



Universitatea  
Politehnica  
din Bucureşti



Facultatea de  
Automatică și  
Calculatoare



Departamentul  
Calculatoare

# BAZE DE DATE -1

**Conf. dr. ing. Alexandru Boicea**  
**[alexandru.boicea@upb.ro](mailto:alexandru.boicea@upb.ro)**

# **PONDERI EVALUARE FINALA**

## **➤ EVALUARE EXAMEN**

- Evaluare in cursul semestrului – **60%**
  - Prezenta si activitate la curs – 10%
  - Evaluare curs(testare la mijlocul semestrului, fara degrevare) – 20%
  - Evaluare laborator – 30%
- Evaluare finala (examen) – **40%**

## **➤ EVALUARE LABORATOR**

- Prezenta laborator – 10%
- Teste laborator – 40%
- Colocviu laborator – 50%

## **➤ CONDITII PROMOVARE EXAMEN**

- Minim 50% din punctajul de laborator
- Minim 50% din punctajul de examen



# Capitolul 1

## Concepte de baze de date



# Definitii

- **Baza de date** reprezinta o modalitate de stocare a datelor pe un suport extern de memorie, intr-o anumita structura, in vederea interogarilor concurente si prelucrarilor ulterioare pentru extragerea informatiilor.
- **Sistem de gestiune a bazei de date(SGBD)** este un ansamblu de programe software care permit utilizatorilor definirea obiectelor, administrarea bazei de date si accesul concurrent la baza de date. Cele mai cunoscute si utilize sisteme de gestiune sunt Oracle, Microsoft SQL Server, IBM DB2, IBM Informix, MySQL, PostgreSQL, Sybase, etc.



# Definitii

- **Dictionarul bazei de date** este acea componenta a unui SGBD care contine informatii despre obiectele din baza de date si parametrii de sistem, cum ar fi:
  - Structurile tabelelor, indecsii si constrangerile de integritate definite pe ele;
  - Spatiul fizic si organizarea logica a fisierelor;
  - Userii creati pe baza de date si drepturi de acces.
- **Organizarea datelor** reprezinta procesul de definire, structurare si relationare a datelor in colectii de date.
- **Colectie de date** este un ansamblu de date organizate pe anumite criterii de functionalitate, denumita tabela (pentru bazele de date relaționale) și obiect (pentru bazele de date orientate obiect).



# Definitii

- **Structura bazei de date** reprezinta o colectie de descrieri statice ale tipurilor de entitati impreuna cu relatiile logice stabilite intre ele.
- **Entitate** reprezinta un obiect al bazei de date care are o reprezentare unica(de ex. tabelele *Studenti*, *Grupe*, *Catalog*, etc.).
- **Atribut** este o proprietate ce descrie o anumita caracteristica a unei entitati(de ex. pentru tabela *Studenti* atrbute pot fi *nr\_matricol*, *nume*, *grupa*, *specializare*, etc.).



# Definitii

- **Relatiile logice** reprezinta asocierile dintre mai multe entitati care respecta anumite restrictii de functionalitate (un ex. de asociere intre tabelele *Studenti si Grupe* ar fi *nr\_matricol, nume, cod\_grupa*)
- **Fișier de date** reprezinta o colectie de date aflate in asociere, definit printr-o structura logica si una fizica.



# Modele de baze de date

Regulile și conceptele care permit descrierea structurii unei baze de date formează **modelul datelor**:

- **Modelul ierarhic** - datele sunt organizate sub forma unui arbore, nodurile constând din înregistrări(date) iar arcele fiind referințe(pointeri) către alte noduri(datele sunt organizate într-o structură arborescentă de tip tata/fiu).
- Datele sunt grupate în înregistrări ce descriu o ierarhie de tip 1-1 sau 1-N (un parinte poate avea unul sau mai mulți fii, în timp ce un fiu poate avea doar un parinte).



# Modele de baze de date

- Datele sunt organizate în entități și fiecare entitate poate avea mai multe atribută.
- Modelul ierarhic este rar folosit în bazele de date moderne, cel mai cunoscut fiind sistemul de fisiere din Windows.



# Modele de baze de date

- **Modelul retea** - datele sunt organizate sub forma unui graf orientat.
  - Nodurile si arcele au aceeasi semnificatie ca la modelul ierarhic.
  - Modelul retea completeaza modelul ierarhic iar datele sunt grupate in inregistrari ce descriu o ierarhie de tip 1-1, 1-N, K-N (un parinte poate avea unul sau mai multi fii, iar un fiu poate avea mai multi parinti).



# Modele de baze de date

- Modelul retea a fost conceput de Charles Bachman si datele sunt structurate sub forma unui graf orientat, fiecare nod putand avea mai multe inregistrari parinte si mai multi fii.
- Este conceput ca o metoda flexibila de reprezentare a obiectelor si relatiilor dintre ele.
- In modelul retea fiecare inregistrare poate avea parinti si fii mulți, formând o latice care permite o mai bună modelare a relatiilor dintre entități.



# Modele de baze de date

- **Modelul Entitate-Asociere** - este folosit in proiectarea conceptuala de nivel inalt a bazelor de date.
  - Modelul consta intr-o abordare grafica a proiectarii bazelor de date.
  - Datorita simplitatii si expresivitatii sale a fost adoptat de comunitatea stiintifica precum si de producatorii de software si va fi tratat in detaliu intr-un capitol separat.



# Modele de baze de date

- **Modelul Relational** - datele sunt organizate sub forma de tabele ce pot fi relateionate intr-o ierarhie de tip 1-1, 1-N, K-N.
  - Modelul este sustinut si de un limbaj standard de manipulare a datelor.
  - Echivalentul unei entitati intr-o baza de date relationala este tabela, iar a unui atribut este coloana.
  - Este cel mai utilizat model de baze de date si de aceea va fi tratat mai in detaliu intr-un capitol ulterior.



# Modele de baze de date

- **Modelul obiect** - datele sunt reprezentate sub forma de obiecte si sunt folosite in programarea orientata pe obiecte.
- Un obiect este o unitate de program care este folosita in constructia blocurilor de program.
- Fiecare obiect este capabil sa receptioneze parametri, sa prelucreze date si sa transmita rezultate altor obiecte.



# Modele de baze de date

- **Modelul obiect-relational** - se bazeaza pe tehnici de programare *Object-Relational Mapping* ( O/RM, ORM, si O/R mapping) pentru conversia datelor din diferite sisteme de baze de date si limbaje de programare orientate pe obiecte.
- Ele au ca efect crearea unei “baze de date virtuale” care poate fi accesata in limbaje de programare specifice.
- Pentru aceasta sunt disponibile pe piata diferite pachete de programe dar se pot crea si propriile tools-uri ORM.



# Modele de baze de date

- Cel mai raspandit model de baze de date este cel relational, in care datele sunt stocate in tabele. Popularitatea acestui model se datoreaza simplitatii sale (din punct de vedere al utilizatorului) si a posibilitatii de definire a unor limbaje neprocedurale de descriere si manipulare a datelor.
- Termenul de relatie (care da denumirea modelului) provine din matematica, iar reprezentarea intuitiva a unei relatii este o tabela de asociere intre doua coloane a doua tabele diferite.
- In cazul modelului relational descrierea structurii unei baze de date consta in principal din descrierea tabelelor componente: denumire, lista de coloane si tipul datelor stocate in acestea.



# Sistemul de gestiune a bazei de date (SGBD)

- Sistemele de gestiune a bazelor de date structurate au avut, aproximativ, urmatoarea evolutie de-a lungul timpului:
  - Sisteme bazate pe fisiere de date (1950)
  - SGBD bazate pe modelul de date ierarhic(1960)
  - SGBD bazate pe modelul de date retea (1970)
  - SGBD entitate-asociere (1975)
  - SGBD relationale (1980, varianta comerciala)
  - SGBD obiect-relationale(1990)
  - SGBD orientate spre aplicatii (web, baze de date spatiale, temporale, multimedia, etc.) (2000)
  - SGBD de depozitare a datelor (data warehousing, data mining, etc.) (2010)



# Sistemul de gestiune a bazei de date (SGBD)

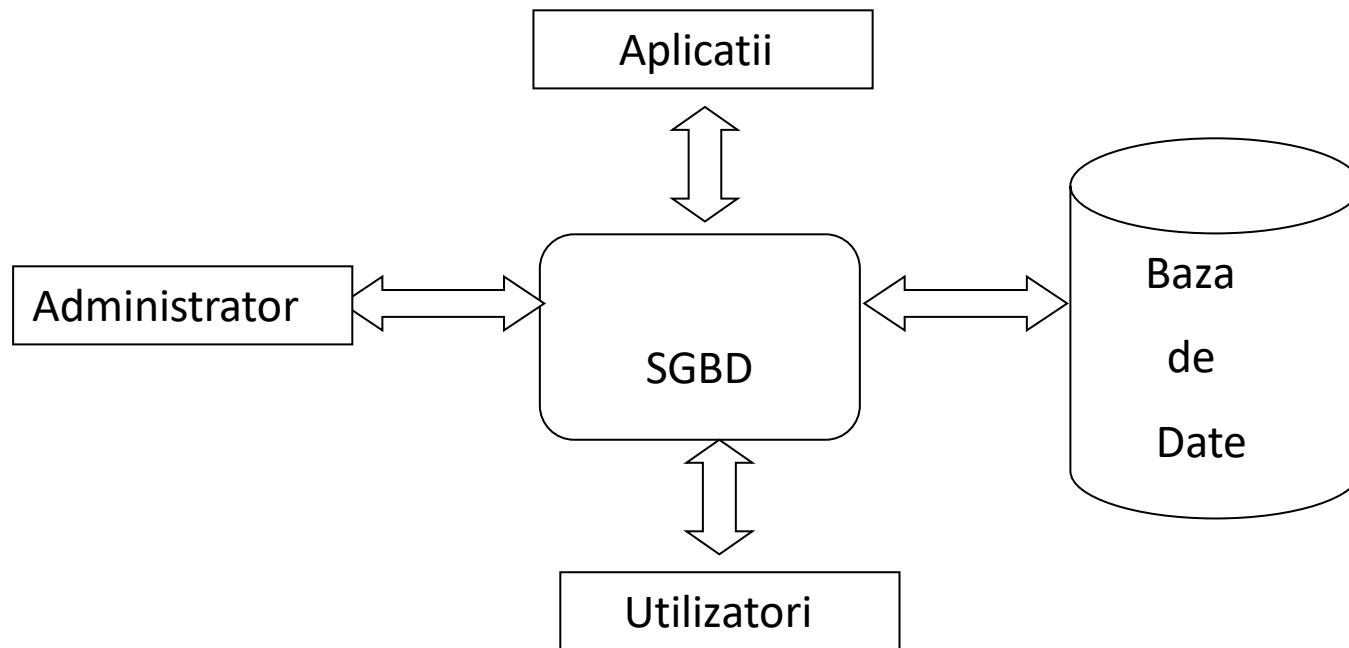


Figura 1. Schema bloc de functionare a unui SGBD



# Sistemul de gestiune a bazei de date (SGBD)

- Functiile principale ale unui SGBD sunt:
  - Definirea structurilor tabelare;
  - Definirea relatiilor dintre tabele;
  - Definirea constrangerilor de integritate a datelor;
  - Manipularea datelor cu ajutorul unui limbaj SQL;
  - Controlul drepturilor de acces la baza de date;
  - Controlul accesului concurrent al userilor la date;
  - Administrarea automata a catalogului bazei de date;
  - Administrarea spatiului fizic si logic de stocare;
  - Controlul backup-urilor (copii de siguranta).



# Definirea structurilor tabelare

- Un SGBD trebuie sa includa posibilitatea definirii structurii obiectelor care formeaza baza de date.
- In cazul bazelor de date relationale aceasta consta in principal in posibilitatea crearii si modificarii structurii tabelelor si constrangerilor de integritate asociate acestora.
- Limbajul prin care se realizeaza aceste operatii se numeste DDL (*Data Definition Language*)
- In sistemele relationale bazate pe SQL aceste operatii au fost incluse in limbaj prin intermediul comenzilor de tip CREATE (pentru creare), ALTER (modificare) sau DROP(stergere din dictionar).



# Definirea structurilor tabelare

- Structura unei tabele este data de urmatoarele specificatii de definire:
  - Definirea coloanelor si tipurile acestora;
  - Definirea constrangerilor de integritate;
  - Definirea tablespace-lui unde se creeaza tabela;
  - Definirea parametrilor logici si fizici.
- Definirea unei tabele trebuie sa respecte anumite reguli:
  - Sa nu existe duplicare de randuri si coloane ;
  - Nu trebuie respectata o anumita ordine a coloanelor;
  - Tipurile de coloane trebuie definite in concordanta cu tipurile de date ;
  - Relatiile intre tabele se fac pe coloane de acelasi tip .



# Integritatea datelor

- Constrainterile de integritate reprezinta anumite reguli pe care datele trebuie sa le respecte la nivel de tabela sau in relatiile cu alte tabele.
- Aceste reguli sunt verificate automat in cazul operatiilor de inserare, stergere si modificare, iar in cazul in care nu se valideaza se genereaza o eroare si tranzactia nu se efectueaza.
- In felul acesta este asigurata o mai mare siguranta in ceea ce priveste corectitudinea datelor si reducerea erorii umane.



# Integritatea datelor

- Constrangerile de integritate pot fi :
  - **Valori nenule** – inregistrarile nu pot contine valori nule;
  - **Cheie unica** – defineste o cheie unica pe una sau mai multe coloane ( nu pot fi mai multe inregistrari cu aceleasi valori pe coloanele respective, dar se accepta valori nule);
  - **Cheie primara** – defineste o cheie primara la nivel de coloana sau tabela ( nu pot fi mai multe inregistri cu aceeasi cheie primara, care nu accepta valori nule);
  - **Cheie externa** – defineste o cheie externa ( tabela se relateaza cu alta tabela pe o cheie unica sau cheie primara);
  - **Verificare** – se forteaza o verificare de indeplinire a unor conditii pe o coloana.



# Manipularea datelor

- Manipularea datelor se refera la operatiile de lucru cu datele inregistrate intr-o baza de date si se realizeaza cu ajutorul unui limbaj DML (*Data Manipulation Language*) care este inclus in limbajul SQL;
- Operatii principale de manipulare a datelor sunt urmatoarele:
  - **Inserarea de date**(adaugarea de linii noi in tabele);
  - **Stergerea de date** (stergerea de linii din tabele);
  - **Modificarea datelor** (modificarea continutului unor linii existente in tabele);
  - **Interogarea datelor** (selectarea liniilor dupa anumite criterii de interogare).



# Categorii de utilizatori

- **Utilizatori privilegiati** - Acestia sunt utilizatori care au dreptul de a afectua toate tipurile de operatii puse la dispozitie de catre sistem (**administratorul bazei de date**);
- **Utilizatori neprivilegiati** - Acestia sunt utilizatorii obisnuiti ai SGBD-ului si dispun de drepturile de acces care le-au fost alocate de catre administratorul bazei de date;
- **Dezvoltatori de aplicatii** - In aceasta categorie intra toti cei implicați in crearea unei aplicatii si in activitatea de mentenanta;
- **Utilizatori de aplicatii** - Sunt cei care au acces la baza de date prin intermediul interfetelor pentru care sunt autorizati.



# Niveluri de reprezentare a datelor

- Exista mai multe niveluri de reprezentare a datelor, fiecare avand caracteristici specifice in functie de perspectiva sub care este vazuta, asa cum se vede in Figura 2:
  - **Nivelul intern** – descrie organizarea interna a datelor;
  - **Nivelul conceptual** – descriere modelul de date(de exemplu modelul relational);
  - **Nivelul extern** - nivelul utilizatorilor care acceseaza baza de date.



# Niveluri de reprezentare a datelor

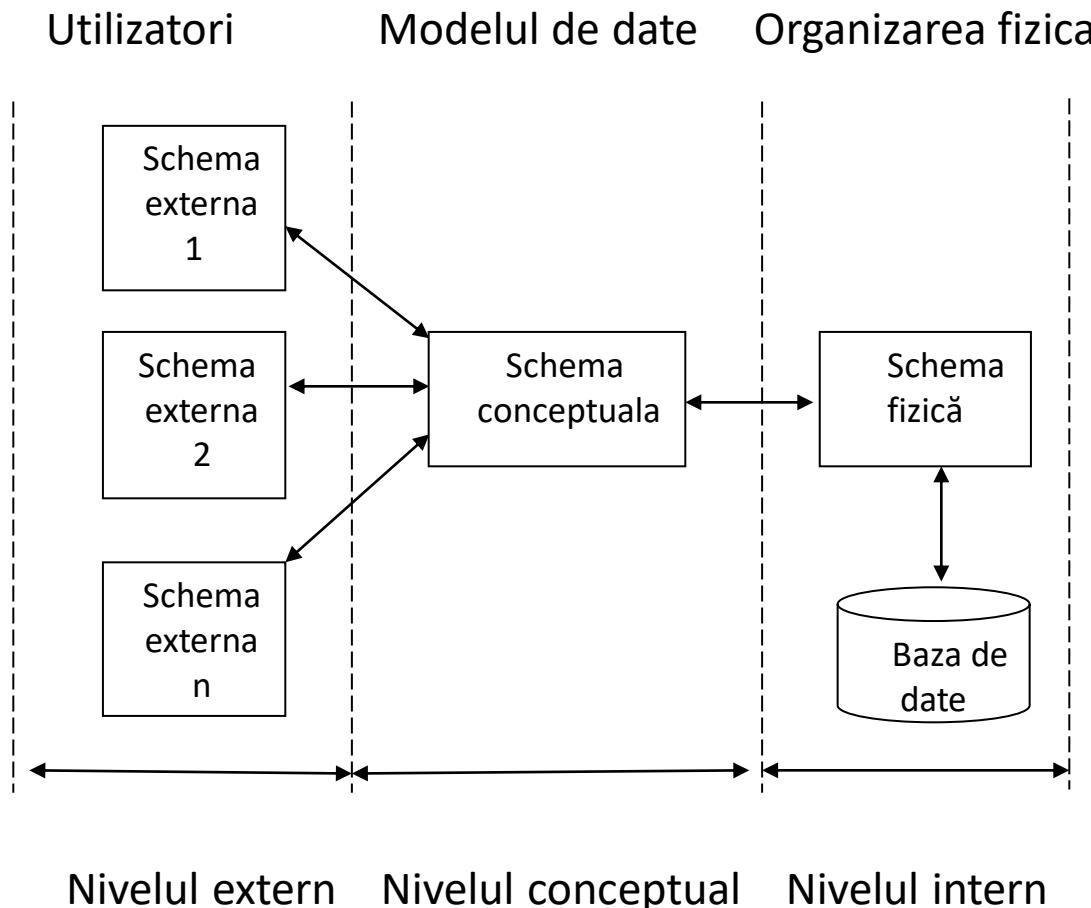


Figura 2. Niveluri de reprezentare a datelor



## Nivelul intern

- Descrierea bazei de date la **nivel intern** poarta numele de **schema fizica** si sistemul de gestiune a bazelor de date pune la dispozitie facilitati pentru inregistrarea si modificarea acestora.
- La acest nivel baza de date este descrisa din perspectiva stocarii sale pe dispozitivele fizice: identificarea discurilor si a cailor unde este stocata, numele fisierelor care formeaza baza de date, structura fizica a acestora, etc.



# Nivelul conceptual

- Descrierea bazei de date la **nivel conceptual** poarta numele de **schema conceptuala** (numita uneori si **schema logica**) a bazei de date.
- In modelul relational schema consta in:
  - Tabelele care formeaza baza de date;
  - Structura (coloanele) fiecarei tabele;
  - Tipul de date asociat coloanelor;
  - Elementele pe baza carora se realizeaza interconectarea tabelelor (coloane comune);
  - Constrainteri de integritate;
  - Operatii declansate automat la modificarea unor elemente ale bazei de date;



# Nivelul extern

- Diferite categorii de utilizatori ai unei baze de date au nevoie în activitatea lor numai de anumite parti specifice ale schemei conceptuale.
- **Nivelul extern** se ocupă de descrierea acestor parti și poartă numele de **scheme externe** (echivalentul userilor creati pe baza de date).
- O baza de date are asociată o singură schema fizică și o singură schema conceptuală, dar mai multe scheme externe.
- Pentru administratorului bazei de date schema externă coincide cu schema conceptuală, deoarece are toate drepturile de acces.
- Celelalte categorii de utilizatori accesează baza de date doar prin intermediul schemelor externe specifice acestora, pe baza drepturilor de acces alocate.



# Independenta datelor

- Existenta celor trei niveluri de descriere permite definirea conceptului de independenta intre datele stocate in baza de date si aplicatiile care utilizeaza aceste date.
- Conceptul de independenta a datelor a aparut odata cu dezvoltarea sistemelor complexe de aplicatii, pentru care cablarea informatiilor structurate in cadrul programului constituie o bariera in calea dezvoltarii si modificarii acestora.



# Independenta datelor

- ***Independenta logica*** reprezinta posibilitatea de schimbare a schemei conceptuale a bazei de date fara modificarea schemelor externe. Conditia este ca modificarea sa nu elimine niciunul dintre elementele necesare translatiei de la schema externa la schema conceptuala.
- Unele operatii implica insa si modificarea definirii schemelor externe.
- Exemple de modificare a schemei conceptuale ar fi:
  - Adaugarea de noi tabele in baza de date;
  - Adaugarea de noi coloane in tabelele existente;
  - Adaugarea de noi constrangeri de integritate;
  - Modificarea numelor tabelelor si coloanelor existente;
  - Modificarea in anumite limite a tipurilor de date;
  - Restructurarea bazei de date.



# Independenta datelor

Exemple:

- Sa consideram o baza de date continand la nivel conceptual o tabela cu date despre studenti cu urmatoarea structura:  
**Studenti (*nr\_matricol*, *nume*, *cod\_facultate*, *medie*)**  
si  $n$  scheme externe continand tabelele virtuale  
**Studenti-1, Studenti-2, ..., Studenti-n** definite astfel:  
**Studenti-n (*nr\_matricol*, *nume*, *cod\_facultate*, *medie*)**  
care contine liniile din tabela **Studenti** unde *cod\_facultate* =  $n$ .
- In cazul modificarii bazei de date prin adaugarea unei noi tabele cuprinzand lista specializarilor din cadrul facultatii si a unei noi coloane in tabela **Studenti** , pentru a specifica la ce specializare este inscris fiecare student, tabelele cu datele studentilor din schemele externe vor ramane aceleasi, daca schimbam structura astfel:



# Independenta datelor

- Baza de date conceptuala:

**Studenti (*nr\_matricol, nume, cod\_facultate, medie, cod\_specializare*)**

**Specializari (*cod\_specializare, den\_specializare*)**

- Schemele externe:

**Studenti-n (*nr\_matricol, nume, cod\_facultate, medie*)** va contine in continuare valorile de pe coloanele *nr\_matricol, nume, cod\_facultate* si *medie* din liniile din tabela **Studenti** unde *cod\_facultate = n*.

- Independenta logica implica folosirea de catre SGBD a informatiilor de definire a schemelor externe, stocate in catalogul sistemului, pentru conversia oricarei operatii din structura schemei externe a aplicatiei sau utilizatorului care a lansat-o, in structura schemei conceptuale a bazei de date.



# Independenta datelor

- ***Independenta fizica*** reprezinta posibilitatea de schimbare a schemei fizice a bazei de date fara modificarea schemei conceptuale si implicit a schemelor externe. Aceasta da posibilitatea reorganizarii fizice a bazei de date fara afectarea aplicatiilor care o folosesc.
- Alte operatii pot fi suportate numai prin modificarea catalogului sau a fisierelor de configurare pe care SGBD-ul le foloseste pentru a face translatia de la schema conceptuala la schema fizica, cum ar fi:
  - Schimbarea dispozitivelor fizice pe care este stocata baza de date;
  - Schimbarea numelor fisierelor fizice in care este stocata baza de date sau a directoarelor unde acestea sunt plasate;



# Independenta datelor

- Adaugarea de noi structuri de cautare rapida (indecsi) pentru cresterea vitezei de executie a anumitor operatii;
- Schimbarea in anumite conditii a structurii fizice a fisierelor bazei de date;
- Schimbarea unor parametri ai sistemului de gestiune care afecteaza modul in care datele sunt stocate la nivel fizic, de exemplu dimensiunea blocului de date.
- Deoarece nu toate sistemele de gestiune a bazelor de date implementeaza total cele trei niveluri de descriere, posibilitatea de a asigura cele doua tipuri de independenta a datelor este conditionata de facilitatile oferite de catre sistemul de gestiune.



# Bibliografie

- Ramez Elmasri, Shamkant B. Navathe, *Fundamentals of Database Systems*, Pearson Education, Boston 2011
- Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer D. Widom, *Database Systems: The Complete Book*, Prentice-Hall, Englewood Cliffs, NJ, 2002.
- J.D. Ullman, J. Widom: *A First Course in Database Systems*, Prentice Hall, first edition 1997, second edition 2002.
- Florin Radulescu, Alexandru Boicea, *Baze de date online*, Editura Oamenilor de Stiinta din Romania, 2011
- Alexandru Boicea, *Baze de date – Note de curs*, Universitatea Politehnica din Bucuresti
- Alexandru Boicea, *Oracle SQL, SQL\*Plus*, Editura Printech, 2007



# Capitolul 2

## Introducere în algebra relațională



# Elemente de algebra relationala

- Algebra relationala este unul dintre cele doua limbaje formale de interogare ale modelului relational si ofera mijloace puternice de a construi relatii noi din alte relatii date. Atunci cand relatiile date sunt reprezentate de date stocate, relatiile construite cu mijloacele algebrei pot fi raspunsuri la fraze de interogare asupra acestor informatii.
- Orice algebra permite construirea de expresii prin aplicarea unor operatori asupra unor operanzi atomici sau asupra altor expresii algebrice.
- In algebra relationala, **operanzii** sunt:
  - **variabile**, care reprezinta relatii;
  - **constante**, care sunt relatii finite.



# Elemente de algebra relationala

- În algebra relatională “clasică” toti operanții și toate rezultatele expresiilor sunt multimi. Vom grupa **operatiile din algebra relatională** în patru clase:
  - **operatii specifice teoriei multimilor** (reuniune, intersecție, diferență), dar aplicate asupra relațiilor;
  - **operatii care îndepărtează parti ale unei relații** (selectie, proiecție);
  - **operatii care asociază tuplurile a două relații** (produs cartezian, jonctiune);
  - **operatia prin care sunt atribuite nume noi atributelor relației și/sau relației.**



# Elemente de algebra relationala

- O proprietate fundamentală în algebra relatională constă în faptul că fiecare operator acceptă instantele unei relații (sau a două) în calitate de argumente și întoarce ca rezultat o alta instanță de relație. Aceasta proprietate permite folosirea compusă a operatorilor (operatia de compunere) pentru a forma fraze de interogare complexe.
- O astfel de fraza de interogare corespunde unei expresii algebrice relationale, care se definește recursiv ca fiind o relație și un operator algebric unar aplicat unei singure expresii ( sau ca un operator algebric binar aplicat la două expresii).



# Operatii pe multimi aplicate relatiilor

## ■ Reuniunea

Fie  $R, S$  relatii.

Reuniunea celor doua relatii este

$T = R \cup S$ , unde

$T = \{ \text{tupluri } t_i, \text{ a. i. } t_i \in r \vee t_i \in s, \forall t_i \in t \}.$

Conditii:  $R, S$  au multimi identice de atribute, cu aceleasi domenii de valori.

Observatie:  $T$  nu contine duplicate.



# Operatii pe multimi aplicate relatiilor

Exemplu:

**R**

Nr_matricol	Nume_student	Cod_grupa	Media
1011	Ionescu Silvia	331CC	9,50
1022	Popescu Daniel	332CC	9,15

**S**

Nr_matricol	Nume_student	Cod_grupa	Media
1011	Ionescu Silvia	331CC	9,50
1033	Popa Cornel	332CC	9,05

**T = R ∪ S**

Nr_matricol	Nume_student	Cod_grupa	Media
1011	Ionescu Silvia	331CC	9,50
1022	Popescu Daniel	332CC	9,15
1033	Popa Cornel	332CC	9,05



# Operatii pe multimi aplicate relatiilor

## ■ Intersectia

Fie  $R, S$  relatii.

Intersectia relatiilor este  $T = R \cap S$ , unde

$T = \{ \text{tupluri } t_i, \text{ a. i. } t_i \in r \wedge t_i \in s, \forall t_i \in t \}$ .

Observatie:  $T$  nu contine duplicate.

Exemplu:

$$T = R \cap S$$

Nr_matricol	Nume_student	Cod_grupa	Media
1011	Ionescu Silvia	331CC	9,50



# Operatii pe multimi aplicate relatiilor

## ■ Diferenta

Diferenta poate fi definita astfel:

$$T = R - S, \text{ unde}$$

$$T = \{ \text{tupluri } t_i, \text{ a. i. } t_i \in r \wedge t_i \notin s, \forall t_i \in t \} \quad \text{sau}$$

$$T = S - R, \text{ unde}$$

$$T = \{ \text{tupluri } t_i, \text{ a. i. } t_i \in s \wedge t_i \notin r, \forall t_i \in t \}.$$

Exemplu:

$$T = R - S$$

Nr_matricol	Nume_student	Cod_grupa	Media
1022	Popescu Daniel	332CC	9,15

$$T = S - R$$

Nr_matricol	Nume_student	Cod_grupa	Media
1033	Popa Cornel	332CC	9,05

Observatie:  $R - S \neq S - R$ .



# Operatii pe multimi aplicate relatiilor

## ■ Proiectia

Fie relatia  $R$  cu attributele  $A_1, A_2, \dots, A_n$ .

Fie, de asemenea, attributele  $A_1, A_2, \dots, A_k$ , a. î.

$\{A_1, A_2, \dots, A_k\} \subset \{A_1, A_2, \dots, A_n\}$ .

Atunci, proiectia relatiei  $R$  pe attributele  $A_1, A_2, \dots, A_k$  (sau pe valorile acestor attribute), notata cu  $\pi_{A_1, A_2, \dots, A_k}(R)$ , este relatia obtinuta din  $R$  prin extragerea valorilor atributelor  $A_1, A_2, \dots, A_k$  (π - pi)



# Operatii pe multimi aplicate relatiilor

Exemplu:

$\pi_{\text{ nume\_student, media}}(S)$

Nume_student	Media
Ionescu Silvia	9,50
Popa Cornel	9,05



# Operatii pe multimi aplicate relatiilor

- **Proiectia** se poate defini formal ca fiind relatie:

$\pi_{i_1, i_2, \dots, i_k} = \{ \text{tupluri } t_i, \text{ a. } \hat{t}. t_i = (a_1, a_2, \dots, a_k),$   
iar in  $R$   $\exists$  tuplul  $t_j = (b_1, b_2, \dots, b_n)$ ,  $a_j = b_j$  pentru  
 $j = 1 \dots k \}$

Mai sus  $a_j$ , respectiv  $b_j$ , sunt valori ale atributelor corespunzatoare din multimile  $\{ A_1, A_2, \dots, A_k \}$  si, evident,  $\{ A_1, A_2, \dots, A_n \}$ . Simbolurile  $a_1, a_2, \dots$  reprezinta coloane din  $R$ .



# Operatii pe multimi aplicate relatiilor

## ■ Selectia

Prin definitie, operatia de selectie aplicata unei relatii  $R$ , notata cu  $\sigma_F(R)$ , consta in extragerea din  $R$  a acelor tupluri care indeplinesc formula (clauza)  $F$ . Schema relatiei obtinuta este aceeasi cu schema relatiei  $R$ , atributele fiind aranjate, prin conventie, in aceeasi ordine. Operanzii continuti in clauza  $F$  sunt constante sau atribut din schema relatiei  $R$ . Operatorii sunt fie operatori aritmetici uzuali, fie operatori logici(de comparatie, negatie, etc). *( $\sigma$  –sigma)*



# Operatii pe multimi aplicate relatiilor

Exemplu:

$\sigma$  media > 9  $\wedge$  media < 9.50 (**T**) , unde **T**= **R**  $\cup$  **S** ;

Nr_matricol	Nume_student	Cod_grupa	Media
1022	Popescu Daniel	332CC	9,15
1033	Popa Cornel	332CC	9,05



# Operatii pe multimi aplicate relatiilor

## ■ Produsul cartezian

Fie relatiile **R** si **S** de aritati  $R_1, R_2$  (domenii de valori).

Fie in **R** si **S** tuplurile  $(r_{i1}, r_{i2}, \dots, r_{ik})$ , respectiv  $(s_{j1}, s_{j2}, \dots, s_{jp})$ .

Formal, produsul cartezian **T** = **R** × **S** al relatiilor **R** si **S** se defineste prin:

$T = \{ \text{tupluri } t_{ij}, \text{ a. i. } t_i = (r_{i1} r_{i2} \dots r_{ik} s_{j1} s_{j2} \dots s_{jp}), \text{ unde}$   
 $(r_{i1} r_{i2} \dots r_{ik}) \in R \text{ si } (s_{j1} s_{j2} \dots s_{jp}) \in S \}$ .

- Tuplurile din **T** reprezinta toate asocierile posibile dintre tuplurile din **R** si tuplurile din **S**.



# Operatii pe multimi aplicate relatiilor

Exemplu:

R

Nr_matricol	Nume_student	Cod_spec
1011	Ionescu Silvia	1
1022	Popescu Daniel	2
1033	Popa Cornel	1

S

Cod_spec	Den_specializare
1	TI
2	AIS

T = R x S

Nr_matricol	Nume_student	R.Cod_spec	S.Cod_spec	Den_specializare
1011	Ionescu Silvia	1	1	TI
1011	Ionescu Silvia	1	2	AIS
1022	Popescu Daniel	2	1	TI
1022	Popescu Daniel	2	2	AIS
1033	Popa Cornel	1	1	TI
1033	Popa Cornel	1	2	AIS



# Operatii pe multimi aplicate relatiilor

- Observatie:

Numarul de atribute ale produsului cartezian

$T = R \times S$  este suma atributelor relatiilor R si S

( $3 + 2 = 5$  atribute) iar numarul de tupluri este produsul numerelor de tupluri ale relatiilor R si S

( $3 \times 2 = 6$  tupluri).



# Operatii pe multimi aplicate relatiilor

## ■ Jonctiunea (*Join*)

Jonctiunea (numita *Join* sau *Theta-join*) este o operatie compusa, care implica efectuarea unui produs cartezian si a unei selectii.

Fie relatiile **R** si **S**, joinul lor (notat  $R \bowtie_F S$ ) se obtine din produsul cartezian al relatiilor R si S urmat de o selectie dupa conditia F (numita si *conditie de join*). Operatorii din conditia F pot fi operatori aritmetici sau operatori logici.

$$R \bowtie_F S = \sigma_F (R \times S)$$



# Operatii pe multimi aplicate relatiilor

Exemplu:

- Sa facem o selectie pentru studentii din **R** care au specializarea TI (cod\_spec=1), facand apoi un join cu **S** pentru a extrage denumirea specializarii.
- ✓ In pasul intai se calculeaza produsul cartezian:

$$T = R \times S$$

Nr_matricol	Nume_student	R.Cod_spec	S.Cod_spec	Den_specializare
1011	Ionescu Silvia	1	1	TI
1011	Ionescu Silvia	1	2	AIS
1022	Popescu Daniel	2	1	TI
1022	Popescu Daniel	2	2	AIS
1033	Popa Cornel	1	1	TI
1033	Popa Cornel	1	2	AIS



# Operatii pe multimi aplicate relatiilor

- ✓ În pasul al doilea se face o selectie pe relatia  $T = R \times S$  cu conditia de join

$$F = r.cod\_spec=1 \wedge r.cod\_spec = s.cod\_spec$$

$$R \bowtie_{r.cod\_spec=1 \wedge r.cod\_spec=s.cod\_spec} S$$

Nr_matricol	Nume_student	R.Cod_spec	S.Cod_spec	Den_specializare
1011	Ionescu Silvia	1	1	TI
1033	Popa Cornel	1	1	TI

- În cazul în care condiția de join este una de egalitate, joinul se mai numește *equi-join*. În restul cazurilor se folosește sintagma *non-equi-join*.
- În limbajul SQL condițiile de join se pun în clauza WHERE a unei cereri SELECT sau UPDATE.



# Operatii pe multimi aplicate relatiilor

## ■ Jonctiunea naturală( *Natural Join*)

Join-ul natural pentru doua relatii **R** si **S** (notat  $R \bowtie S$  )se obtine facand joinul celor doua relatii dupa conditia “coloanele cu acelasi nume si de acelasi tip au valori egale” si eliminand prin proiectie atributele duplicat (coloanele dupa care s-a facut join-ul).

- Observatie: Join-ul natural nu tine cont de semnificatia coloanelor dupa care se face join.
- In limbajul SQL se specifica in clauza WHERE prin sintaxa NATURAL JOIN .



# Operatii pe multimi aplicate relatiilor

Exemplu:

Pentru exemplul anterior, daca se face un join natural dupa atributul Cod\_spec, care este comun ambelor relatii, obtinem:

$R \bowtie S$

Nr_matricol	Nume_student	Cod_spec	Den_specializare
1011	Ionescu Silvia	1	TI
1022	Popescu Daniel	2	AIS
1033	Popa Cornel	1	TI



# Operatii pe multimi aplicate relatiilor

## ■ Jonctiune externa (*Outer Join*)

Am vazut in exemplele anterioare ca un join simplu sau join natural returneaza acele linii care indeplinesc conditia de join (join simplu) sau conditia de egalitate a atributelor (denumire si valori) in cazul unui join natural. Joinul extern se foloseste atunci cand in rezultat dorim sa apara si liniile care nu indeplinesc o conditie de join (din lipsa de date), de exemplu studentii care nu au specificata specializarea pe coloana Cod\_spec sau specializarile care nu au studenti asignati. In acest caz coloanele care nu au valori de corespondenta apar cu valoare nula.



# Operatii pe multimi aplicate relatiilor

- Exista trei tipuri de join extern:
  - ✓ **Join extern stanga** (*left outer join*), in care in rezultat apar toate tuplurile relatiei din stanga operatorului.  
Notatia este:  $R \bowtie_L S$ .
  - ✓ **Join extern dreapta** (*right outer join*), in care in rezultat apar toate tuplurile relatiei din dreapta operatorului.  
Notatia este:  $R \bowtie_R S$
  - ✓ **Join extern complet** (*full outer join*), in care in rezultat apar toate tuplurile relatiilor din stanga si din dreapta operatorului. Notatia este:  $R \bowtie S$
- Observatie: In rezultatul joinului extern sunt intotdeauna continute tuplurile (liniile) din rezultatul joinului general dupa aceeasi conditie.



# Operatii pe multimi aplicate relatiilor

Exemplu:  $R$

Nr_matricol	Nume_student	Cod_spec
1011	Ionescu Silvia	1
1022	Popescu Daniel	2
1033	Popa Cornel	1
1044	Toma Diana	4

S	Cod_spec	Den_specializare
1	TI	
2	AIS	
3	SI	

$$T = R \times S$$

Nr_matricol	Nume_student	R.Cod_spec	S.Cod_spec	Den_specializare
1011	Ionescu Silvia	1	1	TI
1011	Ionescu Silvia	1	2	AIS
1011	Ionescu Silvia	1	3	SI
1022	Popescu Daniel	2	1	TI
1022	Popescu Daniel	2	2	AIS
1022	Popescu Daniel	2	3	SI
1033	Popa Cornel	1	1	TI
1033	Popa Cornel	1	2	AIS
1033	Popa Cornel	1	3	SI
1044	Toma Diana	4	1	TI
1044	Toma Diana	4	2	AIS
1044	Toma Diana	4	3	SI



# Operatii pe multimi aplicate relatiilor

**R**◁○▷  $L(r.\text{cod\_spec}=s.\text{cod\_spec})$  **S**

Nr_matricol	Nume_student	R.Cod_spec	S.Cod_spec	Den_specializare
1011	Ionescu Silvia	1	1	TI
1022	Popescu Daniel	2	2	AIS
1033	Popa Cornel	1	1	TI
1044	Toma Diana	4	null	null

- Observatie: Rezultatul contine linii din **R**(studenti) care nu au **Cod\_spec** corespondent in **S**(specializari).



# Operatii pe multimi aplicate relatiilor

**R**▷◁ O ▷◁ **R(r.cod\_spec=s.cod\_spec)** **S**

Nr_matricol	Nume_student	R.Cod_spec	S.Cod_spec	Den_specializare
1011	Ionescu Silvia	1	1	TI
1022	Popescu Daniel	2	2	AIS
1033	Popa Cornel	1	1	TI
null	null	null	3	SI

- Observatie: Rezultatul contine linii din **S**(specializari) care nu au Cod\_spec corespondent in **R**(studenti).



# Operatii pe multimi aplicate relatiilor

**R▷◁O▷◁ S**  
(r.cod\_spec=s.cod\_spec)

Nr_matricol	Nume_student	R.Cod_spec	S.Cod_spec	Den_specializare
1011	Ionescu Silvia	1	1	TI
1022	Popescu Daniel	2	2	AIS
1033	Popa Cornel	1	1	TI
1044	Toma Diana	4	null	null
null	null	null	3	SI

- Observatie: Rezultatul contine linii din **R(studenti)** care nu au Cod\_spec corespondent in **S(specializari)**, dar si linii din **S(specializari)** care nu au Cod\_spec corespondent in **R(studenti)**.



# Operatii pe multimi aplicate relatiilor

## ■ Semi-jonctiunea (*Semi-Join*)

Prin definitie, semi-jonctiunea relatiilor  $R$  si  $S$ , notata  $R \triangleright < S$  este proiectia jonctiunii naturale a celor doua relatii pe atributele din  $R$ :  $R \triangleright < S = \pi_R R \bowtie S$

O formula echivalenta pentru realizarea acestei operatiuni este  $R \triangleright < S = R \bowtie \pi_{R \cap S}(S)$ . Deci semi-join-ul lui  $R$  in raport cu  $S$  este o relatie care contine multimea tuplurilor lui  $R$  care participa la joinul natural cu  $S$ . In general, un semi-join nu este simetric, deci  $R \triangleright < S \neq S \triangleright < R$ . Diferenta intre join-ul conventional si semi-join este ca un semi-join retuneaza cel mult o linie din prima tabela atunci cand gaseste mai multe linii corespondente in a doua tabela care respecta conditia de join, adica nu retuneaza dubluri de linii. In limbajul SQL semi-join-ul se foloseste impreuna cu operatorii EXIST si IN.



# Operatii pe multimi aplicate relatiilor

Exemplu:

Pentru exemplul anterior, daca se face un semi-join natural dupa atributul Cod\_spec, care este comun ambelor relatii, obtinem:

$R \triangleright\!< S$

Nr_matricol	Nume_student	Cod_spec
1011	Ionescu Silvia	1
1022	Popescu Daniel	2
1033	Popa Cornel	1

$S \triangleright\!< R$

Cod_spec	Den_specializare
1	TI
2	AIS



# Operatori pe multiset-uri

- **Multiset** este o multime in care se admit aparitii multiple ale unor elemente. Multimile conventionale, inclusiv relatiile, nu contin duplicate. In practica bazelor de date intr-o tabela sau un rezultat al unei cereri de interogare de date pot sa apară linii dupicat. In acest caz nu mai putem vorbi de relatiile (care nu permit tupluri dupicat) ci de multiseturi (*bags*). Primele SGBD care au folosit modelul relational au continut limbaje de interogare bazate pe algebra relationala. Aceste sisteme tratau relatiile ca multiset-uri, nu ca multimi conventionale, din ratiuni de eficienta a timpului de interogare. Cu alte cuvinte, daca utilizatorul nu cerea explicit ca tuplurile dupicat (prezente in realitate) sa fie eliminate, relatiile rezultate puteau contine duplicate.
- Prezentam pe scurt efectul unora dintre operatorii de mai sus aplicati multiset-urilor:



# Operatori pe multiset-uri

## ■ Reuniunea

Rezultatul operatiei este asemanator cu al reuniunii din algebra relationala dar din rezultatul final nu se elimina duplicatele.

Exemplu:

**R** (contine notele studentilor care au urmat cursul BD1)

Nr_matricol	Nume_student	Nota
1011	Ionescu Silvia	9
1022	Popescu Daniel	7
1033	Popa Cornel	10
1044	Toma Diana	8

**S** (contine notele studentilor care au urmat cursul BD2)

Nr_matricol	Nume_student	Nota
1011	Ionescu Silvia	8
1022	Popescu Daniel	7
1033	Popa Cornel	10



# Operatori pe multiset-uri

$T = R \cup S$  (contine notele studentilor care au urmat cursurile BD1 si BD2)

Nr_matricol	Nume_student	Nota
1011	Ionescu Silvia	9
1022	Popescu Daniel	7
1033	Popa Cornel	10
1044	Toma Diana	8
1011	Ionescu Silvia	8
1022	Popescu Daniel	7
1033	Popa Cornel	10

In urma operatiei de reuninune se obtine multiset-ul  $T$  care contine linii dupicat.



# Operatori pe multiset-uri

## ■ Intersectia

Rezultatul operatiei este asemanator cu al intersectiei din algebra relationala dar din rezultatul final nu se elimina duplicatele.

Exemplu:

$T = R \cap S$  (contine notele studentilor care au urmat ambele cursuri si au obtinut aceeasi nota la fiecare dintre ele)

Nr_matricol	Nume_student	Nota
1022	Popescu Daniel	7
1033	Popa Cornel	10
1022	Popescu Daniel	7
1033	Popa Cornel	10

- In urma operatiei de intersectie se obtine multiset-ul  $T$  care contine linii dupicat.



