp, putând delega şi alte persoane să fie administratori.

Comanda în SQL de acordare a drepturilor utilizatorilor este

```
GRANT SELECT, UPDATE ,... ON nume_tabela TO
nume_utilizator;
```

Exemplu: Să se confere utilizatorului cu numele Zita şi cu parola BDZ dreptul de conectare la BD, precum şi unele drepturi de acces la una dintre tabelele bazei de date.

```
GRANT SELECT, UPDATE ON DESCRIERE_IMOBIL TO
Zita IDENTIFIED BY 'BDZ';
```

➢ *Comenzi pentru retragerea drepturilor de acces la baza de date*

Comanda SQL pentru retragerea drepturilor de acces la BD este

```
REVOKE UPDATE ON nume_tabela FROM nume_utilizator;
```

Exemplu: Să se retragă drepturile utilizatorului Zita de actualizare a datelor.

REVOKE UPDATE ON DESCRIERE_IMOBIL **FROM** Zita;

CURS 11. Limbaje relaţionale de manipulare a datelor (LMD) - Interogarea datelor

Limbajele de manipulare a datelor trebuie să ofere o serie de facilităţi pentru prelucrarea datelor din relaţiile bazei de date şi anume:

- interogări
- inserări
- modificări
- ştergere.

1. Interogarea datelor

Interogarea bazei de date reprezintă principala funcţie a unui limbaj relaţional de manipulare a datelor.

Comanda fundamentală a standardului SQL este SELECT, aceasta permiţând interogarea unei baze de date.

Interogarea reprezintă o întrebare care îşi extrage răspunsul din baza de date. Componentele interogării se numesc *clause*.

Sintaxa generală a comenzii SELECT este următoarea:

```
SELECT [ALL/DISTINCT/UNIQUE] listă de selecţie
FROM listă de relaţii (tabele)
WHERE condiţie de căutare asupra liniilor
GROUP BY listă de atribute care permit partiţionarea
HAVING condiţie asupra partiţiilor
ORDER BY listă de atribute;
```

Clauzele SELECT şi FROM sunt obligatorii. SELECT specifică datele care se selectează, iar clauza FROM specifică relaţiile din care se selectează. Restul clauzelor sunt opţionale.

Exemplul 1: Să se selecteze toate persoanele împreună cu toate datele personale ale acestora existente în baza de date.

```
SELECT * FROM DATE_PERSOANA;
```

Exemplul 2: Să se selecteze toate ofertele/cererile înregistrate în data de 2006-07-03.

```
SELECT * FROM CERERI_OFERTE
WHERE data_inreg='2006-07-03';
```

➢ *Interogarea datelor folosind operatorii IS şi IS NOT*

Exemplu: Să se selecteze numele tuturor persoanelor care nu au completată adresa de email, apoi să se afişeze numele tuturor persoanelor care au numărul de telefon completat.

```
SELECT numele FROM DATE_PERSOANA
WHERE email IS NULL;
SELECT numele FROM DATE_PERSOANA
```

WHERE nr_telefon **IS NOT** NULL;

- *Interogarea datelor folosind operatorii logici AND, OR, NOT*

Sintaxa pentru interogarea care utilizează un operator logic este

condiţie 1 AND condiţie 2;
condiţie1 OR condiţie 2;
NOT condiţie;

Exemplu: Să se determine numărul facturii şi codul numeric personal pentru ofertele soluţionate după date de 2006-05-01 şi cu un preţ final mai mare sau egal ca 100.000.

```
SELECT cnp,nr_factura FROM FACTURI
WHERE data_factura='2006-08-01' AND total>='100000';
```

- *Interogarea datelor folosind operatorul IN*

Sintaxa este

```
SELECT valoare_câmp IN (valoare1, valoare2,...);
```

Această sintaxă a operatorului IN este similară cu următoarea listă de disjuncţii:

Valoare_câmp=valoare1 OR valoare_câmp=valoare2 OR ...;

Exemplu: Să se selecteze numărul facturii, id-ul cererii/ofertei, data facturii, valoarea totală a facturii, valoarea TVA şi codul numeric personal pentru cererile/ofertele soluţionate cu valoarea totală de 70.000,80.000, 90.000.

```
SELECT * FROM FACTURI
WHERE total IN (70000.00,80000.00,90000.00);
```

- *Interogarea datelor folosind sintaxa DISTINCT*

Pentru a selecta seturi de valori distincte, adică eliminarea valorilor duplicat, în SQL se foloseşte sintaxa DISTINCT, micşorând astfel setul de date. Sintaxa acestei comenzi este

```
SELECT DISTINCT nume_câmp1, nume_câmp2,... FROM
nume_tabela
WHERE comenzi;
sau
SELECT DISTINCT * FROM nume_tabela;
```

Sintaxa DISTINCT se referă la o înregistrare care poate cuprinde unul sau mai multe câmpuri.

Exemplu: Să se afişeze toate datele distincte în care s-au înregistrat cereri sau oferte.

```
SELECT DISTINCT data_inreg FROM CERERI_OFERTE;
```

➢ *Interogarea datelor folosind operatorul LIKE*

Se cunosc mai multe modalităţi de utilizare a expresiei LIKE, şi anume:

- pentru o expresie care începe cu o anumită literă, de exemplu litera 'A': LIKE 'A%';
- pentru o expresie care se termină cu o anumită literă, de exemplu litera 'A': LIKE '%A';
- pentru o expresie care include o anumită literă, de exemplu litera 'A': LIKE '%A%';

Exemplu: Să se selecteze numele, adresa şi emailul tuturor persoanelor feminine care au adresă de email pe yahoo sau personal.

```
SELECT numele, adresa, email FROM DATE_PERSOANA
WHERE adresa LIKE '%BAIA MARE%' AND (email LIKE
'%yahoo%' OR email LIKE '%personal%');
```

➢ *Interogarea datelor folosind operatorul BETWEEN*

Operatorul se utilizează în combinaţie cu două valori între care se află valoarea la care se referă operatorul. Sintaxa este

```
val BETWEEN minim AND maxim;
```

sau

```
val>=min AND val<=max;
```

Cele trei expresii val, min, max pot fi de tip numeric (numeric, decimal, int, smalint etc.) sau de tip dată calendaristică.

Exemplu: Să se selecteze codurile tuturor cererilor/ofertelor înregistrate în perioada 1 ianuarie 2006 şi 1 mai 20006.

```
SELECT id_co FROM CERERI_OFERTE
WHERE data_inreg BETWEEN '2006-01-01' AND '2006-05-
01';
```

➢ *Interogarea datelor folosind funcţiile calendaristice YEAR, DAY, MONTH*

Funcţiile YEAR, DAY, MONTH reţin dintr-un câmp de tip dată calendaristică anul, ziua, respectiv luna.

Exemplu: Să se vizualizeze codurile tuturor cererilor/ofertelor înregistrate în luna mai.

```
SELECT id_co FROM CERERI_OFERTE
WHERE MONTH(data_inreg)=05;
```

➢ *Interogarea datelor folosind ordonarea*

Datele se pot ordona după orice câmp. Ordonarea se poate face atât crescător cât şi descrescător. Sintaxa pentru interogarea

- ordonată crescător este

```
ORDER BY nume_câmp (ASC);
```

- ordonată descrescător este

```
ORDER BY nume_câmp (DESC);
```

Dacă ORDER BY nu este urmat de ASC sau DESC, ordonarea se face implicit crescător.

Exemplu: Să se vizualizeze lista persoanelor în ordine alfabetică.

```
SELECT numele FROM DATE_PERSOANA
ORDER BY numele;
```

➢ *Interogarea datelor din mai multe tabele*

Interogarea datelor din mai multe relaţii este strâns legată de noţiunea de cheie primară, cheie secundară, restricţii de integritate, asocieri între relaţii.

Exemplu: Să se afişeze ofertele şi denumirile oraşelor corespunzătoare ofertelor.

```
SELECT id_co, nume_loc
FROM CERERI_OFERTE, LOCALITATI
WHERE CERERI_OFERTE.tipul='oferta' AND
      CERERI_OFERTE.cod_loc=LOCALITATI.cod_loc;
```

Observaţii: Clauza FROM specifică două relaţii. Clauza SELECT cuprinde valori din relaţia CERERI_OFERTE şi din relaţia LOCALITATI, prin urmare trebuie definite câmpurile în funcţie de tabela din care face parte. Se utilizează sintaxa

```
nume_tabel.nume_câmp
```

Clauza WHERE include condiţii care exprimă o egalitate între valorile identificatorului nume_câmp a relaţiei nume_tabel şi a celei ale referinţei la acest identificator în tabela referită.

CURS 12. Limbaje relaţionale de manipulare a datelor (LMD) - Interogarea datelor din mai multe relaţii

Atunci când în clauza FROM a unei comenzi SELECT apar mai multe tabele se realizează *produsul cartezian* al acestora. De aceea numărul de linii rezultat creşte considerabil, fiind necesară restricţionarea acestora cu o clauza WHERE.

Atunci când este necesară obţinerea de informaţii din mai multe tabele se utilizează condiţii de *join*. În acest fel liniile dintr-un tabel pot fi puse în legătura cu cele din alt tabel conform valorilor comune ale unor coloane. Condiţiile de corelare utilizează de obicei coloanele cheie primară şi cheie externă.

Tipuri de asocieri pentru relaţii

Rolul unei relaţii fiind acela de a modela entităţi, între relaţii există aceleaşi tipuri de asocieri ca şi între entităţi, prezentate la începutul cursului, şi anume asocieri unu la unu, unu la mai mulţi, mulţi la mai mulţi.

- *Asocieri de la unu la unu*

Două relaţii stochează informaţii în *asocierea unu la unu* dacă unei înregistrări din relaţia A îi corespunde (sau nu) o singură înregistrare din B.

Acest tip de asociere este utilizată mai rar. Există, totuşi, cazuri în care este necesară şi utilă stabilirea unei astfel de relaţii.

