

Baze de Date - partea 2

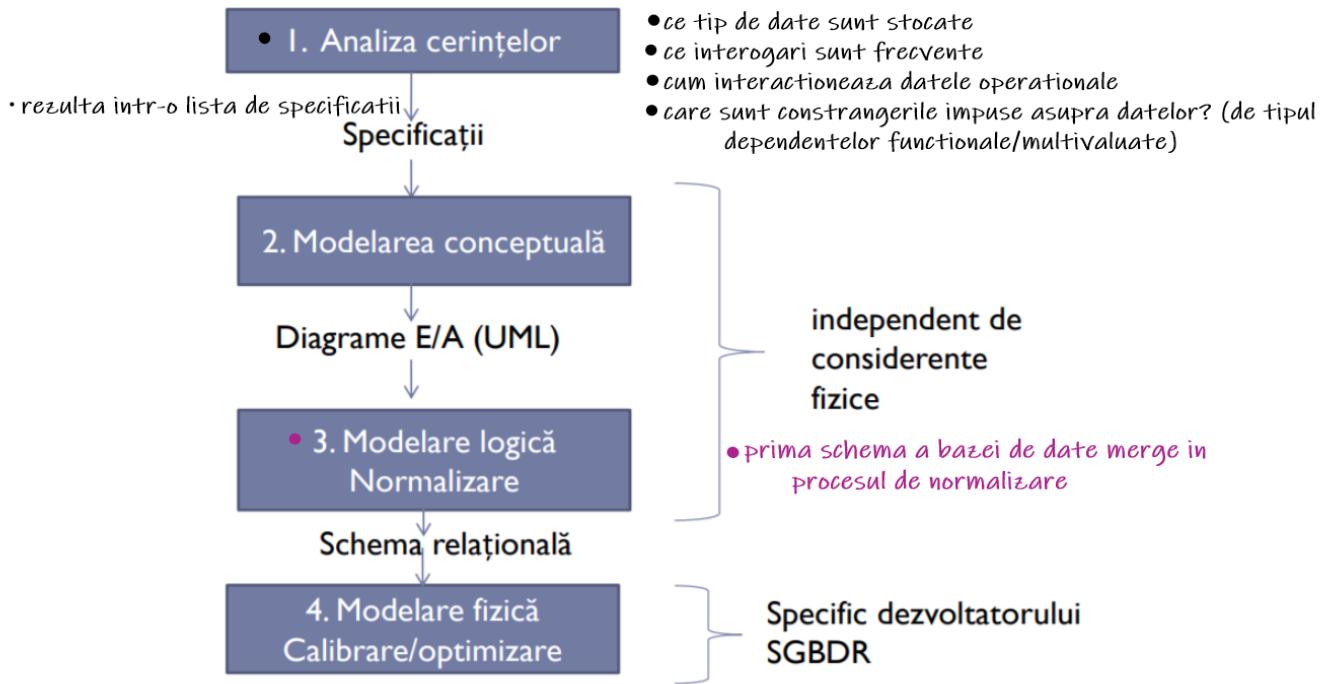
—

January 2021

Contents

| | |
|--|----|
| 1 CURS 9: redundanta, eficienta, consistenta, compositie, agregare, UML | 2 |
| 2 CURS 10 & 11: constrangeri, tranzactii, referinte, views | 31 |
| 3 CURS 12 & 13: tipuri de indecsi, B^+ arbori (formule, nr nivele etc) | 47 |
| 4 CURS 14: algebra relationala, estimarea costurilor | 84 |

1 CURS 9: redundanta, eficienta, consistenta, compozitie, agregare, UML



MODELARE CONCEPTUALA

Rezultatul modelarii este o diagrama (graf):

- entitati
- asocieri
- atribute
- alte notiuni de rol etc.

Traducerea diagramei \Rightarrow obtinerea unei prime scheme de baze de date.

MODELARE LOGICA - NORMALIZARE

Se iau din prima schema a bazei de date dependentele functionale si cele multivaluate si se verifica ca primul set de tabele sa satisfaca formele normale.

Daca un tabel nu satisface o forma normala, se descompune si se aduce in forma normala respectiva.

Dupa normalizarea tabelelor \Rightarrow un numar si mai mare de tabele \Rightarrow ingreuneaza joinul intre tabele.

In urma optimizarii se poate ajunge sa unim impreuna 2 tabele descompuse in urma normalizarii si sa avem un declansator / trigger / procedura care sa se asigure ca (,) constrangerea (“normalizarea”) este respectata in continuare.

MODELARE FIZICA - CALIBRARE/OPTIMIZARE

S-a obtinut baza de date finala si se incepe procesul de modelare fizica (interrogari pe bd, fraze “create”). Se pot pune constrangeri (pe cheile candidat).

NOTATIE: E/A = Entitate/Asociere

PROIECTAREA SCHEMEI BD-ului

REDUNDANTA = stocare pe spatiu suplimentar, generare de diverse “anomalii” (ex: typos)

EFICIENTA = nivelul de optimizare (ex: cat timp ii ia sgbd-ului sa faca o anumita interogare)

CONSISTENTA = sbgd-ul trebuie sa satisfaca mereu anumite constrangeri impuse (este evaluata relativ la anumite constrangeri asupra continutului bd-ului)

Exista 2 abordari:

- descompunere / normalizare
- modelarea E/A (chen)

De obicei sunt aplicate impreuna: se incepe cu modelarea E/A si se continua cu normalizarea.

<http://www.cs.utsa.edu/~cs3443/uml/uml.html>

In plus la UML (fata de chen) sunt:

COMPOZITIE = romb negru; NU are pk; trebuie sa depindă de clasa cu care sunt “asociate” ;

In addition to an aggregation relationship, the lifetimes of the objects might be identical, or nearly so. For example, in an idealized world of electronic books with DRM (Digital Rights Management), a person can own an ebook, but cannot sell it. After the person dies, no one else can access the ebook. [This is idealized, but might be considered less than ideal.]

AGREGARE = romb alb/gol; are pk; nu depinde neaparat de clasa cu care sunt “asociate” ;

One object A has or owns another object B, and/or B is part of A. For example, suppose there are different Book objects for different physical copies. Then the Person object has/owns the Book object, and, while the book is not really part of the person, the book is part of the person’s property. In this case, each book will (usually) have one owner. Of course, a person might own any number of books.

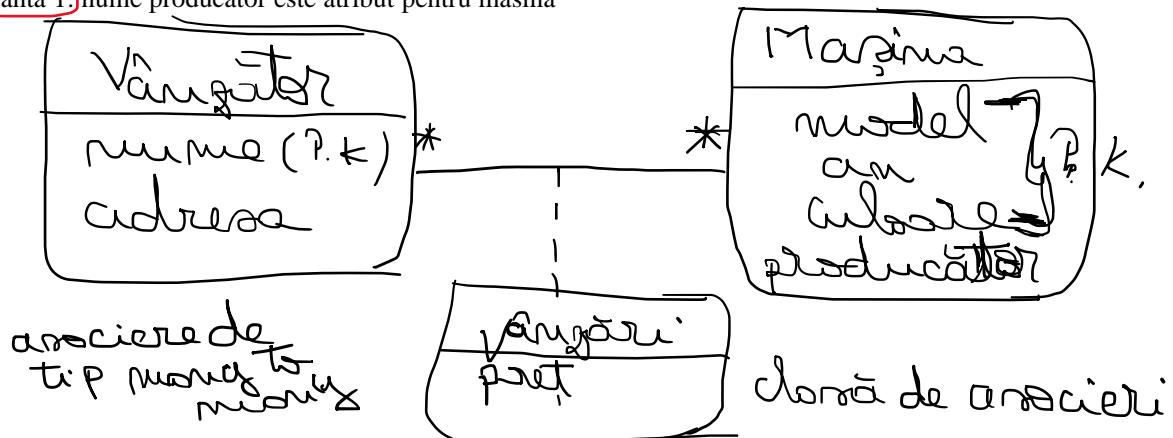
Laborator 13:

1) Pas I: Intai facem proiectarea entitate-asociere (facem diagrama UML)

Din enunt putem sa stabilim clar ca avem de a face cu entitatile Vanzator, Masina si intre acestea avem asocierea care sa aiba un atribut propriu pret, deoarece pretul unei masini (cu anumite caracteristici) e in functie de vanzator (dealer).