# Operatori pe multiset-uri

## ■ Proiectia

Are acelasi mod de calcul, ca si in cazul relatiilor, dar la final nu se elimina liniile dupicat.

Exemplu:

S

Nr_matricol	Nume_student	Disciplina	Nota	Data_examen
1011	Ionescu Silvia	BD1	9	6.06.2020
1022	Popescu Daniel	BD1	7	6.06.2020
1033	Popa Cornel	BD1	10	6.06.2020
1044	Toma Diana	BD1	8	6.06.2020
1011	Ionescu Silvia	BD2	8	1.02.2021
1022	Popescu Daniel	BD2	7	1.02.2021
1033	Popa Cornel	BD2	10	1.02.2021



# Operatori pe multiset-uri

$T = \pi_{nr\_matricol, nume\_student, nota}(S)$

Nr_matricol	Nume_student	Nota
1011	Ionescu Silvia	9
1022	Popescu Daniel	7
1033	Popa Cornel	10
1044	Toma Diana	8
1011	Ionescu Silvia	8
1022	Popescu Daniel	7
1033	Popa Cornel	10

- Se observa ca multisetul  $T$ , obtinut prin proiectie, contine linii dupicat pentru studentii care au obtinut aceeasi nota la disciplinele BD1 si BD2. In acest caz o multime conventionala  $S$ (care nu contine linii dupicat) se poate transforma prin proiectie intr-un multiset  $T$ .



# Operatori pe multiset-uri

- În mod similar relațiilor se calculează și **diferenta, produsul cartezian, selectia, joinul, joinul natural, joinul extern**, dar cu urmatoarele observații:
  - multiset-urile operand pot să contină linii dupăcat;
  - multiset-ul rezultat poate să contină liniile dupăcat.
- Observație: În cazul acestor operații nu pot apărea linii dupăcat decât dacă operanții contin linii dupăcat.



# Operatori extinsi ai algebrei relationale

- Majoritatea limbajelor de interogare moderne se bazeaza pe definitiile operatiilor relationale, precum si pe cele privind tratarea multiset-urilor. Totodata, in practica actuala, limbajele de interogare SQL permit efectuarea unor operatii suplimentare, importante in aplicatii, cum ar fi:

- **Redenumirea**

Exista doua modalitati de a face redenumirea tabelelor si/sau coloanelor:

- **Operatorul de redenumire ( $\rho$ )** - permite atat redenumirea relatiilor/multiseturilor cat si a atributelor acestora:

Fiind data relata  $R(A_1, A_2, \dots, A_n)$ , putem obtine alta relatie  $S(B_1, B_2, \dots, B_n)$ , care are acelasi continut ca si  $R$  dar atributele se numesc  $B_1, B_2, \dots, B_n$ , unde  $S = \rho_{R(A_1, A_2, \dots, A_n)} \cdot$  (ρ - rho)

- Intr-un limbaj SQL se foloseste pentru definirea alias-urilor de coloana/tabela folosite in cererile de interogare SELECT.



# Operatori extinsi ai algebrei relationale

- **Constructorul** - permite redenumirea atributelor in rezultatul unei expresii relationale sau pe multiseturi, de exemplu putem redenumi intr-un rezultat un atribut prin constructia:

Nume\_vechi  $\rightarrow$  Nume\_nou

Exemplu: Fie o relatie

$R=(\text{cod\_grupa}, \text{nr\_matricol}, \text{nume\_stud}, \text{nota}).$

In rezultatul proiectiei

$\pi_{\text{cod\_grupa} \rightarrow \text{Grupa}, \text{nume\_stud} \rightarrow \text{Nume}, \text{media\_notelor} \rightarrow \text{Media\_gen}}(R)$   
atributele se vor numi : Grupa, Nume, Media\_gen.

- In SQL se foloseste pentru aliasul de coloana intr-o cerere SELECT.



# Operatori extinsi ai algebrei relationale

## ■ Eliminare duplicate

Acest operator se poate aplica doar pe multiseturi (relatiile contin tupluri duplikat) iar efectul este eliminarea duplikatelor din multiset.

- Notatia operatorului este urmatoarea:

Fiind dat un multiset  $R$ , atunci  $\delta(R)$  este un multiset fara duplicate (practic devine o relatie).

( $\delta$  - *delta*)

- In SQL se foloseste optiunea DISTINCT intr-o cerere SELECT.



# Operatori extinsi ai algebrei relationale

Exemplu:

T – multiset

Nr_matricol	Nume_student	Nota
1011	Ionescu Silvia	9
1022	Popescu Daniel	7
1033	Popa Cornel	10
1044	Toma Diana	8
1011	Ionescu Silvia	8
1022	Popescu Daniel	7
1033	Popa Cornel	10

$\delta(T)$  – relatie obtinuta din multiset-ul T

Nr_matricol	Nume_student	Nota
1011	Ionescu Silvia	9
1022	Popescu Daniel	7
1033	Popa Cornel	10
1044	Toma Diana	8
1011	Ionescu Silvia	8



# Operatori extinsi ai algebrei relationale

## ■ Grupare

Acest operator este folosit pentru gruparea rezultatelor operatiilor dupa anumite criterii si se poate aplica atat relatiilor cat si multiseturilor.

Sintaxa operatorului de grupare este urmatoarea:

$\gamma_{\text{atribute\&functii}} (R)$  *( $\gamma$  - gamma)*

unde:

- **attribute** - sunt criterii de grupare si apar in rezultatul returnat de operator;
- **functii** - sunt functii de grup (de ex.: MIN, MAX, SUM, AVG, COUNT) care se calculeaza la nivelul fiecarui grup si de asemenea apar in rezultatul operatorului.
- In SQL operatorul se aplica prin functii de grup impreuna cu clauza GROUP BY.



# Operatori extinsi ai algebrei relationale

Exemplu:

S

Nr_matricol	Nume_student	Disciplina	Nota	Data_examen
1011	Ionescu Silvia	BD1	9	6.06.2020
1022	Popescu Daniel	BD1	7	6.06.2020
1033	Popa Cornel	BD1	10	6.06.2020
1044	Toma Diana	BD1	8	6.06.2020
1011	Ionescu Silvia	BD2	8	1.02.2021
1022	Popescu Daniel	BD2	7	1.02.2021
1033	Popa Cornel	BD2	10	1.02.2021

$T = \gamma_{disciplina, Count(*) \rightarrow Nr\_stud, AVG(nota) \rightarrow Media\_discip}(S)$

Disciplina	Nr_stud	Media_discip
BD1	4	8.50
BD2	3	8.33



# Operatori extinsi ai algebrei relationale

## ▪ Sortare

Sintaxa operatorului de sortare este urmatoarea:

$$\tau_{\text{lista\_atribute}}(\mathbf{R}) \quad (\tau - tau)$$

- Operatorul are ca efect sortarea(ordonarea) relatiei sau multisetului  $\mathbf{R}$  in functie de atributele din lista.
- Intr-o relatie, sau multiset, datele nu sunt neaparat ordonate (si nici nu este nevoie). Operatorul se aplica, de regula, atunci cand se fac interogari pentru intocmirea de liste ordonate dupa anumite atributе.
- In limbajul SQL se foloseste clauza ORDER BY dintr-o cerere SELECT.



# Operatori extinsi ai algebrei relationale

Exemplu:

R

Nr_matricol	Nume_student	Cod_grupa
1011	Ionescu Silvia	331CC
1022	Popescu Daniel	332CC
1033	Popa Cornel	333CC
1044	Ene Diana	334CC

$$T = \tau_{\text{nume\_student}}(R)$$

Nr_matricol	Nume_student	Cod_spec
1044	Ene Diana	334CC
1011	Ionescu Silvia	331CC
1033	Popa Cornel	333CC
1022	Popescu Daniel	332CC



# Operatori extinsi ai algebrei relationale

## ▪ Proiectia extinsa

Acest operator este analog proiectiei obisnuite dar permite definirea de atribute(coloane) noi, obtinute prin calcule cu ajutorul unor expresii pe relatii sau multiseturi. Sintaxa este urmatoarea:

$$\pi_{\text{expresie}_1, \text{expresie}_2, \dots \text{expresie}_n} (R)$$

- Observatie: Dupa ce se calculeaza rezultatele, duplicatele se elimina sau nu se elimina, in functie de rezultatul dorit (relatie sau multiset).
- In SQL acest operator este implementat in cererea SELECT folosind operatori sau functii aplicate pe atribute(coloanele) deja definite.



# Operatori extinsi ai algebrei relationale

Exemplu:

S

Nr_matricol	Nume_student	Disciplina	Nota	Data_examen
1011	Ionescu Silvia	BD1	9	6.06.2020
1022	Popescu Daniel	BD1	7	6.06.2020
1033	Popa Cornel	BD1	10	6.06.2020
1044	Toma Diana	BD1	8	6.06.2020
1011	Ionescu Silvia	BD2	8	1.02.2021
1022	Popescu Daniel	BD2	7	1.02.2021
1033	Popa Cornel	BD2	10	1.02.2021

$T = \pi_{nr\_matricol, nume\_student, AVG(nota) \rightarrow media\_BD} (S)$

Nr_matricol	Nume_student	Media_BD
1011	Ionescu Silvia	8.50
1022	Popescu Daniel	7
1033	Popa Cornel	10
1044	Toma Diana	8



# Capitolul 3

## Modelul relational



# Modelul relational

- Modul de organizarea a datelor intr-o baza de date trebuie sa respecte un anumit model, iar operatia in sine se numeste **modelarea datelor**.
- Asa cum am aratat intr-un capitol anterior, au existat mai multe modele de baze de date in evolutia sistemelor de gestiune a bazelor de date, fiecare aducand ceva nou pentru modelarea datelor si o crestere a performantelor.
- **Modelul relational** a fost introdus in anul 1970 de Edgar Frank Codd si este modelul cel mai utilizat de catre sistemele de gestiune aparute dupa anul 1980.
- Obiectele bazei de date se numesc **entitati** si sunt relationate intre ele prin anumite **constrangeri** care formeaza asa numita **diagrama de relatii**.



# Modelul relational

- In **Modelul relational (MR)** datele sunt stocate sub forma tabelara si aduce o serie de avantaje fata de modelele anterioare:
  - datele sunt stocate sub forma de linii intr-o tabela;
  - s-a renuntat la folosirea pointerilor si navigarea prin structuri arborescente;
  - pentru accesarea datelor exista limbaje specializede de nivel inalt, de exemplu limbajul SQL;
  - permite introducerea de constrangeri de integritate, ceea ce duce la reducerea anomalilor si a redundantei datelor;
  - asigura un control riguros al drepturilor de acces;
  - ofera facilitati sporite de securitate a datelor.



# Elementele modelului relational

## ■ Domeniul

**Domeniul** reprezinta o multime de valori si este identificat printr-un nume.

- Un domeniu se poate defini fie prin enumerarea elementelor sale, fie prin specificarea unor caracteristici definitorii ale acestora.

Exemplu:

- Culori = {rosu, galben, albastru, violet, verde}
- Note = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} sau  
 $\text{Note} = \{n \in \mathbb{N}^* \mid n \geq 1 \text{ si } n \leq 10\}$
- Char40 = {Multimea sirurilor de maxim 40 caractere}
- Intreg5 = {Multimea numerelor intregi pozitive din intervalul [0, 99999]}



# Elementele modelului relational

- **Produsul cartezian pe domenii** este definit similar cu cel din teoria multimilor:

Fiind date  $n$  domenii  $D_1, D_2, \dots, D_n$  produsul cartezian al domeniilor este multimea sirurilor de domenii:

$$D_1 \times D_2 \times \dots \times D_n = \{ (v_1, v_2, \dots, v_n) \mid v_i \in D_i, i = 1, \dots, n \}$$

- Trebuie mentionat ca in sirul de domenii care participa la un produs cartezian unele se pot gasi in mod repetat, de exemplu:

$$PC = \text{Intreg} \times \text{Char40} \times \text{Intreg} \times \text{Char40}$$



# Elementele modelului relational

## ■ Relatia

**Relatia** reprezinta o submultime a unui produs cartezian avand asociat un nume.

- Un exemplu de relatie apartinand produsului cartezian  $PC = \text{Studenti} \times \text{Specializari}$  este:

**Specializari \_stud**

```
{   (1011, 'Ionescu Silvia', 1, 'CTI'),  
    (1022, 'Popescu Daniel', 2, 'AI'),  
    (1033, 'Popa Cornel', 1, 'CTI') }
```

- **Elementele unei relatii** sunt denumite in literatura de specialitate **tupluri** (tuple). Relatia de mai sus contine 3 dintre elementele produsului cartezian din care provine (3 tupluri).



# Elementele modelului relational

- O reprezentare intuitiva pentru relatia de mai sus, in care fiecare element al relatiei devine o linie a unei tabele si fiecare coloana corespunde unui domeniu din produsul cartezian de baza, este urmatoarea :

## **Specializari\_stud**

1011	Ionescu Silvia	1	CTI
1022	Popescu Daniel	2	AII
1033	Popa Cornel	1	CTI

- Deci o relatie se poate reprezenta ca o tabela de date, fiecare coloana avand asociat un anumit tip de date, impus de domeniul din care provine.



# Elementele modelului relational

## ■ Atributul

**Atributul** reprezinta o coloana a unei relatii avand asociat un nume.

- Deoarece o relatie are o reprezentare tabelara putem vorbi de “coloana a unei relatii” si putem sa facem urmatoarea similitudine: “atributele unei relatii” reprezinta “coloanele unei tabele”.
- Pentru relativa **Specializari\_stud** putem defini urmatoarele atribute:
  - Nr\_matricol – matricolul studentului
  - Nume\_student – numele studentului
  - Cod\_spec – codul specializarii
  - Den\_specializare – denumirea specializarii



# Elementele modelului relational

## ■ Schema

**Schema unei relatii reprezinta structura relatiei compusa din **nume, attribute, domeniul atributelor si constrangerile de integritate** asociate.**

- Datele dintr-o relatie se pot schimba in timp deoarece se pot adauga, modifica sau sterge linii, dar structura relatiei ramane constanta.
- Exista mai multe modalitati prin care se poate specifica schema unei relatii. In exemplele urmatoare prezentam cateva dintre acestea cu referire la relatia Medii:



# Elementele modelului relational

**Medii** = Nr\_matricol, Nume\_student, Cod\_grupa, Media

**Medii** (Nr\_matricol, Nume\_student, Cod\_grupa, Media)

**Medii** (Nr\_matricol: Intreg, Nume\_student: Char40,  
Cod\_grupa: Char5, Media: Numar)

**Medii** (Nr\_matricol intreg, Nume\_student char(40),  
Cod\_grupa char(5), Media numar)

- În teoria bazelor de date relationale se foloseste și notatia urmatoare pentru o relatie:

**R** = ABCDE ,

care specifică ca schema relației **R** conține 5 attribute notate cu A, B, C, D și E.



# Elementele modelului relational

- **Cheia**

**Cheia unei relatii** este o multime minimala de atribute ale caror valori identifica in mod unic un tuplu al relatiei respective. Deoarece o relatie nu accepta linii dupicat inseamna ca ele pot fi identificate in mod unic prin una sau mai multe coloane. Cheia unei relatii este o caracteristica a schemei acesteia si se defineste ca o constrangere de integritate.

- In tabela **Medii** tuplurile pot fi identificate in mod unic prin atributele (Nr\_matricol, Nume\_student) sau (Nr\_matricol, Nume\_student, Cod\_grupa) dar ele nu indeplinesc si conditia de minimalitate.



# Elementele modelului relational

- Daca consideram ca numarul matricol este unic pentru fiecare student atunci putem defini atributul Nr\_matricol ca fiind cheia relatiei Medii, fiind indeplinita si conditia de minimalitate.
- Niciunul dintre celelalte atribute, luate separat, nu poate fi cheie, de exemplu Nume\_student nu este cheie deoarece pot exista doi studenti cu acelasi nume si in acest caz nu pot fi identificate unic tuplurile .
- Daca facem referire la tabele, rezulta ca nu pot exista doua linii avand aceeasi combinatie de valori pe coloanele care formeaza cheia tabelei respective (proprietaate denumita in literatura de specialitate si unicitatea cheii).



# Elementele modelului relational

- Trebuie specificat ca intr-o relatie se pot identifica uneori mai multe chei. Sa luam ca exemplu relatia: **Studenti( Nr\_matricol, Nume\_student, Adresa, CNP, Data\_nastere)**
- Dupa cum se stie CNP-ul unei persoane este unic asa ca relatia Studenti poate avea doua chei Nr\_matricol si CNP, ambele indeplinind conditia de unicitate si minimalitate.
- Observatie: Deoarece intr-o relatie nu pot exista doua tupluri identice, rezulta ca multimea tuturor atributelor relatiei formeaza o cheie sau contine cel putin o cheie, deci orice relatie are cel putin o cheie.



# Elementele modelului relational

- În literatura de specialitate și în sistemele de gestiune a bazelor de date există alte trei concepte legate de cheie și care vor fi prezentate în acest capitol și capitolele următoare:
- **Cheie primara** (*Primary Key*)
- **Cheie străină** (*Foreign Key*)
- **Supercheie** (*Superkey*)



# Elementele modelului relational

## ▪ Valori nule

**Valoare nula (null value)** este o valoare care modeleaza o informatie nespecificata si este o informatie inaplicabila pentru prelucrari de date(nu se pot aplica operatori aritmetici pe ea).

- Uneori, unele atribute ale unei relatii nu au nicio valoare specificata si in acest caz se spune ca atributul respectiv are o valoare nula.

Exemplu:

## Studenti

Nr_matricol	Nume_student	CNP	Cod_Grupa
1011	Ionescu Silvia	2901028400772	331CC
1022	Popescu Daniel	<null>	332CC
1033	Popa Cornel	1911115451664	<null>



# Elementele modelului relational

- În exemplul de mai sus studentul Popescu Daniel nu are specificat CNP-ul și spunem că atributul CNP are valoare nula. Înregistrarea poate fi totuși identificată unic după atributul Nr\_matricol care este cheie. Dacă se dorește însă să se identifice studentul după CNP atunci cererea de interogare nu va returna nicio linie.
- Valorile nule pot fi modificate ulterior prin comenzi DML, de exemplu în SQL se folosește comanda UPDATE.
- Observație: Într-o tabelă o cheie care identifică unic un tuplu nu trebuie să contină attribute cu valoare nula.



# Elementele modelului relational

- **Constrangeri de integritate**

Constrangerile de integritate reprezinta un mecanism prin care un SGDB poate controla corectitudinea datelor in cazul operatiilor DML de manipulare a datelor. In cazul violarii acestor constrangeri de integritate SGBD-ul va rejecta orice tranzactie pe tabela respectiva.

- ✓ **Constrangerea de valori nenule( NOT NULL)**

Este o constrangere care se aplica numai la nivel de atribut(coloana) si verifica daca inregistrarile au valori nule pe coloanele respective, fortand un cod de eroare care anuleaza tranzactia. Orice incercare de a adauga o linie care contine valori nule pe acea coloana sau de a modifica o valoare nenula intr-una nula va genera o eroare sistem.



# Elementele modelului relational

- Cand se creeaza constrangeri pe o cheie primara se creeaza automat si o costrangere NOT NULL pe atributele respective (o cheie primara nu trebuie sa contina valori nule pe coloanele care o definesc).

Exemplu:

Daca definim atributul CNP de tip NOT NULL atunci cand se insereaza o linie sau se modifica valoarea CNP-ul in valoare nula (se sterge valoarea) SGDB-ul va genera o eroare si operatia se anuleaza.

## ✓ **Constrangerea de unicitate (UNIQUE )**

Se foloseste cand vrem ca un atribut(coloana) sau perechi de atribut sa nu contina valori duplicate.



# Elementele modelului relational

- Verificarea se face numai pentru inregistrari cu valori nenule deoarece constrangerea de unicitate permite inserarea de valori nule in coloanele respective.
- Daca coloanele definite unice sunt si not null, atunci putem considera unicitatea ca o cheie unica.
- In mod automat se creeaza si un index pe coloanele definite unice, ceea ce duce la cresterea vitezei de interogare pe tabela.

Exemplu: Daca definim atributul CNP de tip UNIQUE, la orice operatie de inserare sau modificare de date, SGDB-ul face o verificare automata de unicitate si in caz de duplicare se genereaza o eroare care anuleaza operatia.



# Elementele modelului relational

## ✓ **Constrangerea de cheie primara (PRIMARY KEY)**

Se foloseste pentru definirea unei chei primare la nivel de coloana (cheia contine o singura coloana) sau la nivel de tabela ( cheia contine una sau mai multe coloane).

- O tabela poate avea o singura cheie primara.
- Nu se accepta valori nule pentru niciuna dintre coloanele care definesc o cheie primara.
- Cand se creeaza o cheie primara timpul de raspuns in cazul unei interogari se reduce foarte mult.



# Elementele modelului relational

Exemplu:

- Daca definim atributul Nr\_matricol cheie de tip PRIMARY KEY atunci cand se face o inserare sau o modificare de linie se face o verificare de unicitate dar si de valori nenule(nu se accepta valori nule ale atributelor care compun cheia primara) pe coloana de matricole.
- In caz de violare a constrangerii se anuleaza operatia.
- Daca avem o cheie primara compusa(pe mai multe coloane) verificarea se face pe toate coloanele care compun cheia primara, nu pe coloane individuale.



# Elementele modelului relational

## ✓ **Constrangerea de cheie strina (FOREIGN KEY)**

Acest tip de constrangere se foloseste pentru relationarea unei tabele cu una sau mai multe tabele, verificand daca valorile atributelor definite ca FOREIGN KEY ( cheie strina sau cheie externa) sunt cuprinse in valorile coloanelor altel tabele care trebuie sa fie definite UNIQUE sau PRIMARY KEY.

Mai exact, o relationare cu cheie externa pe o tabela se poate face numai daca coloanele de referinta formeaza o cheie primara sau au unicitate.



# Elementele modelului relational

- Cand se face relationarea intre doua tabele trebuie avute in vedere urmatoarele reguli de functionare :
  - Inserarea unei linii intr-o tabela relationata (pe care am definit FOREIGN KEY) se poate face daca exista numai o singura linie in tabela de referinta ( in care am definit PRIMARY KEY sau UNIQUE) corespunzator coloanelor de relationare;
  - Coloanele definite FOREIGN KEY accepta valori nule;
  - Stergerea unei linii din tabela de referinta nu se poate face atata timp cat exista linii relationate in tabela relationata, pe linia respectiva;
  - Regulile de mai sus sunt valabile si in cazul relationarii pe coloane.



# Elementele modelului relational

Exemplu: Consideram relatiile:

**Specializari** (Cod\_spec, Den\_specializare) si

**Studenti** (Nr\_matricol, Nume\_student, Cod\_grupa, Cod\_spec).

- Tabela **Specializari** va contine toate specializarile din facultate, fiecare avand un cod unic.
- Daca definim in tabela **Studenti** atributul Cod\_spec ca fiind FOREIGN KEY cu referinta pe atributul Cod\_spec din tabela **Specializari**, in momentul in care se face o operatie de inserare sau modificare de linie in tabela **Studenti** SGDB-ul face o verificare daca valoarea Cod\_spec se afla printre valorile de pe coloana Cod\_spec din tabela **Specializari** , iar daca rezultatul este negativ operatia se anuleaza. Coloana Cod\_spec din tabela **Specializari** trebuie sa contine valori unice, adica trebuie obligatoriu sa fie o cheie primara/unica.



# Elementele modelului relational

## ✓ **Constrangerea de conditie (CHECK)**

Acum tip de constrangere se foloseste pentru a forta valorile unei coloane sa verifice o conditie prestabilita.

Conditia poate sa contina si functii, cu unele exceptii (sysdate, user, unele functii de tip data calendaristica etc.) .

Exemplu: Consideram relatia

**Note(Nr\_matricol, Nume\_student, Cod\_grupa,  
Disciplina, Data\_examen, Nota).**

- Pentru evitarea erorilor de operare se poate defini coloana Nota de tip CHECK cu conditia ( $Nota \geq 1$  and  $Nota \leq 10$ ) sau conditia  $\text{max}(Nota) \leq 10$ .



# Expresii sigure. Domenii de siguranta

- **Definitie:** Relatia  $P = \{ t \mid \neg R(t) \}$  in calculul relational reprezinta toate tuplurile posibile de lungime  $t$  (aritatea lui  $R$ ) care nu apartin relatiei  $R$ .
- Se pune problema cum putem afla “toate tuplurile posibile” si care este domeniul lor de valori.
- **Definitie:** O **expresie**  $\{ t \mid \Psi(t) \}$  poate fi considerata **sigura**, daca se poate demonstra ca fiecare componenta a oricarui tuplu  $t$ , care satisface formula  $\Psi$ , apartine domeniului  $DOM(\Psi)$ .
- In expresie,  $t$  este o **variabila tuplu** iar  $\Psi$  este o **formula** si semnifica “multimea tuturor tuplurilor  $t$  care verifica formula  $\Psi$ ”.
- Se doreste eliminarea expresiilor  $\{ t \mid \neg \Psi(t) \}$  considerate “nesigure” si considerarea numai a expresiilor care sunt “sigure”.



# Expresii sigure. Domenii de siguranta

- **Definitie:** Domeniul expresiei, notat  $\text{DOM}(\Psi)$  reprezinta multimea elementelor care:
  - fie apar explicit in formula  $\Psi$ ,
  - fie sunt componente ale unui tuplu oarecare, al unei relatii  $R$  oarecare, mentionata in formula  $\Psi$ .
- Trebuie remarcat ca  $\text{DOM}(\Psi)$  nu se determina prin simpla inspectare a formulei  $\Psi$ , ci este determinat in functie de relatiile care se substituie efectiv variabilelor din  $\Psi$ .
- $\text{DOM}(\Psi)$  este intotdeauna finit deoarece consideram ca toate relatiile sunt finite.

Exemple:

- 1) Daca  $R$  este o relatie cu atributele  $A_1, A_2$  si daca  $\Psi[t] = R[t] \wedge t[1] = c \wedge R(t)$ , atunci  $\text{DOM}(\Psi)$  este o relatie unara data de formula algebraica relationala:  $\{c\} \cup \pi_{A_1}(R) \cup \pi_{A_2}(R)$ .



# Expresii sigure. Domenii de siguranta

2) Fie  $\Psi(t)$  o formula care nu incalca regulile de siguranta.

- Atunci, orice expresie de forma  $\{ t \mid R(t) \wedge \Psi(t) \}$  este sigura, deoarece orice tuplu  $t$  care satisface  $R(t) \wedge \Psi(t)$  este in  $R$ , de unde rezulta ca fiecare dintre componentele sale este in  $\text{DOM}(R(t) \wedge \Psi(t))$ .

3) Formula pentru diferenta a doua multimi  $R$  si  $S$

$\{ t \mid R(t) \wedge \neg S(t) \}$ , este de forma mentionata mai sus, cu  $\Psi(t) = \neg S(t)$ .

Formula pentru selectie este, de asemenea, de aceeasi forma.



# Expresii sigure. Domenii de siguranta

- Generalizare a celor de mai sus: observam ca orice formula  $\{ t \mid R_1(t) \vee R_2(t) \vee \dots \vee R_k(t) \wedge \Psi(t) \}$  este, de asemenea, sigura;  $t[i]$  trebuie sa fie un simbol care apare in componenta  $i$  a unui tuplu oarecare al unei relatii oarecare  $R_j$ . Remarcam, de asemenea, ca formula data pentru reuniune intr-un capitol anterior, este de aceasta forma, dar lipsind  $\Psi$ . Cu alte cuvinte, putem lua pe  $\Psi$  ca fiind o formula intotdeauna adevarata, ca  $t[1] = t[1]$ .
- O expresie sigura se poate aplica si pe o proiectie a lui  $R$ :  
$$\{ t^{(m)} \mid (\exists u_1) (\exists u_2) \dots (\exists u_k) (R_1(u_1) \wedge R_2(u_2) \wedge \dots \wedge R_k(u_k) \wedge t[1] = u_{i1}[j_1] \wedge t[2] = u_{i2}[j_2] \wedge \dots \wedge t[m] = u_{im}[j_m] \wedge \Psi(t, u_1, u_2, \dots, u_k)) \}$$
. Aici, componenta  $t[1]$  este constransa la a fi un simbol care apare in cea de-a  $j_1$  componenta a unui tuplu.



# Calcul relational pe tupluri

- Pe langa algebra relationala, cererile de regasire a informatiei intr-o baza de date relationala pot fi exprimate si prin **calcul relational pe tupluri** (CRT) sau **calcul relational pe domenii** (CRD). In acest paragraf sunt prezentate pe scurt aceste doua modalitati de exprimare a cererilor.
- **Calcul relational pe tupluri**

In calcul relational pe tupluri o cerere se exprima printr-o **expresie** de forma:  $\{ t \mid \Psi(t) \}$  unde **t** este o variabila tuplu, iar  **$\Psi$**  o formula. Semnificatia expresiei este “multimea tuturor tuplurilor **t** care verifica formula  **$\Psi$** ”.



# Calcul relational pe tupluri

Formula  $\Psi$  este compusa din elemente (numite si atomi) care pot fi de trei tipuri:

- Elemente de tip  $R(s)$ , unde  $R$  este un nume de relatie iar  $s$  o variabila tuplu. Semnificatia este “ $s$  este un tuplu din  $R$ ”;
- Elemente de tip  $s[i] \theta v[j]$ , unde  $s$  si  $v$  sunt variabile tuplu iar  $\theta$  un operator prin care se poate compara componenta  $i$  a variabilei tuplu  $s$  cu componenta  $j$  a variabilei tuplu  $v$  ;
- Elemente de tip  $s[i] \theta a$  sau  $a \theta s[i]$ , prin care componenta  $i$  a variabilei tuplu  $s$  se compara cu constanta  $a$ .  
*( $\theta$ - theta)*



# Calcul relational pe tupluri

Pe baza acestor atomi se poate defini recursiv ce este o formula si ce sunt aparitiile libere sau legate ale variabilelor tuplu:

- Orice atom este in acelasi timp formula. Toate aparitiile unei variabile tuplu intr-un atom sunt aparitii libere.
- Daca  $\Psi$  si  $\emptyset$  sunt doua formule, atunci  $\Psi \vee \emptyset$ ,  $\Psi \wedge \emptyset$  si  $\neg\Psi$  sunt formule cu semnificatia “ $\Psi$  sau-logic  $\emptyset$ ”, “ $\Psi$  si-logic  $\emptyset$ ” si respectiv “not  $\Psi$ ”. Aparitiile de variabile tuplu sunt libere sau legate in aceste formule, dupa cum ele sunt libere sau legate in componentele acestora. Este permis ca o aceeasi variabila tuplu sa aiba o aparitie libera in  $\Psi$  si o alta legata in  $\emptyset$ .



# Calcul relational pe tupluri

- Daca  $\Psi$  este o formula atunci si  $(\exists s)(\Psi)$  este formula.
  - Aparitiile variabilelor tuplu  $s$ , care sunt libere in  $\Psi$  sunt legate in  $(\exists s)(\Psi)$ .
  - Celealte aparitii de variabile tuplu din  $\Psi$  raman la fel (libere sau legate) in  $(\exists s)(\Psi)$ .
  - Semnificatia acestei formule este urmatoarea: exista o valoare concreta a lui  $s$  care inlocuita in toate aparitiile libere din  $\Psi$  face ca formula sa fie adevarata.
- Parantezele pot fi folosite in formule dupa necesitati. Precedenta este: intai comparatiile, apoi  $\exists$  si  $\forall$  si in final  $\neg$ ,  $\wedge$ ,  $\vee$  (in aceasta ordine).



# Calcul relational pe tupluri

- Daca  $\Psi$  este o formula atunci si  $(\forall s)(\Psi)$  este formula.
  - Aparitiile variabilei tuplu  $s$  care sunt libere in  $\Psi$  sunt legate in  $(\forall s)(\Psi)$ .
  - Celelalte aparitii de variabile tuplu din  $\Psi$  raman la fel (libere sau legate) in  $(\forall s)(\Psi)$ .
  - Semnificatia acestei formule este urmatoarea: orice valoare concreta a tuplului  $s$ , pusa in locul aparitiilor libere ale acestuia din  $\Psi$ , face ca formula sa fie adevarata.



# Calcul relational pe tupluri

Exemple de expresii si formule:

- Expresia  $\{t \mid R(t) \vee S(t)\}$  este echivalenta reuniunii a doua relatii din algebra relationala. Analog  $\{t \mid R(t) \wedge S(t)\}$  reprezinta intersectia a doua relatii.
- Expresia pentru proiectia lui  $R$  pe atributele  $i_1, i_2, \dots, i_k$  se poate scrie astfel:  $\{t^{(k)} \mid (\exists u)(R(u) \wedge t[1] = u[i_1] \wedge t[2] = u[i_2] \wedge \dots \wedge t[k] = u[i_k])\}$
- Formula  $(\exists s)(R(s))$  spune ca relatia  $R$  este nevida.
- Din pacate, unele dintre expresiile scrise in calcul relational pe tupluri duc la rezultate infinite. De exemplu, daca  $R$  este o relatie finita, expresia  $\{t \mid R(t)\}$  este de asemenea finita, dar expresia  $\{t \mid \neg R(t)\}$  este infinita (exista o infinitate de tupluri care nu apartin lui  $R$ ).



# Calcul relational pe tupluri

- Pentru a evita astfel de rezultate s-a introdus notiunea de **expresii sigure**. Pentru definirea lor se poate folosi si un alt concept similar, numit **domeniul unei formule**:
- **Definitie:** Daca  $\Psi$  este o **formula** atunci domeniul sau, notat cu **DOM( $\Psi$ )**, este **multimea tuturor valorilor** care, fie apar explicit in  $\Psi$ , fie sunt componente ale tuplurilor relatiilor prezente in  $\Psi$ . Cum orice relatie este finita rezulta ca si domeniul oricarei formule este finit.



# Calcul relational pe tupluri

Exemplu:

Fie formula  $\Psi = R(t) \wedge t[1] > 100$

care reprezinta conditia pentru o selectie din  $R$  dupa conditia “valoarea pe prima coloana este mai mare decat 100”.

Atunci:

$\text{DOM}(\Psi) = \{ 100 \} \cup \{\text{multimea valorilor care apar in tuplurile lui } R\}$ .



# Calcul relational pe tupluri

- Se poate deci afirma ca o expresie  $\Psi$  este sigura daca rezultatul sau este compus doar din valori apartinand lui  $\text{DOM}(\Psi)$ . Conform acestei definitii:
  - expresiile  $\{t \mid R(t)\}$ ,  $\{t \mid R(t) \wedge t[1] > 100\}$ ,  
 $\{t \mid R(t) \vee S(t)\}$ ,  $\{t \mid R(t) \wedge S(t)\}$  sau  
 $\{t^{(k)} \mid (\exists u)(R(u) \wedge t[1] = u[i_1] \wedge t[2] = u[i_2] \wedge \dots \wedge t[k] = u[i_k])\}$  sunt sigure;
  - expresiile  $\{t \mid \neg R(t)\}$  sau  $\{t \mid \neg R(t) \wedge \neg S(t)\}$  nu sunt sigure.
- In literatura de specialitate se gaseste demonstratia faptului ca expresiile sigure din CRT sunt echivalente cu expresii din algebra relationala si reciproc.



# Calcul relational pe domenii

## ▪ **Calcul relational pe domenii**

In calculul relational pe domenii in locul variabilor tuplu folosim **variabile de domeniu** care sunt elementele ale tuplurilor. In acest caz rescriem regulile de formare pentru o formula astfel:

- Un **atom** poate fi:
  - $R(x_1, x_2, \dots, x_n)$  , unde R este o relatie iar  $x_i$  sunt variabile de domeniu sau constante;
  - $x \theta y$  , unde x si y sunt variabile de domeniu sau constante , iar  $\theta$  este in continuare un operator de comparatie.



# Calcul relational pe domenii

- Formulele din CRD sunt construite analog cu cele din CRT utilizand de asemenea  $\neg$ ,  $\wedge$ ,  $\vee$  precum si  $\exists$ ,  $\forall$ .
- Notiunea de aparitie libera sau legata a unei **variabile de domeniu** este analoaga cu cele din CRT.
- Analog cu CRT se definesc: **domeniul unei variabile de domeniu**  $\text{DOM}(x)$  si **expresii sigure** in CRD.
- In literatura de specialitate se poate gasi demonstratia faptului ca expresiile sigure din CRD sunt echivalente cu expresii din algebra relationala si reciproc.



# Calcul relational

Exemplu:

✓ Expresiile de mai jos sunt sigure:

- Reuniunea a două relații R și S:

$$\{x_1 x_2 \dots x_n \mid R(x_1 x_2 \dots x_n) \vee S(x_1 x_2 \dots x_n) \}$$

- Intersecția a două relații R și S:

$$\{x_1 x_2 \dots x_n \mid R(x_1 x_2 \dots x_n) \wedge S(x_1 x_2 \dots x_n) \}$$

- Selectia după condiția “valoarea pe prima coloană este mai mare decat 100”:

$$\{x_1 x_2 \dots x_n \mid R(x_1 x_2 \dots x_n) \wedge x_1 > 100 \}$$

✓ Analog cu CRT, expresiile de mai jos nu sunt sigure:

$$\{x_1 x_2 \dots x_n \mid \neg R(x_1 x_2 \dots x_n) \} \text{ si}$$

$$\{x_1 x_2 \dots x_n \mid \neg R(x_1 x_2 \dots x_n) \wedge \neg S(x_1 x_2 \dots x_n) \}$$



# Capitolul 4

## Modelul entitate-asociere



# Modelul Entitate-Asociere

- **Modelul Entitate-Asociere** (*Entity Relationship*) este un model de date folosit în proiectarea conceptuală de nivel înalt a bazelor de date și poate fi transformat în model relational prin aplicarea unor reguli specifice.
- Modelul EA se constituie într-o abordare grafică a proiectării bazelor de date și a fost adoptat de comunitatea științifică, precum și de producătorii de software din domeniu, datorită simplității și expresivității sale.
- Acest model a fost introdus de P. P. Chen în 1976 și a fost utilizat de multe sisteme CASE pentru proiectarea bazelor de date DB2, Oracle, SQL Server, Sybase, s.a.



# Modelul Entitate-Asociere

- Pana la aparitia unor astfel de metodologii se pornea de la o colectie de tabele si de la dependentele functionale sau multivalorice asociate si se ajungea, prin aplicarea unor algoritmi sau metode de normalizare, la schema dorita a bazei de date.
- Aceasta abordare de tip *bottom-up* creeaza insa dificultati in cazul bazelor de date complexe in care numarul de tabele si de dependente este mare.
- Probabilitatea ca unele interdependente intre date sa nu fie sesizate in procesul de proiectare este in aceste cazuri ridicata si pot duce la anomalii in exploatarea aplicatiilor.



# Modelul Entitate-Asociere

- Impactul conceptelor de abstractizare si generalizare precum si elaborarea unui model de descriere informala a datelor, prin modelul Entitate-Asociere (EA), au dus la gasirea unor cai moderne de proiectare a bazelor de date. Modelul EA este unul dintre cele mai utilizate modele datorita intuitivitatii si simplitatii elementelor sale. Im bunatatirile aduse ulterior prin folosirea abstractizarilor si generalizarilor au dus la crearea de variante ale modelului, doua dintre acestea fiind descrise in acest capitol.
- Extensiile modelului EA au aparut si pentru alte necesitati:
  - modelarea cerintelor de secretizare a datelor;
  - elaborarea documentatiei pentru aplicatiile cu baze de date si usurarea comunicarii intre dezvoltatorul de sistem si utilizator;
  - proiectarea bazelor de date complexe pe portiuni si integrarea ulterioara a acestora (asa numita integrare a vederilor).



# Modelul Entitate-Asociere

- Prin aplicarea modelului se obtine *diagrama entitate-asociere* care poate fi apoi translatata in modelul de date folosit de sistemul de gestiune a bazei de date.
- Cateva caracteristici ale modelului sunt urmatoarele:
  - Nu este legat direct de niciunul dintre modelele folosite de sistemele de gestiune a bazelor de date (relational sau orientat obiect) dar exista algoritmi bine pusi la punct de transformare din model EA in celealte modele de date;
  - Este intuitiv, rezultatul modelarii fiind o diagrama care defineste atat datele stocate in baza de date cat si interdependentele dintre acestea;
  - Proiectarea se poate face pe module, diagramele partiale rezultate putand fi apoi integrate pe baza unor algoritmi si metode bine puse la punct;
  - Este usor de inteles si interpretat.



# Elementele modelului Entitate-Asociere

- Modelul entitate-asociere permite reprezentarea informatiilor despre structura bazelor de date folosind trei elemente de constructie: **entitati**, **attribute** ale entitatilor si **asocieri** intre entitati.
- **Entitati**

**Entitatile** modeleaza clase de obiecte concrete sau abstracte despre care se colecteaza informatii, au existenta independenta si pot fi identificate in mod unic. Ele definesc de obicei obiecte sau evenimente cu importanta informationala. Membrii unei clase care formeaza o astfel de entitate poarta numele de **instante** ale acelei entitati.

Exemple de entitati: Studenti, Produse, Facturi, etc.



# Elementele modelului Entitate-Asociere

Entitatea este un obiect generic care reprezinta multimea tuturor instantelor sale si sunt de doua categorii:

- **Entitati independente** (sau tari) sunt cele care au existenta independenta de alte entitati.
- **Entitati dependente** (sau slabe) sunt formate din instante care isi justifica incadrarea in clasa respectiva doar atata timp cat intr-o alta entitate (tata) exista o anumita instanta de care sunt dependente.

Exemplu:

Daca consideram o baza de date universitara putem considera ca entitatea STUDENTI este entitate independenta iar entitatea NOTE entitate dependenta, deoarece notele unui student sunt dependente de existenta studentului in facultatea respectiva.



# Elementele modelului Entitate-Asociere

Element	Tip	Reprezentare	Exemplu
Entitate	Tare		
	Slaba		
Atribut	Identificare		
	Descriere		
Asociere	Asociere $n=2$ entitati	Nume asociere 	Inscris_la 
	Asociere $n > 2$ entitati	 $\dots$ 	Alocare 

Figura 1. Convenția de reprezentare a elementelor modelului EA



# Elementele modelului Entitate-Asociere

## ■ Atribute

**Atributele** modeleaza proprietati atomice distincte ale entitatilor. In procesul de modelare vor fi luate in considerare doar acele proprietati ale entitatilor care sunt semnificative pentru aplicatia respectiva.

Exemplu:

Entitatea STUDENTI poate avea ca atribute Matricol, Nume, CNP, Grupa, etc. Din acest motiv, la entitatea STUDENTI nu vom lua in considerare caracteristici cum ar fi Talia, nefiind necesara pentru baza de date a universitatii (astfel de atribute ar putea exista insa intr-o baza de date privind personalul militar).



# Elementele modelului Entitate-Asociere

Atributele unei entitati sunt de doua feluri:

- **Atribute de identificare** (formand impreuna identificatorul entitatii) - reprezinta acea multime de atribute care permite identificarea unica a instantelor unei entitati;
- **Atribute de descriere** (sau **descriptori**) - sunt folosite pentru specificarea informatiilor suplimentare ale instantelor.
- In cazul entitatii STUDENTI atributul Matricol este atribut de identificare (deoarece nu pot exista doi studenti cu acelasi numar matricol intr-o facultate) pe cand celelalte atribute sunt atribute de descriere.



# Elementele modelului Entitate-Asociere

## ■ **Asocieri**

**Asocierile** modeleaza interdependentele dintre clasele de obiecte reprezentate prin entitati.

De exemplu, intre entitatile STUDENTI si FACULTATI se poate figura o asociere Inscris\_la care descrie apartenenta studentilor pe facultati.

- In crearea diagramei nu vor fi luate in consideratie decat interdependentele care sunt necesare aplicatiei respective, in lumea reala putand exista intre entatile diagramei si alte asocieri care nu sunt semnificative in contextul dat.



# Elementele modelului Entitate-Asociere

- În construcția unei diagrame EA trebuie avute în vedere următoarele:
  - Entitatile se reprezinta prin dreptunghiuri în care este inscris numele entitatii. În cazul entitatilor dependente (slabe), conturul va fi cu linie dubla.
  - Atributele se reprezinta prin cercuri (sau ovale) în interiorul cărora apare numele atributului. Ele sunt conectate cu un segment de dreapta la entitatea de care aparțin. Pentru a distinge attributele de identificare de cele de descriere, numele primelor va fi subliniat.
  - Asocierile se reprezinta prin romburi (daca conecteaza una sau doua entitati) sau poligoane regulate (daca conecteaza mai mult de doua entitati) conectate prin segmente de dreapta la entitatile asociate, avand inscris în interior (sau alături) numele asocierii.



# Extensii ale modelului Entitate-Asociere

- Modelul entitate-asociere clasic are unele lipsuri în ceea ce privește posibilitatea modelării caracteristicilor asociate unor subclase de obiecte modelate prin simple entități.
- Pentru aceasta, la modelul original au fost adăugate două noi concepte: **iерархия de generalizare** și **iерархия de incluziune**. Prima definește partitionarea instantelor unei entități în **n** subclase diferite, iar a doua permite clasarea unor dintre instantele unei entități în **k** subclase care nu reprezintă o partitie în sens matematic.
- Din punct de vedere formal, cele două concepte se pot defini astfel:



# Extensii ale modelului Entitate-Asociere

## ■ Ierarhia de incluziune

**Definitie:** O entitate  $E_k$  este o submultime a entitatii  $E$  (sau este inclusa in entitatea  $E$ ) daca fiecare instanta a lui  $E_k$  este de asemenea o instanta a lui  $E$ .