Exemplu:

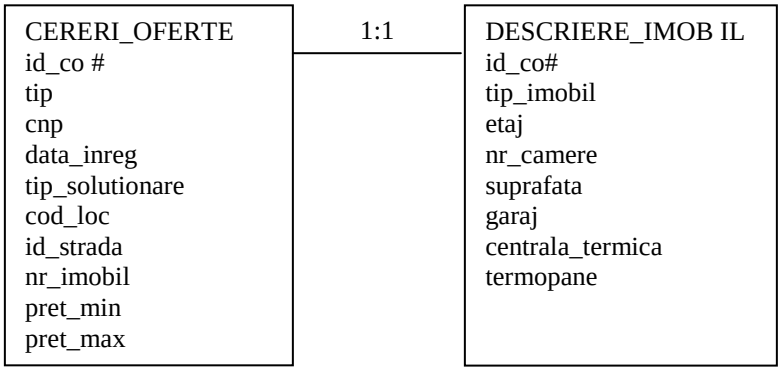


Fig. 12.1. Asociere de tip 1:1

- *Asocieri de la unu la mai mulţi*

O relaţie A se află într-o asociere de *unu la mai mulţi* cu o relaţie B dacă fiecărei înregistrări din A îi corespund una sau mai multe înregistrări din relaţia B. Unei înregistrări din relaţia B nu îi corespunde decât maxim o înregistrare din relaţia A.

Sunt utilizate următoarele denumiri:

- B este *relaţia copil* sau *relaţia care referă* la A sau *relaţia cheie străină*;
- A este *relaţia părinte* (master) sau *relaţia referită* sau *relaţia cheie primară*.

Exemplu:

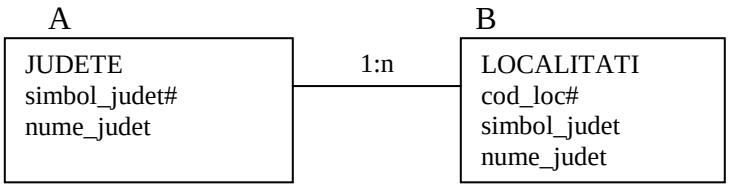


Fig. 12.2. Asociere de tip 1:n

Observaţie: Relaţia A are cheia primară „simbol_judet”, iar relaţia B are atributul „simbol_judet” cheie externă.

- *Asocieri de la mai mulţi la mai mulţi*

O relaţie A se află în asociere de tipul *mulţi la mai mulţi* cu o relaţie B dacă unei înregistrări din relaţia A îi pot corespunde mai multe înregistrări din relaţia B şi unei înregistrări din relaţia B îi pot corespunde mai multe înregistrări din relaţia A.

O asociere *n* la *n* nu se defineşte direct, asocierea construindu-se cu ajutorul unei relaţii de *joncţiune*. În această relaţie se păstrează legătura între cele două relaţii, precum şi informaţiile necesare.

Exemplu:

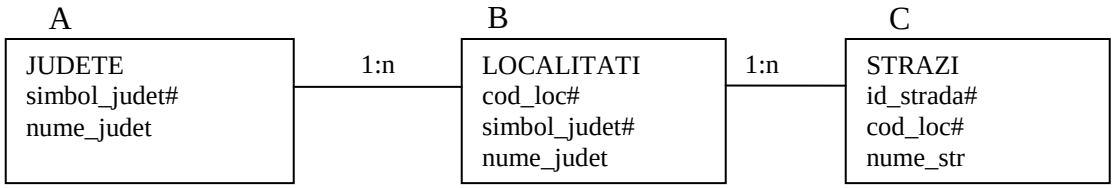


Fig. 12.3. Asociere de tip n:n

Observaţie: În exemplul de mai sus, relaţia LOCALITATI realizează joncţiunea între relaţiile JUDETE şi STRAZI, stocând informaţiile privind numele judeţului „nume_judet”, simbolul judeţului „simbol_judet” şi identificatorul localităţii „cod_loc”.

Astfel, asocierea n la n este vizualizată sub forma a două relaţii de unu la n.

- *Interogarea datelor din mai multe relaţii folosind aliasuri*

Un *alias* este o redenumire fie a unui câmp, fie a unei relaţii. Aliasurile sunt utilizate la eliminarea rescrierii complete a denumirii unei relaţii sau a unui câmp, redenumindu-le într-un mod simplificat. Sintaxa utilizată este:

```
nume_relaţie/camp AS nume_nou;
```

sau

```
nume_relaţie/camp nume_nou;
```

Există posibilitatea de a utiliza aliasuri pentru tabelele din clauza FROM şi utilizarea lor în cadrul comenzii SELECT respective (alias.coloana). Această

identificare (prin 'tabel.coloana' sau 'alias.coloana') este obligatorie atunci când se face referinţă la o coloana ce apare în mai mult de un tabel din clauza FROM.

Exemplul 1: Să se determine toate ofertele de apartamente din oraşul Baia Mare, de pe strada Victoriei.

```
SELECT CO.id_co, L.nume_loc, S.nume_str, CO.nr_imobil,
DI.etaj,  DI.nr_camere, DI.garaj, DI.suprafata,
DI.centrala_termica, DI.termopane, DI.tip_imobil, CO.pret_min,
CO.pret_max
FROM CERERI_OFERTE AS CO, LOCALITATI AS L, STRAZI
AS S, DESCRIERE_IMOBIL AS DI
WHERE CO.tipul='oferta' AND CO.id_co=DI.id_co AND
CO.cod_loc=L.cod_loc AND CO.id_strada=S.id_strada AND
CO.cod_loc=S.cod_loc AND L.cod_loc='MM430' AND
S.id_strada='152' AND DI.tip_imobil='apartament';
```

Lista afişată în urma acestei interogări poate fi de genul:

id_co	nume_loc	nume_str	nr_imobil	etaj	nr_camere	garaj	suprafata	centrala_termica	termopane	tip_imobil	pret_min	pret_max
3	BAIA MARE	VICTORIEI	54	2	3	0	1200.0000	1	0	apartament	242000.00	NULL
5	BAIA MARE	VICTORIEI	3	1	4	1	NULL	1	1	apartament	326700.00	NULL

Fig. 12.4. Lista apartamentelor din Baia Mare, de pe strada Victoriei

Exemplul 2: Să se afişeze toate cererile nesoluţionate de terenuri din localitatea Borşa, data înregistrării, precum şi datele personale ale clienţilor.

```
SELECT CO.id_co, CO.tipul, CO.cnp, CO.data_inreg,
        CO.tip_solutionare, CO.cod_loc, DP.numele,
        DP.adresa, DP.nr_telefon, DP.email,  DI.tip_imobil
FROM CERERI_OFERTE CO, DATE_PERSOANA DP,
      DESCRIERE_IMOBIL DI
WHERE CO.tipul='cerere' AND CO.cnp=DP.cnp
        AND cod_loc='MM435'
        AND CO.tip_solutionare='0'
        AND CO.id_co = DI.id_co
        AND DI.tip_imobil LIKE 'teren';
```

Observaţie: În cazul în care un atribut apare doar într-o relaţie dintre cele menţionate în listă, nu este obligatorie precizarea relaţiei (adică a aliasului) din care face parte atributul respectiv, după cum este „cod_loc='MM435'".

- *Interogarea datelor din mai multe relaţii folosind tipuri de asocieri*

Tipurile de asocieri utilizate în interogarea mai multor relaţii sunt:

- INNER JOIN (joncţiunea internă)
- LEFT OUTER JOIN (semijoncţiunea la stânga)
- RIGHT OUTER JOIN (semijoncţiunea la dreapta)

a) Sintaxa

```
SELECT ...FROM tabel_A INNER JOIN tabel_B (condiţii de join)
```

selectează toate informaţiile din relaţiile A şi B care corespund condiţiilor de asociere.

Exemplul 1: Selectaţi codul ofertei/cererilor şi codul localităţilor fiecărei oferte folosind operaţia de join, apoi utilizând clauza WHERE.

```
SELECT CO.id_co, CO.cod_loc
FROM CERERI_OFERTE CO INNER JOIN LOCALITATI L
        ON (CO.cod_loc=L.cod_loc);

SELECT CO.id_co, CO.cod_loc
FROM CERERI_OFERTE CO, LOCALITATI L
WHERE CO. cod_loc=L.cod_loc;
```

Observaţie: Rezultatul este acelaşi. Valorile NULL vor fi ignorate.

Exemplul 2: Selectaţi numele persoanelor care oferă imobile, codul ofertelor, precum şi denumirile localităţilor, ordonând alfabetic localităţile.

```
SELECT DP.numele, CO.id_co, L.nume_loc
FROM DATE_PERSOANA DP
INNER JOIN CERERI_OFERTE CO ON (DP.cnp=CO.cnp)
INNER JOIN LOCALITATI L ON (CO.cod_loc=L.cod_loc)
WHERE CO.tipul LIKE 'oferta'
ORDER BY L.nume_loc;

SELECT DP.numele, CO.id_co, L.nume_loc
FROM DATE_PERSOANA DP, CERERi_OFERTE CO,
LOCALITATI L
WHERE CO.tipul LIKE 'oferta'
AND DP.cnp=CO.cnp
AND CO.cod_loc=L.cod_loc
ORDER BY L.nume_loc;
```

Observaţie: Sintaxei SELECT-FROM-INNER JOIN i se pot adăuga şi alte condiţii, neincluse în condiţiile de join, dacă acestea se referă la alte câmpuri decât cele care participă la join.

Exemplul 3: Selectaţi numele persoanelor care oferă imobile în judeţul Maramureş, codul ofertelor, tipul acestora, precum şi denumirile localităţilor şi a străzilor, ordonând alfabetic localităţile şi străzile.

1) Folosind INNER JOIN

```
SELECT DP.numele, CO.id_co,  S.nume_str, DI.tip_imobil,
        L.nume_loc
FROM DATE_PERSOANA DP INNER JOIN
        CERERI_OFERTE CO ON  (DP.cnp=CO.cnp)
INNER JOIN STRAZI S ON (CO.id_strada=S.id_strada )
INNER JOIN LOCALITATI L ON (CO.cod_loc=L.cod_loc
        AND  L.cod_loc LIKE 'MM%')
INNER JOIN DESCRIERE_IMOBIL DI ON
        (CO.id_co=DI.id_co AND CO.tipul='oferta')
ORDER BY L.nume_loc, S.nume_str;
```

Observaţie: Toate condiţiile ce se referă la câmpurile din join se vor prezenta în cadrul condiţiilor de join.