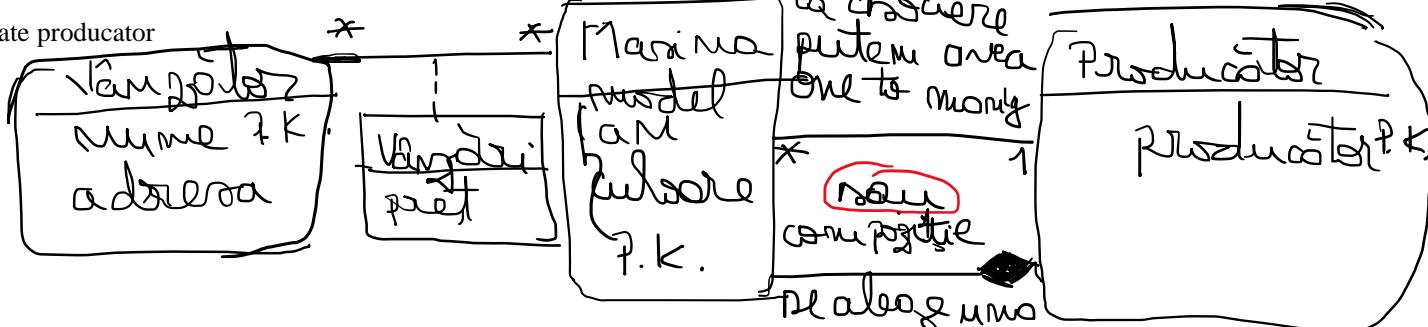
Datorita faptului ca nu avem prea multe detalii despre producator decat numele putem avea doua situatii, una in care pastram numele producatorului ca si atribut al unei clase de entitati, Masina de exemplu, si a doua varianta cand cream clasa de entitati producator.

Varianta 1: nume producator este atribut pentru masina



- Numele vanzatorului identifica in mod unic fiecare vanzator deci e PK.
- Pentru masina, avem de exemplu specificarea ca modelul determina in mod unic producatorul deci pentru alegerea cheii nu o sa luam toate cele patru atribuire: model, an, culoare si producator pentru ca ar fi supercheie, ci stabilim cheie primara doar model, an si culoare.

Varianta 2: entitate producator



Tabelele: Varianta 1Vanzator (nume, adresa)Masina (model, an_culoare, produsator)Vanzari (nume, model, an_culoare, pret)

-Vanzari, fiind asociere nu are marcata o PK atunci
putem avea nume+model+an+cupoare = PK -

Varianta 2 ? . K .Vanzator (nume, adresa)Masina (model, an_culoare, produsator)

Oricare din cele doua asocieri alegem, one-to-many sau compositie se va importa la masina atributul produsator, ca si cheie straina.

Vanzari (nume, model, an_culoare, pret)Produsator (produsator, PK)

- Avantajul acestei scheme este cand avem producatori care nu au masini inca si atunci retinem in baza de date si acesti producatori.
- Vom merge in final cu prima varianta, la enuntul de aici e de ajuns doar cele trei tabele

Pasul 2 – normalizarea (stabilim dependentele)

$\{ \text{name} \rightarrow \text{adresa}$ (DF1)
 $\{ \text{name}, \text{model_an} \rightarrow \text{pret}$ (DF2)
 $\text{model} \rightarrow \text{produsator}$ (DF3)

$\text{model_an} \rightarrow \text{name}$
 (4) \rightarrow adresa
 ce face cu pretul?
 Il trageam la name
 deoarece pretul depinde
 de numarul

$\{ \text{model_an} \rightarrow \text{name}, \text{pret}$
 $\text{model_an} \rightarrow \text{adresa}$ (DF4)

Vânzări \rightarrow (name, model_an, pret) e în 4NF.

șă avem de la (model_an, cultură)

(adresa nu intervine în pret)
 Vândători B.C.

Morine nu este B.C.
 Vândători

Morine DF3 este în 4NF

(model, an, cultură)
 în 4NF.

Vândători

DF2

(name, model_an, cultură)
 în 4NF

DF4

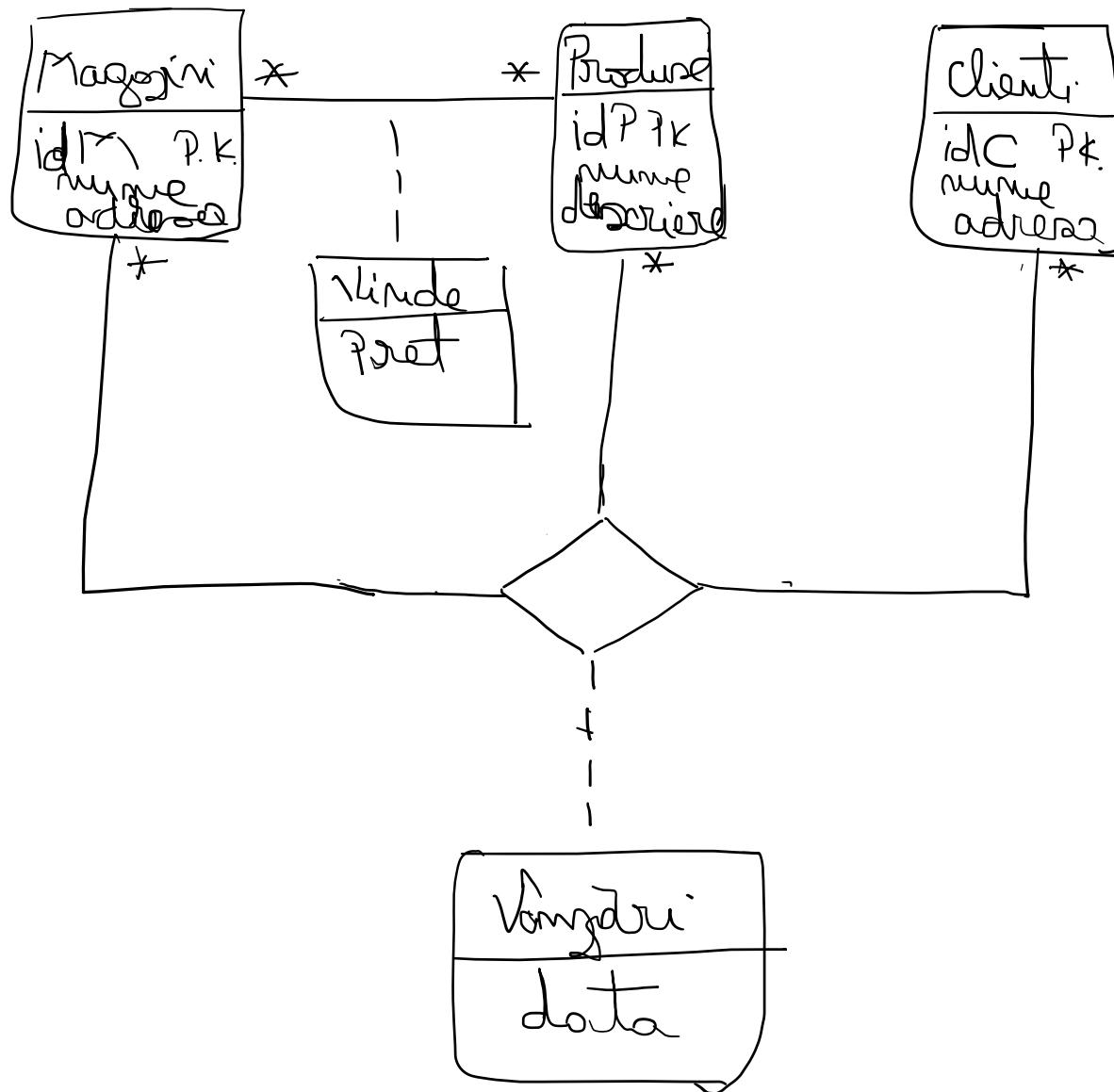
(name, model_an)
 e inclusă în Vândători