- Un exemplu de incluziune este definirea in cadrul entitatii ANGAJATI a unor subclase modelate prin entitatile INGINERI, ECONOMISTI si COLABORATORI.
- In cazul ierarhiei de incluziune entitatile fiu pot sa nu fie disjuncte, de exemplu, pot exista angajati ingineri dar care sunt incadrati cu contract de colaborare. De asemenea, reuniunea lor poate sa nu acopere in intregime entitatea tata, de exemplu, exista angajati care nu sunt ingineri, economisti sau colaboratori.



# Extensii ale modelului Entitate-Asociere

## ■ Ierarhia de generalizare

**Definitie:** O entitate  $E$  este generalizarea entitatilor  $E_1, E_2, \dots, E_n$  daca orice instanta a lui  $E$  este de asemenea instanta in una, si numai una, dintre entitatile  $E_1, E_2, \dots, E_n$ .

- Un exemplu de generalizare este clasarea instantelor entitatii ANGAJATI in subclasele BARBATI si FEMEI.
- O caracteristica a ierarhiei de generalizare este ca din punct de vedere matematic entitatile *fiu* reprezinta o partitie a entitatii *tata*:
  - $E_1 \cup E_2 \cup \dots \cup E_n = E$  si
  - $E_i \cap E_j = \emptyset$  pentru orice  $i \neq j$  din intervalul 1..n
- Ierarhiile de inclusiune si generalizare se folosesc doar in cazul in care, pentru subclasele unor clase modelate prin entitati, este nevoie de stocarea unor informatii suplimentare specifice.



# Extensii ale modelului Entitate-Asociere

Element	Reprezentare	Exemplu
Ierarhie de incluziune	<pre>graph TD; E[E] --&gt; E1[E1]; E --&gt; E2[E2]; E --&gt; E3[E3]</pre>	<pre>graph TD; Angajati[Angajati] --&gt; Economisti[Economisti]; Angajati --&gt; Ingineri[Ingineri]; Angajati --&gt; Colaboratori[Colaboratori]</pre>
Ierarhie de generalizare	<pre>graph TD; E[E] --&gt; Criteriu{Criteriu}; E1[E1] --&gt; Criteriu; E2[E2] --&gt; Criteriu</pre>	<pre>graph TD; Angajati[Angajati] --&gt; Sex{Sex}; Barbati[Barbati] --&gt; Sex; Femei[Femei] --&gt; Sex</pre>

Figura 2. Convenția grafică de reprezentare grafică a ierarhiilor



# Extensiile ale modelului Entitate-Asociere

Exemplu:

- În cazul unei baze de date de personal este nevoie să fie specificat numărul de copii ai fiecarui angajat. Acest fapt se poate modela în două feluri: fie prin adăugarea la entitatea ANGAJATI a unui atribut suplimentar Numar\_copii (care va avea valoarea 0 pentru angajatii fără copii), sau prin crearea unei entități suplimentare COPII aflată într-o relație de incluziune cu entitatea ANGAJATI, care va avea ca atribute de identificare pe cele ale angajatului iar ca atribut descriptiv numărul de copii, acesta fiind atributul specific subclasei.
- Vom alege a doua variantă în cazul în care nu avem nevoie de informații suplimentare despre copii(nume, CNP, etc.) sau numărul angajatilor care au date specifice aceluia atribut este mult mai mic decât numărul total al angajatilor(de exemplu angajati absolvenți a două facultăți), fapt care va duce la economie de spațiu pe disc. În acest caz o nouă entitate(o nouă tabelă a bazei de date) va ocupa mai puțin spațiu decât un atribut suplimentar (o coloană suplimentară) a tabelei ANGAJATI.



# Caracteristici ale elementelor modelului

Așa cum entitatile au atribută care specifică diverse proprietăți ale clasei de obiecte modelate, și asociările au caracteristici care aduc informații suplimentare.

Acestea sunt următoarele:

- **Gradul asocierii**

Este o valoare numerică întreagă și este dat de numarul de entități care participă la acea asociere.

Asociările de grad 1, 2 și 3 se mai numesc și asocieri **unare**, **binare** și respectiv **ternare**.

Exemplu:

Vom considera o bază de date continând informații despre studenți, proiectele realizate de acestia, calculatoarele pe care au alocate ore de lucru și facultățile la care sunt înscrise. De asemenea vom considera că unii dintre studenți au un **tutor** care îi îndrumă, acesta fiind un student dintr-un an mai mare.



# Caracteristici ale elementelor modelului

- Diagrama EA a bazei de date este prezentata in Figura 3. Pentru simplificarea figurii nu s-au reprezentat decat entitatile si asocierile dintre ele, nu si atributele fiecarei entitati in parte.

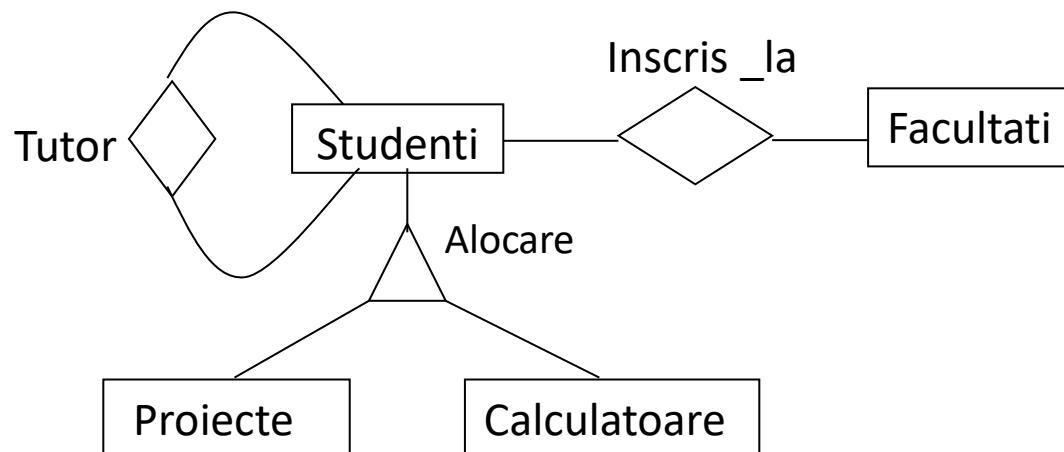


Figura 3. Exemple de asocieri de grad 1, 2 si 3



# Caracteristici ale elementelor modelului

- Asocierea **Tutor** este o asociere unara deoarece la ea participa doar entitatea STUDENTI. Aceasta asociere arata cine este tutorul fiecarui student (daca exista).
- Asocierea **Inscris\_la** este o asociere binara intre entitatile STUDENTI si FACULTATI si arata la ce facultate/facultati este inscris fiecare student.
- Asocierea **Alocare** este o asociere ternara intre entitatile STUDENTI, PROIECTE si CALCULATOARE. Ea modeleaza pe ce calculatoare are alocate ore de lucru fiecare student pentru fiecare proiect.



# Caracteristici ale elementelor modelului

Un exemplu de asociere de grad mai mare ca trei este orarul unui an de studiu al unei facultati. Aceasta este o asociere intre urmatoarele entitati:

- GRUPE - Fiecare grupa are un cod unic.
- SALI - Salile sunt etichetate printr-un indicativ alfanumeric.
- ORE - Un interval orar este un triplet (Zi, De la ora, La ora).
- ACTIVITATE - Este o activitate prezenta in orar (curs, laborator, seminar, proiect).
- PROFESOR - Este cadrul didactic titular pentru o activitate.



# Caracteristici ale elementelor modelului

- Modelarea acestor clase de obiecte ca entitati presupune faptul ca in baza de date sunt stocate si alte informatii despre ele. Diagrama EA este prezentata in Figura 4. Ea modeleaza programarea activitatilor didactice efectuate de profesori pe intervale orare, sali si grupe.

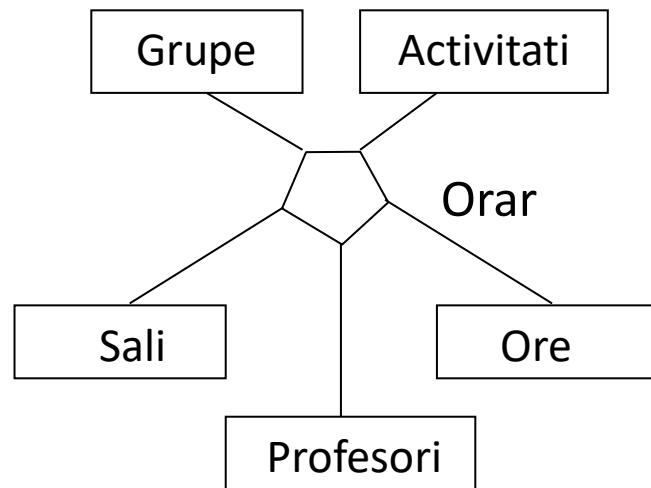


Figura 4. Asociere de grad 5



# Caracteristici ale elementelor modelului

## ■ Conectivitatea asocierii

**Conectivitatea** este specifică fiecărei ramuri a unei asocieri și poate avea una dintre urmatoarele două valori: **unu** sau **multi**. Determinarea ei se face astfel:

- Se alege o entitate **E**;
- Se fixează o entitate **F** care participă la asociere;
- Se determină cu cate instante din entitatea **F** se poate conecta o instantă din **E** ;
- Dacă poate fi cel mult una, conectivitatea ramurii este **unu**, altfel conectivitatea este **multi**.
- Pentru stabilirea conectivității verificarea se face în ambele sensuri între **E** și **F** .



# Caracteristici ale elementelor modelului

Convenții de reprezentare a conectivității:

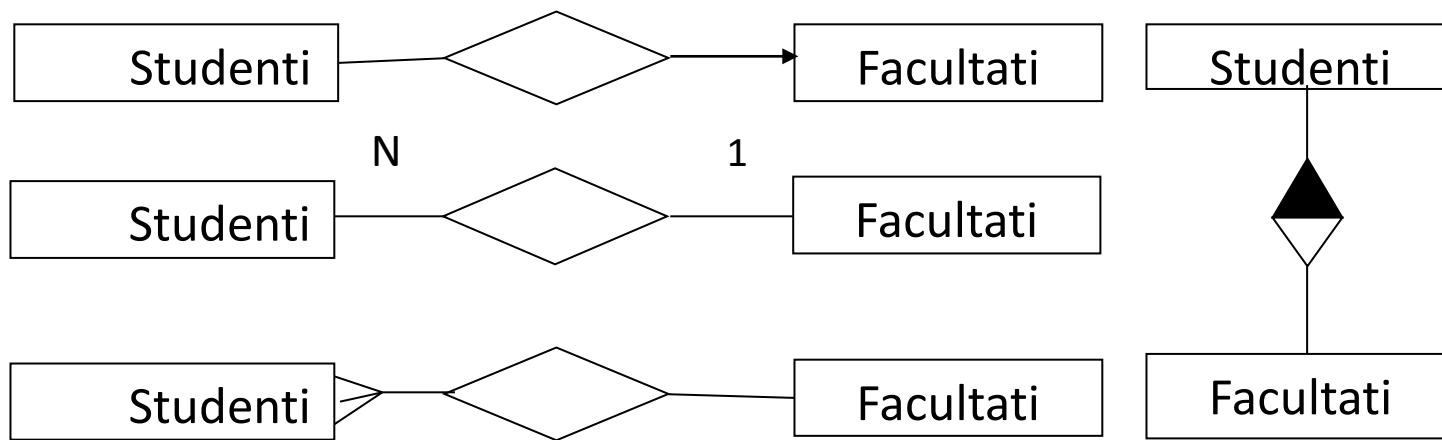


Figura 5. Exemple de reprezentare a conectivitatii



# Caracteristici ale elementelor modelului

Exemplu:

Pentru determinarea conectivitatii trebuie stabilite mai intai cerintele de functionare si modelare.

Revenim la exemplul din Figura 3:

- Asocierea **Tutor** este **unu-unu** sau **multi-unu** dupa cum un student poate fi tutor pentru un singur student sau pentru mai multi studenti de an inferior.
- Asocierea **Inscris\_la** este **multi-unu** (multi spre STUDENTI) sau **multi-multi** dupa cum un student poate fi inscris la una sau mai multe facultati.
- Asocierea ternara **Alocare** (aplicam definitia):



# Caracteristici ale elementelor modelului

- Ramura spre STUDENTI: fiind dat un proiect si un calculator, cati studenti au ore alocate pe acel calculator pentru respectivul proiect? Considerand ca mai multi studenti lucreaza pentru acelasi proiect, pe acelasi calculator, ramura va fi **multi**;
- Ramura spre PROIECTE: fiind dat un student si un calculator, la cate proiecte are acesta alocate ore pe acel calculator? Considerand ca pentru fiecare proiect exista un calculator dedicat, ramura va fi **unu**;
- Ramura spre CALCULATOARE: fiind dat un student si un proiect, pe cate calculatoare are alocate acesta ore pentru realizarea proiectului? Considerand ca la un proiect se lucreaza pe un singur calculator, ramura va fi **unu**.
- Deci asocierea **Alocare** este **multi-unu-unu**.



# Caracteristici ale elementelor modelului

- Observam ca raspunsul la fiecare dintre cele trei intrebari se da in functie de realitatea modelata. Aceeasi asociere poate avea conectivitati diferite in cazuri diferite: daca exista chiar si un singur proiect la care un student are ore alocate pe mai mult de un calculator, ramura spre CALCULATOARE va fi **multi** iar asocierea va fi **multi-unu-multi**.
- Conventia de reprezentare grafica a conectivitatii folosita in aceasta prezentare este urmatoarea: ramurile **unu** vor fi reprezentate cu sageata, iar ramurile **multi** fara segeata.
- In Figura 6 sunt prezentate cele trei asociieri avand figurata si conectivitatea. S-a presupus ca asocierea **Tutor** este **unu-unu** (un student are un singur tutor) iar **Inscris\_la** este **multi-unu** (mai multi studenti sunt inscrisi la o facultate).



# Caracteristici ale elementelor modelului

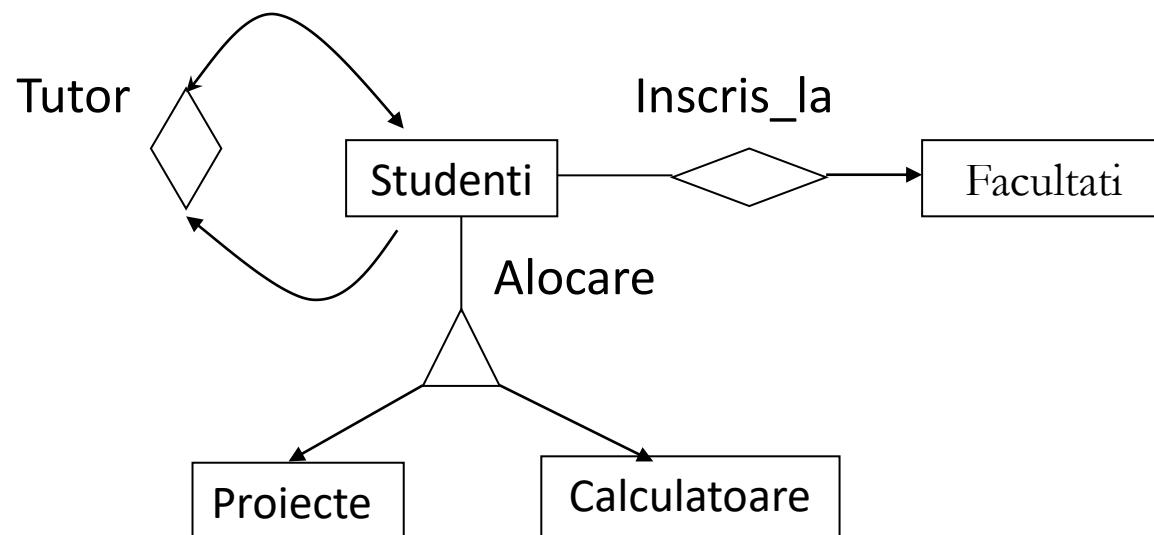


Figura 6. Reprezentarea conectivitatii



# Caracteristici ale elementelor modelului

## ■ Obligativitatea asocierii

Ca si conectivitatea, **obligativitatea** se determina pentru fiecare ramura si poate avea doar una dintre urmatoarele valori: **obigatorie** sau **optionala**.

Determinarea ei se face astfel:

- Se alege ramura spre o entitate **E**;
- Se fixeaza o entitate **F** care participa la asociere;
- Este obligatoriu sa existe o instanta a lui **E** asociata cu o instanta din **F**?
- Daca raspunsul este DA ramura este obligatorie, altfel este optionala.



# Caracteristici ale elementelor modelului

Exemplu:

In exemplul anterior ramurile asocierilor **Tutor** si **Alocare** sunt optionale iar cele ale asocierii **Inscris\_la** sunt obligatorii deoarece:

- Pentru asocierea **Tutor**:
  - exista studenti care nu au un tutor si nici nu sunt tutori pentru alti studenti;
- Pentru asocierea **Alocare**:
  - un student poate sa nu aiba alocate ore pe niciun calculator la un proiect (de exemplu in cazul unui proiect la o materie umanista);
  - un student si un calculator, respectiv un calculator si un proiect, pot sa nu fie asociati prin alocare de ore de lucru (de exemplu pentru calculatoarele din birourile cadrelor didactice).
- Pentru asocierea **Inscris\_la**:
  - nu exista studenti care nu sunt inscrisi la nicio facultate si nici facultati fara studenti inscrisi.



# Caracteristici ale elementelor modelului

- Convenția de reprezentare grafică a clasei de apartenență folosită în continuare este urmatoarea: ramurile **obligatorii** vor fi reprezentate prin linie continuă iar cele **optionale** prin linie interrupță. În Figura 7 este reprezentată obligativitatea.

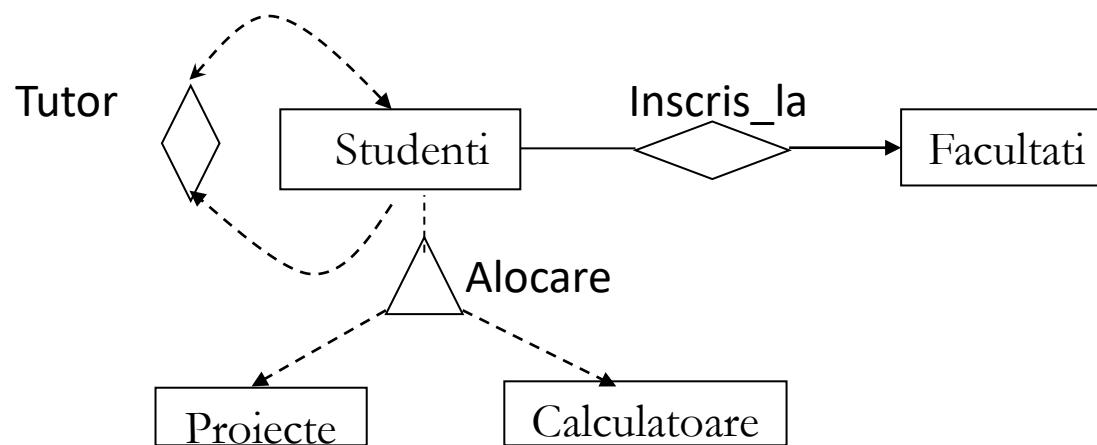


Figura 7. Reprezentarea obligativitatii



# Caracteristici ale elementelor modelului

- Daca gradul si conectivitatea unei asocieri sunt folosite in proiectarea conceptuala a schemei bazei de date, obligativitatea se modeleaza pentru definirea unui criteriu de integritate specificand posibilitatea de aparitie a valorilor nule.
- La transformarea diagramei EA in model relational atributele tabelelor care modeleaza informatia reprezentata de asocieri pot avea sau nu valori nule, dupa cum ramurile acestora sunt optionale sau obligatorii.



# Caracteristici ale elementelor modelului

## ■ Atributele asociierilor

In unele cazuri o anumita informatie descriptiva este asociata cu un ansamblu de clase diferite, nu cu o singura clasa de obiecte, modelate fiecare prin entitati. In acest caz aceasta va fi modelata ca un **atribut al asocierii** dintre entitatile respective.

Exemplu:

Sa luam cazul unei asociieri **A\_absolvit** de tip **multi-multi** intre entitatile STUDENTI si FACULTATI care contine informatii privind facultatile absolvite anterior de unii studenti, asa cum este prezentata in Figura 8.



# Caracteristici ale elementelor modelului

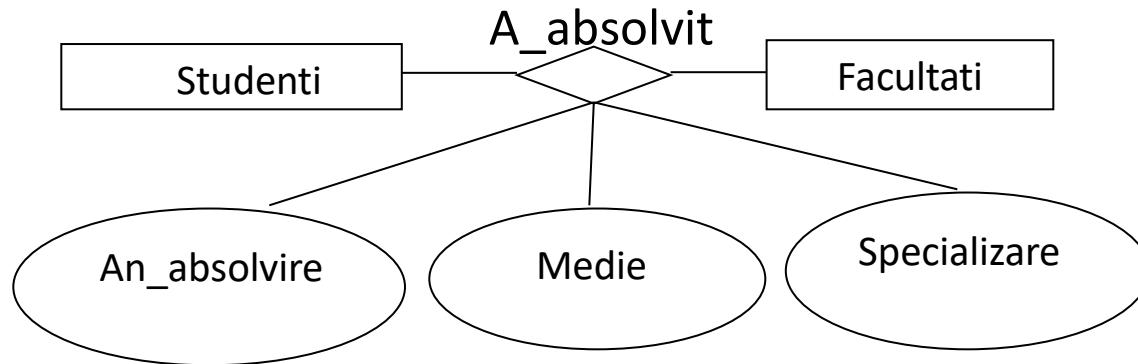


Figura 8. Reprezentarea atributelor asociierilor

- In acest caz informatii ca anul absolvirii, media, specializarea nu pot fi conectate nici la STUDENTI (pentru ca un student poate fi absolventul mai multor facultati in ani diferiti, cu medii diferite, etc.) si din motive similare nici la FACULTATI. Ele descriu asocierea unui student cu o facultate si de aceea vor fi atasate asocierii **A\_absolvit**. Toate atributele unei asocieri sunt attribute descriptive, neexistand in acest caz un identificator al asocierii.



# Caracteristici ale elementelor modelului

## ■ Rolul

In cazul in care de la o asociere pornesc mai multe ramuri catre aceeasi entitate, fiecareia dintre acestea i se poate asocia un **rol**. Acesta arata semnificatiile diferite pe care le are aceeasi entitate in cadrul asocierii respective, asa cum se vede in Figura 9.

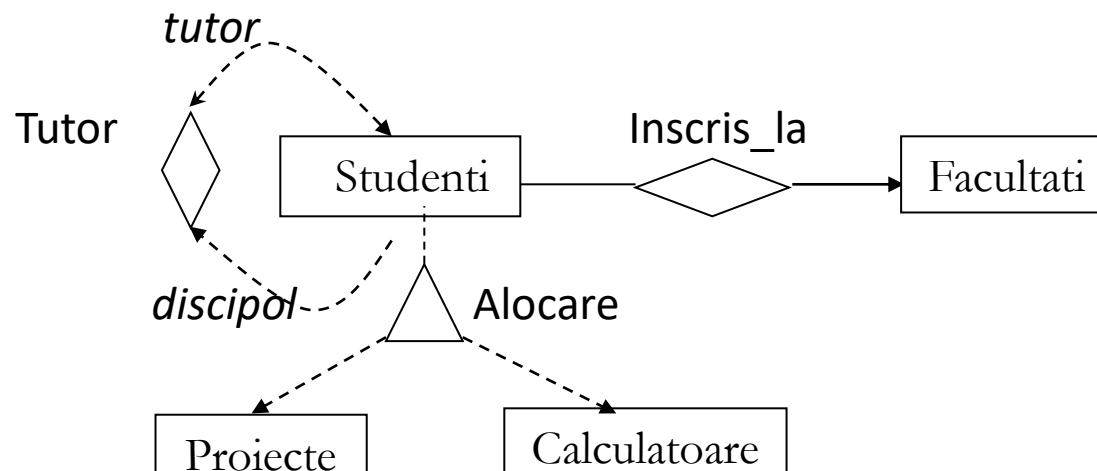


Figura 9. Reprezentarea obligativitatii si a rolurilor

- In cazul asocierii **Tutor** cele doua ramuri pot fi etichetate, de exemplu, cu **tutor** si **discipol**, aratand ca instante diferite ale aceleiasi entitati au roluri diferite.



# Criterii de modelare

## ■ Clasificarea in entitati si attribute

Desi definitia notiunilor de entitate, atribut, asociere este destul de simpla, in practica modelarii apar dificultati in clasificarea diverselor informatii intr-una din aceste categorii. De exemplu, in cazul sediilor unei banchi localizate in diverse orase: obiectul ORASE este entitate distincta sau atribut descriptiv al entitatii SEDII ?

- Pentru a putea clasifica corect informatiile, exista cateva reguli care trebuie respectate si pe care le presentam in continuare.
- Prima regula da un criteriu general de impartire in entitati si attribute, urmatoarele doua semnaleaza exceptii iar ultimele doua reguli au un caracter mai degraba orientativ si mai putin normativ.



# Criterii de modelare

- ***Regula 1.*** Entitatile au informatii descriptive, pe cand atributele nu poseda astfel de informatii.

Daca exista informatii descriptive despre o anumita clasa de obiecte, aceasta va fi modelata ca o entitate.

In cazul in care pentru o clasa de obiecte nu este nevoie decat de un identificator (codul, denumirea, etc), ea va fi modelata ca un atribut.

De exemplu, daca despre un oras este necesara stocarea in baza de date unor informatii ca judet, codul strazilor, numar locuitori, etc. atunci ORASE va fi o entitate. Daca singura informatie necesara este numele atunci Nume\_oras va fi un atribut al altei entitati.



# Criterii de modelare

- **Regula 2. Atributele multivalorice vor fi reclasificate ca entitati.**

Daca la o valoare a unui **identificator** corespund mai multe valori ale unui **descriptor**, atunci descriptorul va fi clasat ca entitate. De exemplu, in cazul unei baze de date privind localizarea in teritoriu a unor banchi, daca se stocheaza date doar despre banci care au un singur sediu, **Localitate** este atribut al entitatii BANCI, asa cum este prezentat in Figura 10.

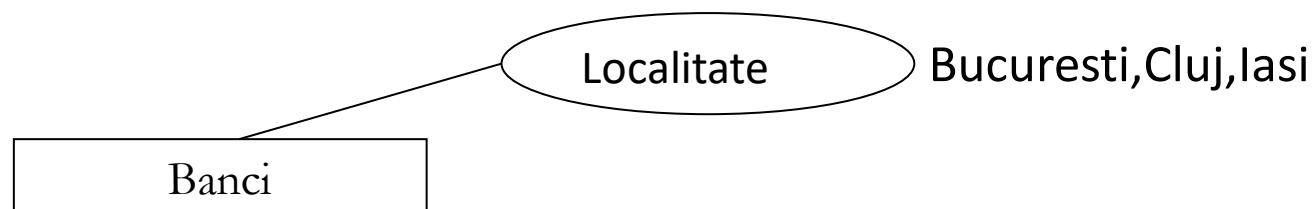


Figura 10. Reprezentarea atributelor multivalorice



# Criterii de modelare

- Daca insa se stocheaza date despre banci care au sucursale si filiale in diverse localitati, deci pentru o singura banca (o valoare a identificatorului entitatii BANCI) avem mai multe localitati in care aceasta are sedii (mai multe valori ale descriptorului Localitate), atunci **LOCALITATI** va fi entitate distincta desi are numai un singur atribut. Pentru a modela localizarea sediilor in diverse localitati intre cele doua entitati va exista o asociere binara unu-multi (unu spre BANCI) numita de exemplu **Are\_sediul\_in**, asa cum se vede in Figura 11.

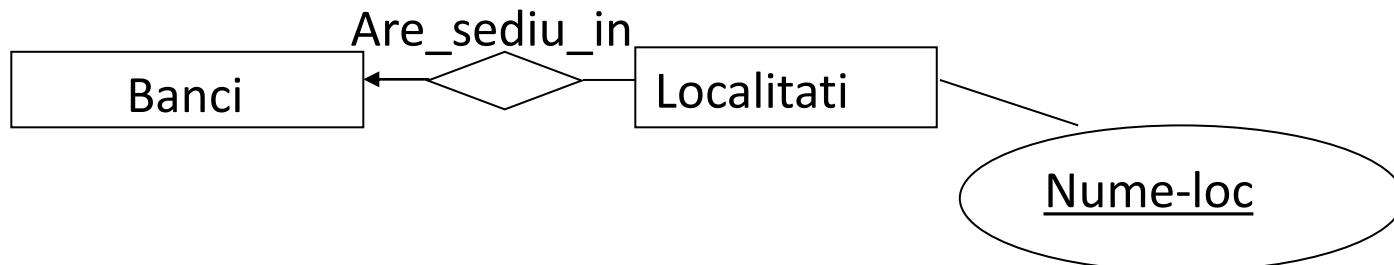


Figura 11. Reprezentarea atributelor multivalorice reclasificate ca entitati



# Criterii de modelare

- **Regula 3.** Atributele unei entitati care au o asociere multi-unu cu o alta entitate vor fi reclasificate ca entitati.

Asa cum am vazut, asocierile pot lega doar entitati. Daca un descriptor al unei entitati este intr-o relatie multi-unu cu o alta entitate, acel descriptor va fi trecut in categoria entitatilor.

De exemplu, daca avem entitatile BANCI avand ca atribut descriptiv monovaloric Localitate si Judet, daca se doreste modelarea apartenentei la judete a localitatilor va exista o asociere multi-unu intre atributul Localitate si entitatea JUDETE, reprezentata in Figura 12.

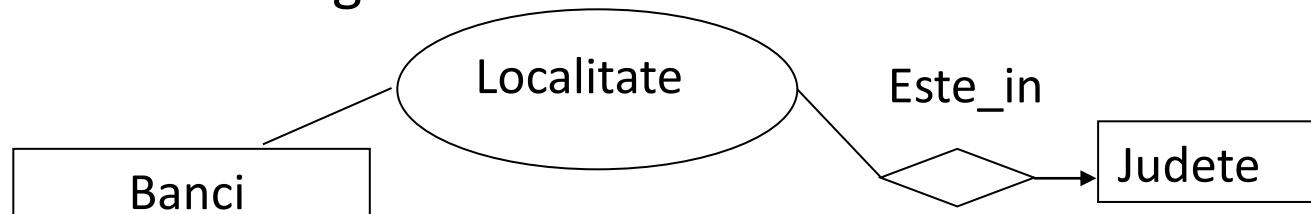


Figura 12. Reprezentarea atributelor multivalorice multi-unu



# Criterii de modelare

- În acest caz atributul Localitate va fi reclasificat ca entitatea LOCALITATI desi nu sunt necesare alte informații în afara numelui localității, ca în Figura 13.

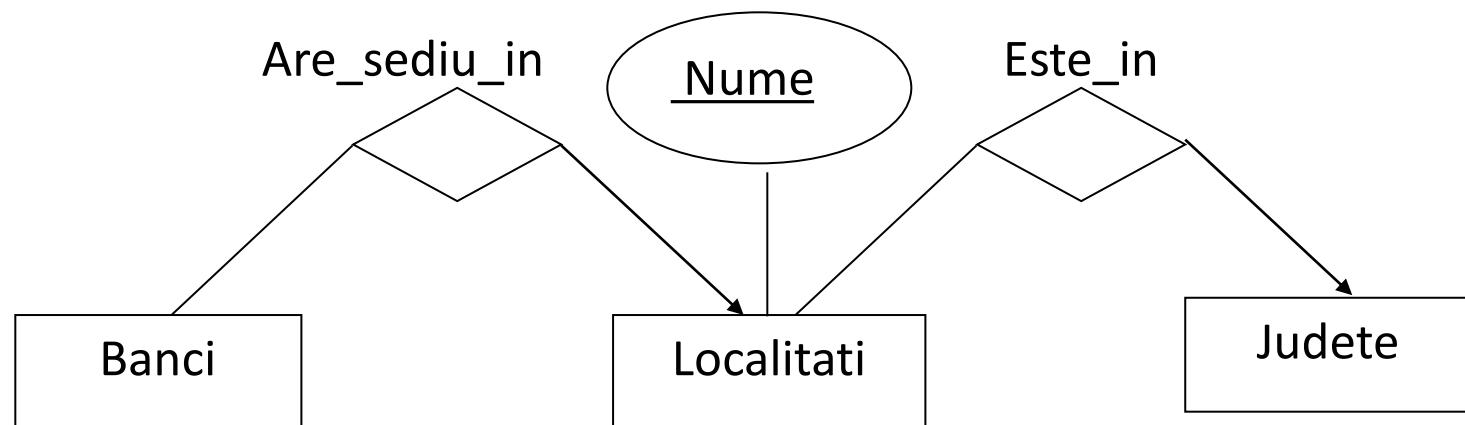


Figura 13. Reprezentarea atributelor multivalorice multi-unu reclasificate



# Criterii de modelare

- ***Regula 4.*** Atributele vor fi atasate la entitatile pe care le descriu in mod nemijlocit. De exemplu, Universitate va fi atasat ca atribut al entitatii FACULTATI si nu al entitatilor STUDENTI sau PROFESORI.
- ***Regula 5.*** Folosirea identificatorilor compusi va fi evitata, pe cat posibil. Identificatorul unei entitati este acea submultime de atribute ale acesteia care identifica in mod unic fiecare instanta a sa. In modelul relational pentru atributele de acest fel se construiesc, de regula, structuri de cautare rapida (indecsi) care functioneaza cu atat mai lent cu cat complexitatea index-ului creste.  
Aplicarea acestei reguli se poate face in diverse moduri:



# Criterii de modelare

- Daca identificatorul unei entitati este compus din mai multe attribute care sunt toate identificatori in alte entitati, acea entitate se elimina. Informatia continuta de aceasta va fi modelata sub forma unei asocieri intre acele entitati.
- Daca identificatorul unei entitati este compus din mai multe attribute care nu sunt toate identificatori in alte entitati, exista doua solutii:
  - Entitatea respectiva se elimina si este inlocuita prin alte entitati si asocieri astfel incat per ansamblu informatia modelata in varianta initiala sa fie pastrata.
  - Entitatea respectiva ramane in forma initiala, cu dezavantaje insa in privinta vitezei tranzactiilor pe baza de date.



## Criterii de modelare

- Se observa ca procedura clasificarii obiectelor in entitati si atribute este iterativa:
  - ✓ Se face o prima impartire conform primei reguli;
  - ✓ O parte dintre atributele astfel obtinute se reclasifica in entitati conform regulilor 2 si 3;
  - ✓ Se face o rafinare finala conform regulilor 4 si 5.



# Criterii de modelare

- **Identificarea ierarhiilor de generalizare si incluziune**  
In cazul in care despre anumite subclase ale unei clase de obiecte exista informatii specifice, clasa si subclasele (care la pasul anterior au fost catalogate ca entitati) sunt interconectate intr-o ierarhie de incluziune sau generalizare, dupa cum este cazul.
- La acest pas se face si o reatasare a atributelor pentru evitarea redundantei, astfel:
  - La entitatea *tata* vor fi atasate atributele care formeaza identificatorul si descriptorii care modeleaza informatii specifice intregii clase.
  - La entatile *fiu* vor fi atasate atributele de identificare (aceleasi ca ale tatalui) plus atributele care modeleaza informatii specifice doar acelei subclase de obiecte.



# Criterii de modelare

Exemplu:

Sa consideram o ierarhie care imparte studentii unei facultati in doua subclase:

- caministi – daca locuiesc in campusul universitar;
- necaministi – daca nu locuiesc in campus.

Diagrama entitate-asociere si ierarhiile de asociere sunt prezentate in Figura 14.



# Criterii de modelare

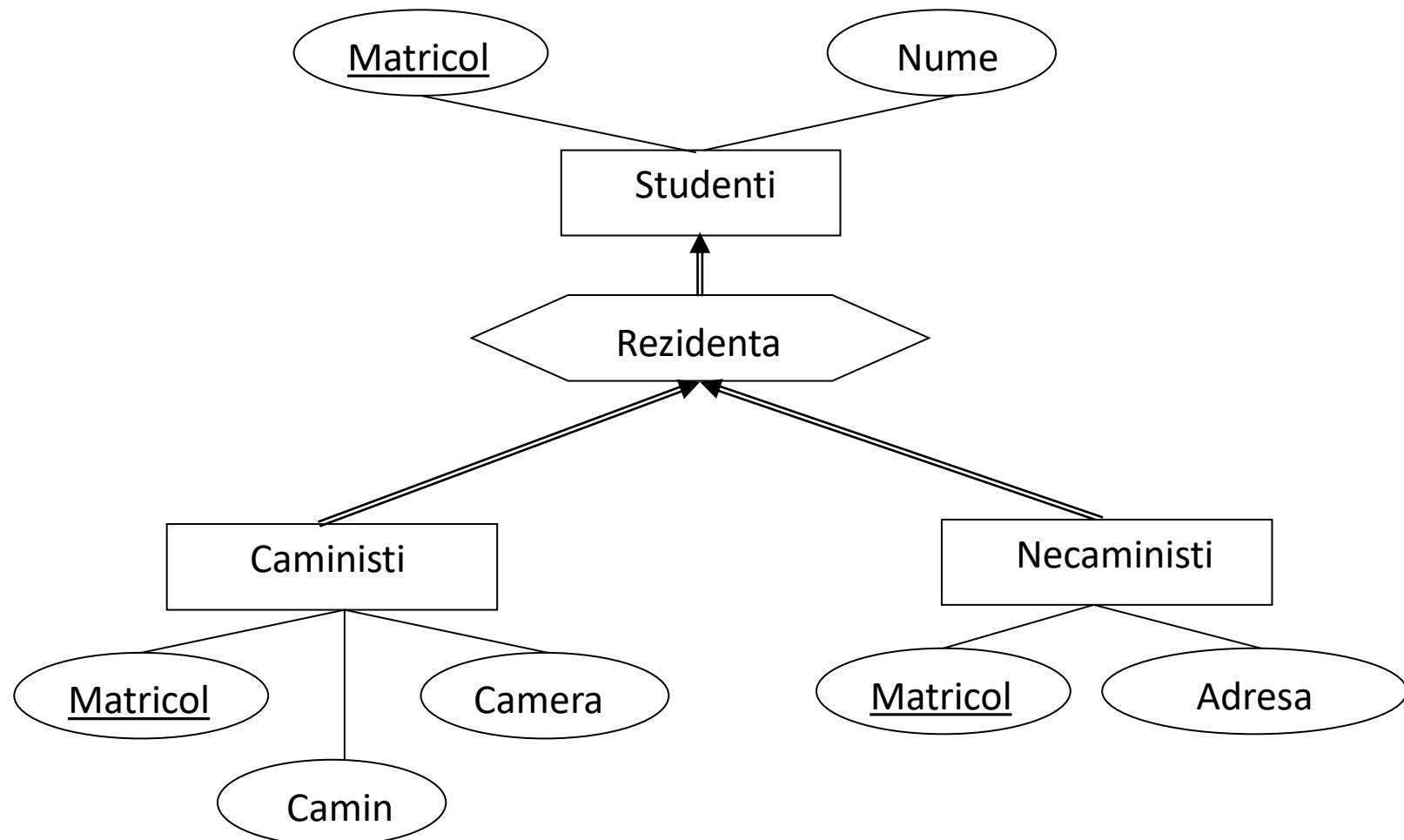


Figura 14. Atributele entitatilor unei ierarhii



# Criterii de modelare

- În acest caz atributul Nume trebuie eliminat de la entitatile fiu CAMINISTI și NECAMINISTI deoarece el este prezent deja la entitatea tata STUDENTI.
- Rezulta urmatoarele reguli:
  - Identifierul tata din cadrul unei ierarhii se regaseste in identifierul tuturor fiilor ierarhiei;
  - Descriptorii care apar si la tata si la fii, se elimina de la fii;
  - Descriptorii care apar la toti fiile unei ierarhii de generalizare, dar nu apar la tata, se muta la tata.



# Criterii de modelare

## ■ Identificarea asocierilor

In aceasta etapa se trateaza informatiile care nu au fost clasificate ca entitati sau atribute ci reprezinta interdependente intre clase de obiecte. Ele sunt modelate ca **asocieri intre entitati.** Pentru fiecare asociere se specifica gradul, conectivitatea, obligativitatea si daca este cazul si atributele asocierii precum si rolurile ramurilor sale.

- Ca si in cazul clasificarii in entitati si atribute, exista cateva reguli de urmat in operatia de definire a asocierilor:



# Criterii de modelare

## ■ Eliminarea asocierilor redundante

In cazul in care o asociere poate fi dedusa din alte asocieri deja catalogate, aceasta se elimina. De retinut ca intre doua entitati pot sa existe oricate asocieri si ele nu sunt considerate redundante atata timp cat au semnificatie diferita.

- Un caz des intalnit de redundanta este cel al compunerii (tranzitivitatii) asocierilor. Prezentam in Figura 15 un exemplu:

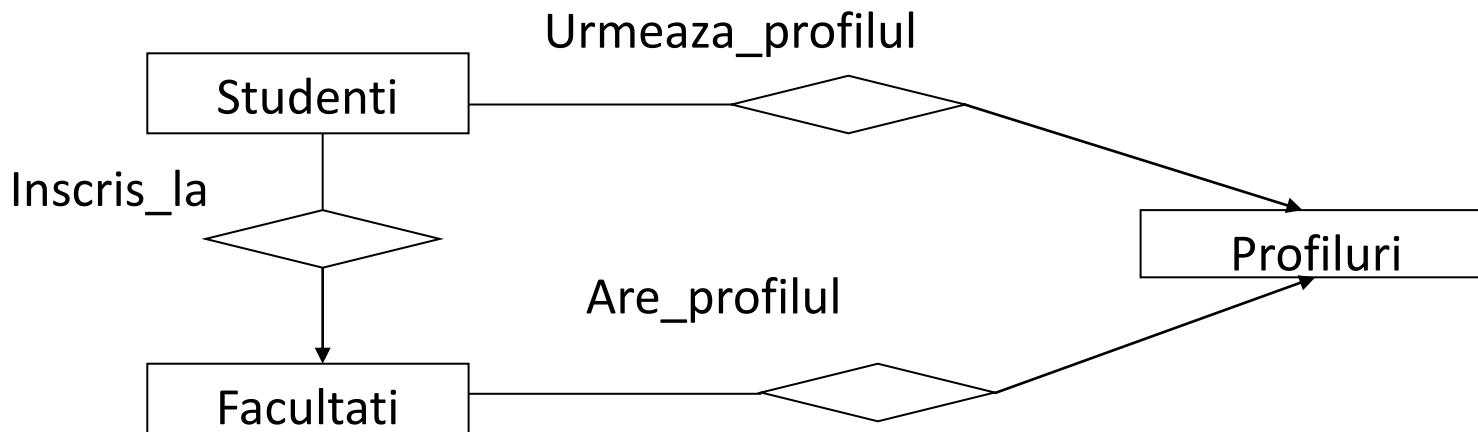


Figura 15. Asocieri redundante



# Criterii de modelare

- În acest exemplu, asocierea **Inscris\_la** modelează apartenența fiecarui student la o facultate a unei universități. Fiecare facultate are un profil unic descris de asocierea **Are\_profilul** (de ex. Electric, Mecanic, Chimic, etc.). Ambele asocieri sunt multi-unu în sensul STUDENTI → FACULTATI → PROFILURI.
- Deoarece asocierile multi-unu (ca și cele unu-unu) sunt din punct de vedere matematic funcții, din compunerea asocierilor putem afla profilul la care este înscris fiecare student.
- Rezulta că asocierea **Urmeaza\_profilul** care are chiar aceasta semnificație este redundanta si trebuie eliminata.



# Criterii de modelare

## ■ Asocieri de grad mai mare ca doi

Asocierile ternare (sau de grad mai mare ca trei) se folosesc doar atunci cand sunt strict necesare.

- Este de multe ori posibil ca o aceeasi informatie sa fie modelata ca o asociere ternara sau ca un ansamblu de asocieri binare si unare. In cazul acesta, este de preferat ca sa se opteze pentru a doua varianta, asa cum se vede in Figura 16 si Figura 17.
- Doar cand asocierile binare nu pot modela intreaga semnificatie dorita se va opta pentru asocieri de grad mai mare decat doi. Aceasta cerinta deriva din faptul ca la trecerea in modelul relational asocierile de grad superior devin scheme de relatii de sine statatoare, marind numarul de tabele din baza de date pe cand cele de grad unu si doi (cu exceptia celor multi-multi) nu au acest efect.