2) Folosind WHERE

```
SELECT DP.numele, CO.id_co,  S.nume_str, DI.tip_imobil,
        L.nume_loc
FROM DATE_PERSOANA DP, CERERI_OFERTE CO,
      STRAZI S,  LOCALITATI L, DESCRIERE_IMOBIL DI
WHERE CO.tipul='oferta' AND
        DP.cnp=CO.cnp AND
        CO.id_strada=S.id_strada AND
        CO.cod_loc=L.cod_loc AND
        CO.id_co=DI.id_co AND
        L.cod_loc LIKE 'MM%'
ORDER BY L.nume_loc, S.nume_str;
```

3) Folosind INNER JOIN şi WHERE

```
SELECT DP.numele, CO.id_co,  S.nume_str, DI.tip_imobil,
        L.nume_loc
FROM DATE_PERSOANA DP INNER JOIN
      CERERI_OFERTE CO ON   (DP.cnp=CO.cnp)
INNER JOIN STRAZI S ON (CO.id_strada=S.id_strada )
INNER JOIN LOCALITATI L ON (CO.cod_loc=L.cod_loc )
INNER JOIN DESCRIERE_IMOBIL DI ON
       (CO.id_co=DI.id_oferta)
WHERE L.cod_loc LIKE 'MM%' AND CO.tipul='oferta'
ORDER BY L.nume_loc, S.nume_str;
```

b) Sintaxa

SELECT ...FROM tabel_A *LEFT OUTER JOIN* tabel_B ON (condiţii de join)

selectează toate informaţiile din A, pe care le completează cu informaţii din B, în măsura în care satisfac condiţiile de join; acolo unde nu vor exista informaţii din B, acestea vor fi completate cu NULL.

Exemplul1: Selectaţi toate ofertele. Dacă există informaţii despre aceste oferte, afişaţi şi aceste informaţii.

```
SELECT *
FROM CERERI_OFERTE CO LEFT OUTER JOIN
   DESCRIERE_IMOBIL DI ON (CO.id_co=DI.id_co )
 WHERE CO.tipul='oferta';
```

Observaţie: Ordinea în care se scrie denumirea relaţiei în sintaxa LEFT OUTER JOIN este foarte importantă. Astfel, relaţia din stânga este *relaţia primară*, adică relaţia pentru care se doreşte returnarea tuturor informaţiilor; relaţia din dreapta este *relaţia secundară*, adică informaţiile din ea sunt necesare doar în măsura în care se potrivesc condiţiilor de asociere. Astfel se explică şi denumirea de *asociere de la stânga spre exterior*.

Exemplul2: Selectaţi toate ofertele, precizând şi numele judeţelor, localităţilor precum şi a străzilor. Dacă există informaţii despre aceste oferte, afişaţi şi aceste informaţii.

```
SELECT L.nume_loc, CO.*, S.nume_str, DI.tip_imobil,
```

```
        DI.nr_camere, DI.suprafata, DI.garaj,
        DI.centrala_termica, DI.termopane
FROM CERERI_OFERTE CO LEFT OUTER JOIN
    DESCRIERE_IMOBIL DI ON(DI.id_co=CO.id_co)
INNER JOIN STRAZI S ON S.id_strada = CO.id_strada AND
                        CO.cod_loc=S.cod_loc
INNER JOIN LOCALITATI L ON CO.cod_loc = L.cod_loc
WHERE CO.tipul LIKE 'oferta';
```

c) Sintaxa

SELECT ...FROM tabel_A *RIGHT OUTER JOIN* tabel_B ON (condiţii de join)

selectează toate informaţiile din B, pe care le completează cu informaţii din A, în măsura în care satisfac condiţiile de join; acolo unde nu vor exista informaţii din A, acestea vor fi completate cu NULL.

Exemplu: Selectaţi toate localităţile şi, în localităţile în care există cereri nesoluţionate, afişaţi numele clienţilor şi tipul de cerere de imobil respectiv.

```
SELECT L.nume_loc, DP.numele, CO.tip_solutionare,
        DI.tip_imobil
FROM  LOCALITATI L RIGHT OUTER JOIN CERERI_OFERTE
CO ON (L.cod_loc=CO.cod_loc)
INNER JOIN DATE_PERSOANA DP ON (DP.cnp=CO.cnp)
INNER JOIN DESCRIERE_IMOBIL DI ON
        (CO.id_co=DI.id_co AND CO.tipul = 'cerere')
WHERE CO.tip_solutionare=0;
```

Observaţie: Sintaxa RIGHT OUTER JOIN este utilizată mai rar; de obicei se utilizează sintaxa LEFT OUTER JOIN.

CURS 13. Limbaje relaţionale de manipulare a datelor (LMD) - Interogarea datelor din mai multe relaţii (continuare)

➢ *Interogarea datelor din mai multe relaţii folosind instrucţiunea UNION*

Sintaxa interogării datelor din mai multe relaţii folosind instrucţiunea UNION este

```
SELECT Câmp 1, Câmp 2, ..., Câmp n
FROM Tabel 1
UNION (ALL)
SELECT Câmp 1A, Câmp 2A,..., Câmp nA
FROM Tabel 2
```

şi returnează înregistrări distincte, dacă este folosită instrucţiunea UNION şi toate înregistrările, dacă se foloseşte UNION ALL. Astfel operatorul UNION elimină duplicatele, iar UNION ALL vizualizează toate înregistrările, inclusiv duplicatele.

Pentru a utiliza această interogare, trebuie să se ţină seama de două cerinţe: domeniile Câmp 1A, Câmp 2A,..., Câmp nA şi Câmap 1, Câmp 2, ..., Câmp n trebuie să fie respectiv aceleaşi şi, numărul de câmpuri din fiecare interogare trebuie să coincidă.

Operatorul UNION se foloseşte atunci când între relaţii nu există o asociere directă.

Exemplul 1: Pentru exemplificare se vor considera relaţiile: PROFESORI (prof_id, nume, prenume), respectiv STUDENTI (stud_id, nume, prenume). Selectaţi lista numelor tuturor profesorilor şi a studenţilor.

```
SELECT  nume, prenume FROM PROFESORI
UNION ALL
SELECT nume, prenume FROM STUDENTI;
```

Rezultatul generat de interogare va fi

nume	**prenume**
POP	VASILE
ION	ANA

Fig. 13.1. Interogarea mai multor relaţii folosind operatorul UNION ALL

Observaţi: Problema mai poate fi soluţionată utilizând alte interogări, dar acestea rămân ca exerciţii individuale.

Exemplul 2: Să se determine care sunt ofertele şi cererile soluţionate prin facturi, afişând într-o listă id_ul cererii/ofertei şi cnp-ul cleintului, atât din tabela CERERI_OFERTE cât şi din tabela FACTURI.

```
SELECT cnp, id_co,data_factura FROM FACTURI
UNION ALL
SELECT cnp, id_co,data_inreg FROM CERERI_OFERTE;
```

Rezultatul generat de interogare va fi o lista greu de urmărit, după cum este şi cea din figura 13.1, deoarece nu se specifică clar care inregistrare corespunde facturilor, şi care tabelei CERERI_OFERTE (acest neajuns va fi înlăturat utilizând concatenarea):

cnp	id_co	data_
2660805270023	1	2006-08-01
2660805270023	2	2006-08-01
2820420223201	5	2006-07-31
1670321778721	3	2006-07-31
2660805270023	1	2006-05-27
1701205230023	2	2006-07-03
2701228450021	3	2006-07-16
1701205230023	4	2006-08-20
1540923832123	5	2006-07-18
2660920219212	6	2006-07-27
2660920219212	7	2006-09-12
1820320223201	8	2006-01-01
2820420223201	9	2006-05-10
1670321778721	10	2006-07-10
1670321778721	11	2006-07-10
2820420223201	12	2006-08-01
1540923832123	13	2006-08-06
1820320223201	14	2006-08-06
1670321778721	15	2006-07-10
2820420223201	16	2006-08-26
2820420223201	17	2006-08-26

Fig. 13.2. Interogarea mai multor relaţii folosind operatorul UNION ALL (cazul neclar)

- *Interogarea datelor mai multor relaţii folosind operatorul de concatenare a două şiruri de caractere*

Rolul operatorului de concatenare a două şiruri de caractere este de a uni două şiruri de caractere într-unul singur. Este utilizat în toate SGBD-urile, cu mici modificări ale simbolului: în Tranzact SQL se foloseşte simbolul ‚+', în Oracle simbolul ‚||' etc.

Se pot concatena o constantă cu un câmp, sau două câmpuri. Câmpurile trebuie să fie de tip text.

Sintaxa pentru concatenarea a două câmpuri este

```
CONCAT(Câmp1, Câmp2)
```

sau inserând virgula, spaţiu sau oricare marcaj de delimitare

```
CONCAT(Câmp1,',', Câmp2) sau CONCAT (Câmp1,' ', Câmp2).
```

Sintaxa

```
CONCAT('Ceva', Câmp)
```

concatenează câmpul şi valoarea returnând o singură valoare.

Sintaxa

```
CONCAT('Ceva1', 'Ceva1')
```

concatenează cele două constante într-una singură 'Ceva1Ceva2'.