În final avem:

Vânzător (nume, adresă)
Mosina (model, producător) \rightarrow nu am demisit ^{FK} Producător
 \rightarrow că nu retine inf. unică
despre producător
Caracteristici-Mosina (model, an, culoare)

Vânzători (nume, model, an, pret)
 \rightarrow FK

3)





SELECT curs
FROM BAZE_DE_DATE

WHERE INITCAP(capitol) = 'Proiectarea Bazelor De Date Relaționale'
AND topic = 'Modelul Entitate/Asociere'

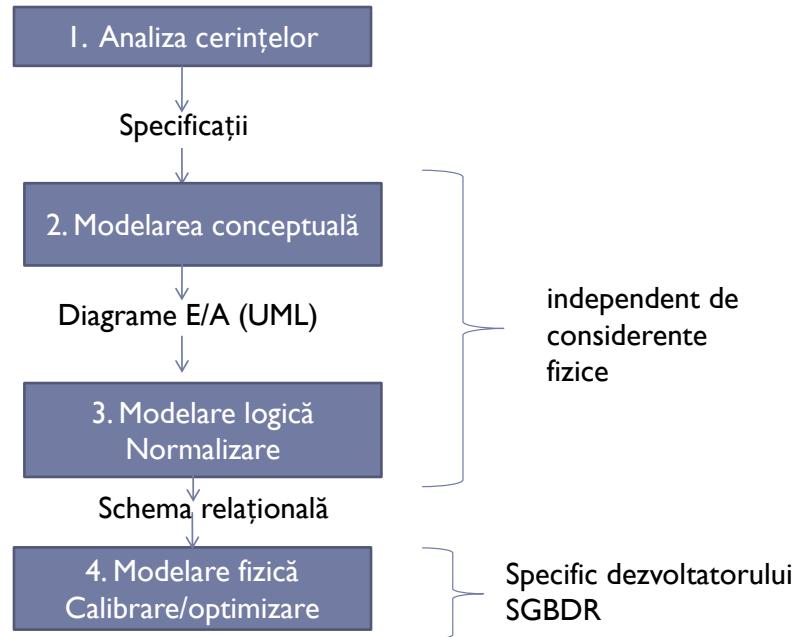
Mihaela Elena Breabă
© FII 2020-2021

Baze de date – cuprinsul cursului

- ▶ Concepte din Baze de date (C1)
 - ▶ Algebra relațională (C2-C3)
 - ▶ Dependențe funcționale și multivaluate (C4-C5)
 - ▶ Proiectarea logică: Normalizare (C6-C7)
 - ▶ Proiectarea conceptuală: Modelarea Entitatea/Asociere (C9)
 - ▶ Proiectarea fizică (C10-C11)
 - ▶ Indexarea (C12-C13)
 - ▶ Procesarea interogărilor (C14)
- Proiectarea
și optimizarea
Bazelor de date
Relaționale

Proiectarea Bazelor de date Relaționale

Metodologie



▶ 3

Planul prelegerii

- ▶ Problematica proiectării schemei
- ▶ Proiectarea E/A în notația lui Chen
 - ▶ Concepțe E/A
 - ▶ Modelarea constrângerilor
 - ▶ Capcane de conexiune
- ▶ Proiectarea E/A în UML
- ▶ Din E/A în schemă relațională

▶ 4

Proiectarea schemei bazei de date

- ▶ Pentru o bază de date putem propune mai multe scheme
 - ▶ Unele sunt (mult) mai bune decât altele
 - ▶ Redundanță?
 - ▶ Eficiență?
 - ▶ Consistență?
- ▶ Cum generăm scheme bune?
- ▶ Două abordări:
 - ▶ Descompunere - normalizare (Codd, '70-'74)
 - ▶ Modelarea E/A (Chen, '76)
- ▶ De obicei sunt aplicate împreună, în doi pași: se începe cu proiectarea E/A și se continuă cu normalizarea

▶ 5

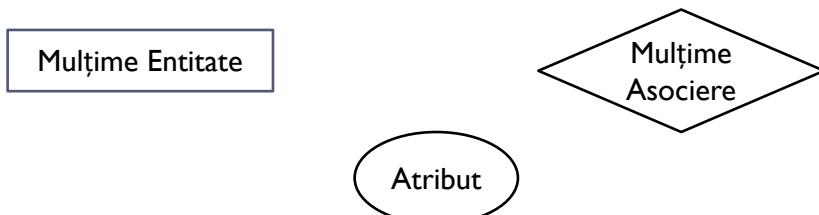
Concepte E/A clasice (Chen 1976)

- ▶ Entitate
 - ▶ Obiect ce trebuie reprezentat în baza de date
 - ▶ Mulțime-entitate - corespunde unui grup de obiecte de același tip, deci unei mulțimi omogene de entități
 - ▶ O instanță – entitate
 - ▶ O instanță **unic identificabilă**
- ▶ Asociere (Relationship)
 - ▶ Conexiune/asociere între două sau mai multe entități de tip diferit sau de același tip
 - ▶ Mulțime-asociere – corespunde unei mulțimi omogene de asocieri, modeleză interacțiuni între mulțimi-entitate
 - ▶ Gradul asocierii = numărul de mulțimi-entitate participante
 - ▶ unare/recursive, binare, ternare...
- ▶ Atribut
 - ▶ Proprietate a unei entități
 - ▶ Pentru asociere
 - ▶ Attribute ale entităților referențiate
 - ▶ Noi attribute (attribute proprii)

▶ 6

Diagrame E/A

- ▶ Reprezentare grafică a conceptelor E/A
 - ▶ Există mai multe standarde grafice, aici varianta Chen



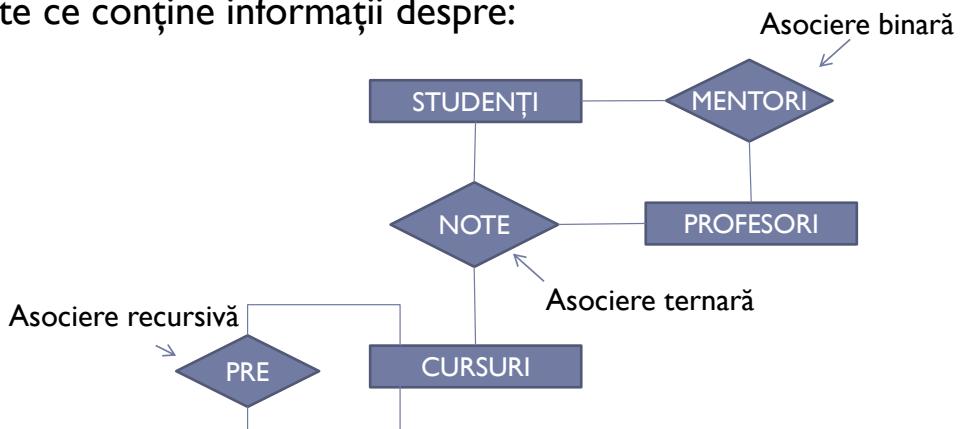
- ▶ Un graf
 - ▶ Mulțimile-entitate, mulțimile-asociere și attributele sunt noduri
 - ▶ Există muchii doar între
 - ▶ noduri-entitate și noduri-asociere
 - ▶ noduri-entitate și noduri-atribute
 - ▶ noduri-asociere și noduri-atribute

▶ 7

Exemplu

- ▶ O bază de date ce conține informații despre:

- ▶ Studenți
- ▶ Profesori
- ▶ Cursuri
- ▶ Note
- ▶ Mentorii



- ▶ Cerințe:

- ▶ Putem determina notele obținute la cursuri pentru orice student, precum și profesorii care au pus notele
- ▶ Putem determina mentorul (profesorul îndrumător al) oricărui student
- ▶ Putem identifica condițiile necesare pentru a studia un curs (fiecare curs ar putea necesita cunoasterea informațiilor din alte cursuri)

▶ 8

Alte concepte E/A

▶ Rol

- ▶ Explică semnificația entităților în asocieri



▶ Cheie primară

- ▶ Un atribut sau o submulțime minimală de atribute ce identifică unic o instanță-entitate sau o instanță-asociere
- ▶ **Obligatorie pentru entități**, pentru a indica care instanțe participă în asocieri

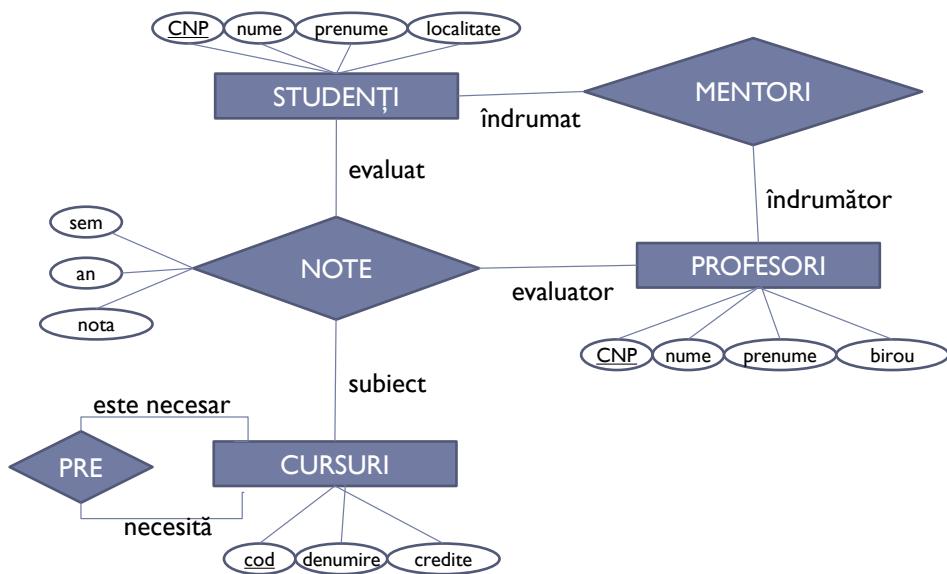
Cheie primară

▶ Cheie străină **pentru o asociere**

- ▶ Un atribut sau o mulțime de atribute care constituie cheie primară pentru entitățile implicate

▶ 9

Exemplu

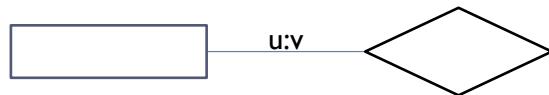


Care sunt cheile străine pentru cele trei mulțimi de asocieri?

▶ 10

Constrângeri de conectivitate/participare

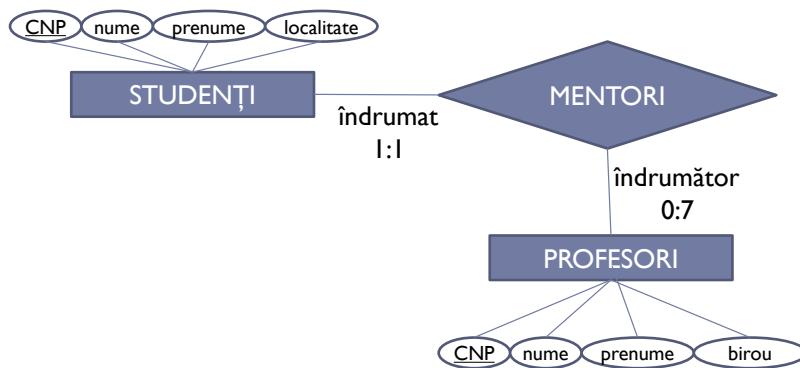
- ▶ Modelul E/A permite declararea de constrângeri asupra numărului de instanțe-asociere în care o instanță-entitate participă
- ▶ Fie R o mulțime-asociere între n mulțimi-entitate E_i , $i=1..n$. Baza de date satisfac constrângerea (E_i, u, v, R) dacă fiecare instanță-entitate din E_i participă în cel puțin u și cel mult v instanțe-asociere din R .



▶ //

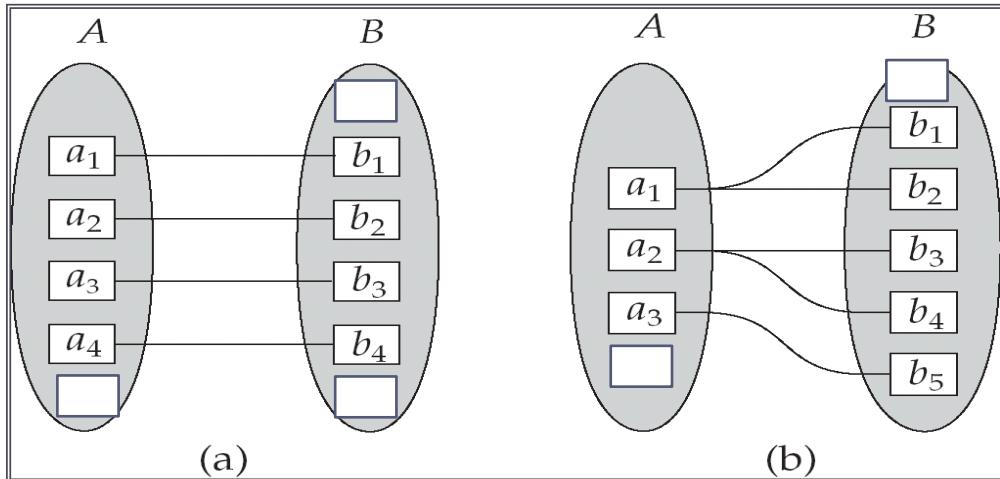
Exemplu

- ▶ (*Studenti*, 1, 1 *Mentori*)
- ▶ (*Profesori*, 0, 7, *Mentori*)
- ▶ Fiecare student are un singur profesor drept mentor iar un profesor poate fi mentor pentru cel mult 7 studenți



▶ 12

Constrângeri de conectivitate pentru asocieri binare (1)



a) Asociere unu la unu
(A,0,1,R) (B,0,1,R)

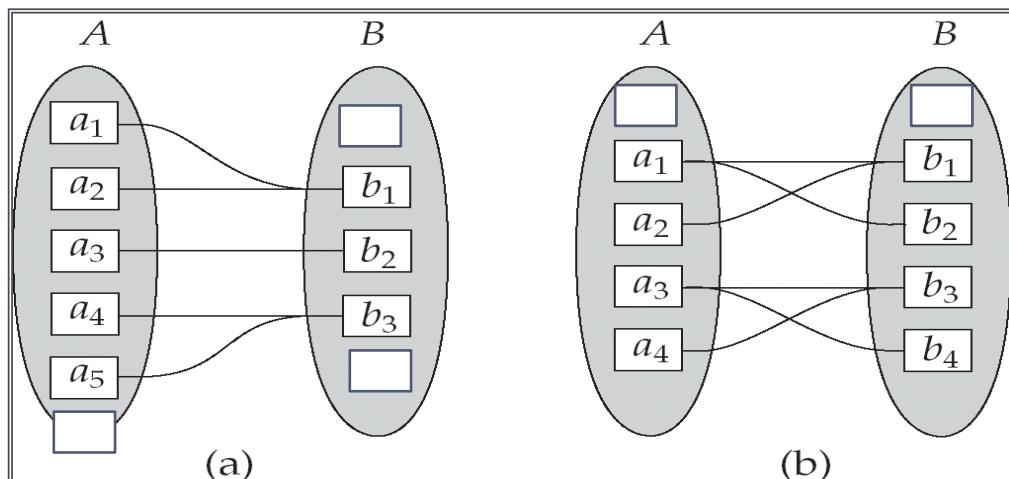


b) Asociere unu la mulți
(A,0,n,R) (B,0,1,R), n>1



▶ 13

Constrângeri de conectivitate pentru asocieri binare (2)



a) Asociere mulți la unu
(A,0,1,R) (B,0,n,R)