# Criterii de modelare

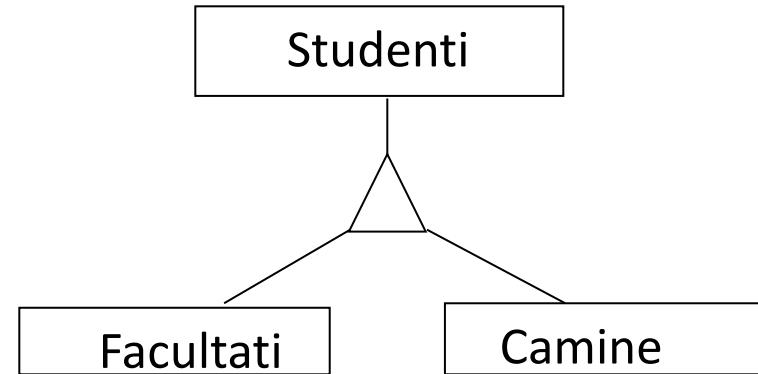


Figura 16. Exemplu de asociere ternara

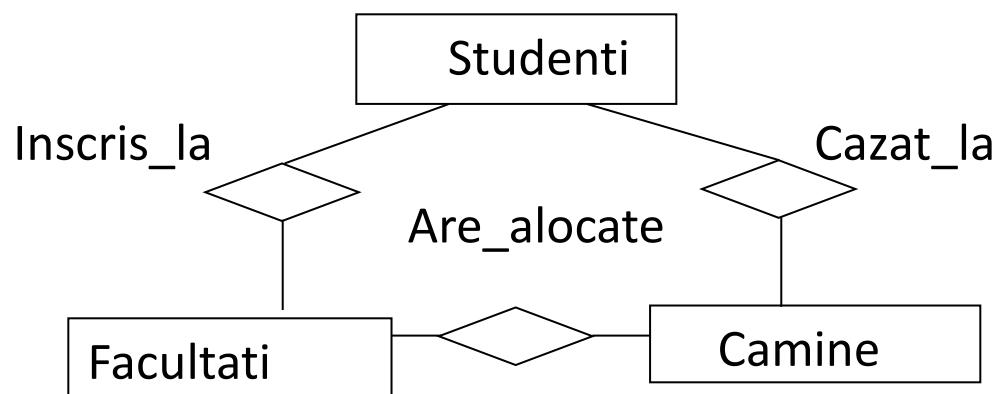


Figura 17. Exemplu de asociere ternara remodelata



# Criterii de modelare

## ■ Integrarea vederilor

In cazul proiectarii bazelor de date complexe, activitatea se desfasoara uneori de catre mai multe colective simultan, fiecare modeland o portiune distincta a bazei de date.

Deoarece in final trebuie sa se obtina o singura diagrama a bazei de date, dupa terminarea modelarii pe portiuni diagramele rezultate sunt integrate, eliminandu-se redundantele si inconsistentele.



# Transformarea diagramelor Entitate-Asociere in Modelul Relational

- In **procesul de transformare** vom pleca de la o diagrama E-A si vom obtine trei tipuri de scheme de relatie in M-R:
  - **Relatii provenite din entitati.** Ele contin aceleasi informatii ca si entitatile din care au rezultat.
  - **Relatii provenite din entitati si care contin chei straine.** Ele contin pe langa informatiile provenite din entitatile din care au rezultat si atributte care in alte entitati sunt identificatori. Este cazul acelor entitati care au asocieri multi-unu si partial din cele care au asocieri unu-unu cu alte entitati.
  - **Relatii provenite din asociieri.** Este cazul celor care apar din transformarea asocierilor binare multi-multi si a asocierilor de grad mai mare ca doi. Ele contin ca atributte reunirea identificatorilor entitatilor asociate si atributtele proprii ale asocierilor.

Procesul de transformare are un algoritm foarte precis si este din acest motiv etapa care se preteaza cel mai bine pentru crearea de instrumente software CASE care sa-l asiste.



# Transformarea diagramelor E-A in M-R

## ■ Transformarea entitatilor

Fiecare entitate a diagramei se transforma intr-o schema de relatie avand:

- **Numele relatiei** = Numele entitatii
- **Atributele relatiei** = Atributele entitatii
- **Cheia relatiei** = Identifierul entitatii

Exemplu:

Fie entitatea ANGAJATI din Figura 18:

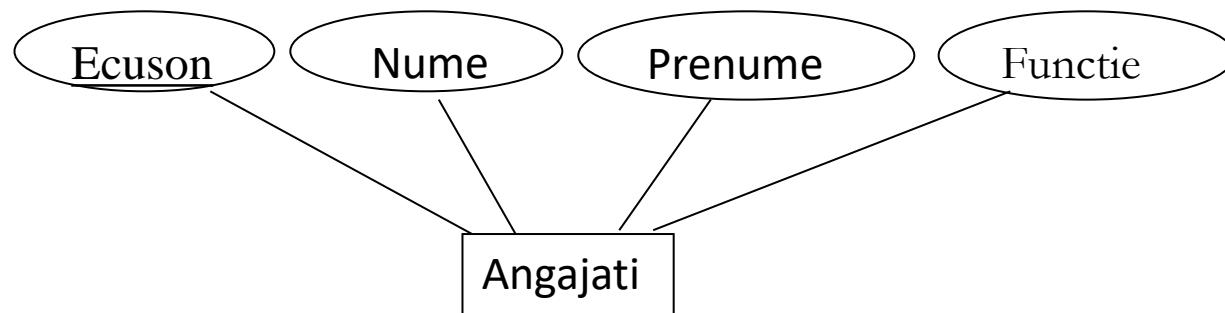


Figura 18. Exemplu de schema de relatie

Schema rezultata este ANGAJATI (Ecuson, Nume, Prenume, Functie).



# Transformarea diagramelor E-A in M-R

- **Transformarea asocierilor unare si binare unu-unu si multi-unu**  
Fiecare asociere din aceasta categorie va avea ca rezultat imbogatirea multimii de atribute descriptive ale uneia dintre cele doua scheme rezultate din entitatile asociate, cu cheia celeilalte scheme. Aceste atribute care se adauga sunt denumite in prezentarea de fata cheie straina (sau cheie externa) deoarece ele sunt cheie dar in alta schema de relatie.  
• In cazul asocierilor **multi-unu**, se adauga identificatorul entitatii unu in schema rezultata din entitatea multi.  
• In cazul asocierilor **unu-unu**, se adauga identificatorul unei entitati in schema rezultata din transformarea celeilalte. Alegerea schemei in care se face adaugarea se poate face dupa doua criterii:
  - fie in acea schema care defineste relatia cu cele mai putine tupluri dintre cele doua;
  - fie pastrandu-se, daca exista, filiatia naturala intre cele doua entitati: identificatorul tatalui se adauga la fiu.



# Transformarea diagramelor E-A in M-R

- In cazul acestui tip de asociere, la schema de relatie care primeste cheia strina se ataseaza una sau doua dependente functionale primare, conform Tabelului 1.

Un exemplu este prezentat in Figura 19.

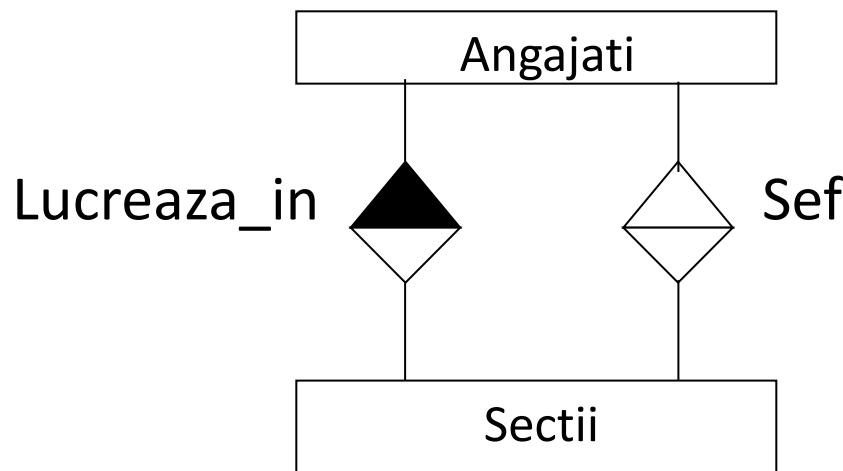
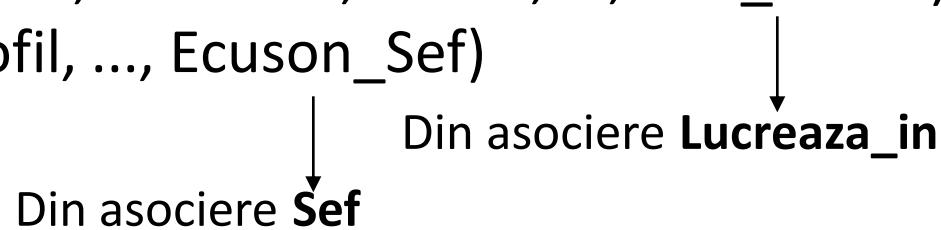


Figura 19. Exemplu de asociere cu doua dependente



# Transformarea diagramelor E-A in M-R

- Intre entitatile ANGAJATI si SECTII exista doua asocieri:
  - **Sef** (unu-unu), care modeleaza faptul ca o sectie este condusa de un angajat (seful de sectie);
  - **Lucreaza\_in** (multi-unu) care modeleaza faptul ca o sectie are mai multi angajati.
- Rezultatul transformarii este cel de mai jos. Atributele aflate dupa punctele se suspensie sunt cele adaugate (chei straine):  
ANGAJATI (Ecuson, Nume, Prenume, Varsta, ..., Cod\_Sectie)  
SECTII (Cod\_Sectie, Profil, ..., Ecuson\_Sef)



In atributul Cod\_Sectie din relatia ANGAJATI se va inregistra pentru fiecare angajat codul sectiei in care acesta lucreaza iar in atributul Ecuson\_Sef din relatia SECTII se va inregistra, pentru fiecare sectie, ecusonul sefului de sectie. Pentru asocierea Sef s-a aplicat primul criteriu (relatia SECTII va avea mult mai putine inregistrari decat ANGAJATI), dar si al doilea criteriu este indeplinit.



# Transformarea diagramelor E-A in M-R

- **Transformarea asocierilor unare si binare multi-multi si a celor de grad mai mare decat doi**

Fiecare asociere binara multi-multi si fiecare asociere cu grad mai mare decat doi se transforma intr-o schema de relatie astfel:

- **Nume relatie** = Nume asociere;
- **Atributele relatiei** = Reuniunea identificatorilor entitatilor asociate la care se adauga atributele proprii ale asocierii ;
- **Cheia relatiei** = Conform Tabelului 2.



# Transformarea diagramelor E-A in M-R

Grad	Conecțivitate	Dependente Funcționale Primare (DFP)
Unare	<ul style="list-style-type: none"> <li>• unu (E) - unu (E)</li> <li>• cheie(E) se introduce ca si cheie straina in tabela E, cu redenumire</li> </ul>	Cheie(E) => Cheie straina(E) Cheie straina(E) => Cheie(E)
	unu (E) - multi (E)	Cheie(E) => Cheie straina(E)
	multi (E) - multi (E)	Cheie(E) => AP
Binare	<ul style="list-style-type: none"> <li>• unu (E1) - unu (E2)</li> <li>• cheie(E2) se introduce in E1</li> </ul>	Cheie(E1) => Cheie straina(E2) Cheie(E2) => Cheie straina(E1)
	unu (E1) - multi (E2)	Cheie(E1) => Cheie straina(E2)
	multi (E1) - multi (E2)	Cheie(E1) + Cheie(E2) => AP
Ternare	unu (E1) - unu (E2) - unu (E3)	Cheie(E1)+Cheie(E2)=>Cheie(E3) sau Cheie(E1)+Cheie(E3)=>Cheie(E2) sau Cheie(E2)+Cheie(E3)=>Cheie(E1)
	unu (E1) - unu (E2) - multi (E3)	Cheie(E1)+Cheie(E3)=>Cheie(E2) sau Cheie(E2)+Cheie(E3)=>Cheie(E1)
	unu (E1)- multi (E2) - multi (E3)	Cheie(E2)+Cheie(E3)=>Cheie(E1)
	multi (E1) - multi (E2) - multi (E3)	Cheie(E1)+Cheie(E2)+Cheie(E3)=>AP

**Tabel 1. Dependente functionale primare rezultate din transformare**

Legenda: X => Y: Multimea de coloane X determina functional multimea de coloane Y

AP : Atribute proprii transformarii sau multimea vida (daca nu exista)

X+Y : Coloanele X impreuna cu coloanele Y



# Transformarea diagramelor E-A in M-R

Grad	Conecțivitate	Cheia relației obținuta din asociere
Unare	multi (E) - multi (E)	Cheie(E) + Cheie(E)
Binare	multi (E1) - multi (E2)	Cheie(E1)+Cheie(E2)
Ternare	unu (E1) - unu (E2) - unu (E3)	Cheie(E1)+Cheie(E2) sau Cheie(E1)+Cheie(E3) sau Cheie(E2)+Cheie(E3)
	unu (E1) - unu (E2) - multi (E3)	Cheie(E1)+Cheie(E3) sau Cheie(E2)+Cheie(E3)
	unu (E1)- multi (E2) - multi (E3)	Cheie(E2)+Cheie(E3)
	multi (E1)-multi(E2) - multi (E3)	Cheie(E1)+Cheie(E2)+Cheie(E3)

**Tabel 2. Cheile schemelor de relație rezultate din asocieri**

Legenda:

X + Y: Multimea de atribute X impreuna cu multimea de atribute Y



# Transformarea diagramelor E-A in M-R

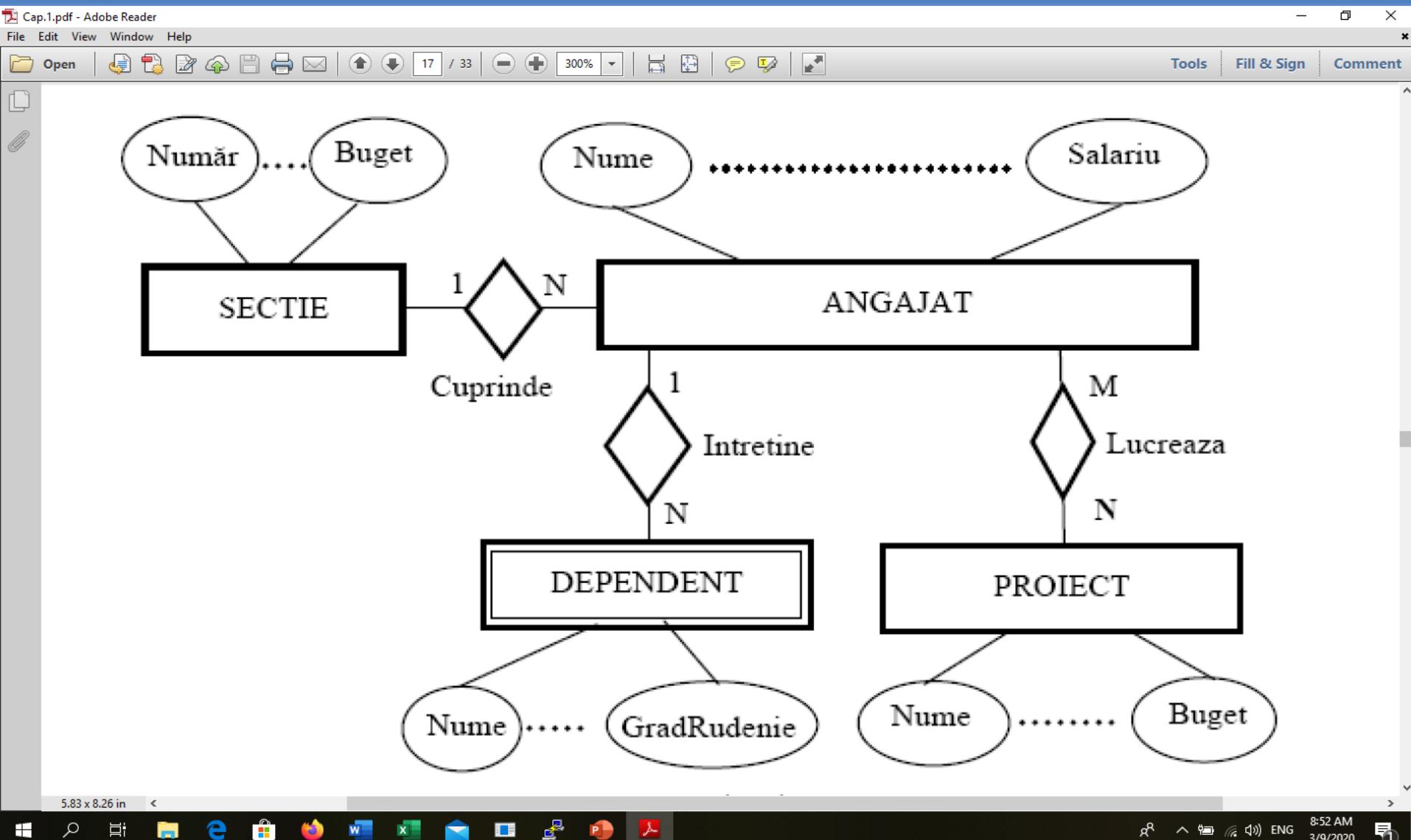


Figura 20. Exemplu de diagrama de relatii in modelul E-A



# Transformarea diagramelor E-A in M-R

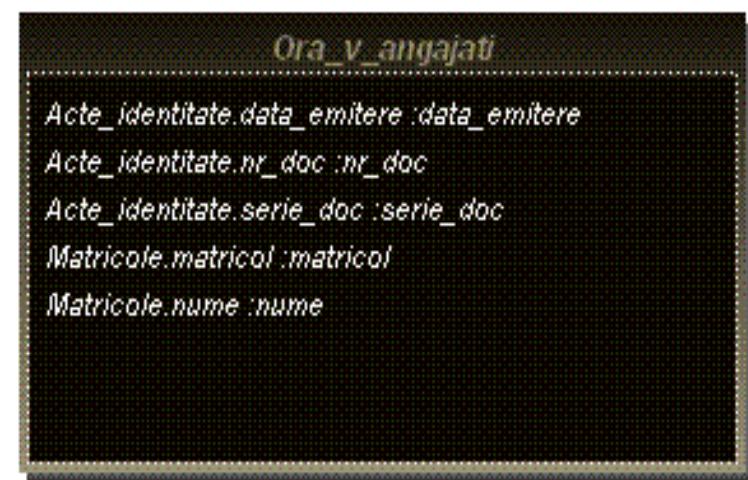
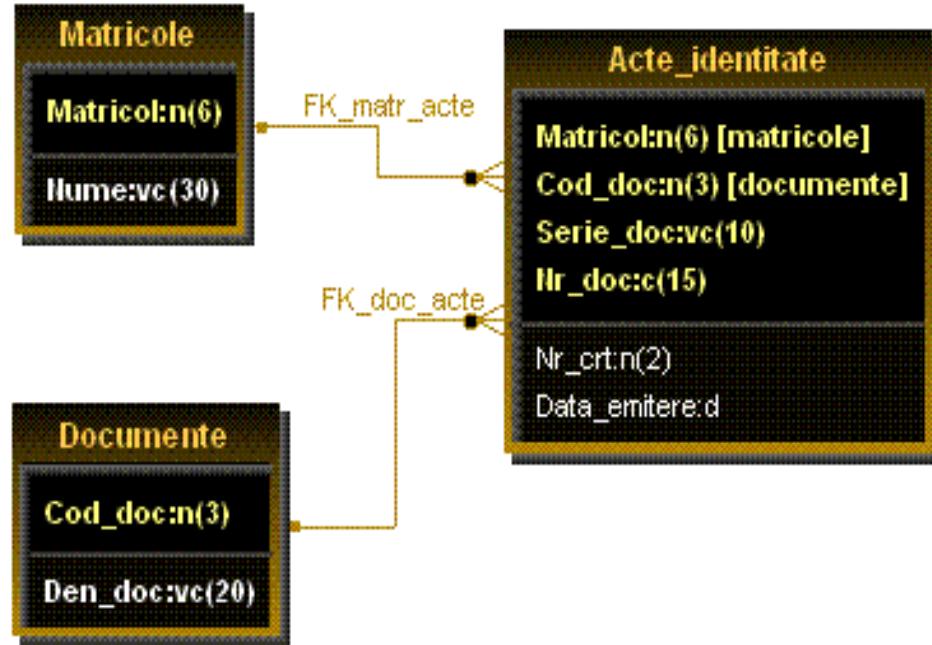


Figura 20. Exemplu de diagrama de relatii in modelul M-R



# Capitolul 5

## Proiectarea bazelor de date



# Proiectarea bazelor de date

Proiectarea corecta a bazei de date este primul pas spre o implementare de succes a unei aplicatii software si de aceea trebuie respectate metodologiile care impun normalizarea schemelor de relatie, pastrarea integritatii datelor si eliminarea anomalilor.

Proiectarea trebuie vazuta sub doua aspecte:

- proiectare conceptuala;
  - proiectare logica.
- Deoarece o schema de relatie poate avea multiple forme, se pune problema alegerii corecte a structurii relatiei printr-o proiectare conceptuala adecvata, dar si respectarea cerintelor de functionare rezultate din analiza si regasite in proiectarea logica.
  - Schema conceptuala descrie in mod abstract forma in care dorim sa stocam datele reale iar procesul de realizare se numeste **modelare conceptuala**.



# Dependente functionale

In paragrafele urmatoare vom folosi o conventie de notare intalnita in literatura de specialitate:

- R, S, T, ...: scheme de relatii;
- r, s, ...: instante ale relatiilor R respectiv S;
- A, B, C, D, ... (litere mari de la inceputul alfabetului): atribute ale unei relatii;
- X, Y, Z, W, U, ... (litere mari de la sfarsitul alfabetului): multimi de atribute dintr-o schema de relatie;
- $X \subseteq R$ : Multimea de atribute X este inclusa in multimea atributelor relatiei R;
- $Y \subseteq X$ : Multimea de atribute Y este inclusa in multimea de atribute X;
- $A \in X$ : Atributul A apartine multimii de atribute X;
- t, t1, t2, ... tupluri ale unei relatii;
- $t[X]$ : valorile atributelor din multimea X aflate in tuplul t;
- F, G, ...: multimi de dependente functionale atasate unei scheme de relatie.



# Dependente functionale

- Termenul **relatie** semnifica atat **schema relatiei** (descrierea structurii acesteia) cat si o **instanta** a acesteia (datele propriu-zise).
- **Dependentă datelor** este o restrictie asupra relatiilor  $R$  care pot constitui valoarea curenta a unei scheme de relatie  $R$ .
- Dependentă trebuie vazuta ca o legatura intre două atribute, in sensul ca valoarea unui atribut determina valoarea celuilalt, de exemplu atributul Matricol determina unic atributul Nume al unui student.



# Dependente functionale

- **Definitie:** Fie:
  - $R$  o schema de relatie si
  - $X, Y \subseteq R$  doua multimi de atribute ale acesteia.
  - Spunem ca  **$X$  determina functional pe  $Y$**  (sau  $Y$  este determinata functional de  $X$ ), prin notatia simbolica  $X \rightarrow Y$  daca si numai daca oricare ar fi doua tupluri  $t_1$  si  $t_2$  din orice instanta a lui  $R$  atunci:
$$t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$$
Cu alte cuvinte, daca doua tupluri au aceleasi valori pe atributele  $X$  atunci ele au aceleasi valori si pe atributele  $Y$ .



# Dependente functionale

Exemple:

- Consideram relatia FURNIZORI cu urmatoarea structura:  
FURNIZORI (ID\_furniz, Den\_furniz, Adresa) unde:
  - ID\_furniz = cheia de identificare unica a unui furnizor;
  - Den\_furniz = denumirea furnizorului;
  - Adresa = adresa furnizorului.
- Relatia COMPOENTE este un catalog de componente pentru calculatoare, cu urmatoarea structura:  
COMPONENTE( ID\_com, Den\_com, ID\_furniz, Pret, Um) unde:
  - ID\_com = cheia de identificare unica a unei componente;
  - Den\_com = denumirea componentei;
  - ID\_furniz = cheia de identificare unica a unui furnizor pentru o componentă;
  - Pret = pretul unitar al componentei ;
  - Um = unitatea de masura (bucati, metri, set, etc.).



# Dependente functionale

- Putem identifica urmatoarele dependente functionale:
- În relația FURNIZORI :
  - $ID_{furniz} \rightarrow Den_{furniz}$
  - $ID_{furniz} \rightarrow Adresa$
- În relația COMPONENTE :
  - $ID_{com} \rightarrow Den_{com}$
  - $ID_{com} \rightarrow ID_{furniz}$
  - $ID_{com} \rightarrow Pret, Um$
  - $ID_{com} \rightarrow ID_{furniz}, Pret$
- Observație: Singura cale de a gasi dependențele functionale valabile pentru o schema R este analiza atenta a inteleșului (semnificatiei) fiecarui atribut al acesteia și a modului în care sunt asignate valori atributelor.



# Dependente functionale

- Dependentă funcțională  $ID_{com} \rightarrow ID_{furniz}$ , Pret specifică ca  $ID_{com}$  identifică unic un singur produs, achiziționat de la un singur furnizor și la un anumit pret. În acest caz, dacă un produs cu același nume este achiziționat de la un alt furnizor, eventual la același pret, trebuie să i se dea un alt  $ID_{com}$ .
- Dependentă funcțională  $ID_{furniz} \rightarrow Den_{furniz}$  specifică ca un  $ID_{furniz}$  identifică unic un furnizor de componente și dacă două componente au același  $ID_{furniz}$  înseamnă că ele sunt achiziționate de la același furnizor.
- Observație: Dependentele funktionale nu se determină prin inspectarea valorilor relaiei ci din semnificatia atributelor acesteia.



# Axiome si reguli

- Pornind de la o multime de dependente functionale atasate unei scheme de relatie se pot deduce alte dependente functionale valide, folosind reguli de inferenta.
- Exista mai multe reguli de inferenta. Pentru a se putea face o prezentare formală a acestora, trei dintre ele au fost alese ca axiome iar restul se pot deduce pornind de la ele.
- Cele trei axiome (numite Axiomele lui Armstrong) sunt urmatoarele:



# Axiome si reguli

- **A1. Reflexivitate:** Fie  $R$  o schema de relatie si  $X \subseteq R$ .

Daca  $Y \subseteq X$  atunci  $X \rightarrow Y$ .

Toate dependentele functionale care rezulta din aceasta axioma sunt numite si **dependente triviale**.  
Ele nu spun nimic in plus fata de setul de dependente initial dar sunt dependente functionale valide.

- **A2. Augmentare:** Fie  $R$  o schema de relatie si

$X, Y, Z \subseteq R$ . Daca  $X \rightarrow Y$  atunci si  $XZ \rightarrow YZ$ .

Aceasta axioma arata ca se poate reuni o aceeasi multime Z in stanga si in dreapta unei dependente functionale valide, obtinand de asemenea o dependenta functionala valida.



# Axiome si reguli

- **A3. Tranzitivitate:** Fie R o schema de relatie si  $X, Y, Z \subseteq R$ . Daca  $X \rightarrow Y$  si  $Y \rightarrow Z$  atunci si  $X \rightarrow Z$ .
- Pe baza acestor axiome se pot demonstra o serie de **reguli de inferenta** pentru dependente functionale dintre care cele mai importante sunt urmatoarele:
- **R1. Descompunere:** Fie R o schema de relatie si  $X, Y, Z \subseteq R$ . Daca  $X \rightarrow Y$  si  $Z \subseteq Y$  atunci si  $X \rightarrow Z$ .

Demonstratie:

$X \rightarrow Y$  este data. Prin ipoteza  $Z \subseteq Y$ , aplicand A1 rezulta  $Y \rightarrow Z$ .

Cu A3, din  $X \rightarrow Y$  si  $Y \rightarrow Z$  obtinem  $X \rightarrow Z$ .



# Axiome si reguli

- Regula descompunerii ne permite sa rescriem un set de dependente functionale astfel incat sa obtinem doar dependente care au in partea dreapta doar un singur atribut. Sa presupunem ca avem o dependenta functionala de forma:

$$X \rightarrow A_1 A_2 A_3 \dots A_n$$

- Atunci ea poate fi inlocuita cu urmatoarele **n** dependente functionale:

$$X \rightarrow A_1$$

$$X \rightarrow A_2$$

$$X \rightarrow A_3$$

...

$$X \rightarrow A_n$$



## Axiome si reguli

- **R2. Reuniune:** Fie  $R$  o schema de relatie si  $X, Y, Z \subseteq R$ .

Daca  $X \rightarrow Y$  si  $X \rightarrow Z$  atunci si  $X \rightarrow YZ$ .

Rezulta si faptul ca din cele  $n$  dependente obtinute prin descompunere se poate obtine dependenta initiala, deci inlocuirea acesteia nu duce la pierderea vreunei corelatii existente.

Demonstratie:

$X \rightarrow Y$  este data. Amplificam cu  $X$  si prin inferenta obtinem  $XX \rightarrow XY$ , sau  $X \rightarrow XY$ . De asemenea,  $X \rightarrow Z$  este data; amplificam cu  $Y$  si  $XY \rightarrow ZY$  sau  $XY \rightarrow YZ$ .

Aplicand A3 rezulta  $X \rightarrow YZ$ .



# Axiome si reguli

- **R3. Pseudotranzitivitate:** Fie R o schema de relatie si  $X, Y, Z, W \subseteq R$ .

Daca  $X \rightarrow Y$  si  $YZ \rightarrow W$  atunci si  $XZ \rightarrow W$ .

Demonstratie:

$X \rightarrow Y$  este data. Amplificam cu Z si obtinem  $XZ \rightarrow YZ$ .

Dar  $YZ \rightarrow W$  si aplicand A3 rezulta  $XZ \rightarrow W$ .



# Inciderea unei multimi de DF

- Pornind de la un set de dependente functionale  $F$  si utilizand axiome si reguli obtinem o multitudine de alte dependente, triviale sau nu. Multimea tuturor dependentelor functionale care se pot deduce din  $F$  se numeste **inchiderea multimii de dependente functionale  $F$** , notata cu  $F^+$ . Definitia formală a acestei inchideri este urmatoarea:

$$F^+ = \{X \rightarrow Y \mid F \Rightarrow X \rightarrow Y\}$$

Unde prin  $\Rightarrow$  am notat faptul ca dependenta respectiva se poate deduce din  $F$  folosind axiome si reguli.



# Inchiderea unei multimi de DF

- Multimea  $F^+$  contine foarte multe dependente, inclusiv dependente triviale ca:  
 $ABC \rightarrow A, ABC \rightarrow B, ABC \rightarrow C, ABC \rightarrow AB, ABC \rightarrow AC,$   
 $ABC \rightarrow BC$  sau  $ABC \rightarrow ABC$
- $F^+$  nu se calculeaza in totalitate, algoritmii care au nevoie de ea ocolesc, intr-un fel sau altul, calculul acesteia.
- Introducerea acestei notiuni s-a facut pentru a explica, in cazul descompunerii unei scheme de relatie, care sunt dependentele mostenite de elementele descompunerii de la relatia initiala si pentru a putea defini formal urmatoarele notiuni:



# Acoperirea unei multimi de DF

- ✓ **Acoperirea unei multimi de DF** - Fie R o schema de relatie si F, G doua multimi de dependente pentru R. Se spune ca F **acopera** pe G daca si numai daca  $G \subseteq F^+$ .
- ✓ **Echivalenta a doua multimi de dependente** - Fie R o schema de relatie si F, G doua multimi de dependente pentru R. Se spune ca F este **echivalenta** cu G daca si numai daca F acopera pe G si G acopera pe F, adica, daca  $G \subseteq F^+$  si  $F \subseteq G^+$ , rezulta  $F^+ = G^+$ .



# Forma canonica a unei multimi de DF

- ✓ **Forma canonica a unei multimi de DF** - Din definitiile de mai sus rezulta ca o multime de dependente poate fi inlocuita cu alta echivalenta continand alte dependente. In cazul aceasta se poate spune ca:
  - **Definitie:** O multime de dependente este in **forma canonica** daca:
    - Orice dependenta are in partea dreapta un singur atribut. Acest lucru se poate obtine aplicand regula descompunerii prezentata anterior.
    - Multimea de dependente este minima, niciuna dintre dependente neputand sa fie dedusa din celelalte (altfel spus, nu exista dependente redundante).



# Forma canonica a unei multimi de DF

Exemplu:

1) Fie  $R = ABCDE$  o schema de relatie si  $F$  multimea de dependente functionale asociata, cu

$$F = \{ AB \rightarrow CDE, C \rightarrow DE \}:$$

Aplicam regula de descompunere si obtinem:

$$F = \{ AB \rightarrow C, AB \rightarrow D, AB \rightarrow E, C \rightarrow D, C \rightarrow E \}$$

Multimea nu este insa minimala deoarece  $AB \rightarrow D$  si  $AB \rightarrow E$  se pot deduce prin tranzitivitate din  $AB \rightarrow C$  impreuna cu  $C \rightarrow D, C \rightarrow E$ .

Rezulta ca forma canonica a lui  $F$  este:

$$F = \{ AB \rightarrow C, C \rightarrow D, C \rightarrow E \}$$



# Forma canonica a unei multimi de DF

2) Pentru relatia

COMPONENTE = ( ID\_com, Den\_com, Pret, ID\_furniz,  
Den\_furniz).

- Multimea de dependente functionale F este:

$$F = \{ ID_{com} \rightarrow Den_{com}, Pret, ID_{furniz}, Den_{furniz},$$
$$ID_{furniz} \rightarrow Den_{furniz} \}.$$

- Forma canonica a lui F este:

$$F = \{ ID_{com} \rightarrow Den_{com},$$
$$ID_{com} \rightarrow Pret,$$
$$ID_{com} \rightarrow ID_{furniz},$$
$$ID_{furniz} \rightarrow Den_{furniz} \}$$

A fost eliminata dependenta redundanta  $ID_{com} \rightarrow Den_{furniz}$ .



# Implicatii logice ale dependentelor

## ➤ Implicatii logice ale dependentelor

- Fie  $F$  o multime de dependente functionale pentru  $R$  si fie  $X \rightarrow Y$  o dependenta functionala, valabila tot pentru schema  $R$ . Atunci,  
 **$F$  implica logic  $X \rightarrow Y$** , daca orice relatie  $r$  pentru  $R$ , care satisface dependentele din  $F$ , satisface si  $X \rightarrow Y$ .

Exemplu:

- Fie  $R$  o schema de relatie, iar  $A, B, C$  atribute in  $R$ . Avem, de exemplu, dependentele  $A \rightarrow B$  si  $B \rightarrow C$  valabile in  $R$ . Se poate arata ca, de asemenea, ca  $A \rightarrow C$  este valabila in  $R$  (tranzitivitate).



# Inciderea unei multimi de DF

- **Inciderea** multimii de dependente  $F$ , notata cu  $F^+$ , se mai defineste ca **multimea dependentelor functionale implicate logic de catre  $F$** .  
Daca  $F^+ = F$ ,  $F$  este o familie completa de dependente.  
Exemplu:  
Fie  $R=ABC$  si  $F$  multimea de dependente. Atunci,  $F^+$  contine toate dependentele  $X \rightarrow Y$ , astfel incat:
  - $X$  contine pe  $A$ , de exemplu  $ABC \rightarrow AB$ ,  $AB \rightarrow BC$  sau  $A \rightarrow C$ .
  - $X$  contine  $B$  dar nu  $A$ , iar  $Y$  nu contine  $A$ , de exemplu,  $BC \rightarrow B$ ,  $B \rightarrow C$ ,  $B \rightarrow \emptyset$ .
  - $X \rightarrow Y$  este una dintre cele doua dependente  $C \rightarrow C$  sau  $C \rightarrow \emptyset$ .



# Cheia si supercheia unei relatii

In acest moment putem sa dam o definitie echivalenta a cheii unei relatii pe baza dependentelor functionale:

- **Definitie:** Fie  $R$  o schema de relatie,  $F$  multimea de dependente functionale asociata si  $X \subseteq R$ . Atunci  $X$  este **cheie** pentru  $R$  daca si numai daca:
  - $F \Rightarrow X \rightarrow R$  (deci  $X \rightarrow R$  se poate deduce din  $F$ );
  - $X$  este minimala: oricare ar fi  $Y \subset X$ ,  $Y \neq X$  atunci  $\neg(F \Rightarrow Y \rightarrow R)$  (deci orice submultime stricta a lui  $X$  nu mai indeplineste conditia anterioara).



# Cheia si supercheia unei relatii

- Deci o cheie determina functional toate atributele relatiei, este minimala si nicio submultime stricta a sa nu determina functional pe R. Se observa faptul ca aceasta definitie este echivalenta cu cea din capitolul anterior: cunoscandu-se valorile pe atributele X, pot fi determinate unic valorile pentru toate atributele relatiei, deci este determinat unic un tuplu din relatie.
- In cazul in care doar prima conditie este indeplinita (fara minimalitate) multimea X se numeste **supercheie**.
- Observatie: Faptul ca o supercheie nu este constransa de minimalitate nu inseamna insa ca ea nu poate fi minimala. Rezulta ca orice cheie este in acelasi timp si supercheie, reciproca nefiind insa adevarata.



# Cheia si supercheia unei relatii

Exemplu:

Fie  $R = ABCDE$  si  $F = \{ AB \rightarrow C, C \rightarrow D, C \rightarrow E \}$ . Atunci AB este cheie pentru R:

- Din  $AB \rightarrow C$ ,  $C \rightarrow D$  si  $C \rightarrow E$  obtinem prin tranzitivitate  $AB \rightarrow D$  si  $AB \rightarrow E$ ;
- Din  $AB \rightarrow C$ ,  $AB \rightarrow D$  si  $AB \rightarrow E$  obtinem prin reuniune  $AB \rightarrow CDE$ ;
- Din  $AB \rightarrow CDE$  obtinem prin augmentare cu  $AB$   $AB \rightarrow ABCDE$ , deci  $AB \rightarrow R$ ;
- Rezulta ca  $AB$  este supercheie pentru  $R$ . Vom arata intr-un alt capitol cum se poate demonstra si ca  $AB$  este minimala, deci este chiar cheie pentru  $R$ , nu numai supercheie.



# Proiectia unei multimi de DF

- Aşa cum s-a mentionat anterior, inchiderea unei multimi de dependente functionale  $F^+$  a fost introdusa si pentru a putea defini setul de dependente functionale mostenite de o schema de relatie obtinuta prin descompunerea unei scheme incorrect proiectata.

- Sa analizam o noua structura a relatiei:

COMPONENTE\_FURNIZORI =

( ID\_com, Den\_com, Pret, ID\_furniz, Den\_furniz, Adresa).

Multimea de dependente functionale F este:

$$F = \{ ID\_com \rightarrow Den\_com, ID\_com \rightarrow Pret,$$
$$ID\_com \rightarrow ID\_furniz, ID\_furniz \rightarrow Den\_furniz,$$
$$ID\_furniz \rightarrow Adresa \}$$



# Proiectia unei multimi de DF

- Prin descompunerea acestei relatii in doua relatii, obtinem relatiile:

COMPONENTE = ( ID\_com, Den\_com, Pret, ID\_furniz)

FURNIZORI = ( ID\_furniz, Den\_furniz, Adresa)

- Atributele relatiei initiale se regasesc fie doar intr-una dintre schemele rezultate, fie in amandoua.

Se pune problema identificarii dependentelor mostenite de cele doua relatii de la relatia initiala.

Pentru aceasta trebuie definita proiectia unei multimi de dependente pe o multime de atribute.



# Proiectia unei multimi de DF

- **Definitie.** Fie o relatie  $R$ , o multime asociata de dependente functionale  $F$  si o submultime de atribute  $S \subseteq R$ .

**Proiectia multimii de dependente  $F$  pe  $S$ ,** notata cu  $\pi_S(F)$  este multimea dependentelor din  $F^+$  care au atributele din partea stanga si cea dreapta incluse in  $S$ .

Formal, putem scrie:

$$\pi_S(F) = \{X \rightarrow Y \in F^+ \mid X, Y \subseteq S\}$$



# Proiectia unei multimi de DF

Exemplu:

Fie relatia  $\text{COMPONENTE\_FURNIZORI} =$

( $\text{ID\_com}$ ,  $\text{Den\_com}$ ,  $\text{Pret}$ ,  $\text{ID\_furniz}$ ,  $\text{Den\_furniz}$ ,  $\text{Adresa}$ )  
avand multimea de dependente functionale:

$F = \{ \text{ID\_com} \rightarrow \text{Den\_com}, \text{ID\_com} \rightarrow \text{Pret}, \text{ID\_com} \rightarrow \text{ID\_furniz}, \text{ID\_furniz} \rightarrow \text{Den\_furniz}, \text{ID\_furniz} \rightarrow \text{Adresa} \}.$

- Aplicand definitia se obtin proiectiile urmatoare:

$F_{\text{COMPONENTE}} = \pi_{\text{COMPONENTE}}(F) = \{ \text{ID\_com} \rightarrow \text{Den\_com}, \text{ID\_com} \rightarrow \text{Pret}, \text{ID\_com} \rightarrow \text{ID\_furniz} \}$

$F_{\text{FURNIZORI}} = \pi_{\text{FURNIZORI}}(F) = \{ \text{ID\_furniz} \rightarrow \text{Den\_furniz}, \text{ID\_furniz} \rightarrow \text{Adresa} \}$



# Proiectia unei multimi de DF

- Observatie: Atunci cand descompunem o schema se poate intampla ca unele dintre dependentele schemei initiale sa se piarda.  
Exemplu: Fie  $R = ABCD$  si  $F = \{ A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A \}$ .  
In cazul in care descompunem  $R$  in  $R_1 = AB$  si  $R_2 = CD$   
atunci:  $F_{R1} = \pi_{R1}(F) = \{ A \rightarrow B, B \rightarrow A \}$  si  
 $F_{R2} = \pi_{R2}(F) = \{ C \rightarrow D, D \rightarrow C \}$
- A doua dependenta din fiecare multime nu este in  $F$  dar este in  $F^+$  (obtinuta prin tranzitivitate).
- Observam insa ca dependentele  $B \rightarrow C$  si  $D \rightarrow A$  nu mai pot fi obtinute nici din  $F_{R1}$ , nici din  $F_{R2}$  si nici din reuniunea lor. Intr-un alt capitol va fi prezentata o metoda prin care se poate verifica daca prin descompunere dependentele initiale sunt pastrate sau nu.



# Inciderea unei multimi de atribute

- **Definitie :** Fie  $R$  o schema de relatie,  $F$  multimea de dependente asociata si  $X \subseteq R$ .

Se poate defini  $X^+$  ca fiind **incliderea multimii de atribute  $X$  in raport cu  $F$**  astfel:

$$X^+ = \{ A \mid X \rightarrow A \in F^+ \}$$

- Deci  $X^+$  contine toate atributele care apar in partea dreapta a dependentelor din  $F$  sau care se pot deduce din  $F$  folosind reguli si axiome.



# Inciderea unei multimi de attribute

## ➤ Algoritm de calcul pentru $X^+$

Fie  $R$  o schema de relatie,  $F$  multimea de dependente asociata si  $X \subseteq R$ .

Pentru calculul inciderii multimii de attribute  $X^+$  se aplica urmatorul **algoritm** iterativ:

- Se porneste cu  $X^{(0)} = X$
- Pentru  $i \geq 1$ ,

$$X^{(i)} = X^{(i-1)} \cup \{ A \mid (\exists) Y \rightarrow A \in F \text{ cu } Y \subseteq X^{(i-1)} \}$$

- Daca  $X^{(i)} = X^{(i-1)}$  sau  $X^{(i)} = R$ , atunci

STOP.

- Scopul introducerii acestei notiuni este si acela de a putea ocoli calculul lui  $F^+$  in alti algoritmi sau definitii.<sub>32</sub>



# Inciderea unei multimi de attribute

Exemplu: Fie  $R = ABCDE$  si  $F = \{ A \rightarrow B, A \rightarrow C, D \rightarrow E \}$ .

Pentru a calcula  $A^+$ ,  $D^+$  si  $(AD)^+$  procedam astfel:

✓ Calcul  $A^+$ :

- $X^{(0)} = \{A\}$
- Din  $A \rightarrow B$  si  $A \rightarrow C$  rezulta ca  $X^{(1)} = X^{(0)} \cup \{B, C\} = \{A\} \cup \{B, C\} = ABC$
- Sigurele dependente care au partea dreapta in  $X^{(1)}$  sunt tot primele doua, deci

$$X^{(2)} = X^{(1)} \cup \{B, C\} = \{A, B, C\} \cup \{B, C\} = ABC$$

- Deoarece  $X^{(2)} = X^{(1)}$   $\Rightarrow$  STOP.

- Rezulta ca  $A^+ = ABC$

✓ Calcul  $D^+$ : In mod similar, rezulta  $D^+ = DE$ .



# Inciderea unei multimi de attribute

✓ Calcul  $(AD)^+$  :

- $X^{(0)} = \{A, D\}$
- Din  $A \rightarrow B$ ,  $A \rightarrow C$  si  $D \rightarrow E$  rezulta ca  
$$X^{(1)} = X^{(0)} \cup \{ B, C, E \} = \{ A, D \} \cup \{ B, C, E \} = ABCDE$$
- Deoarece  $X^{(1)} = R \Rightarrow$  STOP ( oricate iteratii am face nu mai pot sa apara noi attribute).
- Rezulta ca  $(AD)^+ = ABCDE$



# Inciderea unei multimi de atribut

Avem urmatorul rezultat teoretic:

- **Lema.** Fie  $R$  o schema de relatie,  $F$  multimea de dependente asociata si  $X, Y \subseteq R$ .

Atunci  $X \rightarrow Y$  se poate deduce din  $F$  daca si numai daca  $Y \subseteq X^+$ .

Demonstratie:

Fie  $Y = A_1 A_2 A_3 \dots A_n$ . Facem ipoteza ca  $Y \subseteq X^+$ . Prin definitia lui  $X^+$ ,  $X \rightarrow A_i$  este implicat de axiomele Armstrong pentru orice  $i$ .

Dar cum  $Y = A_1 A_2 \dots A_n = \{A_1 \cup A_2 \cup \dots \cup A_n\}$ , prin regula de reuniune rezulta  $X \rightarrow Y$ , deoarece  $X \rightarrow A_i$  pentru orice  $i$ .



# Inciderea unei multimi de atrbute

- Exista teoreme care arata ca “**sistemul axiomelor lui Armstrong este complet si corect**”.
- **Complet** – daca sunt date dependentele functionale din  $F$ , prin axiome deducem toate dependentele functionale din  $F^+$ .
- **Corect** – plecand de la  $F$ , aplicand regulile de inferenta reprezentate de axiomele Armstrong , nu putem deduce dependente functionale care nu sunt in  $F^+$ .