Exemplu: Să se determine care sunt ofertele şi cererile soluţionate prin facturi, afişând într-o listă id_ul cererii/ofertei şi cnp-ul clientului, atât din tabela

CERERI_OFERTE cât şi din tabela FACTURI. De această dată, să se precizeze când este vorba de facturi, respectiv când este vorba de cerere sau ofertă.

```
SELECT CONCAT('F:', ' ', 'cnp;',cnp,'  ', 'id_co:',id_co,'  ','data:',
data_factura)  AS facturi_si_oferte_cereri
FROM FACTURI
UNION
SELECT CONCAT('C_O:',' ', 'cnp;',cnp,'  ', 'id_co:',id_co,'
','data:', data_inreg) FROM CERERI_OFERTE;
```

Rezultatul generat de interogare va fi

feacturi_si_oferte_cereri
F: cnp;2660805270023 id_co:1 data:2006-08-01
F: cnp;2660805270023 id_co:2 data:2006-08-01
F: cnp;2820420223201 id_co:5 data:2006-07-31
F: cnp;1670321778721 id_co:3 data:2006-07-31
C_O: cnp;2660805270023 id_co:1 data:2006-05-27
C_O: cnp;1701205230023 id_co:2 data:2006-07-03
C_O: cnp;2701228450021 id_co:3 data:2006-07-16
C_O: cnp;1701205230023 id_co:4 data:2006-08-20
C_O: cnp;1540923832123 id_co:5 data:2006-07-18
C_O: cnp;2660920219212 id_co:6 data:2006-07-27
C_O: cnp;2660920219212 id_co:7 data:2006-09-12
C_O: cnp;1820320223201 id_co:8 data:2006-01-01
C_O: cnp;2820420223201 id_co:9 data:2006-05-10
C_O: cnp;1670321778721 id_co:10 data:2006-07-10
C_O: cnp;1670321778721 id_co:11 data:2006-07-10
C_O: cnp;2820420223201 id_co:12 data:2006-08-01
C_O: cnp;1540923832123 id_co:13 data:2006-08-06
C_O: cnp;1820320223201 id_co:14 data:2006-08-06
C_O: cnp;1670321778721 id_co:15 data:2006-07-10
C_O: cnp;2820420223201 id_co:16 data:2006-08-26
C_O: cnp;2820420223201 id_co:17 data:2006-08-26

Fig. 13.3. Interogarea mai multor relaţii folosind concatenarea (cazul mai clar)

Observaţie: Concatenarea prezintă dezavantajul afişării câmpurilor null.

- *Interogarea datelor folosind funcţiile totalizatoare*
  - MAX
  - MIN
  - COUNT
  - SUM
  - AVG

a) Interogarea datelor folosind funcţia MAX

Sintaxa:

```
SELECT MAX(Nume_câmp) FROM Tabela
```

returnează un număr egal cu valoarea maximă a câmpului Nume_câmp din relaţia Tabela, valorile null fiind ignorate.

Exemplu: Selectaţi cea mai recentă înregistrare din tabela CERERI_OFERTE, fără a da un nume rezultatului, apoi cu nume pentru câmpul rezultat.

```
SELECT MAX(data_inreg) FROM CERERI_OFERTE;
SELECT MAX(data_inreg)  AS data_ultimei_înregistrari FROM
CERERI_OFERTE;
```

b) Interogarea datelor folosind funcţia MIN

Funcţia MIN este o funcţie similară cu funcţia MAX, cu ajutorul căreia se poate determina valoarea cea mai mică dintr-un câmp.

Atât funcţia MIN cât şi funcţia MAX se poate aplica doar pentru tipurile de date numeric sau dată calendaristică.

c) Interogarea datelor folosind funcţia COUNT

Sintaxa

```
SELECT COUNT (*) FROM Nume_tabela
```

returnează un număr egal cu numărul de înregistrări ale tabelei Nume_tabela.

Exemplu: Precizaţi numărul de oferte înregistrare.

```
SELECT COUNT(*) AS numar_de_oferte
FROM CERERI_OFERTE
WHERE tipul LIKE 'oferta';
```

Sintaxa

```
SELECT COUNT (Nume_câmp) FROM Tabela
```

returnează un număr egal cu numărul de valori nenule ale câmpului Nume_câmp din tabela Nume_tabela. Sunt ignorate valorile null.

Exemplu: Precizaţi numărul de cereri nesoluţionate.

```
SELECT COUNT(tip_solutionare) AS cereri_solutionate
FROM CERERI_OFERTE
WHERE tip_solutionare=1 AND
        tipul='cerere';
```

Sintaxa

```
SELECT COUNT(DISTINCT Nume_câmp) FROM Tabela
```

returnează un număr egal cu numărul de valori distincte nenule ale câmpului Nume_câmp din tabela Nume_tabela. Sunt ignorate valorile null.

Exemplu: Precizaţi numărul de localităţi din care provin ofertele.

```
SELECT COUNT(DISTINCT cod_loc) FROM CERERI_OFERTE
WHERE tipul='oferta';
```

d) Interogarea datelor folosind funcţia SUM

Sintaxa

```
SELECT SUM (Nume_câmp) FROM Tabela
```

returnează un număr egal cu suma tuturor valorilor câmpului Nume_câmp din relaţia Nume_Tabela. Sunt ignorate valorile null.

Exemplu: Precizaţi suma tuturor încasărilor existente pe facturile emise.

```
SELECT SUM(DISTINCT total) FROM FACTURI;
```

Sintaxa

```
SUM (DISTINCT Nume_câmp) FROM Tabela
```

returnează un număr egal cu suma valorilor distincte ale câmpului Nume_câmp din relaţia Nume_Tabela.

Funcţia SUM se aplică acelor câmpuri care au domeniul de valori de tipul FLOAT, DECIMAL, NUMERIC, INT etc. şi nu are sene pentru câmpuri de tip text.

e) Interogarea datelor folosind funcţia AVG

Sintaxa

```
AVG (nume_câmp) FROM Nume_tabela
```

returnează un număr egal cu media aritmetică a tuturor valorilor câmpului Nume_câmp din relaţia Nume_tabela. Valorile null sunt ignorate.

Funcţia AVG se utilizează doar pentru date de tip numeric: INT, FLOAT, NUMERIC.

Exemplu: Selectaţi media valorilor vânzărilor din agenţia imobiliară.

```
SELECT AVG (total) FROM FACTURI;
```

- *Interogarea datelor folosind instrucţiunea GROUP BY*

Prin instrucţiunea GROUP BY se grupează datele după fiecare produs în parte.

Exemplu: Selectaţi fiecare tip de imobil în parte grupându-le alfabetic şi precizaţi numărul de imobile vândute din fiecare tip.

```
SELECT DI.tip_imobil, COUNT(F.id_co) AS suma
FROM DESCRIERE_IMOBIL DI, FACTURI F
WHERE F.id_co=DI.id_co
GROUP BY DI.tip_imobil;
```

Interogarea returnează următoarele informaţii:

descriere	suma
APARTAMENT	2
CASA	1
HALA	1

Fig. 13.4 Rezultatul interogării folosind instrucţiunea GROUP BY şi funcţia SUM

Menţionarea clauzelor SELECT, FROM, WHERE, GROUP BY, ORDER BY în această ordine este obligatorie. Greşeala frecventă care duce la apariţia unor mesaje

de eroare este aceea a introducerii unor câmpuri după care se grupează în clauza SELECT şi neintroducerea lor în clauza GROUP BY.

- *Interogarea datelor folosind instrucţiunea HAVING*

Instrucţiunea HAVING se utilizează numai în combinaţie cu instrucţiunea GROUP BY. Dacă gruparea de date trebuie să satisfacă vreo condiţie, această condiţie se exprimă cu ajutorul sintaxei HAVING.

Clauza HAVING este utilizată când se doreşte filtrarea datelor grupate conform unor criterii. Aceste criterii presupun compararea unor valori obţinute prin apelarea unor funcţii totalizatoare. Aceste tipuri de comparări presupun gruparea datelor. Din această cauză, HAVING cere obligatoriu clauza GROUP BY.

Exemplu: Selectaţi adresele ofertelor grupate după judeţe, localităţi şi străzi care au preţul minim cuprins între 50000 şi 300000.

```
SELECT CO.id_co,J.nume_judet, L.nume_loc, S.nume_str,
CO.pret_min, CO.pret_max
FROM JUDETE J, CERERI_OFERTE CO, LOCALITATI L,
STRAZI S
WHERE CO.cod_loc=L.cod_loc
AND CO.id_strada=S.id_strada
AND L.simbol_judet=J.simbol_judet
GROUP BY CO.id_co,....
HAVING CO.pret_min BETWEEN 50000 AND 300000;
```

Ordinea obligatorie a unei fraze SELECT complete este: SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY.

Curs 14. Funcţii utilizate în interogări

Cele mai des întâlnite funcţii în interogări sunt:

a) funcţii pentru şiruri de caractere
b) funcţii pentru valori numerice
c) funcţii pentru date calendaristice
d) funcţii de conversie dintr-un tip în altul.

a) Funcţii pentru şiruri de caractere

- CONCAT: concatenează două şiruri de caractere

```
SELECT CONCAT('My', 'S', 'QL');
      -> 'MySQL'
SELECT CONCAT('My', NULL, 'QL');
      -> NULL
SELECT CONCAT(14.3);
      -> '14.3'
```

- REPLACE: înlocuirea unui şir de caractere cu un altul într-o expresie de acest tip;

```
SELECT REPLACE('www.mysql.com', 'w', 'Ww');
     -> WwWwWw.mysql.com
```

- CHAR_LENGTH: returnează numărul de caractere dintr-un şir;
- FIELD(str,str1,str2,str3,…): returnează poziţia şirului de caractere „str” în lista şirurilor de caractere „str1,str2,str3”; dacă şirul „str” nu este găsit, returnează valoarea 0.

```
SELECT FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo');
      -> 2
SELECT FIELD('fo', 'Hej', 'ej', 'Heja', 'hej', 'foo');
      -> 0
```

- LEFT(str,n): returnează subşirul fromat dintr-un număr de „n” caractere;

```
SELECT LEFT('paralelipiped', 5);
      -> 'paral'
```

- LOWER(str): toate literele din „str” vor fi convertite în minuscule;

```
SELECT LOWER('LITERE MARI');
     -> 'litere mari'
```

- UPPER(str): toate literele din „str” vor fi convertite în majuscule;

```
SELECT UPPER('LITERE mari');
      -> 'LITERE MARI'
```

- LPAD(str,n,caracter): completează la stânga cu un caracter până la atingerea unei lungimi specificate

```
SELECT LPAD('buna',6,'?');
      ->??buna
SELECT LPAD('buna',1,'??');
      ->b
```

- RPAD(str,n,caracter): completează la dreapta cu un caracter până la atingerea unei lungimi specificate

```
SELECT RPAD('buna',5,'?');
```

->buna?