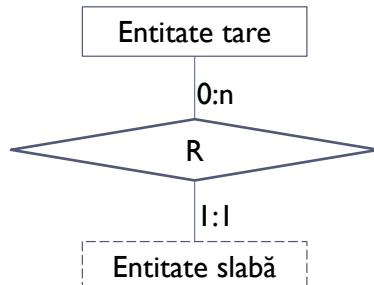
b) Asociere mulți la mulți
(A,0,m,R) (B,0,n,R), m,n>1



▶ 14

Entitate slabă

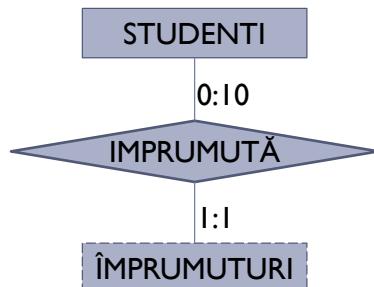
- ▶ O mulțime-entitate este slabă dacă existența instanțelor sale depinde de existența instanțelor altrei mulțimi-entitate (dependență existențială)



- ▶ Nu are cheie
- ▶ Satisfacă constrângerea de conectivitate (Entitate_slabă, I, I, R), deci participă într-o asociere de tip unu la mulți relativ la entitatea tare

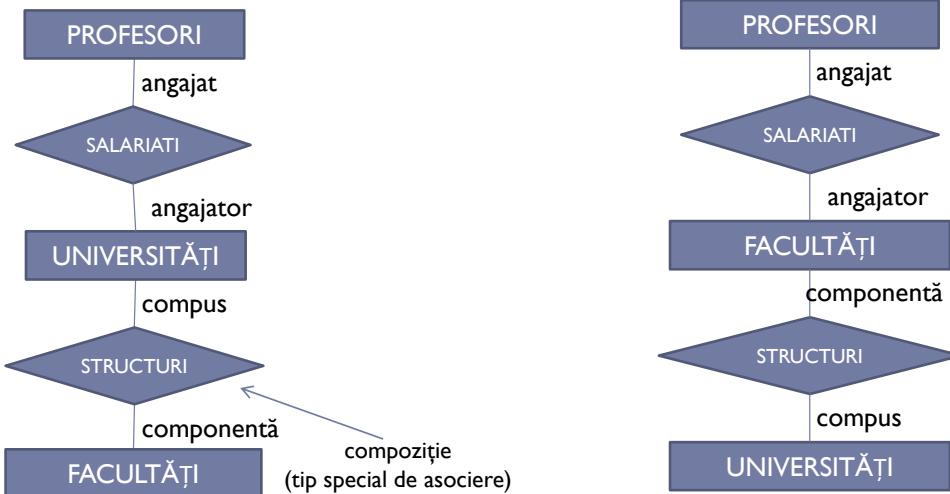
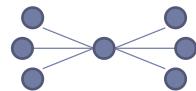
▶ 15

Exemplu



▶ 16

Capcane de conectare (Fan traps)

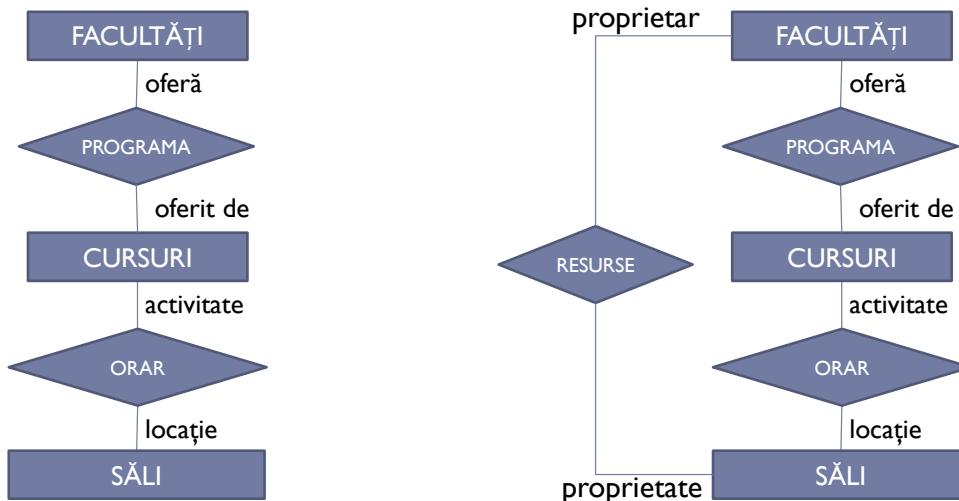
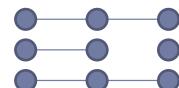


Problema:
La ce departament aparține profesorul X?

Soluția:
Model restructurat

▶ 17

Capcane de conectare (Chasm traps)



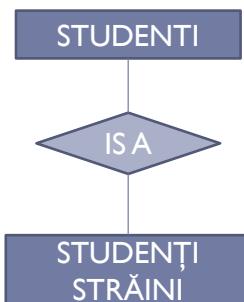
Problema:
Care sunt toate sălile ce aparțin unei facultăți?

Soluția:
Noi asocieri

▶ 18

Modelul E/A extins Specializare

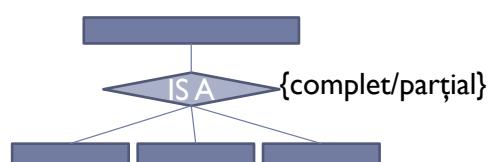
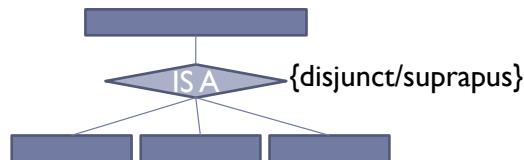
- ▶ Subgrupuri distinctive de instanțe-entități
 - ▶ Au în plus anumite attribute
 - ▶ Participă în asocieri la care nu participă toate instanțele-entități
 - ▶ Corespund unei mulțimi de entități specializate care se află într-o asociere de tip IS-A cu mulțimea de entități de bază



▶ 19

Constrângeri specifice specializării

- ▶ Instanțele specializării moștenesc toate attributele și asocierile mulțimii de entități de bază, inclusiv cheia
- ▶ Clasificare
 - ▶ O instanță a unei mulțimi-entitate poate apartine la una sau la mai multe specializări
 - ▶ Specializări disjuncte (exclusive)
 - ▶ Specializări cu suprapunere
 - ▶ O instanță a unei mulțimi-entitate trebuie sau nu să aparțină la cel puțin o specializare
 - ▶ Complet
 - ▶ Incomplet (parțial)



▶ 20

Modelare UML

► Unified Modeling Language

- ▶ Utilizat în ingineria software
- ▶ Bazat pe concepte orientate obiect
- ▶ Instrument de comunicare cu clientul în termenii utilizați în companie
- ▶ Un limbaj foarte mare, utilizăm un set restrâns de elemente (diagrama de clase) pentru a modela o bază de date.