# Inciderea unei multimi de atribute

Consecinte:

- $X^+$  a fost definit ca multimea de atribute A, astfel incat  $X \rightarrow A$  decurge din F, folosind axiomele lui Armstrong.
- O definitie echivalenta este:  $X^+$  este multimea atributelor A astfel incat F implica logic pe  $X \rightarrow A$ .
- $F^+$  a fost introdusa ca multimea dependentelor implicate logic de catre F. Se poate insa defini  $F^+$  ca multimea dependentelor care decurg din F prin axiomele Armstrong.
- Consecintele pot fi demonstate pe baza informatiilor anterioare.



# O alta definitie pentru cheie

- Pe baza propozitiei din paragraful anterior se poate da o alta definitie pentru cheia sau supercheia unei relatii pe inchiderea unei multimi de attribute  $X^+$  (si nu pe  $F^+$ , ca in subcapitolul anterior ).
- **Definitie:** Fie  $R$  o schema de relatie,  $F$  multimea de dependente functionale asociata si  $X \subseteq R$ .
- $X$  este **cheie** pentru  $R$  daca si numai daca:
  - $X^+ = R$ ;
  - $X$  este minimala: oricare ar fi  $Y \subset X$ ,  $Y \neq X$  atunci  $Y^+ \neq R$  (deci orice submultime stricta a lui  $X$  nu mai indeplineste conditia anterioara).
  - $X$  este **supercheie** pentru  $R$  daca este indeplinita numai prima conditie.



## O alta definitie pentru cheie

Echivalenta acestei definitii cu cea anterioara este evidenta:

- $X^+ = R$  inseamna ca  $X \rightarrow R$  conform lemei;
- Minimalitatea este de asemenea definita echivalent:

Daca  $Y \subset X$ ,  $\neg(F \Rightarrow Y \rightarrow R)$  este echivalenta cu  $\neg(Y^+ = R)$ , adica  $Y^+ \neq R$ .

- Folosind aceasta definitie se poate defini o euristică de gasire a cheilor unei relații:



# Euristica de gasire a cheilor unei relatii

## ➤ **Euristica de gasire a cheilor unei relatii**

Pentru gasirea cheilor unei relatii pornim de la observatia ca atributele care nu sunt in partea dreapta a niciunei dependente nu pot sa apară in procesul de inchidere a unei multimi de atribut, deci ele aparțin oricarei chei a relatiei.

- Fie **R** o schema de relatie si **F** multimea de dependente functionale asociata (F in forma canonica).
  - **Cheia unica sau cheile alternative ale lui R** se calculeaza in pasii urmatori:



# Euristica de gasire a cheilor unei relatii

- 1) Se porneste de la multimea de atribute  $X \subseteq R$  care nu apar in partea dreapta a niciunei dependente.
- 2) Se calculeaza  $X^+$ . Daca  $X^+ = R$  atunci  $X$  este o cheie minimala a relatiei  $R$  si calculul se opreste aici.  
Pasii urmatori se efectueaza doar daca  $X^+ \neq R$ .
- 3) Se adauga la  $X$  cate un atribut din  $R - X^+$  obtinandu-se o multime de chei candidat.
- 4) Se calculeaza  $X^+$  pentru fiecare dintre candidate. Daca se obtin toate atributele lui  $R$  atunci acel  $X$  este o cheie a lui  $R$ .
- 5) Se repeta pasii 3 si 4 pornind de la acele multimi candidat  $X$  care nu sunt gasite ca si chei la pasul anterior. Dintre multimile candidat nu luam niciodata in considerare pe cele care contin o cheie gasita anterior.
- 6) Procesul se opreste cand nu se mai pot face augmentari.



# Euristica de gasire a cheilor unei relatii

Exemple:

1) Fie  $R = ABCDE$  si  $F = \{ A \rightarrow B, A \rightarrow C, D \rightarrow E \}$ .

1.1) Multimea atributelor care nu apar in partea dreapta a niciunei dependente este  $X = (AD)$ .

1.2) Calculam  $(AD)^+ = ABCDE = R$ . Nu mai trebuie verificata minimalitatea deoarece A si D trebuie sa faca parte din orice cheie (oricum,  $A^+ = ABC$  si  $D^+ = DE$ ) .

1.3) Procesul se opreste. Rezulta ca AD este cheie pentru R.

2) Fie  $R = ABCDE$  si  $F = \{A \rightarrow B, B \rightarrow A, A \rightarrow C, D \rightarrow E\}$ .

2.1) Multimea atributelor care nu apar in partea dreapta a niciunei dependente este  $X = D$ .

2.2) Calculam  $(D)^+$ . Obtinem  $(D)^+ = DE$ . Rezulta ca D nu este cheie pentru R.



# Euristica de gasire a cheilor unei relatii

- 2.3) Calculam multimea de candidate: augmentam D cu atribute din  $R - D^+ = ABCDE - DE = ABC$ . Obtinem AD, BD si CD
- 2.4) Calculam inchiderile lor. Obtinem  $(AD)^+ = R$ ,  $(BD)^+ = R$  si  $(CD)^+ = CDE \neq R$ . Rezulta ca AD si BD sunt chei ale lui R dar CD nu este cheie.
- 2.5) Calculam o noua multime de candidate pornind de la CD. Putem augmenta CD cu atribute din  $R - (CD)^+ = ABCDE - CDE = AB$ . Niciuna dintre augmentari nu este insa posibila pentru ca atat ACD cat si BCD contin o cheie gasita anterior (AD respectiv BD).
- 2.6) Procesul se opreste. Singurele chei ale lui R raman AD si BD.



# Acoperiri de multimi de dependente

- **Definitie:** Fie  $F, G$  – multimi de dependente functionale. Daca  $F^+ = G^+$ , spunem ca  $F$  si  $G$  sunt **echivalente**. Daca  $F$  si  $G$  sunt echivalente, spunem ca  $F$  **acopera**  $G$  (si  $G$  acopera  $F$ ).
- Pentru a stabili ca  $F$  si  $G$  sunt echivalente (sau  $F = G$ ), pentru fiecare dependenta  $Y \rightarrow Z$  din  $F$  se verifica daca  $Y \rightarrow Z$  este in  $G^+$ , folosind algoritmul anterior pentru a calcula  $Y^+$  si pentru a verifica apoi daca  $Z \subseteq Y^+$  (potrivit axiomelor, daca  $Z \subseteq Y^+$ , atunci  $Y \rightarrow Z$ ). Daca o dependenta  $Y \rightarrow Z$  din  $F$  nu este in  $G^+$ , atunci sigur  $F^+ \neq G^+$ .



# Acoperiri de multimi de dependente

- Daca fiecare dependenta din  $F$  este in  $G^+$ , atunci fiecare dependenta  $V \rightarrow W$  din  $F^+$  este in  $G^+$ ; pentru a arata ca  $V \rightarrow W$  este in  $G^+$ , se demonstreaza ca fiecare  $Y \rightarrow Z$  din  $F$  este in  $G^+$ , apoi ca  $V \rightarrow W$  este in  $F^+$ .
- Pentru a arata ca fiecare dependenta din  $G$  este, de asemenea, si in  $F^+$ , se procedeaza in mod analog.
- $F$  si  $G$  vor fi echivalente daca fiecare dependenta din  $F$  este si in  $G^+$ , iar fiecare dependenta din  $G$  este si in  $F^+$ .



# Acoperiri de multimi de dependente

- **Lema.** Fiecare multime de dependente functionale  $F$  este acoperita de o multime de dependente  $G$ , in care nicio dependenta nu are in parte dreapta mai mult de un atribut.

Demonstratie:

Fie  $G$  o multime de dependente  $X \rightarrow A$ , astfel incat pentru o dependenta  $X \rightarrow Y$  in  $F$ ,  $A$  este in  $Y$ . Atunci  $X \rightarrow A$  decurge din  $X \rightarrow Y$  prin regula de descompunere. Asadar,  $G \subseteq F^+$ . Dar  $F \subseteq G^+$ , deoarece, daca  $Y = A_1 A_2 \dots A_n$ , atunci  $X \rightarrow Y$  rezulta din  $X \rightarrow A_1$ ,  $X \rightarrow A_2$ , ...,  $X \rightarrow A_n$ , prin regula de reuniiune.



# Multime minimala de dependente

O multime de dependente  $F$  este **minimală** daca:

1. Partea dreapta a fiecarei dependente din  $F$  contine un singur atribut;
2. Pentru nicio dependenta  $X \rightarrow A$  din  $F$ , multimea  $F - \{X \rightarrow A\}$  nu este echivalenta cu  $F$ ;
3. Pentru nicio dependenta  $X \rightarrow A$  din  $F$  si pentru nicio submultime  $Z \subseteq X$ , multimea  $F - \{X \rightarrow A\} \cup \{Z \rightarrow A\}$  nu este echivalenta cu  $F$ .

De fapt, conditia a 2-a garanteaza ca nicio dependenta din  $F$  nu este redundanta, iar a 3-a ca niciun atribut din partea stanga nu este redundant. Desigur, niciun atribut din dreapta nu este redundant, deoarece fiecare parte dreapta contine un singur atribut (conditia 1).



# Multime minimală de dependente

- **Teorema.** Fiecare multime de dependente  $F$  este echivalentă cu o multime  $F'$ , care este minimală.

Exemplu:

Fie:  $F = \{AB \rightarrow C, D \rightarrow EG, C \rightarrow A, BE \rightarrow C, BC \rightarrow D, CG \rightarrow BD, ACD \rightarrow B, CE \rightarrow AG\}$ .

- În ultima lema a fost indicat – la demonstrație – “un fel” de algoritm pentru fragmentarea partilor drepte ale dependentelor funcționale.

Aplicându-l obținem următoarele dependente:

$AB \rightarrow C, C \rightarrow A, BC \rightarrow D, ACD \rightarrow B, D \rightarrow E, D \rightarrow G, BE \rightarrow C, CG \rightarrow B, CG \rightarrow D, CE \rightarrow A, CE \rightarrow G$ .



# Multime minimală de dependente

- Dependentele  $CG \rightarrow B$  și  $CE \rightarrow A$  sunt redundante.
- ✓ Sa aratam ca  $CG \rightarrow B$  este redundanta cu  $ACD \rightarrow B$ :
  - Deoarece  $C \rightarrow A$ , rezulta ca  $ACD \rightarrow B$  este echivalenta cu  $CD \rightarrow B$ ;
  - Acum trebuie sa aratam ca  $CG \rightarrow B$  este redundanta, fata de  $CD \rightarrow B$ :

Pornind de la  $CD \rightarrow B$ ,  
avem  $CG \rightarrow D$ ,  
deci inlocuim pe  $D$  in  $CD \rightarrow B$  si obtinem  
 $C(CG) \rightarrow B$ , adica  $CG \rightarrow B$ .



# Multime minimala de dependente

- Procedand astfel, se obtine **multimea de dependente minimele**:  
 $\{AB \rightarrow C, C \rightarrow A, BC \rightarrow D, CD \rightarrow B, D \rightarrow E, D \rightarrow G, BE \rightarrow C, CG \rightarrow D, CE \rightarrow G\}$ .
- Daca din F eliminam dependentele  
 $CE \rightarrow A, CG \rightarrow D$  si  $ACD \rightarrow B$ ,  
obtinem **acoperirea minimala**:  
 $F' = \{AB \rightarrow C, C \rightarrow A, BC \rightarrow D, D \rightarrow E, D \rightarrow G, BE \rightarrow C, CG \rightarrow B, CE \rightarrow G\}$ .
- Se observa ca cele doua acoperiri minimele contin un numar diferit de dependente.



# Capitolul 6

## Normalizarea schemelor de relatie



# Normalizarea schemelor de relatie

- Dupa cum am aratat in capitolele anterioare, proiectarea unei baze de date se poate face in mai multe feluri si se pune problema care este cea mai buna optiune.
- Un obiectiv important in alegerea modelului de date este realizarea unei reprezentari corecte a datelor, a relatiilor dintre ele si a restrictiilor impuse asupra lor.
- Pentru realizarea acestui obiectiv se utilizeaza tehnica normalizarii, care are ca scop principal identificarea schemei de relatii care sa modeleze cat mai corect realitatea functionala.
- Proiectarea incorecta a schemelor de relatie poate duce la aparitia diferitelor anomalii care complica procesul de dezvoltare sau exploatare a unei aplicatii software.



# Normalizarea schemelor de relatie

- Ca metoda de testare a corectitudinii unei scheme de relatie este verificarea dependentelor functionale atasate schemei.
- Una dintre regulile de baza in proiectare este aceea ca datele nu trebuie sa fie redundante, adica aceleasi date sa fie stocate de mai multe ori in aceeasi relatie sau relatii diferite.
- O alta regula este ca datele care se pot deduce prin prelucrare din alte date nu trebuie stocate in baza de date.
- **Normalizarea** reprezinta un proces de descompunere a unei relatii in mai multe relatii cu scopul eliminarii anomalilor de proiectare si exploatare a bazei de date. Procesul de normalizare se realizeaza cu ajutorul **formelor normale** care impun **regulile de descompunere**.
- O schema de relatie obtinuta in urma normalizarii , care indeplineste un set de standarde si cerintele specifice, se spune ca este intr-o **forma normala**.



# Normalizarea schemelor de relatie

- In anul 1970 Edgar F. Codd a definit formalismul primelor forme normale: Forma Normala 1(FN1), Forma Normala 2(FN2) si Forma Normala 3(FN3).
- In anul 1974, Edgar F. Codd impreuna cu Raymond F. Boyce, au definit FNBC( Forma Normala Boyce-Codd).
- Ulterior au fost definite formele normale FN4, FN5 si FN6 care au avut aplicabilitate destul de redusa in proiectarea bazelor de date.



# Forma normala 1 (FN1)

- **Definitie:** O relatie R se gaseste in **Forma Normala 1 (FN1)** daca si numai daca:
  - Pe toate atributele sale exista doar valori atomice ale datelor(nu exista attribute cu valori multiple);
  - Nu exista attribute sau grupuri de attribute care se repeta.
  - Semnificatia termenului “atomic” este similara cu cea de la modelul entitate asociere: valoarea respectiva este intotdeauna folosita ca un intreg si nu se utilizeaza niciodata doar portiuni din aceasta. In prima forma normala, domeniul fiecarui atribut este construit din valori indivizibile. Cu alte cuvinte, toate atributele trebuie sa fie atomice, adica sa contina o singura informatie.

Exemplu: ABONATI ( Cod\_abonat, Nume, Adresa, Tip\_abon). Aceasta relatie este in FN1 daca atributul Adresa este atomic, adica niciodata nu este nevoie sa fie folosite doar anumite parti ale sale (oras, sector, strada, etc.).



# Forma normala 2 (FN2)

- Fiind data o schema de relatie R si multimea de dependente functionale asociata F, putem defini inca doua concepte.  
Fie A un atribut care nu face parte din cheie si X o multime de atribute din R care formeaza o cheie a relatiei. Atunci:
  - **Definitie:** O dependenta functionala  $Y \rightarrow A$  se numeste **dependenta paritala** daca Y este strict inclusa intr-o cheie a relatiei R.  
( $X$  este cheie,  $Y \subset X$ ,  $Y \rightarrow A$  si  $A$  nu face parte din cheie) sau  
(avem dependenta  $X \rightarrow Y \rightarrow A$ , unde:  $X=cheie$ ,  $Y \subset X$  si  $A \notin X$  ).
  - **Definitie:** O dependenta functionala  $Y \rightarrow A$  se numeste **dependenta tranzitiva** daca Y nu este inclusa in nicio cheie a relatiei R.  
( $X$  este cheie,  $Y \not\subset X$ ,  $Y \rightarrow A$  si  $A$  nu face parte din cheie) sau  
(avem dependenta  $X \rightarrow Y \rightarrow A$ , unde:  $X=cheie$ ,  $Y \not\subset X$  si  $A \notin X$  ).



# Forma normala 2 (FN2)

- **Definitie:** Fie R o schema de relatie si F multimea de dependente functionale asociata. Relatia R este in **Forma Normala 2 (FN2)** daca si numai daca F respecta cerintele FN1 si nu contine dependente partiale (dar poate contine dependente tranzitive).

Cu alte cuvinte, o relatie se gaseste **Forma Normala 2** daca si numai daca:

- Se gaseste in FN1 si
- Orice atribut care nu face parte din cheie va fi identificat de intreaga cheie, nu doar de unele atribute care fac parte din cheie.



# Forma normala 2 (FN2)

Observatie:

- Daca o entitate se gaseste in FN1 si cheia sa este formata dintr-un singur atribut, atunci se gaseste automat si in FN2.
- FN1 si FN2 nu garanteaza eliminarea anomalilor si nu sunt recomandate pentru proiectarea schemelor de relatii ale unei baze de date.



# Forma normala 2 (FN2)

Exemplu:

1) Relatia PRODUSE = Cod\_prod, Den\_prod, Pret,  
Cod\_furniz, Den\_furniz, Adresa

cu dependentele functionale:

$F = \{ \text{Cod\_prod} \rightarrow \text{Den\_prod}, \text{Cod\_prod} \rightarrow \text{Pret},$   
 $\text{Cod\_prod} \rightarrow \text{Cod\_furniz}, \text{Cod\_furniz} \rightarrow \text{Den\_furniz},$   
 $\text{Cod\_furniz} \rightarrow \text{Adresa} \}$  si cheia unica Cod\_prod.

- Relatia este in FN2 deoarece cheia are un singur atribut. Relatia nu are dependente partiale iar ultimele doua dependente sunt dependente tranzitive deoarece Cod\_furniz nu apartine cheii unice Cod\_prod.



# Forma normala 2 (FN2)

## 2) Relatia

NOTE = Nr\_matricol, Nume\_stud, Cod\_discip,  
Den\_discip, Nota, Data\_ex

are cheia (Nr\_matricol, Cod\_discip),

considerand ca un student are o singura nota la o disciplina (nota finala).

- Relatia NOTE nu este in FN2 deoarece dependentele  
 $\text{Nr\_matricol} \rightarrow \text{Nume\_stud}$  si  $\text{Cod\_discip} \rightarrow \text{Den\_discip}$   
sunt dependente partiale ( $\text{Nr\_matricol}$ ,  $\text{Cod\_discip}$  fac parte din cheie).



# Forma normala 3 (FN3)

- Pentru a defini forma FN3 este necesara definirea notiunii de **atribut prim**:
- **Definitie.** Fie R o schema de relatie si F multimea de dependente functionale asociata.

Un atribut  $A \in R$  se numeste **atribut prim** daca el apartine unei chei a lui R.

Exemplu:

Fie  $R = ABCDE$  avand  $F = \{ A \rightarrow B, B \rightarrow A, A \rightarrow C, D \rightarrow E \}$ . Cum cheile relatiei sunt AD si BD rezulta ca in R sunt trei atribute prime: A, B si D.



# Forma normala 3 (FN3)

- **Definitie.** Fie R o schema de relatie si F multimea de dependente functionale asociata.

Relatia R este in **Forma Normala 3(FN 3)** daca si numai daca oricare ar fi o dependenta netriviala

$X \rightarrow A$  din F, atunci:

- X este supercheie pentru R  
sau
- A este atribut prim.



# Forma normala 3 (FN3)

- De remarcat ca daca in F avem dependente care contin mai multe atribute in partea dreapta putem aplica regula de descompunere pentru a obtine dependente care in partea dreapta au cate un singur atribut.
- Observatie : O relatie se gaseste in FN3 daca si numai daca se gaseste in FN2 si, in plus, niciun atribut care nu este parte a unei chei nu depinde de un alt atribut care nu face parte din cheie. Cu alte cuvinte, nu se accepta dependente tranzitive, adica un atribut sa depinda de o cheie in mod indirect.



# Forma normala 3 (FN3)

Exemple:

- 1) Fie  $R=NLAP$ , o schema de relatie pentru furnizori cu atributele (Nume, Localitate, Articol, Pret) si dependentele  $F=\{NA \rightarrow P \text{ si } N \rightarrow L\}$ .
  - Relatia are cheia NA si nu respecta FN3 deoarece are dependenta partiala  $N \rightarrow L$ (de asemenea nu respecta nici FN2). Intr-adevar, fie  $X=NA$ ,  $Y=N$ . Atributul L (localitate) este neprim, deoarece singura cheie este NA. Atunci  $X \rightarrow Y$  si  $Y \rightarrow L$  sunt dependente valabile, pe cand  $Y \rightarrow X$  (adica  $N \rightarrow NA$ ) nu este valabila. Vom observa ca in acest caz,  $X \rightarrow Y$  si  $Y \rightarrow L$  nu numai ca “functioneaza” in  $R$ , dar ele sunt dependente date. In general, este suficient ca  $X \rightarrow Y$  si  $Y \rightarrow L$  sa decurga dintr-o multime data de dependente, chiar daca sunt date ca atare.
- 1) Fie  $R=OSC$ , o schema de relatie pentru localitati cu atributele (Oras, Strada, Cod) si dependentele  $F=\{ OS \rightarrow C \text{ si } C \rightarrow O\}$ . Cheile sunt OS si SC. Relatia are toate atributele prime si este in FN3.



# Forma normala 3 (FN3)

3) Fie  $R=MADS$ , o relatie pentru magazine cu atributele (Magazin, Articol, Departament, Sef). Presupunem ca functioneaza urmatoarele dependente functionale:

- $MA \rightarrow D$  (fiecare articol, in fiecare magazin , este vandut de cel mult un departament/raion);
  - $MD \rightarrow S$  (fiecare departament/raion , din fiecare magazin, are un sef).
- 
- Relatia nu este in FN3 deoarece:
    - Relatia are o singura cheie , pe MA.
    - Daca notam  $X=MA$  si  $Y=MD$ , atunci  $X \rightarrow D$  si  $Y \rightarrow S$  nu respecta regulile care definesc FN3 deoarece D si S nu sunt attribute prime.
  - Relatia este in FN2 pentru ca nu exista dependente partiale (nicio submultime proprie a cheii MA nu determina functional attributele D sau S ).



# Forma normala 3 (FN3)

## ✓ Necesitatea FN3

- Asa cum am aratat, prin FN3 se evita multe dintre probleme legate de redundanta si de anomaliiile de actualizare.
- Putem presupune, astfel, ca dependentele functionale  $X \rightarrow Y$  nu reprezinta numai o restrictie de integritate asupra relatiilor, ci reprezinta, in acelasi timp, o legatura (asociere) pe care baza de date "are intentia sa o memoreze". Cu alte cuvinte, daca atributelor din X le este asignata o multime de valori, consideram important sa stim ce valoare, pentru fiecare atribut din Y, este asociata cu aceasta "asignare" de valori pentru atributele din X.



# Forma normala 3 (FN3)

- Daca avem o dependenta partiala  $Y \rightarrow A$ ,  $X$  fiind o cheie iar  $Y$  o submultime proprie a lui  $X$ , atunci in fiecare tuplu folosit pentru a asocia o valoare, din multimea asignata lui  $X$ , cu valori pentru alte atrbute in afara de  $A$  si de atrbutele din  $X$ , trebuie sa apară aceeasi asociere intre  $X$  si  $A$ .
- Aceasta situatie este usor de evideniat in relatia  $R=NLAP$ , cu cheia NA in care  $N \rightarrow L$  este o dependenta partiala, iar localitatea furnizorului trebuie sa fie repetata pentru fiecare articol livrat de furnizor.  
Evident ca FN3 elimina aceasta posibilitate, precum si redundantele respective si anomaliile de actualizare.



# Forma normala 3 (FN3)

- In caz ca exista o dependenta tranzitiva  $X \rightarrow Y \rightarrow A$ ,  
atunci nu putem asocia o valoare Y cu o valoare X,  
daca nu exista o valoare A asociata cu valoarea Y.  
Aceasta situatie conduce la anomalii de inserare si de  
stergere deoarece nu putem insera o asociere  
X-la-Y fara o asociere Y-la-A, iar daca stergem  
valoarea A asociata cu o valoare Y data vom “pierde  
legatura” unei asocieri X-la-Y.
- De exemplu, in schema de relatie R= MADS, cu  
dependentele F={MA $\rightarrow$ D si MD $\rightarrow$ S}, nu putem  
inregistra un departament oarecare daca acel  
departament nu are sef, iar daca stergem un sef  
“dispare” si departamentul asociat lui.



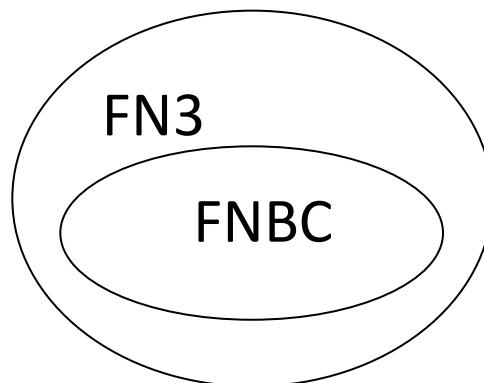
# Forma normala Boyce-Codd (FNBC)

- **Definitie.** Fie R o schema de relatie si F multimea de dependente functionale asociata. Se spune ca R este in **Forma Normala Boyce-Codd (FNBC)** daca si numai daca oricare ar fi o dependenta netriviala  $X \rightarrow Y$  din F atunci X este supercheie pentru R.
- Rezulta ca o relatie este in FNBC daca si numai daca fiecare dependenta din F are in partea stanga o supercheie.
- Nu este obligatoriu ca F sa fie in forma canonica, dar trebuie sa nu contine dependente triviale (obtinute din prima axioma de reflexivitate, de tipul  $AB \rightarrow A$  sau  $AB \rightarrow AB$ ).
- Dependenta triviala  $R \rightarrow R$  este admisa in FNBC(o relatie nu contine linii dupicat si cheia este compusa din toate atributele).



# Forma normala Boyce-Codd (FNBC)

- Observatie: Conditia de FNBC este inclusa in definitia FN3. Din acest motiv orice relatie care este in FNBC este implicit si in FN3. Reciproca nu este adevarata. Rezulta de asemenea ca daca o schema de relatie nu este in FN3 ea nu poate fi nici in FNBC.





# Forma normala Boyce-Codd (FNBC)

Exemplu:

- 1) Relatia  $R = ABCDE$  avand  $F = \{ A \rightarrow B, B \rightarrow A, A \rightarrow C, D \rightarrow E \}$  nu este in forma normala Boyce-Codd deoarece are cheile AD si BD dar nicio dependenta nu are in partea stanga o supercheie a lui R.
- 2) Relatia  $R = ABCD$  avand  $F = \{ AB \rightarrow C, AB \rightarrow D, D \rightarrow A \}$  are cheia unica AB.
  - Relatia este in FN3 deoarece primele doua dependente au in partea stanga o supercheie (AB) iar a treia dependenta are in partea dreapta atributul prim A.
  - Relatia nu este in FNBC deoarece a treia dependenta violeaza definitia pentru aceasta forma normala (nu are in partea stanga o supercheie).



# Forma normala Boyce-Codd (FNBC)

3) Relatia  $R = ABCDE$  avand  $F = \{ A \rightarrow B, B \rightarrow A, A \rightarrow C, D \rightarrow E \}$  are cheile AD si BD. Rezulta ca:

- R nu este in FN3 deoarece dependentele 3 si 4 nu au nici supercheie in partea stanga si nici atribut prim in partea dreapta;
- R nu este in FNBC deoarece nu e in FN3.

4) Relatia  $R = ABCD$  avand  $F = \{ A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A \}$  are cheile A, B, C si D. Rezulta ca:

- R este in FNBC deoarece in partea stanga a dependentelor sunt numai superchei;
- R este in FN3 deoarece este in FNBC.



# Forma normala Boyce-Codd (FNBC)

5) Relatia PRODUSE\_FURNIZORI = Cod\_prod, Den\_prod, Pret,  
Cod\_furniz, Den\_furniz, Adresa  
cu dependentele functionale:

$$F = \{ \text{Cod\_prod} \rightarrow \text{Den\_prod}, \text{Cod\_prod} \rightarrow \text{Pret}, \text{Cod\_prod} \rightarrow \text{Cod\_furniz}, \text{Cod\_furniz} \rightarrow \text{Den\_furniz}, \text{Cod\_furniz} \rightarrow \text{Adresa} \}$$

- Relatia nu este in FNCB deoarece cheia unica este Cod\_prod dar exista dependente care nu au in partea stanga o supercheie:

$$\text{Cod\_furniz} \rightarrow \text{Den\_furniz}, \text{Cod\_furniz} \rightarrow \text{Adresa}$$

- Relatia nu este nici in FN3 deoarece Den\_furniz si Adresa nu sunt atribute prime, in plus aceste dependente sunt si tranzitive.



# Forma normala Boyce-Codd (FNBC)

6) Relatia PRODUSE\_FURNIZORI = Cod\_prod, Den\_prod, Pret, Cod\_furniz, Den\_furniz, Adresa cu dependentele:  
 $F = \{ \text{Cod\_prod} \rightarrow \text{Den\_prod}, \text{Cod\_prod} \rightarrow \text{Pret},$   
 $\text{Cod\_prod} \rightarrow \text{Cod\_furniz}, \text{Cod\_furniz} \rightarrow \text{Den\_furniz},$   
 $\text{Cod\_furniz} \rightarrow \text{Adresa} \}$

- Consideram proiectia

PRODUSE= Cod\_prod, Den\_prod, Pret, Cod\_furniz

$F_{\text{PRODUSE}} = \pi_{\text{PRODUSE}}(F) = \{ \text{Cod\_prod} \rightarrow \text{Den\_prod},$   
 $\text{Cod\_prod} \rightarrow \text{Pret}, \text{Cod\_prod} \rightarrow \text{Cod\_furniz} \}$  care este in FNCB deoarece cheia relatiei este Cod\_prod si toate dependentele au in partea stanga o supercheie (asa cum s-a mentionat, orice cheie este in acelasi timp si supercheie). Proiectia este in FN3 deoarece este in FNCB.



# Anomalii in baze de date

Anomaliiile care pot sa apară într-o bază de date sunt de două feluri:

- Anomalii de **proiectare**: redundante, tipuri gresite de atribut, constrangeri incorecte, etc.
- Anomalii de **funcționare**: sunt cele care apar la operații DML (inserare, modificare, stergere).
- Pentru a înțelege mai bine anomaliiile să considerăm relația NOTE (în care un student are o singură nota la o disciplină):

NOTE = Matricol, Nume, Disciplina, Nota, Data\_ex,  
Cod\_spec, Den\_spec  
cu următoarele date:



# Anomalii in baze de date

Matricol	Nume	Disciplina	Nota	Data_ex	Cod_spec	Den_spec
1011	Ionescu Silvia	BD1	9	08.06.2020	1	CTI
1011	Ionescu Silvia	BD2	10	10.02.2021	1	CTI
1022	Popescu Daniel	PM	8	11.02.2021	2	AII
1022	Popescu Daniel	SO	7	03.06.2020	2	AII
1033	Popa Cornel	BD1	8	08.06.2020	1	CTI
1033	Popa Cornel	BD2	8	10.02.2021	1	CTI



# Anomalii in baze de date

➤ **Redundanta** apare atunci cand datele sunt stocate de mai multe ori in baza de date(in aceeasi relatie sau in relatii diferite).

In relatia NOTE se observa ca atributul Den\_spec este redundant pentru ca este suficient atributul Cod\_spec care specifica codul specializarii studentului si determina unic denumirea acesteia. In acest caz ar trebui sa se descompuna relatia in doua relatii , NOTE si SPECIALIZARI .



# Anomalii in baze de date

Existenta redundantei intr-o baza de date produce urmatoarele efecte negative:

- Alocare suplimentara de spatiu fizic;
- Inconsistenta datelor;
- Cresterea costurilor de acces la baza de date;
- Prelucrari eronate ale datelor;
- Creste probabilitatea erorii umane in timpul actualizarilor pe baza de date.



# Anomalii in baze de date

- **Tipurile atributelor** pot genera anomalii, de exemplu daca atributul Data\_ex este definit de tip Number, in loc de tip Date, atunci cand se insereaza data se va genera eroare de format. Daca este definit Text pot aparea erori la anumite functii de tip data calendaristica.
- **Constrangerile** incorecte apar in momentul definirii relatiilor dar efectul lor se vede, de regula, in timpul functionarii. De exemplu, daca in tabela NOTE se defineste cheia relatiei ca fiind Matricol, atunci nu se poate insera decat nota la o singura disciplina, a doua operatie va genera o eroare de violare de cheie.



# Anomalii in baze de date

- **Inserarea** poate genera anomalii, de exemplu codurile specializarilor sa nu corespunda cu denumirile de specializare sau sa fie folosite coduri diferite pentru aceeasi specializare. Anomalia se inlatura daca se creeaza relatia SPECIALIZARI(Cod\_spec, Den\_spec). O alta anomalie este ca nu se stiu numarul matricol si specializarea unui student pana cand nu se insereaza cel putin o nota la o disciplina.
- **Modificarea** datelor poate duce la inconsistenta datelor, de exemplu daca un student isi schimba specializarea trebuie sa se modifice Cod\_spec, Den\_spec la toate inregistrarile si este posibil sa fie omise unele dintre ele.



# Anomalii in baze de date

- **Stergerea** poate produce anomalii, de exemplu daca se sterg toate inregistrarile aferente unui student atunci nu se mai stie numarul matricol si nici specializarea lui. In acest caz se poate crea relatia STUDENTI(Matricol, Nume, Cod\_spec) si anomalia este evitata.
- Majoritatea anomalilor semnalate au aparut deoarece structura relatiei NOTE contine doua clase de obiecte care trebuie separate. Daca folosim modelul Entitate-Asociere diagrama corecta este cea din Figura 2.



# Anomalii in baze de date

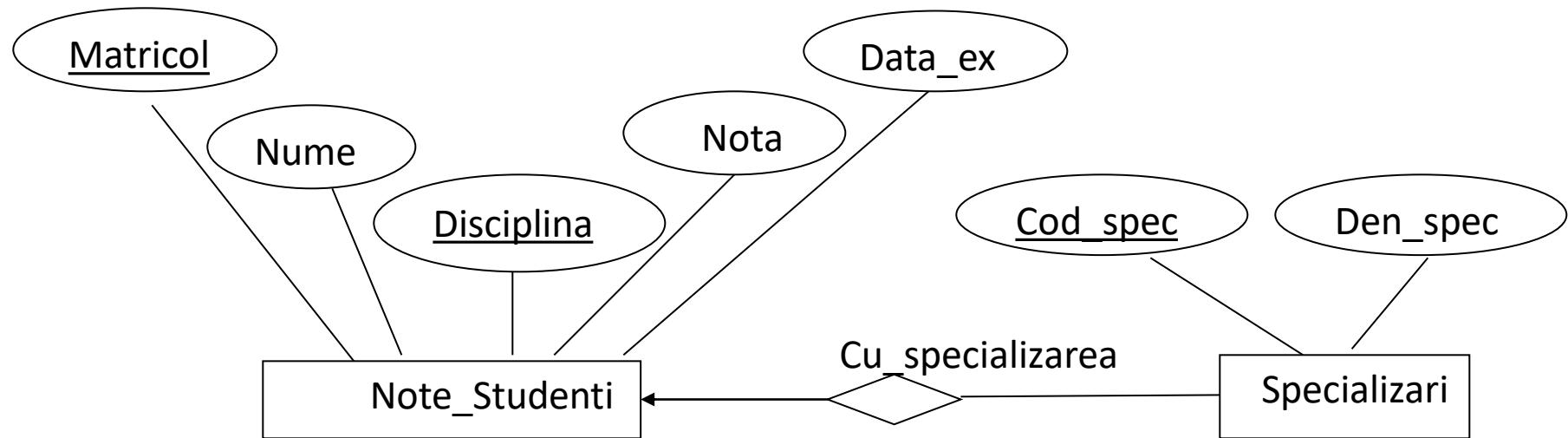


Figura 2. Diagrama entitate-asociere reproiectata



# Anomalii in baze de date

- Prin transformarea diagramei initiale se obtin urmatoarele relatii:  
NOTE\_STUDENTI = Matricol, Nume, Disciplina, Nota,  
Data\_ex, Cod\_spec  
SPECIALIZARI = Cod\_spec, Den\_spec
- In acest caz tabelele vor contine urmatoarele date:

SPECIALIZARI

Cod_spec	Den_spec
1	CTI
2	AII



# Anomalii in baze de date

## NOTE\_STUDENTI

Matricol	Nume	Disciplina	Nota	Data_ex	Cod_spec
1011	Ionescu Silvia	BD1	9	08.06.2020	1
1011	Ionescu Silvia	BD2	10	10.02.2021	1
1022	Popescu Daniel	PM	8	11.02.2021	2
1022	Popescu Daniel	SO	7	03.06.2020	2
1033	Popa Cornel	BD1	8	08.06.2020	1
1033	Popa Cornel	BD2	8	10.02.2021	1

- Procesul de “spargere” a unei tabele, care are o structura incorecta, in doua sau mai multe tabele se numeste descompunerea schemei de relatii si va fi prezentat intr-un alt capitol.



# Anomalii in baze de date

Existenta anomalilor de proiectare poate crea o serie de probleme, cum ar fi:

- Cresterea costurilor prin cresterea spatiului de stocare a datelor;
- Aparitia inconsistentei datelor(valori diferite ale datelor pentru acelasi identificator);
- Nerespectarea standardelor pentru baze de date;
- Imposibilitatea aplicarii constrangerilor de integritate;
- Extragerea de informatii eronate in urma prelucrarii datelor.

Atunci cand se proiecteaza o baza de date trebuie facuta o analiza atenta pentru a vedea cum trebuie grupate atributele in relatii cu scopul de a minimiza redundanta datelor si implicit spatiul fizic alocat.



# Independenta datelor

- O baza de date poate fi accesata de mai multe aplicatii, sau module ale aceleiasi aplicatii, in mod concurrent. In acest caz se pune problema in ce mod sunt afectate aplicatiile de modificarile efectuate in structura bazei de date. In mod normal, modificarile majore de structura(nivel inferior) implica si modificari la nivelul aplicatiei(nivel superior).
- Acest aspect tine de independenta datelor si poate fi vazuta sub doua aspecte:
  - independenta logica;
  - independenta fizica.



# Independenta datelor

- **Independenta logica** a datelor se refera la imunitatea schemelor externe fata de modificarile efectuate in schema conceptuala, cum ar fi:
  - adaugarea de entitati sau atribute noi;
  - modificarea structurii entitatilor;
  - stergerea anumitor entitati sau atribute;
  - adaugarea, modificarea sau stergerea de obiecte in baza de date(tabele,view-uri,proceduri,functii, etc.);
  - activarea sau dezactivarea unor constrangeri.
- Cu alte cuvinte, independenta logica a datelor permite sa se faca modificari in structura bazei de date fara a implica rescrierea programelor aplicatiei.



# Independenta datelor

- **Independenta fizica** de date se refera la imunitatea schemei conceptuale fata de modificarile efectuate in schema interna, cum ar fi:
  - reorganizarea fisierelor de date;
  - introducerea de noi dispozitive de stocare;
  - mutarea bazei de date de pe un server pe altul;
  - modificari in configurarea retelei, in cazul bazelor de date distribuite;
  - replicarea datelor din/pe mai multe servere;
  - utilizarea serverelor redundante pentru o securitate sporita a datelor.



# Independenta datelor

- Independenta datelor, de cele mai multe ori, este o problema foarte grea chiar si in cazul sistemelor de ultima generatie. Limbajele relationale au adus imbunatatiri majore in acest sens insa complexitatea mare a aplicatiilor face ca problema independentei sa fie un concept care trebuie in continuare studiat si imbunatatit.
- Instrumentele moderne de proiectare software ofera facilitatea de modificare in cascada a obiectelor relate, atunci cand unul dintre obiecte sufera alterari de structura.



# Independenta datelor

- În cazul instrumentelor CASE este posibil, uneori, ca o simplă recompilare sau o nouă generare a aplicatiei să preia toate modificările de structură intervenite la nivelul bazei de date. Sunt însă puține astfel de instrumente și de regulă rezolvă parțial problema, în sensul că trebuie făcute și intervenții manuale ale dezvoltatorilor. Pot fi și situații când o aplicatie nu este afectată de modificarea bazei de date, de exemplu dacă se adaugă entități sau atribută noi, funcționarea aplicatiei nu este afectată. Însă aceste modificări sunt făcute, de regulă, cu scopul de a îmbunătăți sau adăuga noi funcționalități în aplicatie și în acest caz trebuie intervenit și la nivel extern.



# Independenta datelor

- In cazul cererilor de interogare sunt situatii cand nu trebuie rescrise, de exemplu daca se adauga noi entitati sau se modifica relatiile dintre entitati fara afectarea datelor existente in baza de date.
- Acelasi lucru se intampla si in cazul vederilor(*view*) create pe una sau mai multe entitati, daca atributele initiale nu sunt alterate nu trebuie recreata vederea, dar si aici trebuie studiate cu atentie implicatiile care pot sa apară.



# Capitolul 7

## Descompunerea schemelor de relatie



# Descompunerea schemelor de relatie

Asa cum s-a mentionat anterior, in cazul in care o relatie din baza de date nu este intr-o forma normala adecvata (FN3, FNBC) pot sa apară diverse anomalii. Solutia este inlocuirea relatiei respective cu doua sau mai multe relatii care sa contina aceleasi informatii dar care, fiecare in parte, sa fie in forma normala dorita de proiectant.

- **Definitia descompunerii unei scheme de relatie**

Procesul prin care se divide o relatie in mai multe relatii se numeste ***descompunerea unei scheme de relatie***.

Formal, putem defini acest concept astfel:



# Descompunerea schemelor de relatie

- **Definitie:** Fie R o schema de relatie,  $R = A_1 A_2 \dots A_m$ . Se spune ca  $\rho = (R_1, R_2, \dots, R_n)$  este o **descompunere** a lui R daca si numai daca

$$R = R_1 \cup R_2 \cup \dots \cup R_n$$

- Schemele  $R_1, R_2, \dots, R_n$  contin deci atribute din R, fiecare atribut  $A_i$  al schemei initiale trebuind sa se regaseasca in cel putin una dintre ele.
- Nu este necesar ca schemele sa fie disjuncte, in practica ele au de multe ori atribute comune.



# Descompunerea schemelor de relatie

Exemple:

In exemplele de mai jos sunt prezentate cateva descompuneri valide ale unor scheme de relatii (unele sunt insa incorecte din punct de vedere al pastrarii datelor si/sau dependentelor initiale):

- 1) Fie relatia  $R = ABCDE$  avand

$$F = \{ A \rightarrow B, B \rightarrow A, A \rightarrow C, D \rightarrow E \}.$$

Putem avea descompuneri ca:

$$\rho_1 = (ABC, DE)$$

$$\rho_2 = (ABCD, DE)$$

$$\rho_3 = (AB, CD, DE)$$



# Descompunerea schemelor de relatie

2) Fie relatia

Produse = IdP, NumeP, Cant, IdF, NumeF, AdresaF  
avand dependentele functionale:

$$F = \{ IdP \rightarrow NumeP, IdP \rightarrow Cant, IdP \rightarrow IdF, \\ IdF \rightarrow NumeF, IdF \rightarrow AdresaF \}$$

- Putem avea mai multe descompuneri, printre care:

$$\rho_1 = ( (IdP, NumeP, Cant, IdF); (NumeF, AdresaF) )$$

$$\rho_2 = ( (IdP, NumeP, Cant, IdF); (IdF, NumeF, AdresaF) )$$

$$\rho_3 = ( (IdP, NumeP); (Cant, IdF); (NumeF, AdresaF) )$$



# Descompunerea schemelor de relatie

- Descompunerea actioneaza deci la nivelul ***schemei*** relatiei. Ce se intampla insa cu ***continutul*** acesteia in cazul unei descompuneri?
- Fiecare relatie rezultata va mosteni o parte dintr-datele relatiei descompuse si anume proiectia acesteia pe multimea de atributi a relatiei rezultata din descompunere.
- Sa consideram o instanta **r** a relatiei din schema **R** (instanta unei relatii este o incarcare cu date corecte a acesteia). Atunci instantele pentru relatiile din descompunerea p sunt:

$$r_i = \pi_{R_i}(r)$$



# Descompunerea schemelor de relatie

Exemplu:

Fie relatia PRODUSE de mai jos:

<b>IdP</b>	<b>NumeP</b>	<b>Cant</b>	<b>IdF</b>	<b>NumeF</b>	<b>AdresaF</b>
<b>101</b>	<b>Imprimanta laser</b>	<b>30</b>	<b>20</b>	<b>Xerox</b>	<b>Str. D. Danielopolu, nr. 4-6, Sector 1, Bucuresti</b>
<b>105</b>	<b>Calculator PC</b>	<b>20</b>	<b>23</b>	<b>IBM</b>	<b>Bd. D.Cantemir, nr.1, Sector 3, Bucuresti</b>
<b>124</b>	<b>Copiator</b>	<b>10</b>	<b>20</b>	<b>Xerox</b>	<b>Str. D. Danielopolu, nr. 4-6, Sector 1, Bucuresti</b>



# Descompunerea schemelor de relatie

1. In cazul descompunerii

$\rho_1 = ( (\text{IdP}, \text{NumeP}, \text{Cant}, \text{IdF}); (\text{NumeF}, \text{AdresaF}) )$   
obtinem:

$r_1$

<b>IdP</b>	<b>NumeP</b>	<b>Cant</b>	<b>IdF</b>
101	Imprimanta laser	30	20
105	Calculator PC	20	23
124	Copiator	10	20

$r_2$

<b>NumeF</b>	<b>AdresaF</b>
Xerox	Str. D. Danielopolu, nr. 4-6, Sector 1, Bucuresti
IBM	Bd. D.Cantemir, nr.1, Sector 3, Bucuresti



# Descompunerea schemelor de relatie

2. In cazul descompunerii

$\rho_2 = ( (\text{IdP}, \text{NumeP}, \text{Cant}, \text{IdF}); (\text{IdF}, \text{NumeF}, \text{AdresaF}) )$

obtinem :

$r_1$

<b>IdP</b>	<b>NumeP</b>	<b>Cant</b>	<b>IdF</b>
101	Imprimanta laser	30	20
105	Calculator PC	20	23
124	Copiator	10	20

$r_2$

<b>IdF</b>	<b>NumeF</b>	<b>AdresaF</b>
20	Xerox	Str. D. Danielopolu, nr. 4-6, Sector 1, Bucuresti
23	IBM	Bd. D.Cantemir, nr.1, Sector 3, Bucuresti



# Descompunerea schemelor de relatie

## 3. In cazul descompunerii

$\rho_3 = ( (\text{IdP}, \text{NumeP}); (\text{Cant}, \text{IdF}); (\text{NumeF}, \text{AdresaF}) )$   
obtinem:

$r_1$

<b>IdP</b>	<b>NumeP</b>
101	Imprimanta laser
105	Calculator PC
124	Copiator

$r_2$

<b>Cant</b>	<b>IdF</b>
30	20
20	23
10	20

$r_3$

<b>NumeF</b>	<b>AdresaF</b>
Xerox	Str. D. Danielopolu, nr. 4-6, Sector 1, Bucuresti
IBM	Bd. D.Cantemir, nr.1, Sector 3, Bucuresti



# Descompunerea schemelor de relatie

- Observam din aceste exemple ca in cazul primei si ultimei descompuneri nu putem reconstrui prin join, sau alti operatori relationali, relatia initiala. In cazul in care descompunerea nu s-a facut corect putem pierde:
  - datele relatiei initiale;
  - dependentele functionale ale relatiei initiale.
- In paragrafele urmatoare sunt prezentati algoritmi prin care putem detecta daca prin descompunere se pierd date sau dependente.