- LTRIM(str): elimină spaţiile de la stânga valorii „str”;

```
SELECT LTRIM('  barbar');
        ->barbar
```

- RTRIM(str): elimină spaţiile de la dreapta valorii „str”;

```
SELECT RTRIM('barbar      ');
        ->barbar
```

- TRIM: eliminarea simultană a spaţiillor la stânga şi la dreapta;

```
SELECT TRIM('  bar   ');
        ->bar
```

- SUBSTR(sir,n): extragerea unei porţiuni dintr-un şir începând cu a n-a literă;

```
SELECT SUBSTR('Paralelipiped',5);
        ->lelipiped
```

b) Funcţii pentru valori numerice

- CEIL(p): întoarce cel mai mic întreg mai mare sau egal cu argumentul p;

```
SELECT CEIL(1.23);
        ->2
SELECT CEIL(-1.23);
        ->-1
```

- FLOOR(p): întoarce cel mai mare întreg mai mic sau egal cu argumentul p;

```
SELECT FLOOR(-1.23);
        ->-2
SELECT FLOOR(1.23);
        ->1
```

- ROUND(p,n): rotunjeşte rezultatul unei expresii (p) la un număr de poziţii fracţionare dacă n este pozitiv, sau, dacă n este negativ, se face la ordinul zecilor, sutelor, miilor etc.

```
SELECT ROUND(-1.23);
        ->-1
SELECT ROUND(-1.58);
        ->-2
SELECT ROUND(1.58);
        ->2
SELECT ROUND(1.298, 1);
        ->1.3
SELECT ROUND(1.298, 0);
        ->1
SELECT ROUND(23.298, -1);
        ->20
```

- TRUNC(p,n): are efect similar funcţiei ROUND, numai că în loc de rotunjire se face trunchiere.

```
SELECT TRUNCATE(1.223,1);
        ->1.2
SELECT TRUNCATE(1.999,1);
        ->1.9
SELECT TRUNCATE(1.999,0);
        ->1
SELECT TRUNCATE(-1.999,1);
```

```
        ->-1.9
SELECT TRUNCATE(122,-2);
        ->100
```

c) Funcţii pentru date calendaristice

- CURRENT_DATE(): furnizează data curentă sub forma ‘YYYY-MM-DD’;

```
SELECT current_date();
        -> 2006-08-15
```

- SYSDATE(): furnizează data curentă şi ora exactă sub forma ‘YYYY-MM-DD HH-MM-SS’;

```
SELECT SYSDATE();
        -> 2006-08-15 15:06:44
```

- CURRENT_TIMESTAMP(), NOW(), SYSTIMESTAMP: sunt sinonime cu SYSDATE;
- DATE_ADD (data,INTERVAL nr.): adună un număr de ani, luni sau zile la data argument;

```
SELECT DATE_ADD('2006-08-15', INTERVAL 1 MONTH);
        -> 2006-09-15
```

- LAST_DAY(data): furnizează ultima zi din luna în care se află data argument;

```
SELECT LAST_DAY('2008-02-15');
        -> 2008-02-29
```

- DATEDIFF(data1,data2): calculează numărul de zile dintre cele două date calendaristice;

```
SELECT DATEDIFF('1997-12-31 23:59:59','1997-12-30');
        ->1
SELECT DATEDIFF('1997-11-30 23:59:59','1997-12-31');
        ->-31
```

- DAYNAME(data): afişează denumirea zilei corespunzătoare datei argument;

```
SELECT DAYNAME(’2006-08-19’);
        -> 'Saturday'
```

- DAYOFMONTH(data), DAYOFYEAR(data): (ziua din lună, ziua din an) furnizează numărul de ordine în cadrul lunii, respective a anului pentru o dată calendaristică.

```
SELECT DAYOFYEAR('2006-08-19');
        ->231
```

d) Funcţii de conversie dintr-un tip în altul

Funcţia de conversie cel mai des utilizată este funcţia CAST.

- CAST: realizează diferite conversii, de exemplu

```
SELECT CONCAT('Data: ', CAST(NOW() AS DATE));
        -> Data: 2006-08-18
```

concatenează şirul de caractere ‘data:’ cu data calendaristică actuală, convertită la şir de caractere.

Exemple

Exemplul 1: Modificaţi toate numerele de telefon din judeţul Maramureş, astfel ca prefixul să nu mai fie 0262 ci 0362, din baza de date AGENTIE_IMOBILIARA.

```
UPDATE DATE_PERSOANA SET
nr_telefon=CONCAT('0362', SUBSTR(nr_telefon, 5))
WHERE SUBSTR(nr_telefon,1,4)='0262';
```

Pentru modificarea datelor s-a folosit comanda

```
UPDATE nume_tabel SET instructiuni
WHERE conditii,
```

iar funcţia

```
SUBSTR(sir,nr)
```

extrage ciferele din întregul număr de telefon, începând cu a cincea poziţie, cifre ce vor fi concatenate cu prefixul ’0362’. De asemenea, prin funcţia

```
SUBSTR(nr_telefon,1,4)
```

se realizează extragerea primelor patru numere din întregul număr de telefon.

Exemplul 2: Afişaţi numele şi data naşterii clienţilor din baza de date AGENTIE_IMOBILIARA, cunoscând codul numeric personal al acestora.

```
SELECT CONCAT('Numele: ',numele) AS numele,
       CONCAT('Anul: ', '19',SUBSTR(cnp,2,2),' , ','Luna:',
              SUBSTR(cnp,4,2), ', ','Ziua: ',
              SUBSTR(cnp,6,2))
              AS data_nasterii
FROM DATE_PERSOANA;
```

Observaţie: În interogarea de mai sus se concatenează şirul ’Numele’ cu numele clientului, apoi se concatenează şirul ’Anul’ cu şirul ’19’ (deoarece se consideră că nu sunt clienţi născuţi după anul 1999) şi cu cele două cifre ale codului numeric personal care desemnează anul naşterii unei persoane (SUBSTR(cnp,2,2)). În mod analog se efectuează şi celelalte concatenări.

numele	data_nasterii
Numele: BABICIU CONSTANTIN	Anul: 1954 , Luna: 09, Ziua: 23
Numele: CHIS GHEORGHE	Anul: 1967 , Luna: 03, Ziua: 21
Numele: SAS IOAN	Anul: 1970 , Luna: 12, Ziua: 05
Numele: VERDES ANDREI	Anul: 1972 , Luna: 04, Ziua: 04
Numele: ACHIM MIHAI	Anul: 1982 , Luna: 03, Ziua: 20
Numele: POP ANA	Anul: 1966 , Luna: 08, Ziua: 05
Numele: ACHIM LOREDANA	Anul: 1966 , Luna: 09, Ziua: 20
Numele: IONESCU MARA	Anul: 1970 , Luna: 12, Ziua: 28
Numele: POP VASICA	Anul: 1982 , Luna: 04, Ziua: 20

Fig. 14.1. Utilizarea funcţiilor CONCAT şi SUSTR

Exemplul 3: Afişaţi preţurile ofertelor (preţul minim şi preţul maxim) folosind alinierea la dreapta şi la stânga.

```
SELECT  LPAD(pret_min,30,' ') AS 'pret minim',
        RPAD(pret_max,30,' ') AS 'pret maxim'
FROM CERERI_OFERTE
WHERE tipul LIKE ’oferta’
ORDER BY pret_min,pret_max;
```

pret minim	pret maxim
NULL	NULL
90000.00	100000.00
120000.00	135000.00
150000.00	160000.00
242000.00	NULL
326700.00	NULL
500000.00	502000.00

Fig. 14.2. Utilizarea funcţiilor LPAD şi RPAD

Exemplul 4: Să se afişeze numărul de cereri de imobile primite în prima jumătate a oricărei luni faţă de cele primite în cea de a doua jumătate.

```
SELECT LTRIM(COUNT(id_co)) AS 'nr de cereri in
       prima jumatate a lunii'
FROM CERERI_OFERTE
WHERE DAYOFMONTH(data_inreg)<16
AND tipul =’cerere’;
```

nr de cereri in prima jumatate a lunii
10

(a)

```
SELECT COUNT(id_co) AS 'nr de cereri in a doua
       jumatate a lunii'
FROM CERERI_OFERTE
WHERE DAYOFMONTH(data_inreg)>15
AND tipul =’cerere’;
```

nr de cereri in a doua jumatate a lunii
1

(b)

Fig. 14.3. Utilizarea funcţiei DAYOFMONTH (a), (b)