► 21

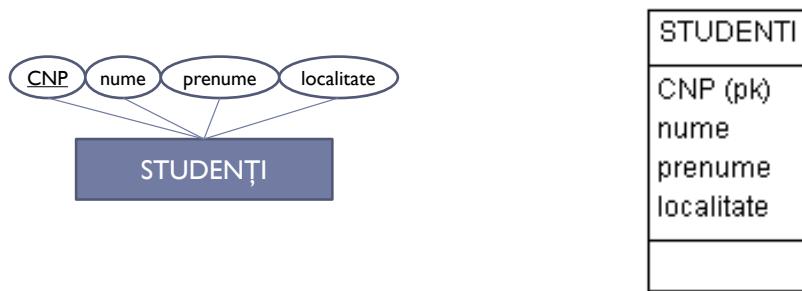
Mapare E/A – UML

| E/R | UML |
|--|------------------------|
| Mulțime-entitate cu atribute | Clasă |
| Mulțime-asociere fără atribute proprii | Asociere |
| Mulțime-asociere cu atribute proprii | Clasă de asociere |
| Specializare | Subclasă |
| | Compoziție și agregare |

► 22

Clase

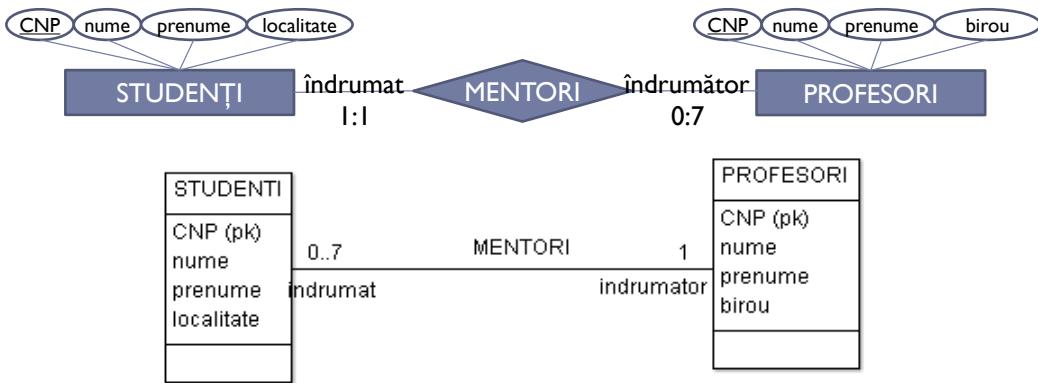
- ▶ Componente: nume, attribute, metode
- ▶ BD: nume, attribute (cheia primară)



▶ 23

Asocieri

- ▶ Exprimă asocierea dintre obiectele aparținând la două clase
- ▶ BD: asocierea dintre instanțele a două mulțimi-entitate



- ▶ Obs: constrângerile de conectivitate se specifică invers decât în diagramele E/A

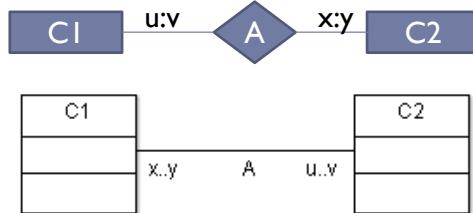
▶ 24

Asocieri

Constrângeri de conectivitate/multiplicitate

► Restricții

- (C1,u,v,A)
- (C2,x,y,A)



- Fiecare obiect din (instanță-entitate) C1 este asociat cu cel puțin u și cel mult v obiecte din (instanță-entitate) C2
- Fiecare obiect din (instanță-entitate) C2 este asociat cu cel puțin x și cel mult y obiecte din (instanță-entitate) C2

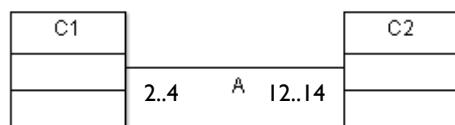
| x..y | u..v | Tip asociere |
|----------|----------|-------------------------|
| 0..1 | 0..1 | unu la unu incompletă |
| 1..1 (1) | 1..1 (1) | unu la unu completă |
| 0..1 | 0..* (*) | unu la multi incompletă |
| ... | ... | ... |

► 25

???

- Modelați asocierea dintre STUDENȚI și UNIVERSITĂȚI. Un student poate studia la cel mult 2 universități și e necesar să studieze la cel puțin una. O universitate primește cel mult 10.000 studenți.

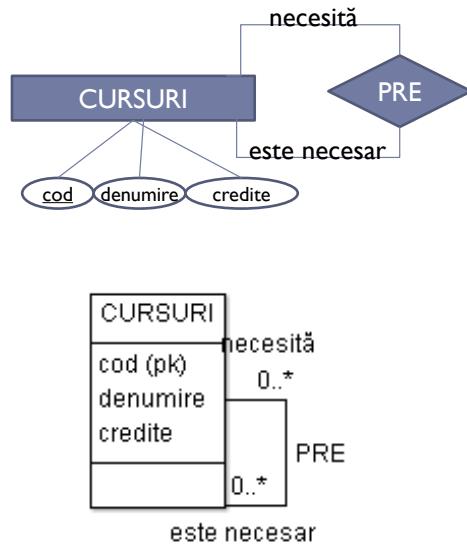
- Fie asocierea



Care e numărul minim de instanțe pentru mulțimea-entitate C1 și pentru C2?

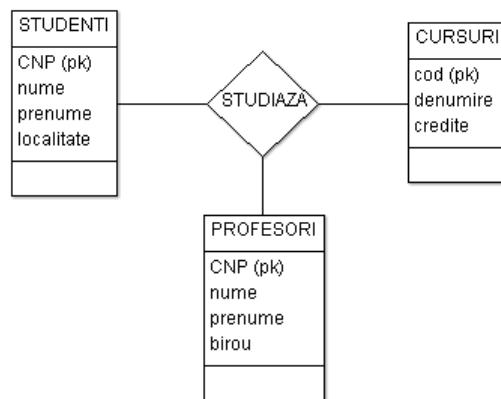
► 26

Asocieri recursive



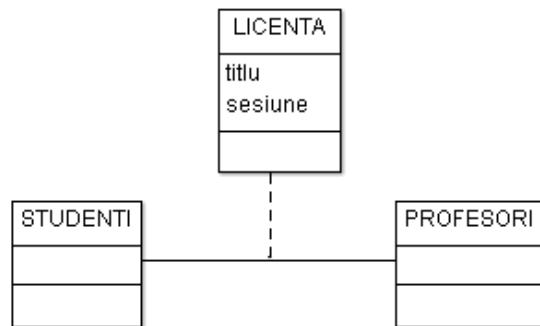
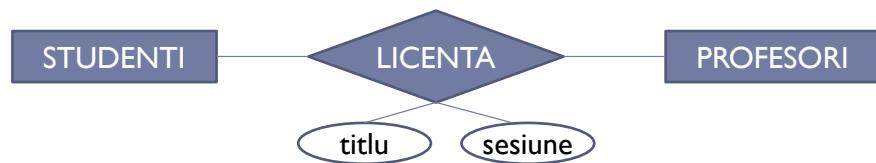
▶ 27