# Descompunere cu join fara pierdere de date

## ➤ Descompunere cu join fara pierdere de date (j.f.p)

Conditia pentru a nu se pierde date prin descompunere este ca relatia initiala sa poata fi reconstruita in totalitate prin join-ul natural al relatiilor rezultante, fara tupluri in minus sau in plus.

Formal, definitia este urmatoarea:

- **Definitie:** Fie R o schema de relatie, F multimea de dependente functionale asociata si o descompunere  $\rho = (R_1, R_2, \dots, R_n)$  a lui R. Se spune ca  $\rho$  este o descompunere ***cu join fara pierderi in raport cu F*** (prescurtat j.f.p.) daca si numai daca pentru orice instanta r a lui R, care satisface dependentele F, avem:

$$r_1 \bowtie r_2 \bowtie \dots \bowtie r_n = r \quad \text{unde} \quad r_i = \pi_{R_i}(r)$$



# Descompunere cu join fara pierdere de date

- În exemplul de la paragraful anterior doar descompunerea  $\rho_2 = ((\text{IdP}, \text{NumeP}, \text{Cant}, \text{IdF}); (\text{IdF}, \text{NumeF}, \text{AdresaF}))$  are proprietatea j.f.p, în cazul celorlalte, din cauza inexistentei coloanelor comune, joinul natural nu se poate efectua.
- Faptul că o descompunere are proprietatea j.f.p se poate testa pornind doar de la lista atributelor relației initiale, lista atributelor relațiilor din descompunere și a multimii de dependente funcționale asociată folosind algoritmul următor:



# Algoritm de testare a proprietatii j.f.p.

- **Algoritm de testare a proprietatii de j.f.p. pentru o descompunere**
- **Intrare:** Schema de relatie  $R = A_1 A_2 \dots A_m$ , multimea de dependente functionale  $F$  si o descompunere  $\rho = (R_1, R_2, \dots, R_n)$ .
- **Iesire:** Decizia daca  $\rho$  are sau nu proprietatea j.f.p.
- **Metoda:**
  - ✓ Se construieste un tabel avand **n** linii si **m** coloane.  
Liniile sunt etichetate cu elementele descompunerii  $\rho$  iar coloanele cu atributele relatiei  $R$ . Elementul  $(i,j)$  al tabelului va fi egal cu  $a_i$  daca  $A_j \in R_i$  sau  $b_{ij}$  in caz contrar.



# Algoritm de testare a proprietatii j.f.p.

- ✓ Se parcurg dependentele  $X \rightarrow Y$  din F. Daca doua (sau mai multe) linii din tabel au aceleasi simboluri pe coloanele X aceste linii se egaleaza si pe coloanele din Y astfel:
  - Daca pe o coloana din Y apare un  $a_j$  atunci toate elementele de pe acea coloana din liniile respective devin  $a_j$ .
  - Daca pe o coloana din Y nu apare niciun  $a_j$  atunci se alege unul dintre elementele de tip  $b_{ij}$  si toate elementele de pe acea coloana din liniile respective devin egale cu acel  $b_{ij}$ .



# Algoritm de testare a proprietatii j.f.p.

- ✓ Procesul se opreste:
  - Fie cand s-a obtinut o linie in tabel care contine doar **a-uri**, caz in care descompunerea **p are proprietatea j.f.p.**
  - Fie cand la o parcurgere a dependentelor nu mai apar schimbari in tabel si nu s-a obtinut o linie doar cu **a-uri**. In acest caz descompunerea **p nu are proprietatea j.f.p.**

In literatura de specialitate se poate gasi demonstratia faptului ca acest algoritm determina corect daca o descompunere are proprietatea j.f.p.



# Algoritm de testare a proprietatii j.f.p.

Exemple:

- 1) Fie  $R = ABCDE$ ,  $F = \{ A \rightarrow B, A \rightarrow C, A \rightarrow D, D \rightarrow E \}$  si o descompunere a lui  $R$ ,  $\rho = (ABCD, DE)$   
Construim tabelul aplicand algoritmul:

	A	B	C	D	E
ABCD	a1	a2	a3	a4	b15 a5
DE	b21	b22	b23	a4	a5

La prima parcurgere, pentru dependentele  $A \rightarrow B$ ,  $A \rightarrow C$ ,  $A \rightarrow D$  nu gasim doua linii cu aceleasi valori pe coloana A dar pentru dependenta  $D \rightarrow E$  cele doua linii sunt egale pe coloana D (simbolul a4). Le egalam si pe coloana E: cum pe aceasta coloana exista a5 rezulta ca b15 devine egal cu a5. S-a obtinut o linie numai cu **a**-uri, deci descompunerea are proprietatea de join fara pierderi.



# Algoritm de testare a proprietatii j.f.p.

2) Fie relatia  $R = ABCDE$  si  $F = \{ A \rightarrow B, AC \rightarrow D, D \rightarrow E \}$  si descompunerea  $\rho = (AB, BC, CDE)$ . Tabelul este urmatorul:

	A	B	C	D	E
AB	a1	a2	b13	b14	b15
BC	b21	a2	a3	b24	b25
CDE	b31	b32	a3	a4	a5

La prima trecere nu apar modificari in tabel.

Procesul se opreste si  $\rho$  nu are proprietatea j.f.p.



# Algoritm de testare a proprietatii j.f.p.

3) Fie  $R = ABCDE$ ,

$$F = \{ C \rightarrow E (1), A \rightarrow C (2), B \rightarrow D (3), D \rightarrow E (4), E \rightarrow B (5) \}$$

(cu dependente numerotate intre paranteze) si  $\rho = (BCE, AB, ACD)$

	A	B	C	D	E
BCE	b11	a2	a3	b14	a5
AB	a1	a2	b23 (2) a3	b24 (3) b14	b25 (4) a5
ACD	a1	b32 (5) a2	a3	a4	b35 (1) a5

Prima trecere:

- Din  $C \rightarrow E$  rezulta  $b35$  devine  $a5$  (1)
- Din  $A \rightarrow C$  rezulta  $b23$  devine  $a3$  (2)
- Din  $B \rightarrow D$  rezulta  $b24$  devine  $b14$  (3)
- Din  $D \rightarrow E$  rezulta  $b25$  devine  $a5$  (4)
- Din  $E \rightarrow B$  rezulta  $b32$  devine  $a2$  (5)

Am obtinut o linie doar cu **a**-uri. Descompunerea este j.f.p.



# Algoritm de testare a proprietatii j.f.p.

## Observatie:

- In exemplele de mai sus a fost suficiente o singura trecere prin dependente. Exista insa situatii cand sunt necesare mai multe treceri pana cand procesul se opreste.
- In cazul in care descompunerea are numai doua elemente se poate testa daca are proprietatea de join fara pierderi si astfel:



# Algoritm de testare a proprietatii j.f.p.

- **Definitie:** Fie  $R$  o schema de relatie,  $F$  multimea de dependente functionale asociata si  $\rho = (R_1, R_2)$  o descompunere a sa.  
Atunci  $\rho$  are proprietatea de **join fara pierderi** daca una dintre dependentele urmatoare se poate deduce din  $F$ :
  - $(R_1 \cap R_2) \rightarrow (R_1 - R_2)$  sau
  - $(R_1 \cap R_2) \rightarrow (R_2 - R_1)$
- Cu alte cuvinte, daca una dintre aceste dependente face parte din  $F$  (sau inchiderea  $F^+$  aplicand axiome si reguli de inferenta) atunci descompunerea  $\rho$  este cu join fara pierderi.



# Algoritm de testare a proprietatii j.f.p.

Exemplu: In prima exemplificare a aplicarii algoritmului de testare aveam o descompunere cu doua elemente:  $R = ABCDE$ ,  $F = \{ A \rightarrow B, A \rightarrow C, A \rightarrow D, D \rightarrow E \}$  si  $\rho = (ABCD, DE)$ . Avem :

- $R_1 = ABCD$ ,  $R_2 = DE$ ,
- $(R_1 - R_2) = ABCD - DE = ABC$
- $(R_2 - R_1) = DE - ABCD = E$
- $(R_1 \cap R_2) = D$

Cele doua dependente sunt:

- $(R_1 \cap R_2) \rightarrow (R_1 - R_2)$  devine  $D \rightarrow ABC$  (se pot obtine prin regula de inferenta – descompunere:  $D \rightarrow A$ ,  $D \rightarrow BC$  si  $D \rightarrow C$ , dar niciuna din aceste dependente nu face parte din  $F$  sau  $F^+$ ).
- $(R_1 \cap R_2) \rightarrow (R_2 - R_1)$  devine  $D \rightarrow E$ . Cum  $D \rightarrow E$  face parte din  $F$  rezulta ca  $\rho$  are proprietatea de join fara pierderi.



# Descompuneri care pastreaza dependentele

## ➤ Descompuneri care pastreaza dependentele

- O a doua problema, in cazul descompunerii unei scheme de relatie R avand dependentele F in mai multe relatii R<sub>1</sub>, R<sub>2</sub>, ..., R<sub>n</sub>, este aceea a pastrarii corelatiilor intre date, corelatii impuse de dependentele functionale din F.
- Fiecare relatie R<sub>i</sub> va mosteni o multime de dependente data de proiectia multimii de dependente functionale F pe R<sub>i</sub>:

$$F_i = \pi_{R_i}(F)$$



# Descompuneri care pastreaza dependentele

Exemplu:

Fie relatia Produse =  $\{IdP, NumeP, Cant, IdF, NumeF, AdresaF\}$  avand dependentele functionale:

$F = \{ IdP \rightarrow NumeP, IdP \rightarrow Cant, IdP \rightarrow IdF, IdF \rightarrow NumeF, IdF \rightarrow AdresaF \}$

- În cazul descompunerii  $\rho_2 = (R1, R2)$  unde:  
 $R1 = (IdP, NumeP, Cant, IdF)$   
 $R2 = (IdF, NumeF, AdresaF)$

cele două relații mostenesc următoarele dependente:

$$F_{R1} = \pi_{R1}(F) = \{ IdP \rightarrow NumeP, IdP \rightarrow Cant, IdP \rightarrow IdF \}$$

$$F_{R2} = \pi_{R2}(F) = \{ IdF \rightarrow NumeF, IdF \rightarrow AdresaF \}$$



# Descompuneri care pastreaza dependentele

- Dupa cum se observa toate dependentele relatiei initiale sunt pastrate fie in  $F_{R1}$ , fie in  $F_{R2}$ . Exista insa si cazuri in care unele dependente din F nu mai pot fi regasite in multimile de dependente asociate schemelor din descompunere si nu se pot deduce din acestea.
- In primul caz se spune ca descompunerea pastreaza dependentele iar in al doilea ca descompunerea nu pastreaza dependentele.
- O definitie formală a acestui concept este urmatoarea:



# Descompuneri care pastreaza dependentele

- **Definitie:** Fie  $R$  o schema de relatie,  $F$  multimea de dependente functionale asociata, o descompunere  $\rho = (R_1, R_2, \dots, R_n)$  a lui  $R$  si  $F_i = \pi_{R_i}(F)$  proiectia multimii de dependente functionale(multimile de dependente functionale ale elementelor descompunerii).

Se spune ca  $\rho$  ***pastreaza dependentele*** din  $F$  daca si numai daca orice dependenta din  $F$  poate fi dedusa din  $\cup_{i=1..n} (F_i)$ .

- Rezulta ca o descompunere pastreaza dependentele daca si numai daca:

$$F \subseteq (\cup_{i=1..n} (F_i))^+$$



# Descompuneri care pastreaza dependentele

- Din pacate, atât proiecția unei multimi de dependente cat și incluziunea de mai sus implica un calcul de inchidere a unei multimi de dependente ( $F$ , respectiv reuniunea multimilor  $F_i$ ).
- Există și în acest caz un algoritm pentru a testa dacă o dependență este pastrată după descompunere, sau nu este, fără a fi necesar calculul efectiv al multimilor  $F_i^+$ .



# Algoritm de testare a pastrarii dependentelor

## ➤ Algoritm de testare a pastrarii dependentelor

- **Intrare:** O schema de relatie R, multimea de dependente functionale asociata F si o descompunere  $\rho = (R_1, R_2, \dots, R_n)$  .
- **Iesire:** Decizia daca  $\rho$  pastreaza sau nu dependentele.



# Algoritm de testare a pastrarii dependintelor

- **Metoda:** Pentru fiecare dependenta  $X \rightarrow Y$  din  $F$  se procedeaza astfel:
  - ✓ Se porneste cu o multime de atribute  $Z = X$ ;
  - ✓ Se parcurg repetat elementele descompunerii  $\rho$ . Pentru fiecare  $R_i$  se calculeaza o noua valoare a lui  $Z$  astfel:
$$Z = Z \cup ((Z \cap R_i)^+ \cap R_i);$$
  - ✓ Procesul se opreste in momentul cand  $Z$  ramane neschimbat la o parcursere a elementelor  $R_i$ . Daca  $Y \subseteq Z$  atunci dependenta  $X \rightarrow Y$  este pastrata, altfel nu e pastrata.
  - ✓ Daca toate dependentele din  $F$  sunt pastrate inseamna ca  $\rho$  **pastreaza dependentele din  $F$ .**



# Algoritm de testare a pastrarii dependintelor

Exemple:

1) Fie  $R = ABCDE$ , multimea de dependente functionale  $F = \{ C \rightarrow E, A \rightarrow C, B \rightarrow D, D \rightarrow E, E \rightarrow B \}$  si descompunerea  $\rho = (BCE, AB, ACD)$ .

- Se observa ca dependentele  $C \rightarrow E$ ,  $A \rightarrow C$  si  $E \rightarrow B$  sunt pastrate: ele apartin proiectiei lui  $F$  pe  $BCE$  (prima si ultima) si  $ACD$  (a doua).
- Raman de testat dependentele  $B \rightarrow D$  si  $D \rightarrow E$ .  
Sa aplicam algoritmul pentru  $B \rightarrow D$ :



# Algoritm de testare a pastrarii dependentelor

- ✓ Initial  $Z = B$
- 1) Prima trecerea prin elementele lui  $\rho$ :
  - Pentru BCE:
$$Z = B \cup ((B \cap BCE)^+ \cap BCE) = B \cup (BDE \cap BCE) = BE$$

deoarece  $(B \cap BCE)^+ = B^+$  iar calculul lui  $B^+$  este urmatorul:

- $X^{(0)} = \{B\}$
- $X^{(1)} = \{B\} \cup \{D\} = BD$
- $X^{(2)} = \{BD\} \cup \{DE\} = BDE$
- $X^{(3)} = \{BDE\} \cup \{DEB\} = BDE$



# Algoritm de testare a pastrarii dependintelor

- Pentru AB:

$$Z = BE \cup ((BE \cap AB)^+ \cap AB) = BE \cup (BDE \cap AB) = BE$$

- Pentru ACD:

$$Z = BE \cup ((BE \cap ACD)^+ \cap AB) = BE \cup \emptyset = BE$$

- ✓ Deoarece  $Z=BE$  ramane neschimbat dupa efectuarea trecerilor, procesul se opreste.
- ✓ Cum  $\{D\} \not\subset BE$  rezulta ca dependenta  $B \rightarrow D$  nu este pastrata, deci  $\rho$  nu pastreaza dependentele.



# Algoritm de testare a pastrarii dependintelor

2) Fie schema de relatie  $R = ABCD$ ,  $F = \{ A \rightarrow B, A \rightarrow C, C \rightarrow D, D \rightarrow A \}$  si o descompunere  $\rho = (ABC, CD)$ . Trebuie sa testam daca  $D \rightarrow A$  este pastrata (celelalte dependente se regasesc direct in proiectiile lui  $F$  pe elementele descompunerii).

- ✓ Initial  $Z = D$
- 1) Prima trecere prin elementele lui  $\rho$ :
  - Pentru  $ABC$ :  $Z = D \cup ((D \cap ABC)^+ \cap ABC) = D \cup \emptyset = D$
  - Pentru  $CD$ :  $Z = D \cup ((D \cap CD)^+ \cap CD) = D \cup (ABCD \cap CD) = CD$
- 2) A doua trecere prin elementele lui  $\rho$ :
  - Pentru  $ABC$ :  $Z = CD \cup ((CD \cap ABC)^+ \cap ABC) = CD \cup (ABCD \cap ABC) = ABCD$ . Stop.
  - ✓ Am obtinut ca  $A \subseteq Z$ , deci dependenta  $D \rightarrow A$  este pastrata, rezulta ca  $\rho$  pastreaza dependentele.



# Algoritmi de descompunere a schemelor de relatie

## ➤ Algoritmi de descompunere

Algoritmii de testare a pastrarii dependentelor si a joinului fara pierderi pot fi aplicati atunci cand descompunerea unei scheme de relatie se face “manual”, pe baza experientei pe care o are proiectantul bazei de date.

- Exista insa si niste algoritmi simpli care, pornind de la o schema de relatie si multimea de dependente functionale asociata , ne duc direct la o descompunere care este in FN3 sau FNBC si, in plus, au proprietatea de join fara pierderi (nu se pierd date prin descompunere) si se pastreaza dependentele functionale.



# Algoritmi de descompunere a schemelor de relatie in FN3

## ■ Algoritm de descompunere in FN3 cu pastrarea dependentelor

Fie R o schema de relatie si F multimea de dependente functionale asociata, cu  $F = \{ X_1 \rightarrow Y_1, X_2 \rightarrow Y_2, \dots X_n \rightarrow Y_n \}$   
Atunci descompunerea  $\rho = (X_1Y_1, X_2Y_2, \dots X_nY_n)$  este o descompunere in FN3 cu pastrarea dependentelor.

Se observa din definitia de mai sus a descompunerii  $\rho$  ca:

- Toate dependentele sunt pastrate: dependenta  $X_i \rightarrow Y_i$  este in proiectia lui F pe  $X_iY_i$
- Pentru a minimiza numarul de elemente din descompunere se aplica regula reuniunii: daca avem mai multe dependente care au aceeasi parte stanga le reunim intr-una singura.
- Daca in descompunere exista doua elemente  $X_iY_i$  si  $X_jY_j$  astfel incat  $X_iY_i \subseteq X_jY_j$  atunci  $X_iY_i$  se elimina.
- **Observatie:** In literatura de specialitate exista demonstratia faptului ca fiecare schema din descompunerea  $\rho$  este in FN3.



# Algoritmi de descompunere a schemelor de relatie in FN3

- Exemple:
- 1)  $R = ABCDE$ ,  $F = \{ A \rightarrow B, A \rightarrow C, A \rightarrow D, D \rightarrow E \}$ . Rescriem prin reuniune multimea de dependente functionale:  $F = \{ A \rightarrow BCD, D \rightarrow E \}$ . Rezulta din algoritm ca descompunerea  $\rho = (ABCD, DE)$  este in FN3 cu pastrarea dependentelor.
- 2) Fie relata Produse = IdP, NumeP, Cant, IdF, NumeF, AdresaF avand dependentele functionale:  
 $F = \{ IdP \rightarrow NumeP, IdP \rightarrow Cant, IdP \rightarrow IdF, IdF \rightarrow NumeF, IdF \rightarrow AdresaF \}$
- Rescriem multimea de dependente. Raman numai doua dependente:  
 $F = \{ IdP \rightarrow NumeP, Cant, IdF; IdF \rightarrow NumeF, AdresaF \}$
- Descompunerea in FN3 cu pastrarea dependentelor va fi:  
 $\rho = ((IdP, NumeP, Cant, IdF), (IdF, NumeF, AdresaF))$



# Algoritmi de descompunere a schemelor de relatie in FN3

- **Algoritm de descompunere in FN3 cu pastrarea dependentelor si join fara pierderi**

Daca la descompunerea obtinuta prin algoritmul anterior adaugam o cheie a relatiei (ca element al descompunerii) vom obtine o descompunere care are atat proprietatea de join fara pierderi cat si pe cea a pastrarii dependentelor.

Formal putem scrie algoritmul astfel:

- **Definitie:** Fie R o schema de relatie si F multimea de dependente functionale asociata, cu

$F = \{ X_1 \rightarrow Y_1, X_2 \rightarrow Y_2, \dots X_n \rightarrow Y_n \}$  si o cheie X pentru R, cu  $X \not\subset X_i Y_i$ . Atunci descompunerea

$\rho = (X, X_1 Y_1, X_2 Y_2, \dots X_n Y_n)$  este o descompunere in FN3 cu pastrarea dependentelor si join fara pierderi.



# Algoritmi de descompunere a schemelor de relatie in FN3

- Pastrarea dependentelor este evidentă, ca mai sus. Demonstratia faptului că descompunerea are și proprietatea de join fără pierderi se găsește în literatura de specialitate.
- **Observatie:**

Daca vreunul dintre elementele de forma  $X_iY_i$  contine deja o cheie a lui R atunci nu este necesara adaugarea unui element suplimentar in descompunere. Deci, daca o cheie este deja inclusa intr-o descompunere atunci nu trebuie adaugata ca element suplimentar.



# Algoritmi de descompunere a schemelor de relatie in FN3

Exemple:

- 1) Pentru relatiile din exemplele de mai sus descompunerea ramane aceeasi, deoarece:
  - In cazul relatiei  $R = ABCDE$  cu descompunerea  $\rho = (ABCD, DE)$ , cheia este A, deja inclusa in ABCD, deci  $\rho$  este in FN3 cu pastrarea dependentelor si join fara pierderi.
  - In cazul relatiei PRODUSE cu descompunerea  $\rho = ((IdP, NumeP, Cant, IdF), (IdF, NumeF, AdresaF))$  cheia este IdP, inclusa de asemenea intr-unul dintre elementele descompunerii, deci  $\rho$  este in FN3 cu pastrarea dependentelor si join fara pierderi.



# Algoritmi de descompunere a schemelor de relatie in FN3

2) Fie  $R = ABCDE$ ,  $F = \{ A \rightarrow B, B \rightarrow A, A \rightarrow C, D \rightarrow E \}$ .  
Cheile relatiei sunt AD si BD.

Rescriem multimea de dependente:

$$F = \{ A \rightarrow BC, B \rightarrow A, D \rightarrow E \}.$$

- Rezulta descompunerea cu pastrarea dependentelor:  
 $\rho = (ABC, AB, DE)$ . Dar AB este inclus in ABC si rezulta in final  $\rho = (ABC, DE)$ .
- Cum elementele descompunerii nu contin vreo cheie a lui R, o adaugam. Obtinem in final descompunerile  $\rho_1 = (AD, ABC, DE)$  si  $\rho_2 = (BD, ABC, DE)$  in FN3 care pastreaza dependentele si au proprietatea j.f.p.



# Algoritmi de descompunere a schemelor de relatie in FNBC

- **Algoritm de descompunere in FNBC cu join fara pierderi**

Fie  $R$  o schema de relatie si  $F$  multimea de dependente functionale asociata,  $F$  in forma canonica:  $F = \{ X_1 \rightarrow A_1, X_2 \rightarrow A_2, \dots X_n \rightarrow A_n \}$ . Putem calcula descompunerea in FNBC cu join fara pierderi iterativ:

- ✓ Initial  $\rho = (R)$  ;
- ✓ La fiecare pas se alege o schema  $T$  care contine o dependenta de forma  $X \rightarrow A$  care violeaza conditiile de FNBC. Schema respectiva este inlocuita in  $\rho$  prin  $T_1$  si  $T_2$  unde  $T_1 = XA$  si  $T_2 = T - \{A\}$  ;
- ✓ Procesul se opreste cand in  $\rho$  nu mai exista elemente care nu sunt in FNBC.



# Algoritmi de descompunere a schemelor de relatie in FNBC

Exemple:

1) Fie relatia  $R = ABCD$  cu  $F = \{ AB \rightarrow C, AB \rightarrow D, D \rightarrow A \}$ . Cheia relatiei este AB. Relatia este in FN3 dar nu este in FNBC din cauza dependentei  $D \rightarrow A$  care nu are in partea stanga o supercheie a lui R.

- ✓ Initial:  $\rho = (R) = (ABCD)$ ;
- ✓ Alegem dependenta  $D \rightarrow A$  care violeaza conditia de FNBC;
- Inlocuim  $T = ABCD$  cu  $T_1 = DA$  si  $T_2 = ABCD - A = BCD$  ;
- $T_1$  mosteneste de la T dependenta  $D \rightarrow A$ , cheia va fi D si  $T_1$  e in FNBC;
- $T_2$  mosteneste de la T dependenta  $\{ BD \rightarrow C$  obtinuta prin augmentarea  $DB \rightarrow AB$  si  $AB \rightarrow C \}$ . Cheia va fi BD si  $T_2$  e in FNBC;
- ✓ Rezulta ca descompunerea in FNBC cu join fara pierderi este  $\rho = (AD, BCD)$ .



# Algoritmi de descompunere a schemelor de relatie in FNBC

Observatii:

- Dependenta mostenita de  $T_2$  este din  $F^+$ . Ea se deduce astfel: Din  $D \rightarrow A$  prin augmentare cu  $B$  obtinem  $DB \rightarrow AB$  si impreuna cu dependenta  $AB \rightarrow C$ , prin tranzitivitate obtinem  $DB \rightarrow C$ .
- Analog din  $AB \rightarrow D$  se deduce  $DB \rightarrow D$  dar aceasta este o dependenta triviala (partea dreapta e inclusa in cea stanga).
- In multe cazuri este nevoie de mai multe iteratii, relatiile de tip  $T_2$  (egale in algoritm cu  $T - A$ ) nefiind uneori in FNBC. Ele se descompun din nou in acelasi fel.



# Algoritmi de descompunere a schemelor de relatie in FNBC

2) Consideram R =(id\_dis, id\_stud, sesiune, data\_ex, nota, id\_prof) un catalog de note si F=(id\_dis->id\_prof, id\_stud->id\_dis, id\_stud->nota, id\_stud->id\_dis, id\_dis->data\_ex, id\_dis->sesiune, sesiune->data\_ex) cu cheia=(id\_dis, id\_stud).

Obs: Consideram ca un student are o singura nota la o disciplina(nota finala) si o disciplina este predata de catre un singur profesor.

- Aplicam algoritmul de descompunere:

a) Alegem  $\text{id\_dis} \rightarrow \text{id\_prof}$  care violeaza conditia FNBC ( $\text{id\_dis}$  nu este supercheie) si rezulta  $\rho_1 = (\text{id\_dis}, \text{id\_prof})$ ,  $\rho_2 = (\text{id\_dis}, \text{id\_stud}, \text{sesiune}, \text{data\_ex}, \text{nota})$

- $\rho_1$  cheia= $\text{id\_dis}$  si  $\text{id\_dis} \rightarrow \text{id\_prof}$ , deci este FNBC.
- $\rho_2$  cheia= $(\text{id\_dis}, \text{id\_stud})$ ,  $\text{id\_dis}, \text{id\_stud} \rightarrow \text{sesiune}, \text{data\_ex}, \text{nota}$ , deci este FNBC.

Putem sa ne oprim aici deoarece am obtinut o descompunere FNBC.

Obs: Daca continuam algoritmul putem obtine alte descompuneri tot in FNBC, dar se urmareste ca descompunerea sa contine cat mai putine relatii.

De exemplu, daca continuam cu  $\text{id\_dis} \rightarrow \text{data\_ex}$  obtinem descompunerea:

$\rho_1 = (\text{id\_dis}, \text{id\_prof})$ ,  $\rho_2 = (\text{id\_dis}, \text{data\_ex})$ ,  $\rho_3 = (\text{id\_dis}, \text{id\_stud}, \text{sesiune}, \text{nota})$  in care relatiile sunt tot in FNBC.



# Capitolul 8

**Dependente multivalorice.  
Formele normale FN4 si FN5**



# Dependente multivalorice

- Exista situatii in care, desi o relatie este in Forma Normala Boyce Codd, instancele sale contin date redundante. Acest fapt se intampla din cauza unei proiectari defectuoase, cand in aceeasi relatie sunt stocate date care apartin mai multor entitati si a cel putin doua asocieri multi-multi.

Sa luam urmatorul exemplu din Figura 1:

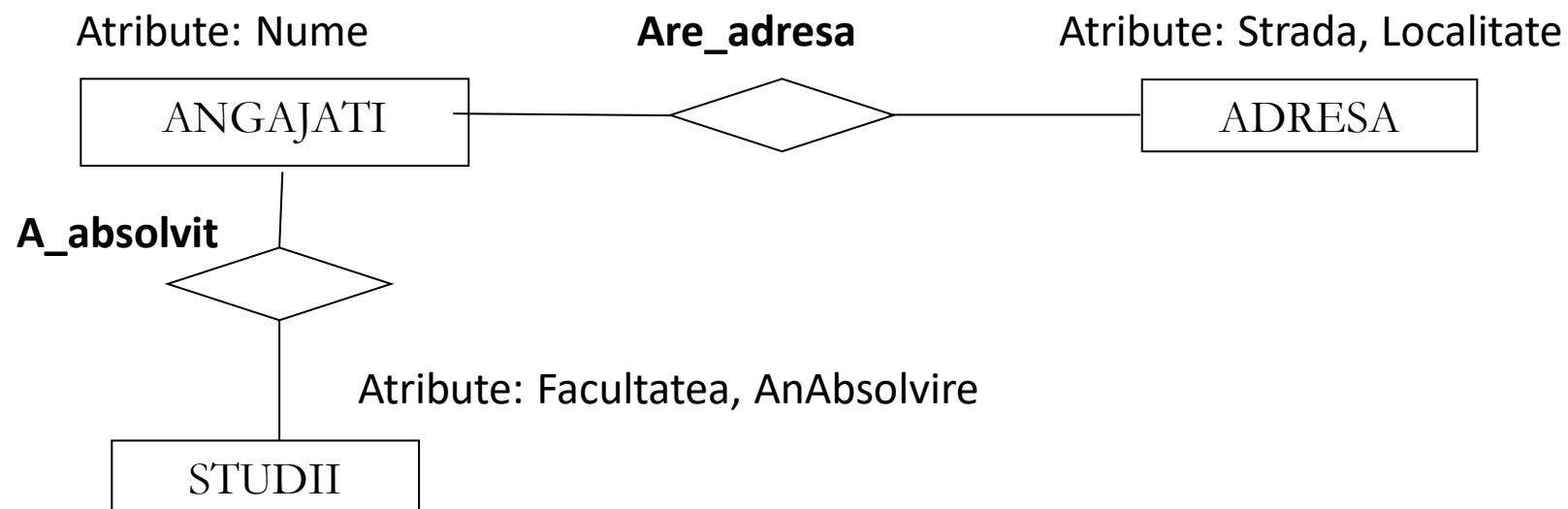


Figura 1. Exemplu de asocieri multi-multi



# Dependente multivalorice

- Ambele asocieri sunt multi-multi: un angajat poate sa fie absolvent al mai multor facultati si in acelasi timp poate avea mai multe adrese (de exemplu una pentru domiciliul stabil si alta pentru rezidenta temporara la un moment dat).
- In cazul in care toate datele din aceasta diagrama sunt stocate intr-o singura tabela putem avea urmatoarea incarcare cu date corecte:

Nume	Strada	Localitate	Facultate	AnAbsolvire
Vasilescu	Viitorului	Ploiesti	Automatica	2000
Vasilescu	Viitorului	Ploiesti	Comert	2004
Vasilescu	Plantelor	Bucuresti	Automatica	2000
Vasilescu	Plantelor	Bucuresti	Comert	2004
Marinescu	Revolutiei	Timisoara	Constructii	1998
Marinescu	Revolutiei	Timisoara	Drept	2003
Marinescu	Revolutiei	Timisoara	Master ASE	2006
Marinescu	Calea Vitan	Bucuresti	Constructii	1998
Marinescu	Calea Vitan	Bucuresti	Drept	2003
Marinescu	Calea Vitan	Bucuresti	Master ASE	2006



# Dependente multivalorice

- Putem observa ca nu exista nicio dependenta functionala netriviala valida pentru aceasta relatie, deci nu exista dependente care sa violeze conditiile FNBC. Ca urmare, relata este in FNBC avand ca singura cheie posibila multimea tuturor atributelor relatiei: din axioma de reflexivitate (A1) putem obtine dependenta:  
Nume,Strada,Localitate,Facultate,AnAbsolvire →  
Nume,Strada,Localitate,Facultate,AnAbsolvire
- Desi relata este in FNBC adresa si facultatea absolventa de un angajat sunt prezente repetat in relatie: adresa pentru fiecare facultate absolventa, iar facultatea pentru fiecare adresa a angajatului.
- Exemplul de mai sus sugereaza faptul ca seturile de attribute {Strada, Localitate} si {Facultate, AnAbsolvire} sunt independente unele de altele, in sensul ca fiecare adresa apare cu fiecare facultate absolventa de un angajat si reciproc. Astfel de situatii sunt modelate cu un nou tip de dependente, numite **dependente multivalorice**.



# Dependente multivalorice

- **Definirea dependentelor multivalorice (DMV)**
- **Definitie:** Fie o relatie R si doua multimi de atribute X si Y incluse in R. Se spune ca X multidetermina Y sau ca exista dependenta multivalorica  $X \rightarrow\rightarrow Y$ , daca si numai daca, ori de cate ori avem doua tupluri ale relatiei t1 si t2 cu  $t1[X] = t2[X]$  atunci exista un tuplu t3 in relatia R pentru care:
  - $t3[X] = t1[X] = t2[X]$
  - $t3[Y] = t1[Y]$  si  $t3[R-X-Y] = t2[R-X-Y]$
  - Cu alte cuvinte, construim un tuplu t3 compus din  $t1[X] + t1[Y] + t2[R-X-Y]$  care trebuie sa se regaseasca deasemenea printre tuplurile din R.



# Dependente multivalorice

Exemplu:

	X	Y	R - X - Y
t1	AAA	BBB	CCC
t2	AAA	DDD	EEE
t3	AAA	BBB	EEE

- **O consecinta** interesanta a acestei definitii este ca, daca inversam tuplurile t1 si t2, rezulta ca exista si un tuplu t4 pentru care

- $t4[X] = t1[X] = t2[X]$
- $t4[Y] = t2[Y]$  si  $t4[R-X-Y] = t1[R-X-Y]$

Cu alte cuvinte, construim un tuplu t4 compus din  $t1[X] + t2[Y] + t1[R-X-Y]$  care trebuie sa se regaseasca deasemenea printre tuplurile din R.



# Dependente multivalorice

Exemplu:

	X	Y	R – X – Y
t1	AAA	BBB	CCC
t2	AAA	DDD	EEE
t4	AAA	DDD	CCC

- Tot din aceasta definitie rezulta ca daca in R exista dependenta multivalorica  $X \rightarrow\rightarrow Y$  atunci exista si dependenta  $X \rightarrow\rightarrow R - X - Y$  ( acest fapt va fi prezentat in paragraful urmator ca axioma de complementare a dependentelor multivalorice).
- Intorcandu-ne la exemplul anterior rezulta ca in relatia continand date despre angajati, studii si adrese, avem urmatoarele dependentele multivalorice (a doua fiind obtinuta din prima prin complementare):
  - Nume  $\rightarrow\rightarrow$  Strada, Localitate
  - Nume  $\rightarrow\rightarrow$  Facultate, AnAbsolvire



# Dependente multivalorice

- Intr-adevar, daca luam in considerare pentru t1 si t2 tuplurile 2 si 3 din relatie, obtinem:

Nume	Strada	Localitate	Facultate	An_absolv
Vasilescu	Viitorului	Ploiesti	Comert	2004
Vasilescu	Plantelor	Bucuresti	Automatica	2000

- Gasim in relatie pe prima pozitie si tuplul t3 de forma:

Nume	Strada	Localitate	Facultate	An_absolv
Vasilescu	Viitorului	Ploiesti	Automatica	2000

- In acelasi timp gasim pe pozitia 4 si tuplul t4:

Nume	Strada	Localitate	Facultate	An_absolv
Vasilescu	Plantelor	Bucuresti	Comert	2004



# Dependente multivalorice

- Sa facem o alta alegere pentru t1 si t2:

Nume	Strada	Localitate	Facultate	An_absolv
Vasilescu	Viitorului	Ploiesti	Automatica	2000
Vasilescu	Viitorului	Ploiesti	Comert	2004

- Atunci t3 si t4 vor fi:

t3

Nume	Strada	Localitate	Facultate	An_absolv
Vasilescu	Viitorului	Ploiesti	Comert	2004

t4

Nume	Strada	Localitate	Facultate	An_absolv
Vasilescu	Viitorului	Ploiesti	Automatica	2000

- Observam ca t3 = t2 si t4 = t1 ceea ce este corect pentru ca in definitia dependentelor multivalorice nu se cere ca t3 sa fie diferit de t1 si t2.



# Dependente multivalorice

- **Consecinta importanta:** orice dependenta functionala este in acelasi timp si o dependenta multivalorica:
- Fie relatia R si o dependenta functionala  $X \rightarrow Y$  pentru R. Atunci, daca doua tupluri t1 si t2 au aceleasi valori pe attributele X , vor avea aceleasi valori si pe attributele Y. Rezulta ca t2 indeplineste conditiile pentru t3 din definitia dependentelor multivalorice:

	X	Y	R – X – Y
t1	AAA	BBB	CCC
t2	AAA	BBB	DDD
t3 (=t2)	AAA	BBB	DDD

- Rezulta ca daca  $X \rightarrow Y$  avem si dependenta multivalorica  $X \rightarrow\rightarrow Y$



# Dependente multivalorice

Exemplu:

Fie relatia Produse:

<b>IdP</b>	<b>NumeP</b>	<b>Qty</b>	<b>IdF</b>	<b>NumeF</b>	<b>AdresaF</b>
101	Imprimanta laser	30	20	Xerox	Str. Daniel Danielopolu 4-6, Sector 1, Bucuresti
105	Calculator PC	20	23	IBM	Bd. Dimitrie Cantemir, nr.1, Sector 3, Bucuresti
124	Copiator	10	20	Xerox	Str. Daniel Danielopolu 4-6, Sector 1, Bucuresti

- In aceasta avem dependenta functionala:  
 $\text{IdF} \rightarrow \text{NumeF}, \text{AdresaF}$



# Dependente multivalorice

- Avem doua tupluri cu aceleasi valori pe IdF:

	<b>IdP</b>	<b>NumeP</b>	<b>Qty</b>	<b>IdF</b>	<b>NumeF</b>	<b>AdresaF</b>
<b>t1</b>	101	Imprimanta laser	30	<b>20</b>	<b>Xerox</b>	<b>Str. Daniel Danielopolu 4-6, Sector 1, Bucuresti</b>
<b>t2</b>	<b>124</b>	<b>Copiator</b>	<b>10</b>	20	Xerox	Str. Daniel Danielopolu 4-6, Sector 1, Bucuresti

- In acest caz putem forma tuplul t3 astfel:
  - Pe IdF valoarea 20;
  - Pe NumeF si adresaF valorile din primul tuplu;
  - Pe restul atributelor valorile din al doilea tuplu.



# Dependente multivalorice

- Obtinem t3 identic cu t2:

	<b>IdP</b>	<b>NumeP</b>	<b>Qty</b>	<b>IdF</b>	<b>NumeF</b>	<b>AdresaF</b>
t3	124	Copiator	10	20	Xerox	Str. Daniel Danielopolu 4-6, Sector 1, Bucuresti

- Ca si in cazul dependentelor functionale, exista si in cazul DMV o serie de axiome si reguli care ne permit ca pornind de la un set de dependente sa obtinem alte dependente.



# Axiome pentru DMV

- **Axiome si reguli pentru DMV**

Urmatoarele axiome sunt specifice DMV. Le numerotam incepand cu A4 deoarece intr-o schema de relatie pot fi atat dependente functionale (carora li se aplica axiomele A1-A3 descrire anterior) cat si dependente multivalorice.

## **A4. Complementare:**

Fie R o schema de relatie si  $X, Y \subseteq R$ .

Daca  $X \rightarrow\rightarrow Y$  atunci si  $X \rightarrow\rightarrow (R - X - Y)$ .

## **A5. Augmentare pentru DMV:**

Fie R o schema de relatie si  $X, Y, Z, W \subseteq R$ .

Daca  $X \rightarrow\rightarrow Y$  si  $Z \subseteq W$  atunci si  $XW \rightarrow\rightarrow YZ$ .

## **A6. Tranzitivitate pentru DMV:**

Fie R o schema de relatie si  $X, Y, Z \subseteq R$ .

Daca  $X \rightarrow\rightarrow Y$  si  $Y \rightarrow\rightarrow Z$  atunci si  $X \rightarrow\rightarrow (Z - Y)$ .



# Axiome pentru DMV

- Ultimele doua axiome leaga dependentele multivalorice cu cele functionale:
- **A7.** Fie  $R$  o schema de relatie si  $X, Y \subseteq R$ .  
Daca  $X \rightarrow Y$  atunci si  $X \rightarrow\rightarrow Y$ .
- **A8.** Fie  $R$  o schema de relatie si  $X, Y, Z, W \subseteq R$  cu  $W \cap Y = \emptyset$   
Daca  $X \rightarrow\rightarrow Y, Z \subseteq Y, W \rightarrow Z$  atunci si  $X \rightarrow Z$ .
- **Observatie importantă:** orice dependenta functionala este in acelasi timp si o dependenta multivalorica insa reciproca nu este adevarata: exista dependente multivalorice pentru care in schema relatiei nu avem o dependenta functionala corespondenta.



# Axiome pentru DMV

Exemplu:

Consideram dependenta multivalorica existenta in tabela de angajati din paragraful anterior:

Nume →→ Strada, Localitate

In relatie nu exista insa si o dependenta functionala echivalenta de tipul:

Nume → Strada, Localitate

Rezulta ca:

- Putem folosi si axiomele A1-A3 dar numai pentru dependente multivalorice care sunt in acelasi timp si dependente funktionale.
- Pentru restul dependentelor multivalorice putem folosi doar A4-A6.



# Reguli pentru DMV

- Există de asemenea o serie de reguli care se pot deduce din axiome. Toate consideră existența unei scheme de relație R iar X, Y, Z, W sunt submultimi ale lui R:
  - **R1. Reuniune:** Dacă  $X \rightarrow\rightarrow Y$  și  $X \rightarrow\rightarrow Z$  atunci  $X \rightarrow\rightarrow YZ$ .
  - **R2. Pseudotranzitivitate:** Dacă  $X \rightarrow\rightarrow Y$  și  $WY \rightarrow\rightarrow Z$  atunci  $WX \rightarrow\rightarrow Z - WY$ .
  - **R3. Pseudotranzitivitate mixta:** Dacă  $X \rightarrow\rightarrow Y$  și  $XY \rightarrow\rightarrow Z$  atunci  $X \rightarrow\rightarrow Z - Y$ .
  - **R4. Diferenta:** Dacă  $X \rightarrow\rightarrow Y$  și  $X \rightarrow\rightarrow Z$  atunci
    - $X \rightarrow\rightarrow Y - Z$ ;
    - $X \rightarrow\rightarrow Z - Y$ .



# Reguli pentru DMV

- **R5. Intersectie:** Daca  $X \rightarrow\rightarrow Y$  si  $X \rightarrow\rightarrow Z$  atunci  
$$X \rightarrow\rightarrow Y \cap Z$$
- **R6. Eliminare atribute comune:** Daca  $X \rightarrow\rightarrow Y$  atunci  
$$X \rightarrow\rightarrow Y - X .$$
- **R7. Acoperire completa de atribute:** Daca  $X \cup Y = R$  atunci  
$$X \rightarrow\rightarrow Y \text{ si } Y \rightarrow\rightarrow X .$$
- **R8. Reflexivitate:** Daca  $Y \subseteq X$  atunci  
$$X \rightarrow\rightarrow Y .$$
- Aceste axiome si reguli se pot folosi pentru calculul inchiderii unei multimi de dependente functionale si multivalorice.