SINTEZĂ ŞI EXERCIŢII

1. Cei opt operatori iniţiali

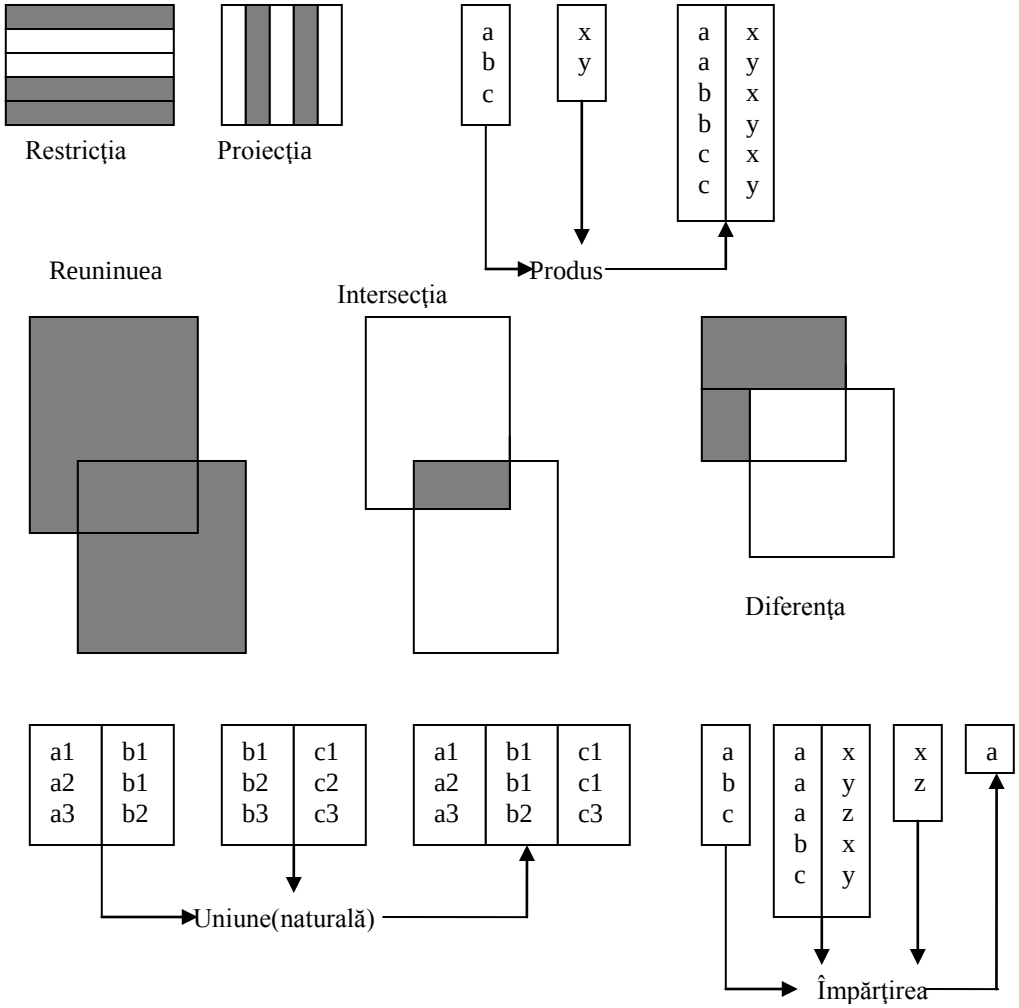


Fig. 1. Reprezentarea operatorilor relaţional

2. Teste

Curs 1 - Exerciţii

1. Prin tipul unei date înţelegem:
a) mulţimea valorilor pe care le poate lua data;
b) structura, mulţimea valorilor, operaţiile admise şi modul de tratare a erorilor;
c) structura, mulţimea valorilor, operaţiile admise, regulile de protecţie şi securitatea datelor.

2. Prin valoarea atomica, înţelegem:
a) o valoare reprezentată printr-un singur simbol;
b) o valoare din care nu putem elimina nici o componentă fără să denaturăm semnificaţia datei;
c) o valoare cu format rigid

3. Atributul reprezintă:
a) mulţimea valorilor posibile ale unei date;
b) o caracteristică a unei entităţi;
c) formatul de reprezentare a unei date

4. Prin baza de date, înţelegem:
a) orice colecţie de date;
b) o colecţie de date conectate logic, exhaustivă şi neredundantă;
c) orice colecţie de date materializată sub forma unor fişiere de disc magnetic

5. Prin sistem de gestiune a bazelor de date, înţelegem:
a) modul de organizare al unui sistem informatic, care manipulează date din baze de date;
b) un program sau pachet de programe, care permite crearea, păstrarea şi eliminarea bazelor de date;
c) un sistem de programe, care permite utilizatorului definirea, crearea şi întreţinerea bazei de date, precum şi accesul controlat la aceasta.

6. Definiţi următorii termeni:
- bază de date
- sistem de gestiune de bază de date
- entitate
- diagramă E-R
- atribut
- asociere

7. Care sunt etapele de realizare a unei baze de date?

Curs 2- Exerciţii

8. O relaţie este:
a) orice tabel bidimensional cu valori atomice;
b) orice tabel;
c) orice tabel bidimensional

9. Se numeşte grad al unei relaţii:
a) numărul domeniilor distincte ale relaţiei;
b) numărul de tupluri distincte ale relaţiei;
c) numărul de atribute ale relaţiei

10. Cardinalitatea unei relaţii reprezintă:
a) numărul de atribute ale relaţiei;
b) numărul de tupluri ale relaţiei;
c) numărul de atribute identificator, ale relatiei.

11. Care sunt componentele unei diagrame E-R?

12. Ce se înţelege prin cardinalitate?

13. Numiţi trei tipuri de asocieri între entităţi.

14. Pentru baza de date FURNIZORI_COMPONENTE descrisă mai jos, în figura 2, precizaţi tipurile de relaţionări între relaţiile F,C, respectiv FC, apoi realizaţi diagrama E-R.

15. Daţi exemple de:
- o relaţie de tip mulţi la mulţi
- o relaţie de tip unu la unu
- o relaţie de tip unu la mulţi.

16. Relaţiile de tip mai multe la mai multe sunt dificil de implementat într-un proiect eficient de baze de date. Ce este de făcut, în acest caz?

17. Desenaţi diagrama entitate-relaţie (precizând şi tipul asocierilor) pentru următoarea bază de date COLECTIE_MUZICA: presupuneţi că aveţi o colecţie muzicală rock, formată din CD-uri şi DVD-uri şi casete audio şi doriţi să construiţi o bază de date care să permită să găsiţi înregistrările pe care le aveţi pentru un anumit interpret (de exemplu, Joe Cocker), pentru un anumit chitarist (de exemplu, Joe Satriani), pentru un anumit baterist (de exemplu, Lars Ulrich), pentru un anumit album (de exemplu, Master of Puppets), o anumită formaţie (de exemplu, Metallica).

Curs 3- Exerciţii

18. Se numeşte domeniu al unui atribut:
a) mulţimea valorilor posibile ale unui atribut;
b) tipul datelor de atribut;
c) mulţimea valorilor actuale ale unui atribut

19. Se numeşte bază de date relaţională:
a) un set de tabele;
b) un set de tabele normalizate;
c) un set de tabele bidimensionale

20. Definiţi următorii termeni:
- cheie primară
-cheie externă
- BDR
- domeniu
- entitate
- atribut
- relaţie

21. Care sunt conceptele utilizate pentru a descrie elementele de bază ale organizării unei BDR?

Curs 4- Exerciţii

22. Două relaţii R1 şi R2 sunt compatibile cu reuniunea, dacă:
a) au acelaşi număr de atribute;
b) au acelaşi număr de atribute, iar atributele care ocupă aceeaşi poziţie au acelaşi nume;
c) au acelaşi număr de atribute, iar atributele care ocupă aceeaşi poziţie au acelaşi domeniu.

23. Pentru a putea determina intersecţia relaţiilor R1 şi R2 este necesar ca:
a) relaţiile să fie compatibile cu intersecţia;
b) relaţiile să fie compatibile cu reuniunea;
c) relaţiile să aibă cel puţin un tuplu comun.

24. Dacă R1 are n1 tupluri, R2 are n2 tupluri, atunci R1 U R2 are:
a) cel puţin n1 + n2 tupluri;
b) n1 + n2 tupluri;
c) cel mult n1 + n2 tupluri.

25. Operatorul de proiecţie aplicat unei relaţii, permite:
a) eliminarea unor coloane;
b) modificarea poziţiei coloanelor;
c) eliminarea unor coloane şi modificarea poziţiei lor

26. Pentru a putea realiza joncţiunea naturală între relaţiile R1 şi R2 este necesar ca:
a) relaţiile R1 şi R2 să fie compatibile cu reuniunea;
b) relaţiile R1 şi R2 să aibă câte un atribut cu acelaşi domeniu;
c) relaţiile R1 şi R2 să aibă cel puţin câte un atribut cu acelaşi nume şi acelaşi domeniu.

27. Se consideră relaţiile:

STUDENT (NR_LEG, NUME, PRENUME, GRUPA) şi
TELEFON (NR_LEG, PREFIX, TELEFON, TIP_TELEFON).

Se presupune că fiecare student are cel mult un telefon mobil. Pentru a rezolva cererea “să se afişeze toţi studenţii unei grupe precizate, iar acolo unde este cazul să se afişeze şi telefonul mobil”, este necesar să utilizam:
a) o joncţiune naturală;
b) o joncţiune de egalitate;
c) o joncţiune externă.

28. Rezultatul aplicării unui operator relaţional, este întotdeauna:
a) o relaţie;
b) o relaţie sau o valoare;
c) o relaţie sau un set de relaţii.

29. Definiţi următorii operatori:
- reuniunea
- diferenţa
- produsul cartezian
- proiecţia
- selecţia
- intersecţia
- diviziunea

30. Enumeraţi cele trei tipuri de joncţiune.

31. Care dintre operatorii relaţionali definiţi în cursul 4 au o definiţie care nu se bazează pe aceeaşi structură?

32. Fie baza de date FURNIZORI_COMPONENTE modelată de următoarele valori eşantion:

F

F#	numeF	Stare	oras
F1	Pop	20	Bucuresti
F2	Achim	10	Ploiesti
F3	Ardelean	30	Ploiesti
F4	Popescu	20	Bucuresti
F5	Ionescu	30	Vaslui

C

C#	numeC	Culoare	Masa	oras
C1	piulita	Rosu	12.0	Bucuresti
C2	Bolt	Verde	17.0	Ploiesti
C3	Surub	albastru	17.0	Arad
C4	Surub	Rosu	14.0	Bucuresti
C5	Cama	Albastru	12.0	Ploiesti
C6	Roata dintata	rosu	19.0	Bucuresti

FC

F#	C#	cant
F1	C1	300
F1	C2	200
F1	C3	400
F1	C4	200
F1	C5	100
F1	C6	100
F2	C1	300
F2	C2	400
F3	C2	200
F4	C2	200
F4	C4	300
F4	C5	400

Fig. 2. Baza de date FURNIZORI_COMPONENTE (valori eşantion)

Furnizorii şi componentele sunt identificate în mod unic prin numărul furnizorului F, respectiv prin numărul componentei C. Fiecare furnizor are un număr „F", un nume „numeF", care nu este neapărat unic, o valoare de cotare sau a stării „stare" şi o localizare „oras". Se presupune că fiecare furnizor este localizat în exact un singur oraş. Fiecare componentă are un număr „C" care este unic, un nume „numeC", o culoare „culoare", o masă „masă" şi localitate în care sunt stocate componentele. Relaţia FC reprezintă livrările şi se exprimă astfel: Furnizorul F livrează componenta C în cantitatea „cant".