Asocieri n-are



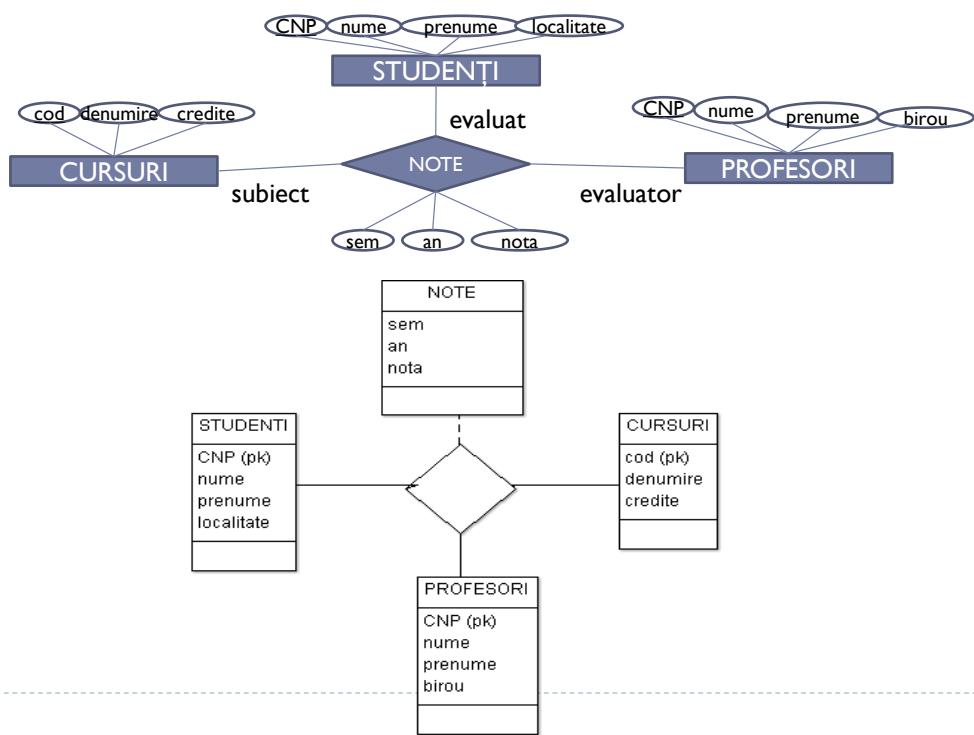
▶ 28

Clase de asociere



▶ 29

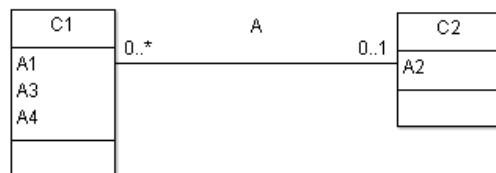
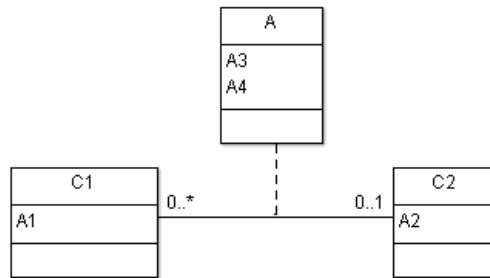
Clase de asociere



▶ 30

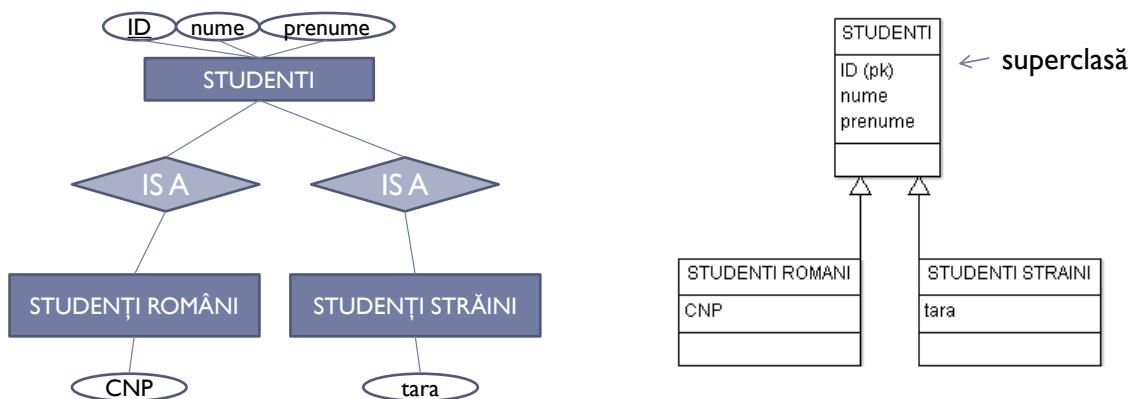
Eliminarea claselor de asociere

- Atunci când avem multiplicitate 0..1 sau 1..1



▶ 31

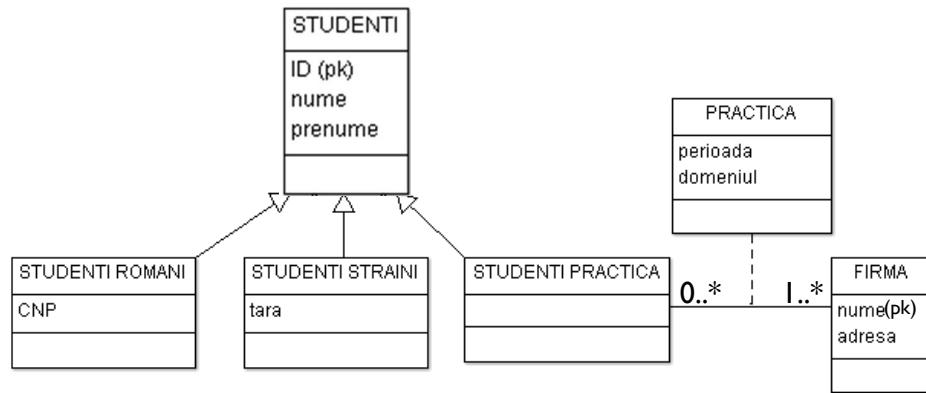
Subclasă (1)



(Specializare completă, disjunctă)

▶ 32

Subclasă (2)

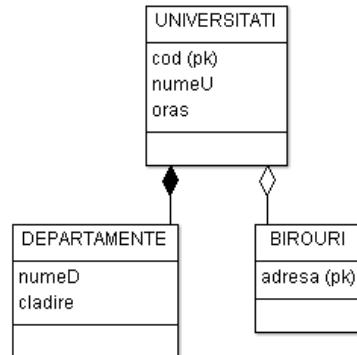


(Specializare completă, cu suprapunere)

▶ 33

Compoziție și agregare

- ▶ Obiecte dintr-o clasă aparțin obiectelor din altă clasă
- ▶ Tipuri speciale de asociere



- Compoziția: **toate** obiectele unei clase *părți* aparțin obiectelor dintr-o clasă *compusă*; clasei *părți* îi corespunde de obicei o entitate slabă (multiplicitate **1..1**; fără cheie primară);
- Agregarea: **unele** obiecte dintr-o clasă aparțin obiectelor din altă clasă (multiplicitate **0..1**)

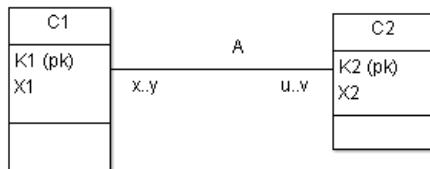
▶ 34

Mapare E/A, UML -> schema BD relațională

| E/A | UML | Schema relațională |
|--|------------------------|---|
| Mulțime-entitate cu atribute | Clasă | Relație cu cheie primară |
| Mulțime-asociere fără atribute proprii | Asociere | Relație cu chei străine |
| Mulțime-asociere cu atribute proprii | Clasă de asociere | Relație cu chei străine și alte atribute |
| Specializare | Subclasă | Relație cu cheie primară (cea a superclasei) și atribute particulare/specializate |
| | Compoziție și agregare | Relație cu cheie străină și atribute particulare |

▶ 35

Mulțimi-entitate/clase și asocieri



{C1(K1, X1), C2(K2, X2), A(K1, K2)}

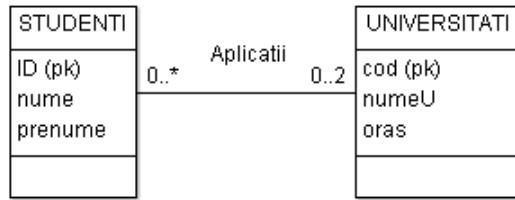
- ▶ Cheia primară pentru asociere depinde de multiplicitate

| x..y | u..v | Cheia primară pt A | Observații |
|------|------|--------------------|---|
| 0..1 | * | K2 | Nu e necesară relația A {C1(K1, X1), C2(K2, X2, K1)} |
| 1..1 | | | |
| * | 0..1 | K1 | Nu e necesară relația A {C1(K1, X1, K2), C2(K2, X2)} |
| 1..1 | | | |
| * | * | (K1, K2) | |

▶ 36

???

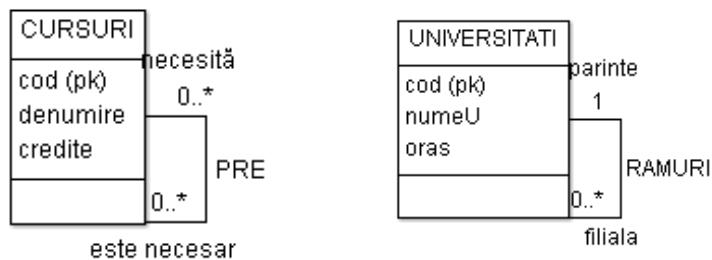
► Fie diagrama



► Mai este posibilă renuntarea la relația corespunzătoare asocierii?