# Inchiderea multimii DMV

Definitia inchiderii este aceeasi ca la dependentele functionale:

- **Definitie:** Fie  $R$  o schema de relatie si  $G$  multimea de dependente functionale si multivalorice asociata.

Atunci ***inchiderea multimii de dependente***  $G^+$ , notata  $G^+$ , este o multime de dependente (DF si DMV) care sunt in  $G$ , sau se pot deduce din  $G$  folosind axiomele si regulile specifice pentru dependente functionale si multivalorice.



# Proiectia unei multimi DMV

Analog cu cazul dependentelor functionale, se poate defini si proiectia unei multimi de dependente functionale si multivalorice pe o multime de atribute:

- **Definitie.** Fie o relatie R, o multime asociata de dependente functionale si multivalorice G si o submultime de atribute  $S \subseteq R$ . **Proiectia multimii de dependente G pe S**, notata cu  $\pi_S(G)$  este multimea dependentelor din  $G^+$  care au si partea stanga si pe cea dreapta incluse in S.
- In momentul in care o schema de relatie se descompune in doua sau mai multe subscheme, fiecare subschema va mosteni o multime de dependente functionale si multivalorice obtinuta prin proiectia multimii initiale G pe atributele din subschema respectiva.



# Forma normala 4 (FN4)

- Pentru a preintampina redundantele prezentate la inceputul acestui capitol este bine ca schemele de relatie sa fie intr-o forma normala superioara FNBC. Aceasta forma care considera, pe langa dependentele functionale acceptate de FNBC, si dependente multivalorice se numeste **Forma Normala 4 (FN4)**.
- Definitia ei este similara cu cea pentru FNBC dar conditia se pune pentru dependentele multivalorice ale relatiei respective:
- **Definitie:** O schema de relatie R este in **Forma Normala 4** daca orice dependenta multivalorica netriviala  $X \rightarrow\rightarrow Y$  are in partea stanga o supercheie.



# Forma normala 4 (FN4)

- Dependentele multivalorice triviale sunt acceptate in FN4 si sunt de doua feluri:
  - ✓ Dependente provenite din R8:  
 $X \rightarrow\rightarrow Y$  unde  $Y \subseteq X$  (sunt cele in care partea dreapta este inclusa in partea stanga);
  - ✓ Dependente provenite din regula R7:  
 $X \rightarrow\rightarrow Y$  pentru  $X \cup Y = R$  (contin toate atributele relatiei, deci sunt dependente la care partea stanga reunita cu partea dreapta acopera intreaga relatie);
- Conditia de FN4 spune deci ca orice DMV care nu intra intr-una dintre categoriile de mai sus are in partea stanga o supercheie.



# Forma normala 4 (FN4)

Exemple:

## 1. Relatia

Angajati=(Nume, Strada, Localitate, Facultate, An\_absolv)  
are dependenta multivalorica netriviala

Nume →→ Strada, Localitate

Cum cheia relatiei e multimea tuturor atributelor  
acesteia, rezulta ca relatia nu este in FN4 deoarece  
atributul Nume nu este supercheie.

## 2. Relatia

Produse=(Cod\_prod, Den\_prod, Pret, Furnizor, Adresa)  
are cheia Cod\_prod si dependenta multivalorica  
netriviala Cod\_prod →→ Furnizor, Adresa  
Deoarece Cod\_prod este si supercheie rezulta ca relatia  
este in forma normala FN4.



## Forma normala 4 (FN4)

- Relatia dintre formele normale FN3, FNBC si FN4 este una de incluziune, in aceasta ordine, asa cum se vede in Figura 2.
- Orice relatie in FN4 este in acelasi timp in FNBC si FN3 daca dependentele multivalorice au corespondent in cele functionale .

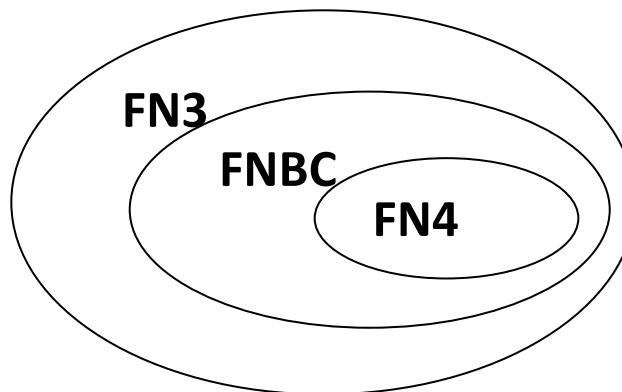


Figura 2. Relatia de incluziune dintre FN3, FNBC si FN4



# Forma normala 4 (FN4)

## ✓ Algoritm de descompunere in FN4

Acest algoritm este similar cu cel de descompunere in FNBC dar ia in considerare dependentele multivalorice care violeaza FN4.

- **Observatie:** Dependentele multivalorice ale unei relatii sunt atat cele care provin prin axioma A7 din dependente functionale (daca  $X \rightarrow Y$  atunci  $X \rightarrow\rightarrow Y$ ), cat si dependente multivalorice care nu au corespondent in multimea celor functionale.



# Forma normala 4 (FN4)

- Fie R o schema de relatie si G multimea de dependente multivalorice asociata (consideram ca din G au fost eliminate dependentele triviale). Putem calcula descompunerea in FN4 iterativ:
  - Initial  $\rho = (R)$
  - La fiecare pas se alege o schema T care contine o dependenta de forma  $X \rightarrow\rightarrow Y$  care violeaza conditia pentru FN4. Schema respectiva este inlocuita in  $\rho$  prin  $T_1$  si  $T_2$  unde  $T_1 = XY$  si  $T_2 = X(T - Y)$  .
  - Procesul se opreste cand in  $\rho$  nu mai exista elemente care nu sunt in FN4 .



# Forma normala 4 (FN4)

Exemplu:

Pentru relatia Angajati care nu era in FN4 :

- Initial

$\rho = (\text{Nume}, \text{Strada}, \text{Localitate}, \text{Facultate}, \text{An\_absolv})$

- Alegem dependenta

$\text{Nume} \rightarrow\rightarrow \text{Strada}, \text{Localitate}$

care violeaza conditia pentru FN4.

Obtinem :

- $T_1 = \text{Nume}, \text{Strada}, \text{Localitate} \text{ si}$
- $T_2 = \text{Nume}, \text{Facultate}, \text{An\_absolv}$



## Forma normala 4 (FN4)

- Obtinem  $\rho = ( (\text{Nume}, \text{Strada}, \text{Localitate}), (\text{Nume}, \text{Facultate}, \text{An\_absolv}) )$ . Fiecare subschema mosteneste cate o dependenta multivalorica:
    - $T_1$  : dependenta  $\text{Nume} \rightarrow\rightarrow \text{Strada}, \text{Localitate}$
    - $T_2$  : dependenta  $\text{Nume} \rightarrow\rightarrow \text{Facultate}, \text{An\_absolv}$
  - Cum cele doua dependente multivalorice mostenite de  $T_1$  si  $T_2$  sunt triviale (contin toate atributele relatiei respective) rezulta ca cele doua relatii sunt in FN4 deoarece nu exista dependente care violeaza conditia de FN4. Stop.
- Observatie:** Asa cum s-a mentionat anterior, fiecare subschema  $T_i$  obtinuta din descompunere mosteneste de la relatia originala  $T$  projectia multimii de dependente a lui  $T$  (DF si DMV) pe  $T_i$ .



# Forma normala 5 (FN5)

- **Definitie:** Atunci cand o relatie poate fi reconstruita fara pierderi de dependente din unele proiectii ale sale se spune ca avem o **dependenta jonctionala**. Daca jonctiunile includ si relatia, avem de-a face cu o **dependenta jonctionala triviala**.
  
- **Definitie:** O relatie este in **Forma Normala 5 ( FN5 )** daca:
  - dependenta jonctionala care reconstruieste schema originala este o dependenta jonctionala triviala;
  
  - sau
  - fiecare relatie in dependenta jonctionala constituie (cu toate atributele sale) o supercheie a relatiei originale.



# Forma normala 5 (FN5)

Exemplu:

Fie relatia R de mai jos:

DPC

Departamente	Proiecte	Componente
D1	P1	C1
D1	P1	C2
D1	P2	C2
D2	P3	C2
D2	P4	C2
D2	P4	C4
D2	P5	C4
D3	P2	C5

Schema de relatie DPC este in FNBC, deoarece cheia sa contine toate atributele. De asemenea, DPC este in FN4 deoarece nu contine DMV netriviale.



# Forma normala 5 (FN5)

- Pe relatia de mai sus, vom efectua acum operatia de proiectie si obtinem relatiile  $DP=(\text{Departamente}, \text{Proiecte})$ ,  $DC=(\text{Departamente}, \text{Componente})$  si  $PC=(\text{Proiecte}, \text{Comp})$ :

DP

Depart	Proiecte
D1	P1
D1	P1
D1	P2
D2	P3
D2	P4
D2	P4
D2	P5
D3	P2

DC

Depart	Comp
D1	C1
D1	C2
D1	C2
D2	C2
D2	C2
D2	C4
D2	C4
D3	C5

PC

Proiecte	Comp
P1	C1
P1	C2
P2	C2
P3	C2
P4	C2
P4	C4
P5	C4
P2	C5

- Examinand relatiile obtinute, observam ca nu exista posibilitatea de a reconstrui relatiea DPC prin operatiuni de jonctiune, pe niciuna dintre perechile de relatii de mai sus.



# Forma normala 5 (FN5)

- Prin operatiuni de jonctiune, vor rezulta jonctiuni cu pierderi. Reprezentam mai jos numai tuplurile “nelegate” ale acestor jonctiuni:

DP  $\bowtie$  DC

Depart	Proiecte	Comp
D1	P2	C1
D2	P3	C4

DP  $\bowtie$  PC

Depart	Proiecte	Comp
D3	P2	C2
D2	P1	C2

DC  $\bowtie$  PC

Depart	Proiecte	Comp
D1	P3	C2
D1	P4	C2
D2	P5	C2
D2	Pr2	C2

- Niciunul dintre tuplurile de mai sus nu exista in relatia DPC. Pentru a reconstrui DPC fara pierderi, trebuie sa efectuam jonctiunea tuturor celor trei scheme: DP  $\bowtie$  DC  $\bowtie$  PC



# Forma normala 5 (FN5)

- Preferam o schema cu DP, DC si PC in loc de DPC, deoarece relatiile descompuse au cardinalitate inferioara celei a relatiei DPC (sau egala cu aceasta). Aceasta inseamna ca DPC va conduce la anomalii de actualizare din cauza redundantei sale, deoarece DPC, desi este in FNBC si in FN4, nu este in FN5.
- Deoarece avem dependenta jonctionala, DPC nu este in FN5 (niciuna dintre aceste relatii nu formeaza o supercheie pentru DPC). Prin urmare, schema descompusa cu DP, DC si PC este in FN5, intrucat aceste relatii satisfac oricare dintre conditii (de fapt, ambele conditii) si intrucat fiecare relatie reprezinta o dependenta jonctionala pentru ea insasi, iar atrbutele sale sunt superchei (am vazut in acest exemplu ca toate cele trei relatii au cheile formate din toate atrbutele lor).



# Forme normale

- Relatiile de incluziune dintre formele normale sunt prezentate in Figura 3.

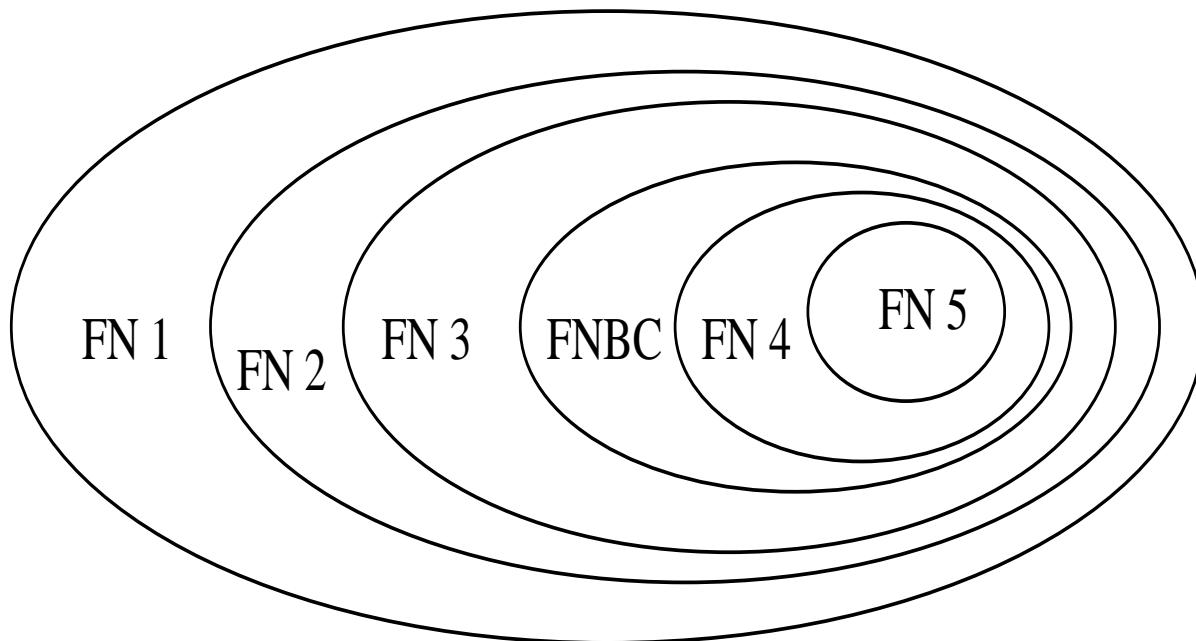


Figura 3. Relatiile de incluziune dintre formele normale



# Capitolul 9

## Optimizarea cererilor de interogare



# Optimizarea costurilor de executie

- Pentru prelucrarea datelor dintr-o baza de date trebuie scrise cereri care sunt execute de catre Sistemul de Gestiune a Bazei de Date (SGBD). Cererile sunt scrise in limbajul SQL(*Structured Query Language*) care este un limbaj de nivel inalt dar neprocedural. Sintaxa unei cereri specifica informatiile care trebuie extrase, eventualele operatii care se executa asupra datelor, dar nu specifica modul de acces la date, aceasta fiind sarcina SGBD-lui. In functie de structura bazei de date pot exista mai multe modalitati de acces si prelucrare, fiecare modalitate avand anumite costuri care, de regula, se exprima in timpul de acces.
- Reducerea timpului de acces la baza de date la o valoare minima reprezinta o metoda de optimizare a costurilor de executie. In acest sens trebuie optimizate in primul rand cererile de interogare dar si celealte comenzi DML(*Data Manipulation Language*) trebuie optimizate, cum ar fi cele de inserare, modificare si stergere de date.



# Echivalenta expresiilor in algebra relationala

- Executia cererilor de interogare incepe prin construirea unei structuri arborescente destinate evaluarii expresiilor algebrice continute in cerere.
- Pentru rescrierea unei expresii in algebra relationala, in vederea optimizarii costurilor de executie, se pot folosi urmatoarele reguli de echivalenta a expresiilor:

**R1.** Comutativitatea produsului cartezian si a join-lui

- $E1 \times E2 \equiv E2 \times E1$
- $E1 \bowtie E2 \equiv E2 \bowtie E1$
- $E1 \bowtie_F E2 \equiv E2 \bowtie_F E1$



# Echivalenta expresiilor in algebra relationala

**R2.** Asociativitatea produsului cartezian si a joinului

- $E_1 \times (E_2 \times E_3) \equiv (E_1 \times E_2) \times E_3$
- $E_1 \bowtie (E_2 \bowtie E_3) \equiv (E_1 \bowtie E_2) \bowtie E_3$
- $E_1 \bowtie_{F_1} (E_2 \bowtie_{F_2} E_3) \equiv (E_1 \bowtie_{F_1} E_2) \bowtie_{F_2} E_3$

**R3.** Cascada de proiectii

- $\pi_{A_1 \dots A_n} (\pi_{B_1 \dots B_m}(E)) \equiv \pi_{A_1 \dots A_n}(E)$

Conditia este bineintelea ca  $\{A_i\} \subseteq \{B_j\}$

**R4.** Cascada de selectii

- $\sigma_{F_1}(\sigma_{F_2}(E)) \equiv \sigma_{F_1 \wedge F_2}(E)$



# Echivalenta expresiilor in algebra relationala

## R5. Comutativitatea selectiilor cu proiectiile

Daca F contine doar atribute din A<sub>1</sub>...A<sub>n</sub>, atunci:

- $\pi_{A_1 \dots A_n}(\sigma_F(E)) \equiv \sigma_F(\pi_{A_1 \dots A_n}(E))$

Sau, in cazul general, daca F contine si alte atribute B<sub>1</sub>...B<sub>m</sub>:

- $\pi_{A_1 \dots A_n}(\sigma_F(E)) \equiv \pi_{A_1 \dots A_n}(\sigma_F(\pi_{A_1 \dots A_n, B_1 \dots B_m}(E)))$

## R6. Comutativitatea selectiei cu produsul cartezian

Daca toate atributele din F fac parte din E<sub>1</sub>, atunci:

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

Daca F = F<sub>1</sub>  $\wedge$  F<sub>2</sub>, cu F<sub>1</sub> continand doar atribute din E<sub>1</sub> si F<sub>2</sub> doar din E<sub>2</sub>, atunci:

- $\sigma_F(E_1 \times E_2) \equiv \sigma_{F_1}(E_1) \times \sigma_{F_2}(E_2)$

Daca F = F<sub>1</sub>  $\wedge$  F<sub>2</sub>, cu F<sub>1</sub> continand doar atribute din E<sub>1</sub> dar F<sub>2</sub> este o expresie generala, atunci:

- $\sigma_F(E_1 \times E_2) \equiv \sigma_{F_2}(\sigma_{F_1}(E_1) \times E_2)$



## Echivalenta expresiilor in algebra relationala

### R7. Comutativitatea selectie – reunioane

- $\sigma_F(E1 \cup E2) \equiv \sigma_F(E1) \cup \sigma_F(E2)$

### R8. Comutativitatea selectie – diferența

- $\sigma_F(E1 - E2) \equiv \sigma_F(E1) - \sigma_F(E2)$

### R9. Comutativitatea proiecție – produs cartezian

Daca in lista A<sub>1</sub>...A<sub>n</sub>, atributele A<sub>1</sub>...A<sub>k</sub> sunt din E<sub>1</sub> iar A<sub>k+1</sub>...A<sub>n</sub> din E<sub>2</sub>, atunci:

- $\pi_{A_1 \dots A_n}(E1 \times E2) \equiv \pi_{A_1 \dots A_k}(E1) \times \pi_{A_{k+1} \dots A_n}(E2)$

### R10. Comutativitatea proiecție – reunioane

- $\pi_{A_1 \dots A_n}(E1 \cup E2) \equiv \pi_{A_1 \dots A_n}(E1) \cup \pi_{A_1 \dots A_n}(E2)$



# Echivalenta expresiilor in algebra relationala

- Aceste reguli permit definirea unor strategii generale de optimizare a cererilor de interogare, strategii care se refera la posibilitatile de crestere a performantelor de executie a cererilor de interogare prin reordonarea operatiilor implicate de aceste cereri.
- De exemplu, prin deplasarea operatiilor de selectie cat mai in stanga expresiilor algebrice se reduce numarul de tupluri care trebuie manipulate in procesul de executie a cererii.



## Strategii generale de optimizare a cererilor

- Se pot mentiona urmatoarele strategii generale de optimizare a cererilor de interogare:
  - ✓ *Executia cererilor de selectie inaintea operatiilor de join.* Join-ul si produsul cartezian actioneaza ca generatori de tupluri. Prin selectie se reduce dimensiunea relatiilor la care se aplica acesti generatori de tupluri, deci si a rezultatelor obtinute. Pentru deplasarea selectiei in fata join-ului se vor aplica proprietatile mentionate anterior, tinand cont de faptul ca un join poate fi exprimat sub forma unui produs cartezian urmat de o selectie, iar in cazul join-ului natural printr-un produs cartezian urmat de o selectie si o proiectie.



## Strategii generale de optimizare a cererilor

✓ *Executia operatiilor de proiectie inaintea operatiilor de join.*

Este realizata cu ajutorul proprietatii de comutare a selectiei cu produsul cartezian, regula R6. Prin executia proiectiei inaintea join-ului se obtin unele avantaje si anume:

- Daca proiectia lasa intacta cheia, operatia de join se poate realiza mai rapid intrucat s-a redus numarul de tupluri ce trebuie stocate si eventual sortate;
- Proiectia implica eliminarea tuplurilor duplicate, care se poate realiza relativ usor, de exemplu cu ajutorul unui index. Dupa eliminarea tuplurilor duplicate, join-ul va fi mai simplu, pentru ca relatiile sunt mai mici.



## Strategii generale de optimizare a cererilor

### ✓ *Executia cererilor de selectie inaintea proiectiilor.*

Regula de mutare a selectiilor in fata proiectiilor este data de proprietatea de comutare a selectiei cu proiectia, regula R5. Realizarea deplasarii este utila atunci cand:

- Exista o serie de facilitati in realizarea selectiei, precum accesul direct, facilitati care ar putea fi inhibate prin operatia de proiectie;
- Eliminarea tuplurilor duplicate obtinute prin proiectie se realizeaza prin sortarea tuplurilor. Selectia reduce numarul de tupluri care trebuie sortate, facilitand astfel realizarea operatiei de proiectie.



# Algoritm de optimizare a expresiilor relationale

J.D. Ullman a propus urmatorul algoritm de optimizare a expresiilor relationale:

- **Intrare:** un arbore reprezentand o expresie relationala.
- **Iesire:** program pentru evaluarea expresiei.
- **Metoda:**
  - 1) Fiecare selectie este transformata intr-o cascada de selectii, folosind regula de echivalenta R4 :
    - $\sigma_{F_1 \wedge F_2 \wedge \dots \wedge F_n}(E) \equiv \sigma_{F_1}(\sigma_{F_2}(\dots(\sigma_{F_n}(E))\dots))$
  - 2) Fiecare selectie este deplasata in jos folosind regulile R4-R8 cat mai aproape de frunze.



# Algoritm de optimizare a expresiilor relationale

- 3) Folosind regulile R3, R5, R9 si R10, fiecare proiectie este deplasata cat mai jos posibil in arborele sintactic. Regula R3 face ca unele proiectii sa dispara, regula R5 sparge o proiectie in doua astfel incat una poate migra spre frunze. Daca o proiectie ajunge sa fie facuta dupa toate atributele, atunci va fi eliminata.
- 4) Cascadele de selectii si proiectii sunt combinate intr-o singura selectie folosind regulile R3-R5, o singura proiectie sau o selectie urmata de o proiectie. Aceasta abordare tine de eficienta: e mai putin costisitor sa se faca o singura selectie si apoi o singura proiectie decat daca s-ar alterna de mai multe ori selectii cu proiectii.



# Algoritm de optimizare a expresiilor relationale

- 5) Nodurile interioare ale arborelui rezultat sunt impartite in grupuri. Fiecare nod intern care corespunde unei operatiuni binare devine un grup impreuna cu predecesorii sai imediati cu care sunt asociate operatiuni unare. Din bloc face parte de asemenea din orice lant de noduri succesoare asociate cu operatiuni unare si terminate cu o frunza, cu exceptia cazului cand operatia binara este un produs cartezian neurmat de o selectie cu care sa formeze un equi-join.
- 6) Se evaluateaza fiecare grup astfel incat niciunul nu este evaluat inaintea descendantilor sai. Rezulta astfel un program de evaluare a expresiei relationale initiale.

# Algoritm de optimizare a expresiilor relationale

Exemplu:

- Fie o baza de date care implementeaza o diagrama Entitate-Asociere formata din doua entitati (Studenti, Discipline) si o asociere binara Note, de tip multi-multi, continand notele studentilor si data obtinerii lor, ca atribute proprii.

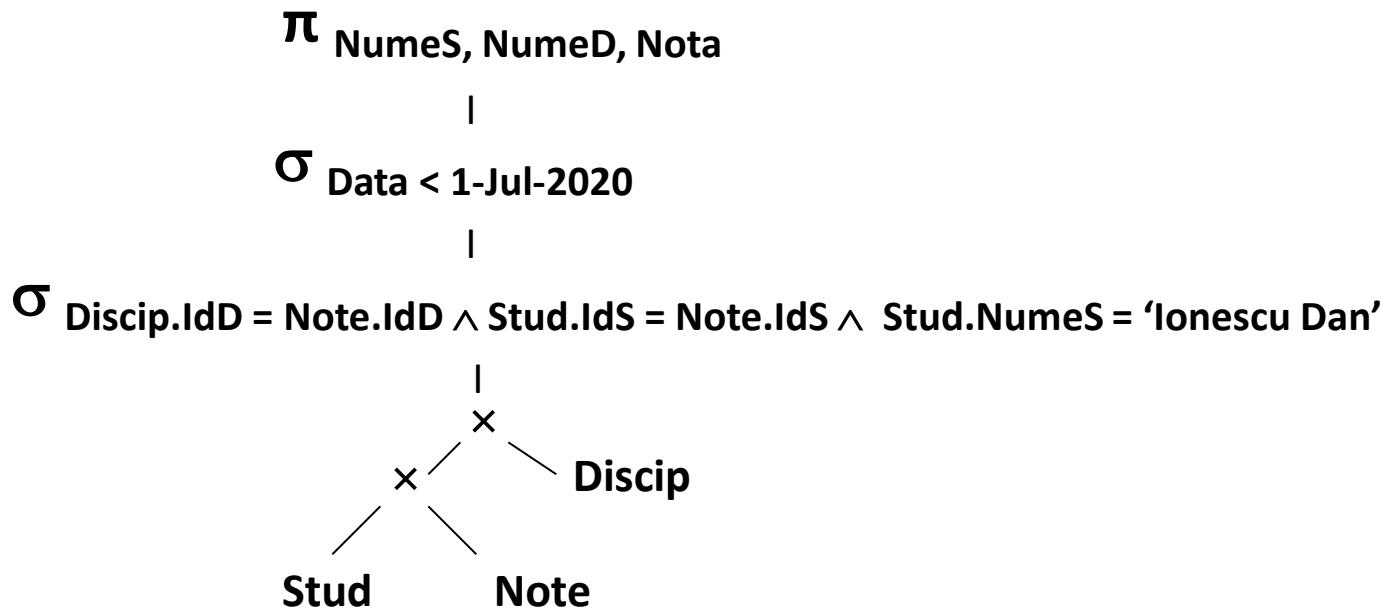


Figura 1 - Exemplu de expresie relationala



# Algoritm de optimizare a expresiilor relationale

- Expresia relationala de mai sus afiseaza numele studentului Ionescu Dan si notele sale la toate disciplinele, cu conditia ca notele sa fie pana la 1 iulie 2020.

Structura simplificata a bazei de date este urmatoarea:

- Stud (IdS, NumeS)
- Discip (IdD, NumeD)
- Note (IdS, IdD, Nota, Data)



# Algoritm de optimizare a expresiilor relationale

- Aplicam algoritmul de mai sus:
- ✓ Selectia dupa data va ajunge pe ramura tablei Note.
- ✓ Conditii se vor sparge in trei, doua conditii de join separate(una pentru equi-joinul Stud cu Note si alta pentru equi-joinul rezultatului primului join cu Discip) si o a treia conditie de sortare dupa numele studentului care coboara pe ramura respectiva.
- ✓ Regula R5 generalizata va duce la adaugarea unor proiectii care lasa sa treaca mai departe de la frunze doar atributele necesare operatiilor ulterioare.
- ✓ Rezultatul este prezentat in Figura 2. Se observa formarea a doua grupuri care se evaluateaza in ordinea mentionata in algoritm: intai grupul de jos si apoi grupul de sus care foloseste rezultatul primului grup. Pentru fiecare grup produsul cartezian si selectia se pot combina intr-un equi-join

# Algoritm de optimizare a expresiilor relationale

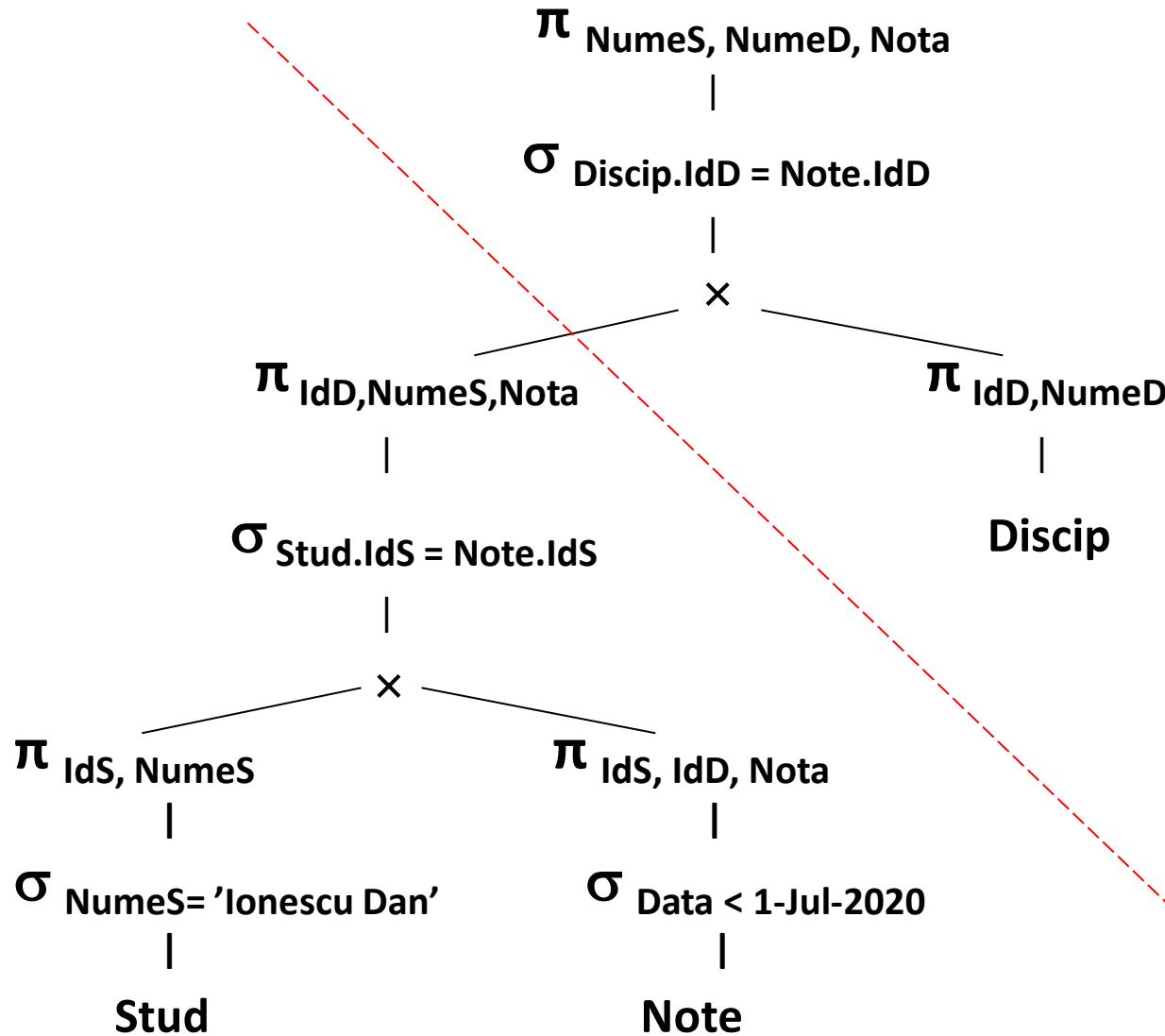


Figura 2. Rezultatul optimizării expresiei relationale



# Optimizarea cererilor intr-o baza de date relationala

- Optimizarea cererilor intr-o baza de date relationala presupune parcurgerea mai multor etape:
  - ✓ Translatarea cererilor cu ajutorul algebrei relationale (expresii algebrice relationale).
  - ✓ Transformarea in expresii echivalente care pot sa fie executate cat mai eficient. Transformarile echivalente se fac din punct de vedere:
    - **Logic** – se aplica prioritatea operatiilor (executia operatiilor de proiectie inaintea operatiilor de join, executia selectiilor inaintea proiectiilor, combinarea selectiilor multiple, etc.);
    - **Semantic** - transformarile sunt bazate pe proprietatile atributelor.



# Optimizarea cererilor intr-o baza de date relationala

- ✓ Reprezentarea expresiilor echivalente cu ajutorul grafurilor de strategii.
- ✓ Pentru fiecare cerere se estimeaza costul de executie, bazandu-se pe estimari ale parametrilor relevanti (de exemplu, numarul tuplurilor).
- ✓ In final se identifica costul de executie cel mai mic.



# Graful de strategii

- Graful de strategii reprezinta o metoda pentru studierea tehnicilor de optimizare a interogarilor si reprezinta un instrument foarte util pentru descrierea si studierea optimizarii.
- In literatura de specialitate sunt cunoscute mai multe tipuri de grafuri de interogari, graful de strategii fiind unul dintre ele.
- Primul graf de strategii a fost prezentat in 1984 de catre Lonely Runner si Joachim Rosenthal.



# Graful de strategii

- Pentru construirea grafurilor de strategii se folosesc dreptunghiuri pentru reprezentarea tabelelor si cercuri pentru operatori.
- Nodurile sunt asimilate tabelelor si operatorilor.
  - Graful are un nod rezultat, fiind un nod tabela care reprezinta rezultatul interogarii.
  - Nodurile la care nu sosesc arcuri se numesc noduri de baza, iar celelalte sunt noduri intermediare.
- Arcele au urmatoarea semnificatie:
  - Daca tabela din nodul n1 este intrare pentru operatorul din nodul n2, atunci avem arc  $n1 \rightarrow n2$ ;
  - Daca tabela din nodul n3 rezulta prin aplicarea operatorului din nodul n2, atunci avem arc  $n2 \rightarrow n3$ .

- Daca fiecare nod tabel are cel mult o intrare, graful va reprezenta o singura strategie.
- Daca sunt mai multe intrari intr-un nod tabela, atunci graful de strategii poate reprezenta complet mai multe strategii candidat pentru interogarea propusa.

Exemplu:

Consideram proiectia  $\pi_{A,B,C}(\sigma_{F>0}(E1) \bowtie \sigma_{D \neq F}(E2))$   
care are graful de strategii pentru interogare prezentat  
in Figura 3.

Graful reprezinta o singura strategie.

# Graful de strategii

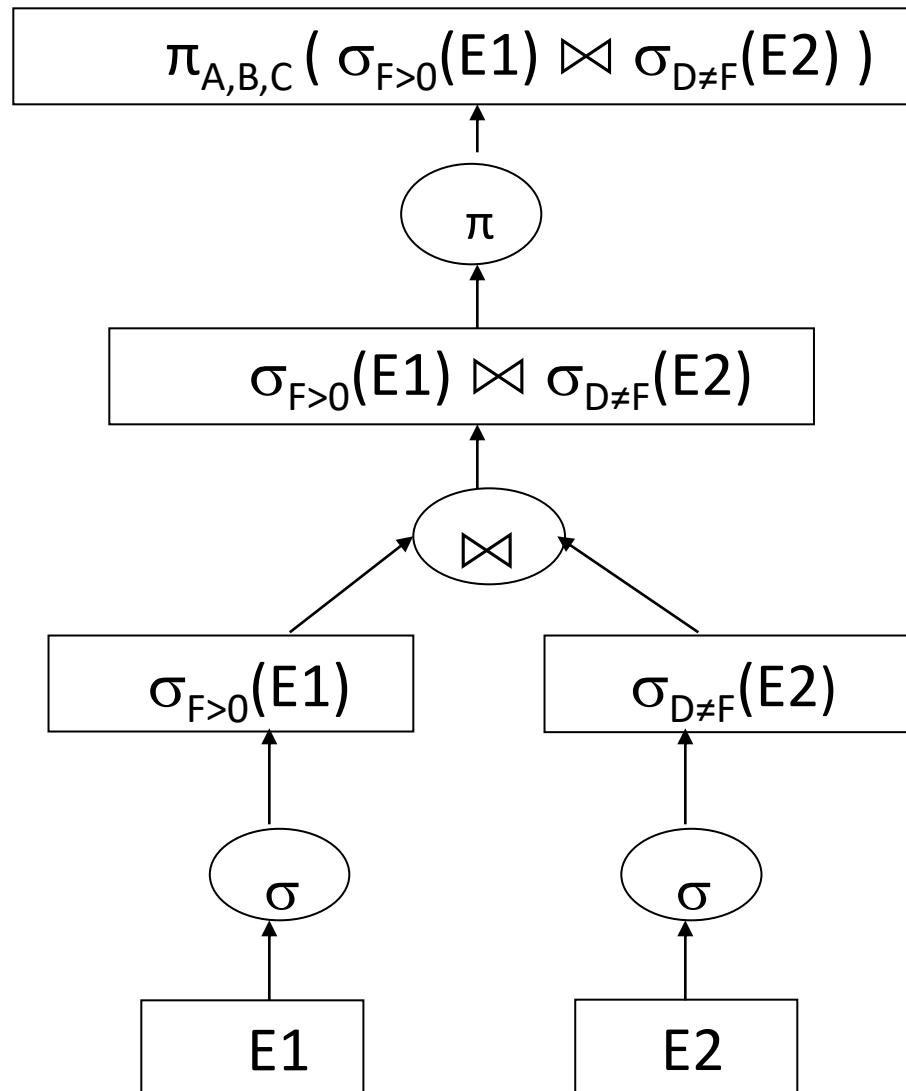


Figura 3. Exemplu de graf cu o singura strategie de interogare



# Elemente de cost impuse de strategie

- In cazul bazelor de date de dimensiuni mari, un aspect foarte important in contextul optimizarii cererilor este minimizarea numarului de accesari ale hard discului.
- Sa luam exemplul unui produs cartezian intre doua relatii **R** si **S**, de cate doua atribute, avand **r** respectiv **s** tupluri.

Fie **nr**, respectiv **ns**, numarul de tupluri din fiecare relatie care incap intr-un bloc pe disc si **b** numarul de blocuri disponibile in memoria interna a calculatorului.



# Elemente de cost impuse de strategie

- Sa presupunem ca procesul de calcul se desfasoara astfel: citim numarul maxim posibil de blocuri din R (adica  $b-1$  blocuri) si pentru fiecare inregistrare din R citim in intregime, in ultimul bloc disponibil, relatia S.

Atunci:

- Numarul de blocuri citite pentru a parurge relatia R este:  $r/nr$  ;
- Numarul de parcurgeri ale relatiei S este:  $r/((b-1)*nr)$  ;
- Pentru o parcuregere a relatiei S se citesc:  $s/ns$  blocuri;



# Elemente de cost impuse de strategie

- Numarul total de accesari ale discului este :

$Nr\_acc = \text{Numarul de blocuri din } R + (\text{Numarul de parcurgeri ale lui } S * \text{Numarul de blocuri din } S)$   
adica:

$$Nr\_acc = (r/nr) + [ r/((b-1) * nr) ] * (s/ns)$$

care se poate rescrie si astfel:

$$Nr\_acc = (r/nr) * (1 + s/((b-1) * ns))$$

- In cazul relatiilor cu numar mare de tupluri aceasta expresie denota timpi mari pentru executia unei astfel de cereri. In cazul de mai sus, deoarece numarul de accesari este mai puternic influentat de  $r/nr$  decat de  $s/ns$ , vom considera ca relatie  $R$  pe cea care are acest raport mai mic , dintre cele doua relatii.



# Elemente de cost impuse de strategie

- Pentru implementarea joinurilor, se folosesc de asemenea diverse metode:
  - Sortarea uneia dintre relatii dupa atributele implicate in join (in cazul equi-joinului);
  - Folosirea indecsilor (daca exista) pe atributele implicate in join, pentru acces direct la tuplurile care dau elemente ale rezultatului;
  - Micsorarea numarului de tupluri luate in calcul, prin aplicarea mai intai a selectiilor, daca acestea sunt prezente in cerere.



# Elemente de cost impuse de strategie

- În general, există o serie de principii (strategii) care duc la micsorarea numărului de accesari ale discului. Ele sunt urmatoarele\*:
  - ✓ Realizarea cu prioritate a selectiilor, pe cat este posibil.
  - ✓ Combinarea anumitor selectii cu produse carteziene adiacente pentru a forma un join.
  - ✓ Combinarea secentelor de operatii unare (selectii și proiecții) într-o singură (o selectie sau/si o proiecție).
  - ✓ Cautarea subexpresiilor comune, pentru a fi evaluate o singură dată.



# Elemente de cost impuse de strategie

- ✓ Folosirea indecsilor sau sortarea relatiilor, daca se obtine o crestere a performantelor.
- ✓ Evaluarea diverselor strategii posibile inainte de a incepe procesul de calcul efectiv (in cazul in care sunt posibile mai multe metode de calcul) pentru a alege pe cea mai eficienta.
- Pe baza acestor principii, exista o serie de algoritmi de optimizare pentru evaluarea expresiilor relationale.

\*(vezi Ullman, J.D. *Principles of Database Systems*, Second Edition, Computer Science Press)



## Etapele executiei unei cereri SQL

- Pentru a minimiza costurile, o cerere SQL este transformata din codul sursa intr-o forma executabila, asa cum se arata in Figura 4, iar principaliii pasi de procesare sunt:
  - **Parsarea** - detectarea eventualele erori de sintaxa și cautarea unei copii deja parsata in memoria *shared pool*;
  - **Optimizarea** – generarea mai multor planuri de executie si identificarea celui optim;
  - **Generarea** – generarea de cod pentru planul de executie optim;
  - **Executia** – cererea este executata cu costuri minime.



# Etapele executiei unei cereri SQL

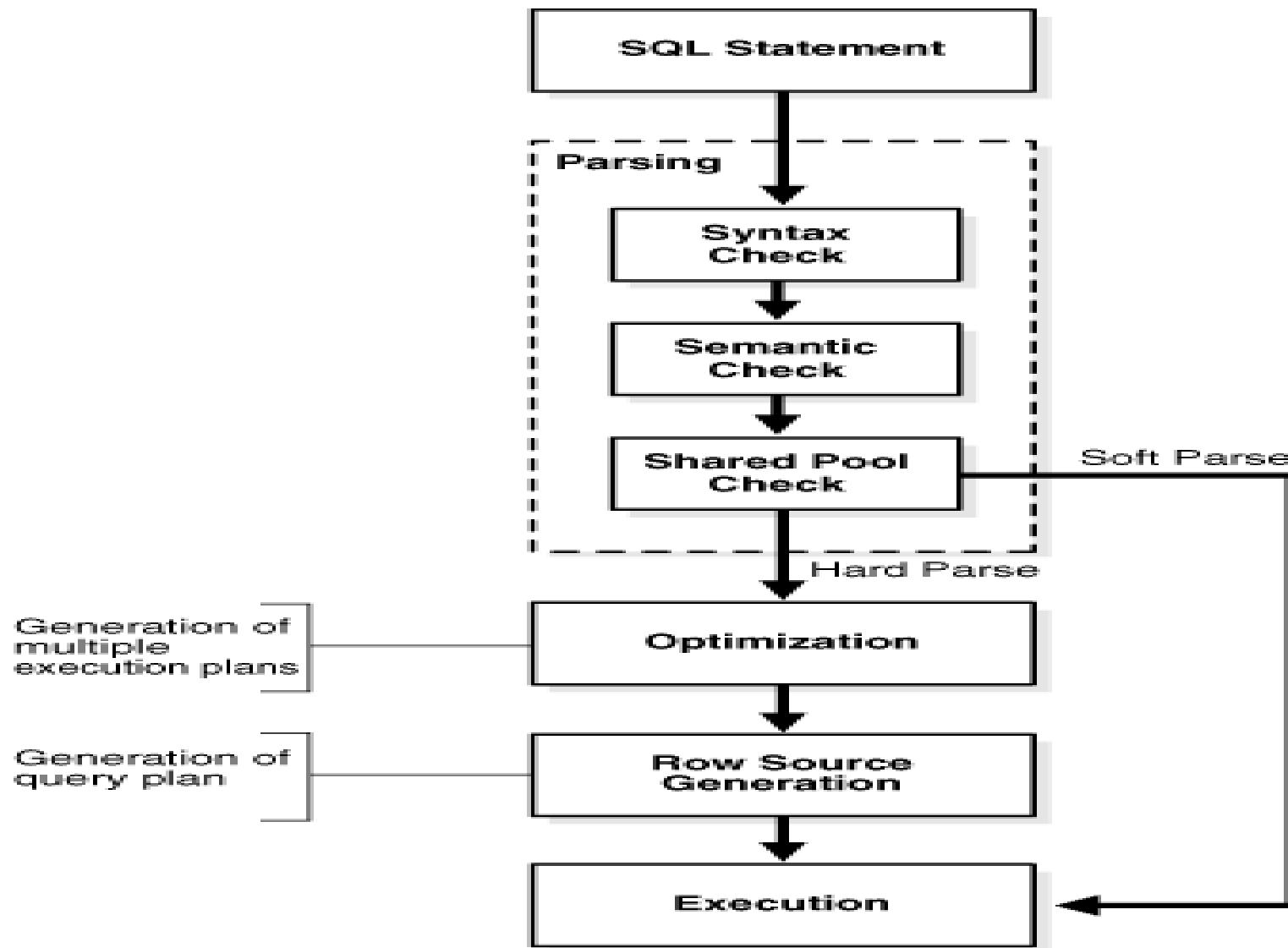


Figura 4. Etapele de executie pentru o cerere SQL



# Etapele executiei unei cereri SQL

## ➤ Parsarea

Aceasta etapa implica anumite verificari, similare compilarii, in vederea detectarii unor eventuale erori:

- **Sintaxa** – se verifica constructia cererii si daca incalca o regula de sintaxa sistemul de gestiune genereaza o eroare;
- **Semantica** – verifica in dictionar obiectele specificate in cerere si structura lor, de exemplu numele tabelelor si coloanelor.
- **Existenta** - verifica daca cererea a mai fost executata de curand si exista deja prelucrata in memoria *Shared Pool*.
- Unele erori nu pot fi detectate in timpul parsarii dar vor aparea in timpul executiei, de exemplu corelarea intre tipul de coloane si valoarea datelor.