Care este valaorea expresiei F JOIN FC JOIN C?

33. Fie R o relaţie de gardul n. Câte proiecţii diferite ale relaţiei R există?

34. Reuniunea, intersecţia, produsul şi joncţiunea sunt atât comutative, cât şi asociative. Verificaţi aceste proprietăţi pe baza unor exemple construite de dumneavoastră.

35. Fie expresia a JOIN b. Dacă relaţiile a şi b au anteturi disjuncte, atunci această expresie este echivalentă cu a TIMES b; dacă au acelaşi antet, atunci este echivalentă cu a INTERSECT b. Verificaţi aceste afirmaţii pe baza unor exemple construite de dumneavoastră.

36. În aritmetică, înmulţirea şi împărţirea sunt operaţii inverse. TIMES şi DIVIDEDBY sunt operaţii inverse în algebra relaţională?

Curs 5- Exerciţii

37. Fie A si B două atribute (simple sau compuse) ale relaţiei R. Se spune că atributul B este dependent funcţional de A, dacă:

a) fiecărei valori a lui A îi este asociată exact o valoare a atributului B;

b) fiecărei valori a lui A îi este asociată cel puţin o valoare nenulă a atributului B;

c) fiecare valoare a lui B este asociată unei valori a lui A.

38. Explicaţi următoarele noţiuni:

- restricţii de integritate
- cheia primară a unei relaţii
- cheia externă a unei relatii
- relaţie care referă
- relaţie referită.

39. În baza de date FURNIZORI_COMPONENTE, daţi exemple de

a) chei primare şi secundare
b) chei simple şi chei compuse
c) relaţie care referă şi relaţie referită.

40. Precizaţi care sunt restricţiile de integritate minimală ale modelului relaţional, apoi enunţaţi aceste restricţii.

41. Ce înseamnă dependenţă funcţională? Daţi două exemple de DF.

42. Se defineşte o relaţie ORAR, cu următoarele atribute:

Z ziua din săptămână (de la 1 la 5)
T perioada din zi (de la 1 la 6)
C numărul sălii de clasă
P numele profesorului
L numele lecţiei.

Tuplul (z,t,c,p,l) apare în această relaţie dacă şi numai dacă la momentul (z,t) lecţia l este predată de profesorul p în sala de clasă c. Se presupune că lecţiile au durata de o perioadă şi că ficare lecţie are un nume, care este unic pentru toate lecţiile predate într-o săptămână. Ce dependenţe funcţionale conţine această relaţie? Care sunt cheile candidat?

43. Fie relaţia F_O_C din figura următoare:

F_O_C

F#	oras	C#	cant
F1	Brasov	C1	100
F1	Brasov	C2	100
F2	Iasi	C1	200
F2	Iasi	C2	200
F3	Iasi	C2	300
F4	Brasov	C2	400
F4	Brasov	C4	400
F4	Brasov	C5	400

Fig. 3. Relaţia F_O_C (valori eşantion)

Determinaţi dependnţele funcţionale.

Curs 7- Curs 8- Exerciţii

44. Se spune că o relaţie este în forma normală 1, dacă:
a) nu conţine atribute compuse;
b) fiecare atribut are numai valori atomice;
c) conţine cel puţin o cheie candidat.

45. Se spune că o relaţie se află în a doua formă normală, dacă:
a) se află în prima formă normală şi fiecare atribut care nu este cheie primară este total dependent de cheia primara;
b) se află în prima formă normală şi orice atribut care nu este cheie candidat este total dependent de cheia primară;
c) se află în prima formă normală şi orice atribut care nu este determinat este total dependent de cheia primară.

46. Se consideră relaţia EXAMEN (NR_LEG, DATA_EXAMEN, ID_MATERIE, SALA, NOTA). Această relaţie se află în:
a) a II-a formă normală;
b) a III-a formă normală;
c) prima formă normală.

47. Există numai trei forme normale? Enumeraţi formele normale.

48. Definiţi următoerele noţiuni:
- atribut simplu (atomic)
- atribut compus.

49. Să se normalizeze bazele de date prezente pe parcursul acestui capitol: COLECTIE_MUZICALA, FURNIZORI_COMPONENTE.

Curs 9- Exerciţii

50. Limbajul SQL reprezintă:
a) un limbaj procedural de descriere şi manipulare a datelor într-o bază de date;
b) un limbaj neprocedural de descriere şi manipulare a datelor într-o baza de date;
c) un limbaj neprocedural de descriere şi manipulare a datelor într-o baza de date relaţională, sau obiect – relatională.

51. Ce este SQL?

52. Care sunt limbajele SQL?

53. Explicaţi care este rezultatul returnat de fiecare dintre următoarele funcţii: MAX, MIN, COUNT, SUM, AVG.

54. Care sunt domeniile celor mai utilizate tipuri de variabile numerice? Dar pentru tipul şir de caractere?

Curs 10- Exerciţii

55. Precizaţi care sunt rezultatele generate de comanda ALTER TABLE...ADD.
56. Care este comanda prin care o cheie primară, o cheie externă sau o constrângere este exprimată?
57. Creaţi tabelul „salariat” având următoarea structură:

Nume	Caracteristici	Tipul
cod_angajat	NOT NULL	INTEGER(4)
nume		VARCHAR(25)
prenume		VARCHAR(25)
functia		VARCHAR(20)
sef		INTEGER(4)
data_angajarii		DATE
varsta		NUMBER
email		CHAR(10)
salariu	Valoare implicită 0	DECIMAL(9,2)

Fig. 4. Structura tabelului SALARIAT (prima formă)

58. Pentru baza de date COLECTIE_MUZICA (vezi problema 9), să se execute toate comenzile din cursul 10, şi anume
- crearea bazei de date
- crearea tabelelor indicând cheile
- modificarea numelui unei tabele
- adăugarea unui atribut
- modificarea unui atribut
- adăugarea/ suprimarea unor restricţii de integritate
- modificarea unei constrângeri
- acordarea/ retragerea drepturilor de acces la baza de date.

59. Exersaţi aceleaşi comenzi pentru baza de date FURNIZORI_COMPONENTE.
60. Să se definească o constrângere la nivel de coloană prin care să se specifice cheia primară şi cheia externă prin construirea tabelei F din baza de date FURNIZORI_COMPONENTE descrisă în problema 15.
61. Să se definească o constrângere la nivel de tabel prin care să se specifice cheia primară şi cheia externă prin construirea tabelei FC din baza de date FURNIZORI_COMPONENTE descrisă în problema 15.
62. După ce tabela F a fost creată în problema 33, suprimaţi cheia primară a tabelei F din baza de date FURNIZORI_COMPONENTE apoi recreaţi cheia primară a tabelei.
63. Aceeaşi cerinţă ca în problema precedentă, pentru cheia străină.
64. Ştergeţi şi apoi creaţi din nou tabelul „salariat” cu următoarea structură:

NUME	TIP	CONSTRÂNGERE
cod_ang	INTEGER(4)	Cheie primară
nume	VARCHAR(25)	NOT NULL
prenume	VARCHAR(25)	
data_nasterii	DATE	data_nasterii<data_angajarii
functia	VARCHAR(9)	NOT NULL
sef	INTEGER(4)	Referă ca şi cheie externă cod_ang din acelaşi tabel
data_angajarii	DATE	
email	VARCHAR(20)	unic
salariu	DECIMAL(12,3)	> 0
cod_dept	INTEGER(4)	NOT NULL
		Combinaţia NUME + PRENUME să fie unică

Fig. 5. Structura tabelului SALARIAT (a doua formă)

65. Ştergeţi tabelul „salariat”, iar apoi recreaţi-l implementând toate constrângerile la nivel de tabel.

Observaţie: Constrângerea de tip NOT NULL se poate declara doar la nivel de coloană.

Curs 11- Exerciţii

66. Se consideră comanda SQL

```
SELECT * FROM (STUDENT INNER JOIN ADRESA ON
NR_LEG = NR_LEG)
INNER JOIN TELEFON ON NR_LEG = NR_LEG
```

Care dintre următoarele afirmaţii este corectă:

a) comanda este corectă;
b) comanda va genera un mesaj de eroare;
c) comanda se execută dar nu se afişează nimic.

67. Se consideră comanda SQL

```
SELECT * FROM STUDENT
LEFT OUTER JOIN TELEFON ON STUDENT. NR_LEG =
        TELEFON.NR_LEG WHERE GRUPA = 7710;
```

Se presupune că în grupa selectată sunt 30 de studenţi, 5 studenţi au un telefon şi 4 studenţi au câte două telefoane. Atunci, în urma execuţiei se va afişa:

a) o listă cu 30 linii;
b) o listă cu 34 de linii;
c) o listă cu 13 linii.

68. Să se afişeze numele clienţilor care au achiziţionate între 3 şi 10 imobile, din baza de date AGENTIE_IMOBILIARA.

69. Pentru baza de date FURNIZORI_COMPONENTE realizaţi următoarele interogări:

A) afişarea tuturor furnizorilor şi a oraşelor unde sunt situate, ordonând descrescător după numele furnizorilor;
B) afişarea tuturor furnizorilor din Ploieşti;
C) afişarea componentelor care au masa sub 14 u.m, inclusiv;
D) afişarea componentelor de culoare roşie şi albastră care nu sunt produse în oraşul Ploieşti;
E) afişarea acelor furnizori care au livrat componenta C2, ordonând după furnizori;
F) afişarea mediei cantităţii livrate de furnizorul F1; (utilizaţi dicţionarul pentru a afla detalii despre funcţia MIN, MAX, SUM, COUNT, AVG);
G) afişarea furnizorilor care au livrat cel mai mare număr de componente de un anumit tip;
H) afişarea furnizorilor care au livrat cea mai mică cantitate de componente, afişând numele acestora oraşul de proveniență al furnizorilor şi componenta livrată;
I) afişarea tuturor culorilor folosite pentru componentele livrate de furnizori;
J) afişarea tuturor informaţiilor furnizorilor a căror nume începe cu “A”;
K) afişarea numelui furnizorilor care au starea cuprinsă între 10 si 20;

70. Să se vizualizeze toate denumirile albumelor existente în baza de date COLECTIE_MUZICA, ordonate alfabetic.

71. Să se realizeze interogări asemănătoare cu cele de la exerciţiul 39 pentru baza de date COLECTIE_MUZICA.

Curs 12- Exerciţii

72. În cazul bazei de date FURNIZORI_COMPONENTE, să se realizeze următoarele interogări:

a) Care sunt numele furnizorilor ce au livrat componenta C2 şi care sunt oraşele din care provin aceşti furnizori?
b) Care componentă roşie s-a livrat în cea mai mare cantate?
c) Din ce oraş provine furnizorul cu cele mai puţine componente vândute, şi care sunt aceste componente?
d) Să se vizualizeze toate informaţiile legate de furnizorii care au livrat componente, precum şi toate informaţiile despre aceste componente.

73. Afişaţi lista cu toţi interpreţii şi cu albumele lor existente în baza de date COLECTIE_MUZICA, ordonând după interpreţi.

74. Vizualizaţi aceeaşi listă de mai sus existentă pe casete audio.

75. Care dintre interpreţi figurează şi pe CD şi pe casete audio?

76. Descrieţi tipurile de JOIN utilizate în procesul de interogare a relaţiilor unei baze de date.

77. Pentru toate interogările de mai sus, să se formuleze fraze SELECT atât cu ajutorul clauzei WHERE cât şi cu ajutorul operaţiilor de JOIN.

78. Fie baza de date H-R cu diagrama reprezentată în figura 6.

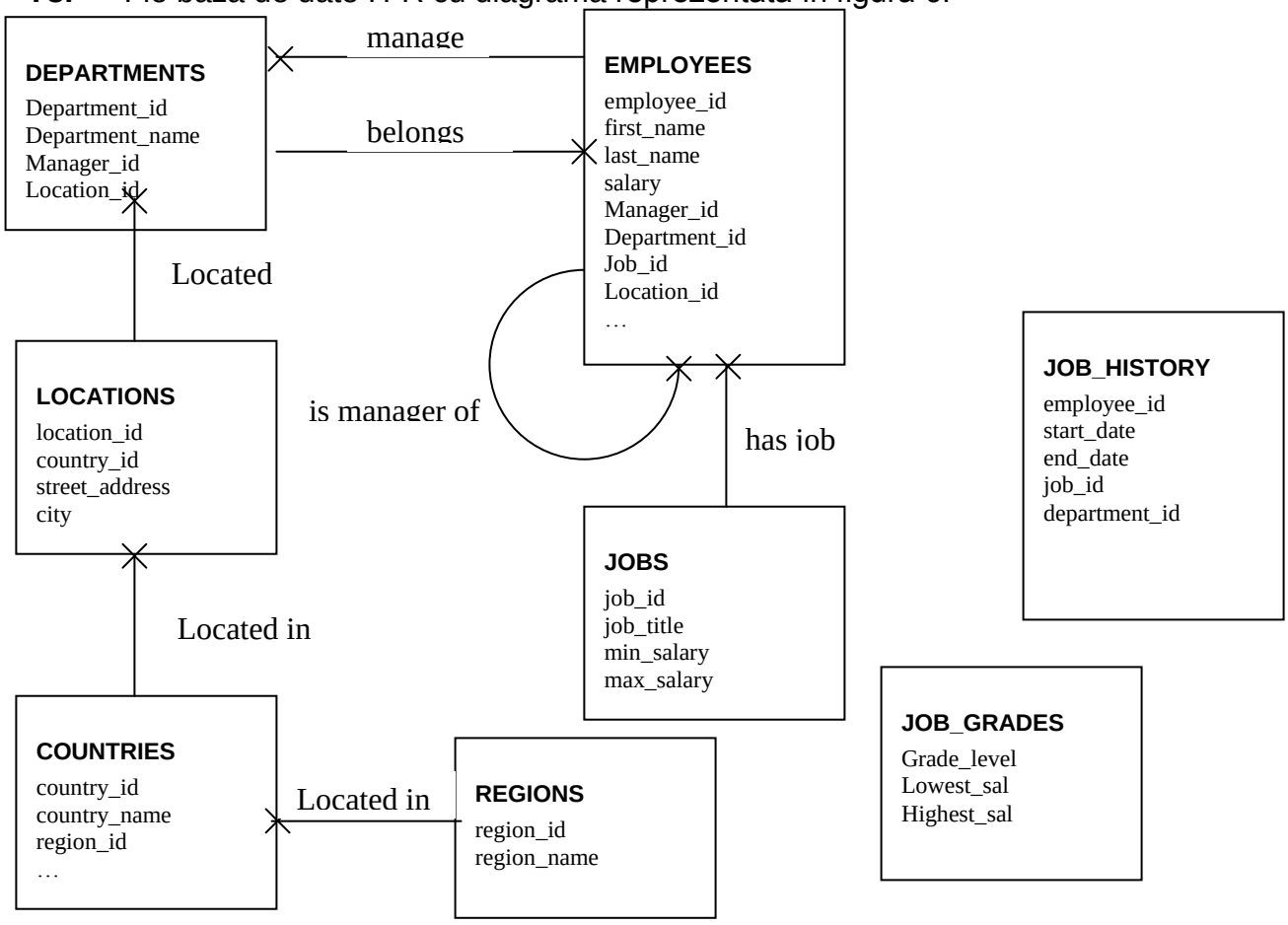


Fig. 6. Diagrama bazei de date H-R

În ipoteza în care tabelele au fost poulate cu date, să se afişeze numele salariaţilor şi numele departamentelor în care lucrează. Se vor afişa şi salariaţii care nu lucrează într-un departament. (right outher join).

80. Să se afişeze numele departamentelor şi numele salariaţilor care lucrează în ele. Se vor afişa şi departamentele care nu au salariaţi. (left outher join).

Curs 13- Exerciţii

81. Din câte localităţi provin furnizorii? (BD utilizată este FURNIZORI_COMPONENTE).

82. Pentru baza de date AGENTIE_IMOBILIARA, precizaţi:

A) care sunt localităţile cu cele mai multe solicitări (cereri) de imobil? Dar cu cele mai multe oferte?

B) în ce lună a anului s-au înregistrat cele mai multe oferte? Dar cele mai puţine cereri?

C) Care este clientul ce a achiziţionat cele mai multe imobile?

83. În ce interogări este necesară utilizarea cuvântului HAVING?

a) când este necesar să eliminăm linii duble din rezultat;

b) când este necesar să ordonăm mulţimea rezultat;

c) când este necesar să efectuăm un calcul pe grup;

d) când este necesar să restricţionăm grupurile de linii returnate.

84. Cu care clauză se utilizează în permanenţă clauză HAVING?

85. Precizaţi deosebirea dintre clauzele ORDER BY şi GROUP BY.

Curs 14- Exerciţii

86. Care sunt funcţiile pentru adăugarea şi eliminarea caracterelor suplimentare (inclusiv a spaţiilor albe) din şirurile de caractere?
87. Se pot folosi mai multe funcţii într-o singură instrucţiune?
88. Afişaţi ultima zi a lunii curente.
89. Să se afişeze data curentă şi data peste trei luni.
90. Afişaţi data peste un an, 2 luni şi 3 zile de la data curentă.
91. Să se afişeze lista clienţilor din baza de date AGENTIE_IMOBILIARA ce trebuie felicitaţi de Sfânta Maria.
92. Extrageţi din fiecare nume a clientului patru caractere, începând cu al cincilea, apoi să se afişeze numărul de caractere care intră în componenţa numelui fiecărui client (se va utiliza baza de date AGENTIE_IMOBILIARA).
93. Scrieţi cu majuscule numele furnizorilor din baza de date FURNIZORI_COMOPNENTE.
94. Înlocuiţi literele „F" şi „C" cu „furniz", respecitv cu „compon" în baza de date FURNIZORI_COMOPNENTE.
95. Să se afişeze lista furnizorilor şi componentelor din baza de date FURNIZORI_COMOPNENTE sub forma „Furnizorul ... a livrat componenta ... în cantitatea de ...".
96. Folosiţi toate cele patru funcţii numerice (CEIL, FLOOR, ROUND, TRUNC) pentru suma de achitat lunar în decursul unui an, sumă ce se obţine împărţind preţul total (de pe factură) al unei tranzacţii imobiliare la numărul de luni dint-un an (din baza de date AGENTIE_IMOBILIARA).
97. Calculaţi numărul de zile cuprins între data de înregistrare a ofertelor şi data de soluţionare a acestora, adică data de pe factură (în cazul în care ofertele au fost soluţionate) utilizând BD AGENTIE_IMOBILIARA.
98. Câte oferte au fost primite în timpul primelor trei zile ale săptămânii (de-a lungul tuturor lunilor) din baza de date AGENTIE_IMOBILIARA? Dar în restul zilelor săptămânii?
99. Afişaţi în ce zile ale săptămânii s-au înregistrat oferte în baza de date AGENTIE_IMOBILIARA. Aceeaşi cerinţă pentru numele lunilor (folosiţi funcţia MONTHNAME).

Bibliografie

1. "*Baze de date-Organizare, proiectare şi implementare*"- Ion Lungu, Constanta Bodea, Georgeta Bădescu, Crsitian Ioniţă, Ed. ALL Educational, Bucureşti, 1995.
2. "*Proiectarea bazelor de date. Normalizare şi postonormalizare. Implementări SQL şi Oracle*"- Marin Fotache, Ed. Polirom.
3. "*Baze de date relaţionale - proiectare şi implementare*"- Ileana Popescu, Editura Universităţii din Bucureşti, 1996.
4. "*Transact SQL*"- Ştefan Ardeleanu, Ed. Niculescu.
5. "*SQL. Dialecte DB2, Oracle şi Visual FoxPro*"- Marin Fotache, Polirom, 2001.
6. "*Oracle 9i2. Ghidul dezvoltării aplicaţiilor profesionale*"- Marin Fotache, Cătălin Strîmbei, Liviu Creţu, Polirom, 2003.