# Etapele executiei unei cereri SQL

## ➤ Optimizarea

- Optimizarea este procesul de a alege cele mai eficiente mijloace de a executa o comanda SQL. Optimizatorul, componenta a sistemului de gestiune, optimizeaza interogarile bazandu-se pe statisticile colectate anterior. Pentru optimizare se iau in calcul mai multi parametri, cum ar fi numarul de randuri, dimensiunea setului de date, tipurile de operatori, etc. si se genereaza mai multe planuri de executie. Pentru fiecare plan de executie se calculeaza un cost si in final se alege acela cu costul cel mai mic.
- Optimizatorul trebuie sa efectueze cel putin o data un *hard parse* pentru fiecare comanda DML și efectueaza optimizarea timpului acesteia. Comenzile DDL nu sunt optimizate, cu exceptia cazului in care contin incluse comenzi DML , cum ar fi o subcerere care necesita optimizare.



# Etapele executiei unei cereri SQL

## ➤ Generarea

- Generatorul de cod primește planul de executie optim de la optimizator și produce un plan iterativ, numit plan de interogare, care este utilizat de catre sistemul de gestiune. Planul iterativ este codul sursa al comenzi si poate fi executat pe masina pe care este stocata baza de date. Planul de interogare se executa pas cu pas prin folosirea rezultatelor din pasul anterior si la final se returneaza datele specificate in cerere.
- *Row source este un rand returnat de un pas din planul de executie impreuna cu o structura de control*, care poate procesa iterativ randurile.



## Etapele executiei unei cereri SQL

- *Row Source Generation* produce un arboare row source, care contine urmatoarele informatii:
  - Identificarea tabelelor referite in declaratii;
  - Metode de acces pentru fiecare tabela mentionata in declaratie;
  - Metode de asociere pentru tabelele specificate in join;
  - Operatii efectuate pe date, cum ar fi filtrarea, sortarea sau agregarea.



# Etapele executiei unei cereri SQL

## ➤ Executia

- Executia este etapa finala in care se executa fiecare row source din arborele produs de generatorul Row Source Generation.
- Rezultatul unui *row source* poate fi o linie extrasă dintr-o tabelă( sau view), o linie extrasă din mai multe tabele relateionate printr-un join, rezultatele prelucrarilor de date folosind funcții de sistem, operații efectuate pe datele extrase, etc.



# Optimizatorul bazat pe cost intr-o baza de date Oracle

- Optimizatorul bazat pe cost *CBO(Cost Based Optimization)* este o componenta importanta a sistemului de gestiune Oracle si influenteaza fiecare interogare care este executata pe baza de date.  
Optimizatorul evalueaza fiecare comanda SQL si genereaza cel mai bun plan de executie.
- Sintagma “cel mai bun plan de executie” poate depinde de la o baza de data la alta, in functie de mai multe criterii. Pentru o anumita baza de date un plan de executie bun se poate referi la returnarea rezultatului interogarii intr-un timp cat mai mic, pe cand la o alta baze de date poate fi returnarea rezultatului cu un consum cat mai mic de resurse. De obicei, cel mai bun plan de executie urmareste o medie a celor doua.



# Optimizatorul bazat pe cost intr-o baza de date Oracle

- Pentru a se obtine performanta, trebuie sa fie luati in calcul toti factorii care influenteaza optimizatorul, atat interni cat si externi.
- Cei mai importanți factori sunt parametrii si statisticile optimizatorului.
- CBO este influentat de diferite setari de configurare. Aceste setari pot avea un impact important asupra performantei CBO-ului. Primul dintre acestea este modul de optimizare (*optimizer-mode*).
- *Optimizer\_mode* poate fi setat pentru toate sesiunile, pentru o singura sesiune, sau pentru o singura cerere SQL, direct din SQL\*Plus ca mai jos:



# Optimizatorul bazat pe cost intr-o baza de date Oracle

```
SQL> alter system set optimizer_mode = first_rows_10;  
SQL> alter session set optimizer_goal = all_rows;  
SQL> select * /*+ first_rows(100) */ from employees;
```

- Desi *optimizer-mode* este cel mai important parametru pentru CBO, mai exista si alti parametri care influenteaza comportamentul CBO-ului. Valorile implicite ale acestora pot fi schimbate, dar Oracle nu recomanda schimbarea lor decat cand se cunoaste foarte bine ceea ce implica acest lucru pentru ca schimbarea valorilor poate influenta radical planurile de executie.



# Optimizatorul bazat pe cost intr-o baza de date Oracle

- Modurile de optimizare dintr-o baza de date Oracle sunt urmatoarele:
  - ***rule***
  - ***choose***
  - ***all\_rows***
  - ***first\_rows***

## ➤ Modul ***rule***

- Acest mod bazat pe reguli este unul invecit si nu este recomandat pentru folosire deoarece nu foloseste imbunatatirile care au aparut in noile versiuni Oracle , cum ar fi indexul de tip bitmap.

## ➤ Modul *choose*

- Modul *choose* este cel implicit. Atunci cand parametrul *optimizer\_mode* este setat pe *choose*, va fi ales optimizatorul bazat pe cost cand sunt disponibile statistici si optimizatorul rule in caz contrar.

## ➤ Modul *all\_rows*

- Acest mod va intoarce rezultatul incercand sa minimizeze resursele folosite si este util cand nu este nevoie de un rezultat partial.
- In acest caz se prefera scanarile de tip full-table si nu cele care folosesc un index. Trebuie evitat atunci cand aplicatia are nevoie de rezultate intermediare in timp real.

## ➤ Modul *first\_rows*

- Acest mod va intoarce rezultatul initial intr-un timp cat mai scurt fara sa tina cont de resursele utilizate, si chiar cu posibilitatea ca rezultatul final sa fie obtinut intr-un timp mai mare. De obicei, acest mod alege sa foloseasca un index pentru returnarea rezultatului. Este util atunci cand utilizatorul are nevoie sa vada o mica parte a rezultatului intr-un timp cat mai scurt.

## ➤ Modul *first\_rows\_n*

- Functioneaza similar cu *first\_rows* dar are cateva optiuni predefinite: *first\_rows\_1*, *first\_rows\_10*, *first\_rows\_100* si *first\_rows\_1000*. Precizand dinainte cate randuri se doreste a fi intoarse optimizatorul este ajutat sa faca o decizie in privinta alegerii, sau nu, a unui index.



# Optimizatorul bazat pe cost intr-o baza de date Oracle

- Modul *all\_rows* este folosit pentru minimizarea resurselor si favorizeaza scanarile de tip *full-table* iar modul *first\_rows* realizeaza mai multe operatiuni de tip I/O dar returneaza mai repede liniile (primele  $n$  dintre acestea).
- Atunci cand o interogare are ca sursa mai multe tabele, Oracle intocmeste o ordine, in functie de anumite criterii. De exemplu, atunci cand tabelele au constrangeri de tipul UNIQUE sau PRIMARY KEY, aceste tabele vor fi primele. Dupa ce a stabilit ordinea tabelelor, Oracle genereaza un set de planuri de executie, carora le ataseaza costurile si il alege pe acela cu costul cel mai mic.



# Optimizatorul bazat pe cost intr-o baza de date Oracle

- De multe ori dezvoltatorii considera ca este de ajuns sa fie scrisa o interogare care returneaza rezultatul corect fara sa tina cont de performanta. In realitate, scrierea unei astfel de interogari nu reprezinta nici jumata din timpul necesar scrierii unei interogari corecte. Accesarea unei baze de date intr-un mod optim se dovedeste de multe ori a fi un pas important pentru succesul unei aplicatii.
- Uneori, rescrierea unui SQL, tinand cont de cele prezentate, poate reduce atat de mult timpul de executie incat de la cateva ore se poate ajunge la cateva secunde.



# Optimizatorul bazat pe reguli intr-o baza de date Oracle

- Un alt tip de optimizator folosit de un SGDB Oracle este Optimizatorul bazat pe reguli RBO(*Rule Based Optimization*). Aceasta este o tehnica de optimizare mai veche bazata pe reguli predefinite de executie a unei cereri de interogare. De exemplu, daca o tabela are un index atunci orice interogare implica folosirea indexului, dar nu este totdeauna cea mai buna strategie deoarece, uneori, un index poate creste timpul de executie a unei interogari.
- Trebuie specificat ca acest tip de optimizator foloseste numai reguli predefinite, fara a se baza pe statistici, ceea ce face ca uneori rezultatele sa nu fie cele scontate. Modurile de optimizare *rule*, *choose* sunt folosite de RBO dar nu au mai fost imbunatatite incepand cu versiunea Oracle 10g.



# Capitolul 10

## Tranzactii si acces concurrent



# Tranzactii pe baza de date

- O tranzactie reprezinta o prelucrare logica asupra uneia sau mai multor date dintr-o baza de date, executata prin intermediul unui program.
- Tranzactia poate fi vazuta si ca o unitate de prelucrare sequentiala a datelor care respecta consistenta bazei de date.
- In cazul accesului la aceleasi date se spune ca executiile programelor sunt concurente sau ca exista un acces concurrent la date.
- In mod normal, o tranzactie completa lasa baza de date asupra careia a actionat intr-o stare consistenta.
- Atunci cand mai multe programe opereaza simultan pe aceleasi date pot sa apara situatii in care continutul bazei de date devine inconsistent.
- Gestiunea, controlul si executia tranzactiilor este sarcina exclusiva a sistemului de gestiune.



# Tranzactii pe baza de date

- Tranzactia completa este unitatea de prelucrare executata integral, ea reprezentand parcurgerea si prelucrarea corecta a tuturor seventelor de program care o genereaza.
- Tranzactia incompleta este o tranzactie “abandonata”, respectiv executata fara succes.
- In cazul abandonarii executiei, toate rezultatele actiunilor efectuate si terminate pana in acel moment sunt anulate.
- Dintre cauzele abandonarii unei tranzactii, pot fi amintite:
  - ✓ Aparitia unei anomalii in timpul executiei care determina SGBD sa abandoneze tranzactia(daca sursa anomaliei a fost interna, iar intreruperea executiei a fost decisa de catre SGBD, executia tranzactiei este reluata automat).
  - ✓ Tranzactia in curs de executie intalneste o situatie neprevazuta, cum ar fi, citirea unei valori nule sau unei date in format gresit.
  - ✓ Aparitia unei probleme hardware a serverului, a retelei sau a unei pene de curent.



# Tranzactii pe baza de date

- Sa consideram un exemplu de comert online in care doi clienti vor sa cumpere acelasi produs.

Timp (t)	Client 1 (T1)	Client 2 (T2)	Stoc (S)
t1	READ S		100
t2		READ S	100
t3	S = S - 1		100
t4		S = S - 1	100
t5	WRITE S		99
t6		WRITE S	99

- Se observa ca stocul initial era 100 iar stocul final 99, ceea ce nu este corect deoarece s-au vandut 2 bucati din produsul respectiv.
- Aceasta anomalie apare din cauza ca tranzactiile T1 si T2 nu au fost executate in ordinea corecta.
- In acest caz spunem ca dupa executia tranzactiilor T1 si T2 avem o inconsistenta a bazei de date.



# Tranzactii pe baza de date

- In exemplul anterior tranzactiile T1 si T2 au fost rezultatul executiei aceleiasi operatii pe baza de date. Sunt situatii in care pot fi operatii diferite, de ex. Client 1 executa operatia de cumparare(T1) si Client 2 executa operatia de renuntare la cumparare (T2).

Timp (t)	Client 1 (T1)	Client 2 (T2)	Stoc (S)
t1	READ S		100
t2		READ S	100
t3	S = S - 1		100
t4		S = S + 1	100
t5	WRITE S		99
t6		WRITE S	101

- Se observa ca stocul initial era 100 iar stocul final 101, ceea ce nu este corect deoarece s-a vandut o bucată din produsul respectiv.
- Si in acest caz spunem ca dupa executia tranzactiilor T1 si T2 avem o inconsistenta a bazei de date.



# Tranzactii pe baza de date

Observatii:

- Pe o aceeasi baza de date se pot executa simultan mai multe tranzactii, care lucreaza cu una sau mai multe date din baza de date;
- Fiecare tranzactie poate citi mai multe date si poate scrie mai multe date in baza de date (nu neaparat cele citite, pot fi si altele);
- Tranzactiile care nu executa operatii de scriere sau de citire de date din baza de date nu duc la inconsistente;
- In exemplul anterior incrementarea si decrementarea lui S nu sunt cauza inconsistentelor, ci ordinea in care s-au facut scrierile rezultatelor in baza de date;
- De aceea in unele dintre exemplele din acest capitol nu vom mai figura acest tip de operatii. O executie concurrenta pentru patru tranzactii se va reprezenta astfel:



# Tranzactii pe baza de date

Temp	T1	T2	T3	T4
t1	READ A			
t2	READ B			
t3		READ A		
t4		READ B		
t5			WRITE B	
t6		WRITE B		
t7				READ B
t8				READ C
t9	WRITE A			
t10		WRITE C		

- Pot exista mai multe tipuri de tranzactii:
- ✓ Tranzactii care scriu date citite anterior (T1 il scrie pe A, citit anterior);
- ✓ Tranzactii care scriu date care nu au fost anterior citite, calculate eventual pe baza altor date citite din baza de date (T2 scrie pe C pe care nu l-a citit, dar a citit A si B);
- ✓ Tranzactii care doar citesc date fara sa scrie (T4);
- ✓ Tranzactii care doar scriu date fara sa citeasca anterior ceva din baza de date (T3).



# Tranzactii pe baza de date

- **Definitie:** O **tranzactie** este o colectie de operatii efectuate asupra starii logice si fizice a unei baze de date, executate prin intermediul unei comenzi sau a unui program.
- **Definitie:** Un **articol** este o parte a bazei de date care se poate citi, scrie sau bloca/debloca printr-o singura operatie de READ, WRITE sau LOCK / UNLOCK.
- In exemplul anterior am folosit articolele simbolice A, B si C. In cazurile reale un articol poate fi:
  - ✓ O intreaga tabela;
  - ✓ O linie sau mai multe linii dintr-o tabela;
  - ✓ O coloana sau mai multe coloane dintr-o tabela;
  - ✓ O celula dintr-o tabela (valoarea unei coloane de pe o linie a tablei);
  - ✓ Orice alta parte a bazei de date care indeplineste conditia din definitie, in concordanta cu facilitatile puse la dispozitie de catre SGBD-ul respectiv.



# Planificarea tranzactiilor

- Sistemul de gestiune a bazei de date blocheaza automat orice linie modificata de o comanda de tip UPDATE pana cand tranzactia care a efectuat o operatie fie comite modificarile (le face permanente in baza de date), fie le revoca.
- **Definitie:** O **planificare** reprezinta ordinea in care sunt executati de catre SGBD pasii elementari ai unui set de tranzactii.
- Planificarea este deci o lista de pasi pentru un set de tranzactii care se executa concurrent si arata ca SGBD-ul executa acesti pasi exact in acea ordine.
- In continuare vom reprezenta doar pasii care semnifica o interactiune a tranzactiei cu datele din baza de date:
  - ✓ READ – citirea unui articol
  - ✓ WRITE – scrierea unui articol
  - ✓ LOCK – blocarea unui articol (in diverse forme)
  - ✓ UNLOCK – deblocarea unui articol



# Planificarea tranzactiilor

- Există și operații pe baza de date care nu ridică probleme de acces concurrent:
  - ✓ COMMIT – comiterea modificărilor efectuate de o tranzacție;
  - ✓ ROLLBACK – revocarea modificărilor efectuate de o tranzacție.
- O planificare poate fi reprezentată în mai multe moduri:
  - ✓ Sub forma tabelară, ca în exemplele anterioare, în care ordinea execuției este de sus în jos.
  - ✓ Sub forma unei liste în care sunt folosite următoarele notări:
    - $R_i(A)$  - Tranzacția  $T_i$  citește articolul  $A$ ;
    - $W_i(A)$  - Tranzacția  $T_i$  scrie articolul  $A$ .
- Ultimul exemplu tabelar care reprezintă tranzacțiile T1-T4 poate fi reprezentat sub forma unei liste astfel:

$R1(A); R1(B); R2(A); R2(B); W3(B); W2(B); R4(B); R4(C); W1(A); W2(C)$ .



# Planificare seriala

- **Definitie:** O planificare in care pasii fiecarei tranzactii sunt succesivi, fara sa fie intercalati pasi ai altor tranzactii, se numeste **planificare seriala**.
- Exemplu de planificare seriala pentru doua tranzactii T1 si T2:

R1(A); R1(B); R1(C); W1(B); R2(B); R2(C); W2(A); W2(C)

---

Pasii lui T1

Pasii lui T2

- Planificarile seriale nu ridica probleme de consistenta (sunt planificari “bune” din punct de vedere al executiei concurente).
- De aceea unul dintre obiectivele acestui capitol este acela de a gasi planificari care sa se comporte la fel ca o planificare seriala.
- Una dintre metode de a obtine planificari care sa nu ridice probleme privind consistenta datelor dupa executia tranzactiilor este aceea a blocarii articolelor.



# Blocarea articolelor

- **Definitie:** Blocarea unui articol de catre o tranzactie semnifica faptul ca acea tranzactie obtine din partea SGBD-lui drepturi speciale de acces, care impiedica alte tranzactii sa efectueze anumite operatii asupra acelui articol.
- Există două categorii de blocari:
  - ✓ Blocari exclusive: celelalte tranzactii nu pot să execute operatii asupra articolului blocat. Aceste blocari sunt denumite în literatura de specialitate și EXCLUSIVE LOCKS sau WRITE LOCKS;
  - ✓ Blocari partajate: celelalte tranzactii pot să execute doar anumite tipuri de operatii asupra articolului blocat. Aceste blocari sunt denumite în literatura de specialitate și SHARED LOCKS sau READ LOCKS.



# Blocarea articolelor

Observatii:

- Pentru a scrie un articol, o tranzactie trebuie sa obtina anterior un Write Lock asupra acestuia: nicio alta tranzactie nu poate citi sau scrie acel articol pana cand nu este deblocat;
- Pentru a citi un articol o tranzactie trebuie sa obtina anterior un Read Lock asupra lui;
- Mai multe tranzactii pot sa blocheze acelasi articol pentru citire, caz in care nicio alta tranzactie nu poate sa scrie in el.
- O tranzactie poate sa obtina un Write Lock pe articolul respectiv abia dupa deblocarea acestuia de catre toate tranzactiile care l-au blocat pentru citire.
- Articolele pot fi deblocate unul cate unul de catre tranzactia care le-a blocat sau pot fi deblocate toate in cazul unui COMMIT sau unui ROLLBACK, in functie de modelul de blocare folosit.
- O tranzactie mai poate fi definita ca o succesiune de operatii de scriere/citire pe baza de date sau operatii de blocare/deblocare, comitere sau revocare.



# Proprietatile tranzactiilor

- O tranzactie trebuie sa indeplineasca urmatoarele criterii, care au fost sintetizate prin cateva proprietati sub abrevierea ACID:
  - ✓ A – Atomicitate
  - ✓ C – Consistenta
  - ✓ I – Izolare
  - ✓ D – Durabilitate.
- **Atomicitate:** schimbarile starii unei tranzactii sunt atomice, in sensul ca aceste schimbari se executa in totalitate(comise), sau niciuna(revocate). Schimbarile sunt modificari in baza de date, mesaje sau actiuni asupra subsistemelor din mediul extern.
- **Consistenta:** o tranzactie constituie o transformare corecta a starii bazei de date. Actiunile intreprinse in grup nu incalca niciuna dintre restrictiile de integritate specifice starii mentionate. De aici rezulta ca tranzactia trebuie executata de catre un program corect.



# Proprietatile tranzactiilor

- **Izolare:** chiar daca tranzactiile sunt executate in mod concurrent, fiecare tranzactie T “vede” executia celorlalte tranzactii ca fiind produsa fie anterior, fie dupa T, dar nu simultan.
- **Durabilitate:** odata ce o tranzactie a fost executata cu succes (deci a fost “completa” sau “comisa”), schimbarile pe care le-a produs in starea bazei de date nu vor fi afectate de eventuale erori viitoare.

Exemple:

- ✓ Atomicitate: Sa consideram urmatoarea comanda SQL care indexeaza cu 10% salariile in tabela SALARII:  
  
    > UPDATE SALARIU SET SALARIU=1.1\*SALARIU;
- Daca in urma indexarii un salariu va fi mai mare de 5000, toate modificarile vor fi revocate( in ideea ca exista o constrangere de tip CHECK(salariu<=5000) pe tabela SALARII).



# Proprietatile tranzactiilor

- Aceste modificari trebuie comise impreuna sau revocate impreuna.
- Sistemul de gestiune este cel care trebuie sa puna la dispozitie mecanismele prin care sa se asigura atomicitatea tranzactiilor, inclusiv in cazul unor incidente hardware si software care pot interveni in timpul executiei unei tranzactii.
- ✓ Consistenta: Sa consideram ca tabela STUDENTI are o constrangere NOT NULL pe coloana Matricol. Daca se executa comanda INSERT fara a specifica o valoare pentru Matricol, se va genera o eroare si inserarea nu se va efectua.
- In cele mai multe cazuri aceste restrictii sunt modelate sub forma constrangerilor de integritate (NOT NULL, PRIMARY KEY, FOREIGN KEY, etc.).
- Daca o tranzactie contine o operatie care violeaza o constrangere de integritate atunci toate modificarile efectuate de tranzactie vor fi revocate.
- Mecanismele de pastrare a consistentei trebuie asigurate de catre sistemul de gestiune.



# Proprietatile tranzactiilor

- ✓ Izolarea: Sa consideram ca un user insereaza o linie in tabela PRETURI cu comanda:  
    > INSERT INTO PRETURI ...
- Un alt user incearca sa modifice pretul unui produs cu comanda:  
    > UPDATE PRETURI SET ...
- Userul al doilea nu va putea sa execute comanda pana cand primul user nu finalizeaza comanda de inserare prin comiterea tranzactiei.
- O tranzactie trebuie sa se comporte ca si cand operatiile efectuate de ea sunt izolate, independente de operatiile efectuate de alta tranzactie.
- Nicio alta tranzactie nu trebuie sa citeasca date intermediare scrise de tranzactia respectiva.
- Modul de asigurare a izolarii tranzactiilor este tot sarcina sistemului de gestiune.



# Proprietatile tranzactiilor

- ✓ Durabilitatea: Sa consideram ca un user modifica anumite preturi in tabela PRETURI, dupa care comite tranzactia:
  - > UPDATE PRETURI SET ...
  - > COMMIT;
- Odata comise cu succes modificarile efectuate de catre o tranzactie ele vor persista in baza de date si nu mai pot fi revocate.
- Inclusiv in cazul unui incident hardware sau software, efectele tranzactiilor comise sunt regasite in baza de date dupa restartarea sistemului.
- Sistemul de gestiune a bazei de date trebuie sa contine mecanisme prin care efectele tuturor tranzactiilor comise sa fie inregistrate si in fisierele de log, pentru a putea reface starea bazei de date in cazul unui incident.



# Serializabilitatea tranzactiilor

- Asa cum s-a specificat planificarile seriale nu duc la inconsistente.
- Intr-o functionare normala, planificarile pot contine pasi intercalati ai diverselor tranzactii.
- Rezultatul va fi totusi corect, daca efectul executiei planificarii respective este acelasi cu al unei planificari seriale ale acelorasi tranzactii.
- O astfel de planificare se numeste **planificare serializabila**.
- **Definitie:** O planificare este **serializabila** daca produce aceleasi efecte in baza de date ca o planificare seriala.



# Serializabilitatea tranzactiilor

Exemple:

1) Sa analizam urmatoarele planificari:

T1	T2	T3
Read A		
Read B		
		Read C
		Read D
Write A		
		Write C
	Read A	
	Write A	

Planificare seriarizabila

T1	T2	T3
		Read C
		Read D
		Write C
Read A		
Read B		
Write A		
	Read A	
	Write A	

Planificare seriala echivalenta

- Planificarile au acelasi rezultat in urma executiei tranzactiilor, deci planificarea initiala este seriarizabila.



# Serializabilitatea tranzactiilor

2) Consideram urmatoarea planificare:

T1	T2	(baza de date )
Read A		10
	Read A	10
A = A + 1		10
	A = A + 1	10
Write A		11
	Write A	11

Planificare neseriala

- Planificarea aceasta este neserializabila deoarece nu se poate gasi una seriala echivalentă. La final baza de date nu este într-o stare consistentă deoarece ambele tranzacții au incrementat pe A citit initial.



# Serializabilitatea tranzactiilor

- 3) Planificările următoare sunt seriale, dar niciuna nu este echivalentă cu cea anterioară deoarece au ca efect rezultate diferite pentru A (bază de date este într-o stare consistentă la final):

T1	T2	A
Read A		10
A=A+1		10
Write A		11
	Read A	11
	A=A+1	11
	Write A	12

T1	T2	A
	Read A	10
	A=A+1	10
	Write A	11
Read A		11
A=A+1		11
Write A		12

Planificări seriale



# Conflict in planificarea tranzactiilor

- Există și o altă abordare a serializabilității bazată pe conflictelor care pot să apară între pasii a două tranzacții dintr-o planificare.
- **Definție:** Între două operații aparținând unei planificări există un **conflict** dacă:
  - Apărătunor tranzacții diferite;
  - Sunt executate pe același obiect;
  - Una dintre operații este o scriere;
  - Cele două operații sunt succeseive (între ele nu există alta operație cu care una dintre ele este în conflict).
- Rezultă că există trei tipuri de situații conflictuale:  
Fieind date două tranzacții T1 și T2, pot exista conflicte de tipurile:  
**R1-W2, W1-R2 și W1-W2**



# Conflict in planificarea tranzactiilor

- **Definitie:** Doua planificari sunt **conflict-echivalente** daca:
  - Contine aceleasi operatii, ale acelorasi tranzactii;
  - Fiecare pereche de operatii conflictuale apare in aceeasi ordine in cele doua planificari.
- Aceasta definitie nu spune ca nu pot sa apara anomalii in executia celor doua planificari, ci ca pot sa apara aceleasi anomalii in ambele.
- **Definitie:** O planificare este **conflict-serializabila** daca este conflict-echivalenta cu o planificare seriala.
- O definitie echivalenta: O planificare este conflict-serializabila daca poate fi transformata intr-o planificare seriala, prin interschimbari ale operatiilor consecutive din doua tranzactii care nu sunt in conflict .
- O planificare poate fi modificaata prin interschimbarea operatiilor neconflictuale, obtinandu-se o planificare seriala.



# Conflict in planificarea tranzactiilor

Exemplu:

- Plecand de la urmatoarea planificare serializabila, prin aplicarea succesiva a trei interschimbari de operatii se obtine o planificare seriala:

T1	T2
Read A	
Write A	
	Read A
Read B	
	Write A
Write B	
	Read B
	Write B

T1	T2
Read A	
Write A	
Read B	
	Read A
	Write A
Write B	
	Read B
	Write B

Pas 1

T1	T2
Read A	
Write A	
Read B	
	Read A
	Write A
Write B	
	Read A
	Write A
	Read B
	Write B

Pas 2

T1	T2
Read A	
Write A	
Read B	
	Read A
	Write A
Write B	
	Read A
	Write A
	Read B
	Write B

Pas 3



# Testarea conflict-seriazibilitatii

- Algoritmul de testare a conflict-seriazibilitatii se bazeaza pe constructia grafului de dependenta.
- ✓ **Algoritm de testare a conflict-serializabilitatii**
- Se construieste graful de dependenta astfel:
  - Nodurile reprezinta tranzactiile;
  - Pentru orice pereche de operatii  $O_i$  si  $O_j$  aflate in conflict, cu  $O_i$  in  $T_i$  si  $O_j$  in  $T_j$ , avem un arc de la nodul  $T_i$  la  $T_j$  daca  $O_i$  apare in planificare inaintea lui  $O_j$ .
- Daca graful obtinut nu contine cicluri, planificarea este conflict-serializabila, altfel nu este conflict-serializabila.



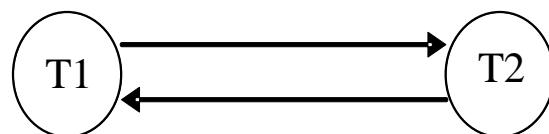
# Testarea conflict-seriazibilitatii

Exemplu:

- Consideram urmatoarea planificare:

T1	T2
<b>Read A</b>	
	<b>Write A</b>
<b>Write A</b>	

- Graful de dependenta are doua noduri si doua arce:



- Avem doua conflicte:

T1 – Read A cu T2 – Write A

T2 – Write A cu T1 – Write A



# Testarea conflict-seriazibilitatii

- Deoarece graful obtinut contine un ciclu, planificarea nu este conflict-serializabila.

Observatii:

- Exista planificari serializabile care nu sunt conflict-serializabile.
- De exemplu, sa presupunem ca in planificare de mai sus tranzactia T2 scrie in A exact valoarea citita de T1. In acest caz planificarea nu este conflict-serializabila dar este serializabila, avand acelasi efect cu planificarea seriala “T2 urmata de T1”:

T1	T2
	Write A
Read A	
Write A	



# Testarea conflict-seriazibilitatii

- Acest fapt se datoreaza insa doar coincidentei intre valoarea scrisa de T2 si cea citita de T1.
- Sistemul de gestiune nu ia insa decizii bazate pe aceste cazuri particulare, planificarea este considerata periculoasa deoarece poate sa duca la inconsistenta bazei de date.
- De aceea in evaluarea planificarilor se considera ca la scriere o tranzactie poate scrie orice valoare si nu doar o valoare particulara.
- Exista si o a treia abordare a serializabilitatii numita **v-serializabilitate** (*view serializability*), similara cu conflict-seriazibilitatea (*conflict serializability*):



# Testarea conflict-seriazibilitatii

- **Definitie:** Doua planificari  $S_1$  si  $S_2$  sunt **v-echivalente** daca pentru orice articol A:
  - Daca Ti citeste valoarea initiala a lui A in  $S_1$ , atunci ea face acelasi lucru si in  $S_2$ ;
  - Daca Ti citeste o valoare a lui A scrisa de  $T_j$  in  $S_1$ , atunci face acelasi lucru si in  $S_2$ ;
  - Daca Ti scrie valoarea finala a lui A in  $S_1$ , atunci ea face acelasi lucru si in  $S_2$ .
- **Definitie:** O planificare este **v-serializabila** daca este v-echivalenta cu o planificare seriala.
- Aceasta definitie accepta planificari de tranzactii conflict-serializabile si planificari care contin tranzactii care scriu date fara sa citeasca ceva din baza de date.



# Testarea conflict-seriazibilitatii

Exemplu:

- Mai jos este prezentata o planificare v-serializabila si o planificare seriala v-echivalenta:

T1	T2	T3
<b>Read A</b>		
	<b>Write A</b>	
<b>Write A</b>		
		<b>Write A</b>

Planificare v-serializabila

T1	T2	T3
<b>Read A</b>		
<b>Write A</b>		
	<b>Write A</b>	
		<b>Write A</b>

Planificare seriala v-echivalenta

- Planificarea nu este conflict-serializabila dar este v-serializabila.



# Blocari in tranzactii

- Pentru a putea asigura serializabilitatea tranzactiilor sistemele de gestiune pun la dispozitie posibilitatea de blocare a articolelor.
- Daca o tranzactie blocheaza un articol, celelalte tranzactii care vor sa aiba acces la acelasi articol pot fi puse in asteptare pana la deblocarea acestuia.
- Exista mai multe modele de blocare, prezentate in continuare.



# Modelul LOCK / UNLOCK

## ➤ **Modelul LOCK/UNLOCK**

- În cadrul acestui model există o singură opțiune de blocare, LOCK, ea ducând la obținerea unui acces exclusiv la articol pentru tranzacția care îl blochează (celelalte tranzacții nu pot nici scrie și nici citi articolul).
- Deblocarea se face cu opțiunea UNLOCK.
- Vom presupune în continuare că o tranzacție:
  - nu blochează un articol deja blocat de ea;
  - nu deblocă un articol pe care nu l-a blocat ea.



# Verificarea serializabilitatii

- Pentru verificarea serializabilitatii se construieste **graful de precedenta G**.

## ✓ **Algoritm de verificarea a serializabilitatii**

- Se construieste graful de precedenta G in care nodurile sunt tranzactiile planificate;
- Daca pentru vreun articol A avem in planificarea S secventa:  
Ti: UNLOCK A  
Tj: LOCK A

atunci vom avea in graf un arc de la nodul Ti la nodul Tj.

Tj este prima tranzactie care blocheaza A dupa deblocarea sa de catre Ti.

- Daca graful G are cicluri, atunci S nu este serializabila.
- Daca graful G nu are cicluri, atunci S este serializabila si planificarea seriala echivalenta se obtine prin **sortarea topologica** a grafului G.



# Sortare topologica



## Algoritm de sortarea topologica

- Se alege un nod care nu are arce de intrare (exista cel putin un astfel de nod, neexistand cicluri);
- Se listeaza tranzactia asociata nodului dupa care acesta este sters din graf impreuna cu toate arcele care ies din el;
- Procesul se reia ciclic;
- Daca nu intervine nicio schimbare fata de starea anterioara , atunci procesul se opreste.



# Sortare topologica

Exemplu:

- Sa consideram urmatoarea planificare:

T1	T2	T3
	<b>Lock A</b>	
	<b>Unlock A</b>	
		<b>Lock A</b>
		<b>Unlock A</b>
<b>Lock B</b>		
<b>Unlock B</b>		
	<b>Lock B</b>	
	<b>Unlock B</b>	

- Graful obtinut nu are cicluri, deci planificarea este serializabila si planificarea seriala echivalenta (obtinuta prin sortare topologica) este T1, T2, T3.





# Protocolul de blocare in doua faze

- **Definitie:** O tranzactie respecta **protocolul de blocare in doua faze** daca toate blocarile preced toate deblocarile.
- Acest protocol garanteaza serializabilitatea: daca toate tranzactiile respecta cerintele protocolului se poate demonstra ca orice planificare a lor e serializabila.
- De asemenea se poate demonstra ca daca o tranzactie nu respecta protocolul pot exista executii neserializabile ale acelei tranzactii in conjunctie cu alte tranzactii.

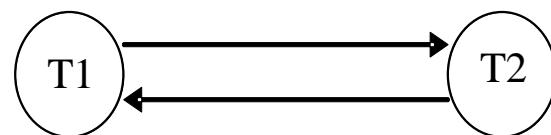


# Protocolul de blocare in doua faze

Exemple:

1) Consideram urmatoarea planificare:

T1	T2
<b>Unlock A</b>	
	<b>Lock A</b>
	<b>Lock B</b>
	<b>Unlock A</b>
	<b>Unlock B</b>
<b>Lock B</b>	



- Graful de precedenta contine un ciclu, deci planificarea nu este serializabila.
- Planificarea nu respecta nici protocolul de blocare in doua faze deoarece pentru T1 avem Unlock A inainte de Lock A.



# Protocolul de blocare in doua faze

2) Protocolul de blocare in doua faze implica uneori operatii de rollback in cascada: In momentul Rollback pentru T1 este necesar Rollback si pentru T2 deoarece T2 a citit date scrise de T1, date care prin Rollback se pierd.

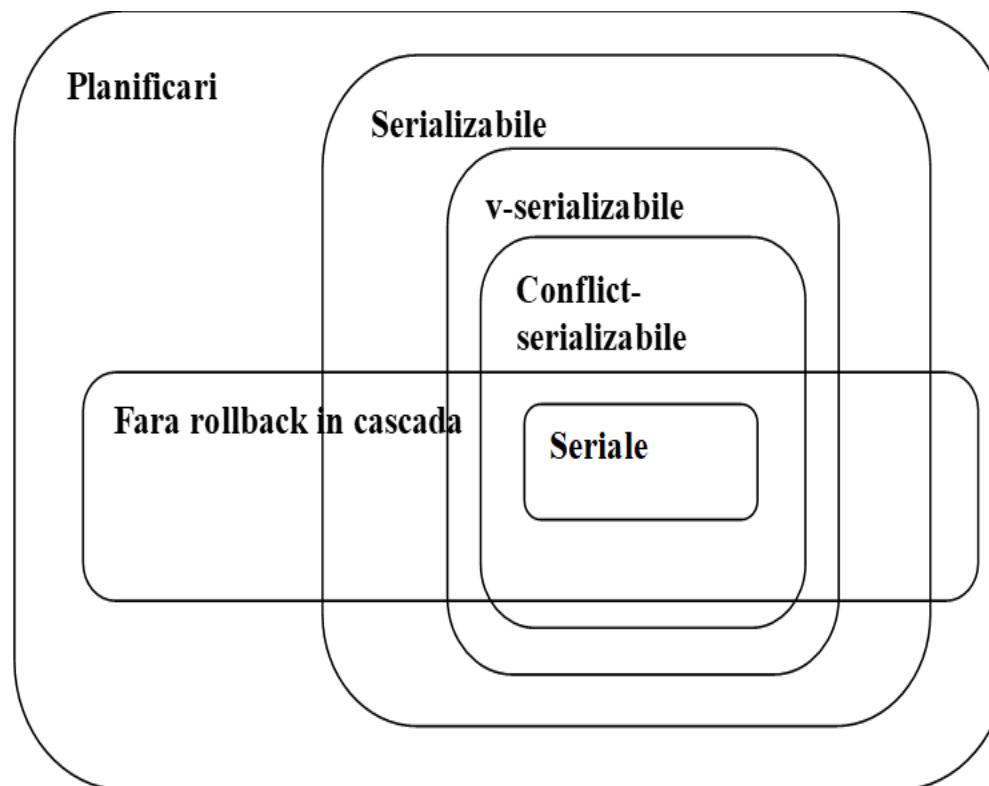
- O astfel de planificare se numeste planificare cu rollback in cascada.
- Exista in acest caz varianta protocolului de blocare stricta in doua faze care implica eliberarea tuturor articolelor blocate la sfarsitul tranzactiei.

T1	T2
Lock A	
Lock B	
Read A	
Write A	
Unlock A	
	Lock A
	Read A
	Write A
	Unlock A
Read B	
Write B	
Rollback	



# Incluziunea planificarilor

- Schema bloc de incluziune a planificarilor este urmatoarea:





# Modelul RLOCK / WLOCK

## ➤ **Modelul RLOCK/WLOCK**

- În cadrul acestui model există o două opțiuni de blocare:
  - RLOCK (blocare pentru citire). Oricate tranzactii pot bloca același articol pentru citire, dar o tranzacție nu poate bloca pentru scriere un articol blocat cu RLOCK;
  - WLOCK (blocare pentru citire). Duce la obținerea unui acces exclusiv la articol pentru tranzacția care îl blochează. Celelalte tranzactii nu mai pot să blocheze cu RLOCK sau WLOCK acel articol.
- Deblocarea pentru ambele tipuri se face cu UNLOCK.
- Vom presupune și în acest caz că o tranzacție:
  - nu blochează un articol deja blocat de ea;
  - nu deblocă un articol pe care nu l-a blocat ea.



# Modelul RLOCK / WLOCK

Observatii:

- Si pentru acest model se poate construi un graf de precedenta (altfel decat in paragraful anterior) din care se poate deduce daca planificarea e serializabila sau nu;
- Si pentru acest model protocolul de blocare in doua faze este valabil;
- Daca toate blocarile (de orice fel, la citire sau la scriere) preced toate deblocarile, planificarea este serializabila.



# Modelul RLOCK / WLOCK

## ✓ Algoritm de verificare a serializabilitatii

- **Intrare:** O planificare P a multimii de tranzactii T<sub>1</sub>, T<sub>2</sub>, ..., T<sub>k</sub>.
- **Ieșire:** Raspunsul daca planificarea este serializabila.
- **Metoda:** Construirea grafului de precedenta G, similar cu modelul anterior: fiecarei tranzactii ii corespunde un nod al grafului iar arcele se traseaza astfel:
  - 1. Fie T<sub>i</sub> tranzactia care executa RLOCK A iar T<sub>j</sub> urmatoarea tranzactie (diferita de T<sub>i</sub>) care face WLOCK A. Trasam atunci un arc de la T<sub>i</sub> la T<sub>j</sub>.
  - 2. Fie T<sub>i</sub> tranzactia care face WLOCK A si T<sub>j</sub> urmatoarea tranzactie (daca exista) care face WLOCK A. Se traseaza un arc de la T<sub>i</sub> la T<sub>j</sub>. De asemenea, in acest ultim caz fie T<sub>m</sub> tranzactia care face RLOCK A dupa ce T<sub>i</sub> elibereaza pe A dar inainte de blocarea acestuia de catre T<sub>j</sub>. Se traseaza un arc de la T<sub>i</sub> la T<sub>m</sub>.

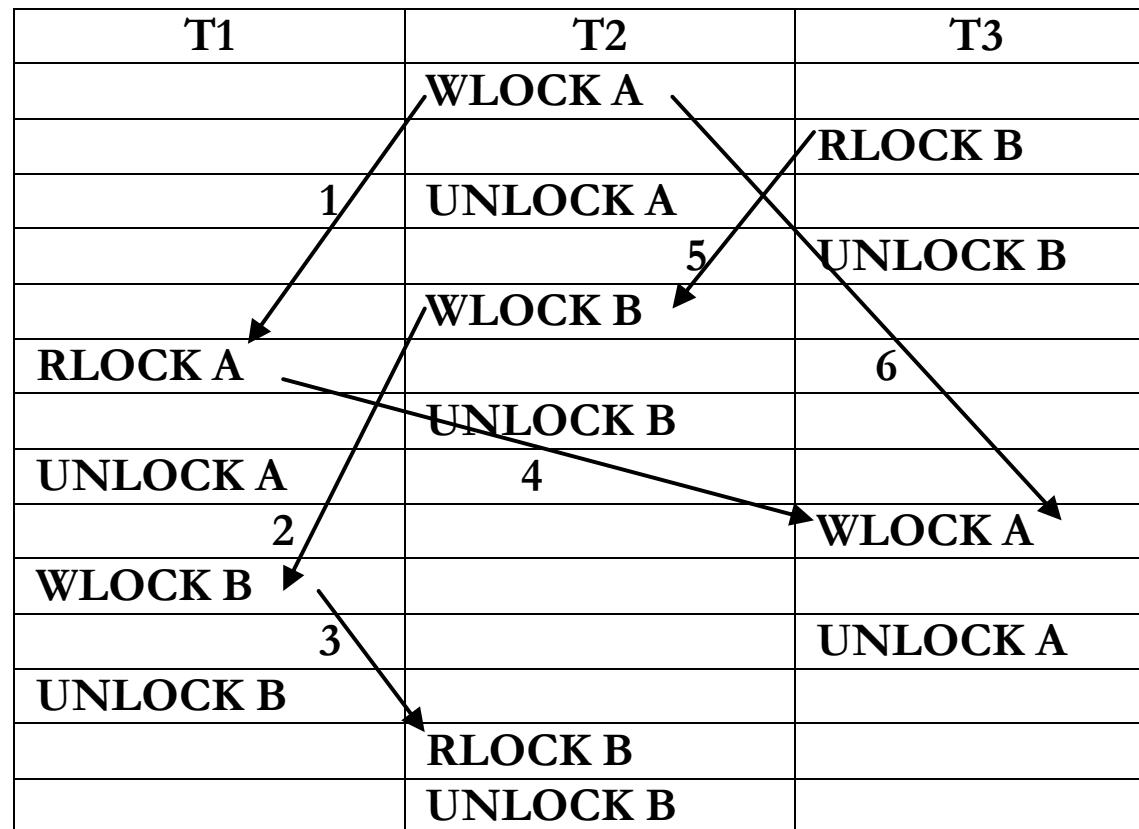
Nota: In cazul 2, daca T<sub>j</sub> nu exista, atunci T<sub>m</sub> este urmatoarea tranzactie care face RLOCK A dupa eliberarea lui A de catre T<sub>i</sub>.

- **Rezultat:** Daca graful obtinut are cicluri, atunci planificarea nu este serializabila, altfel este serializabila.



# Modelul RLOCK / WLOCK

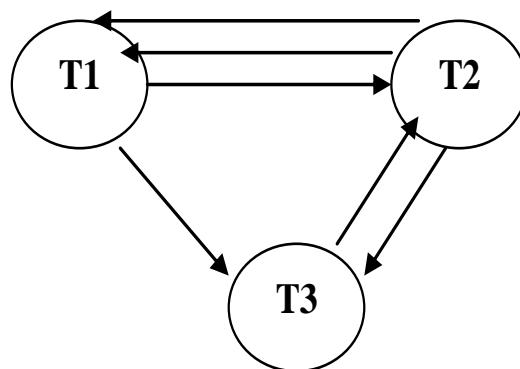
Exemplu:





# Modelul RLOCK / WLOCK

- Aplicand algoritmul se obtine urmatorul graf:



- Arcele s-au trasat pe baza regulilor de mai sus :
  - Prima regula a generat arcele de tip R-W: 4, 5.
  - Prima parte a celei de-a doua reguli a generat arcele de tip W-W: 2, 6.
  - A doua parte a celei de-a doua reguli a generat arcele de tip W-R: 1,3. (arcul 3 a fost generat luand in considerare nota de la a doua regula).
- Deoarece graful are cicluri rezulta ca planificarea nu e serializabila.