► 37

Asocieri recursive

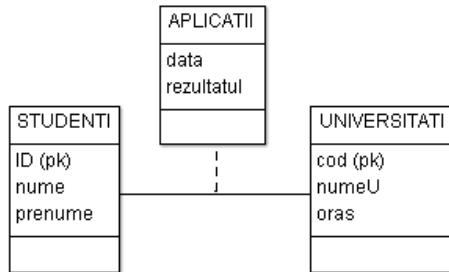


{CURSURI (cod, denumire, credite)
PRE (cod1, cod2)}

{UNIVERSITATI (cod, numeU,oras)
RAMURI (codFiliala, codParinte)}

► 38

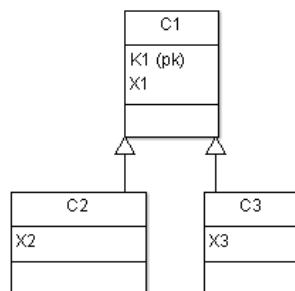
Clase de asociere



{STUDENTI (ID, nume, prenume)
UNIVERSITATI (cod, numeU, oras)
APPLICATII (ID, cod, data, rezultatul)}

▶ 39

Specializare/Subclase



► Posibilități

- ▶ Relații subclasă ce conțin cheia superclasei și atributele specializeze
 $C1(K1, X1)$, $C2(K1, X2)$, $C3(K1, X3)$
- ▶ Relații subclasă ce conțin atributele superclasei (inclusiv atributul cheie) și atributele specializeze; superclasa conține doar tuple nespecializate
 $C1(K1, X1)$, $C2(K2, X1, X2)$, $C3(K2, X1, X3)$
- ▶ O singură relație ce conține atributele din superclasă și subclasă
 $C(K1, X1, X2, X3)$

▶ 40

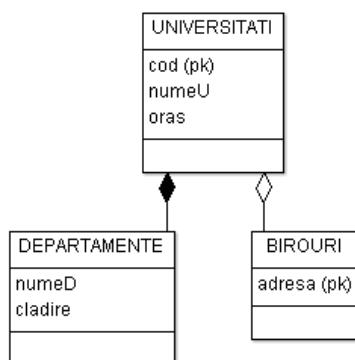
???

- Fie superclasa S cu un număr de subclase. Considerăm că relația de specializare este incompletă și cu suprapunere. Dacă n1, n2 și n3 reprezintă numărul total de tuple necesare fiecărei scheme de decodificare din cele 3 date anterior (în ordinea dată), care este relația dintre cele 3 valori?

- $n_1 < n_2 < n_3$
- $n_1 \leq n_2 \leq n_3$
- $n_3 < n_2 < n_1$
- $n_3 \leq n_2 \leq n_1$

▶ 41

Compoziție și agregare



{ UNIVERSITATI(cod, numeU, oras)
DEPARTAMENTE(codU, numeD, cladire)
BIROURI (codU, adresa) }
NU acceptă NULL
acceptă NULL

▶ 42

Modelare EA/UML

Sumar

▶ PROS

- ▶ Tehnică populară de modelare conceptuală
- ▶ Construcții expresive, descriu punctul de vedere personal asupra aplicației
- ▶ Permite exprimarea unor tipuri de constrângeri (chei primare, străine, multiplicitate, exclusivitate...)

▶ CONS

- ▶ Tehnică subiectivă (entitate sau atribut, entitate sau asociere, subclasare sau nu, compoziție sau nu)
- ▶ Nu permite modelarea tuturor dependentelor
- ▶ Necesară utilizarea ulterioară a normalizării

▶ 43

Bibliografie

- ▶ Capitolele 11 și 12 în Thomas Connolly, Carolyn Begg: *Database Systems: A Practical Approach to Design, Implementation and Management, (5th edition)* Addison Wesley, 2009
- ▶ Hector Garcia-Molina, Jeff Ullman, Jennifer Widom: *Database Systems: The Complete Book (2nd edition)*, Prentice Hall; (June 15, 2008)
- ▶ Instrumente:
 - ▶ <https://creately.com> (diagramă EA, diagramă UML de clasă)
 - ▶ <http://diagramo.com/> (diagramă EA)
 - ▶ <http://argouml-downloads.tigris.org/nonav/argouml-0.32.2/ArgoUML-0.32.2.zip> (diagramă UML de clasă)

▶ 44

2 CURS 10 & 11: constrangeri, tranzactii, referinte, views

CONSTRANGERI

- primary key = unique + not null
- NU pot fi 2 pk intr-un tabel, doar pk format din 1 sau mai multe atribute declarate anterior

O *tranzacție* reprezintă un grup de comenzi de modificare de date (*DML*) care trebuie executate împreună, pentru a garanta consistența datelor. Eșuarea oricărei comenzi din cadrul unei tranzacții determină revenirea la starea inițială, dinaintea tranzacției.

O tranzacție începe de la prima comandă DML executată și se încheie la întâlnirea uneia dintre comenziile COMMIT și ROLLBACK, la întâlnirea unei comenzi DDL, la închiderea sesiunii sau la o eroare a sistemului.

TRANZACTII:MAI MULTE DETALII PUTIN USELESS UNDER PRESSURE :”)

Comanda ROLLBACK încheie tranzacția readucând baza de date la starea de dinaintea începerii tranzacției. O cădere a sistemului rezultă tot într-o comandă de tip ROLLBACK, adică revenirea la starea de dinaintea începerii tranzacției; este modalitatea prin care este protejată integritatea datelor. Restul situațiilor enumerate mai sus care determină încheierea unei tranzacții, fac ca modificările efectuate asupra datelor să fie permanente, fără posibilitatea revenirii la o stare anterioară.

Imediat ce o tranzacție s-a încheiat, prima comandă DML lansată marchează începutul uneia noi.

În cadrul acestui laborator ați executat comanda ROLLBACK înainte de a experimenta cu comanda CREATE TABLE, adică înainte de prima comandă de tip DDL lansată. Rezultatul a fost revenirea bazei de date la starea existentă la începutul sesiunii de lucru. Dacă nu ar fi fost lansată comanda ROLLBACK, modificările efectuate asupra datelor ar fi devenit permanente în momentul executării comenzi DDL (așa cum s-a specificat mai sus, orice comandă DDL încheie tranzacția marcând modificările ca fiind permanente).

Pe parcursul unei tranzacții pot fi adăugați niște marcatori/indicatori cu ajutorul cărora să putem reveni la stări intermediare. Adăugarea unui indicator se realizează cu ajutorul comenzii SAVEPOINT *nume;indicator* iar revenirea la starea bazei de date din acel moment se realizează cu ajutorul comenzi ROLLBACK TO SAVEPOINT *nume;indicator*.

Orice comandă DDL este considerată a forma o tranzacție.

!!NU MAI BAT CAMPII CU TRANZITII:

Integritate referentiala de la A la B: adica elementele din A nu prea pot exista fara cele din B (A tabela mica, B tabela mare)

- coloana referentiata din A = cheie straine
- valoarea coloanei referentiate din A trebuie sa apară și în B
- coloana din B la care se face referință trebuie să fie primary key sau (dacă) unique
- se pot referenția mai multe coloane = chei straine multi-atribut

Comenzi care pot strica tabelele de genul:

- inserări în A (aia mică)
- stergeri în B (aia mare)
- update pe A sau B (oricare din tabele)

TABELE VIRTUALE / VIEWs

Pentru a fi corect un view acesta trebuie să fie "actualizabil"/"updatable":

- view-ul creat nu are *DISTINCT* în *SELECT*
- *SELECT*-ul din *CREATE VIEW* NU are un *join* de tabele
- restul atributelor din tabela implicată în view care nu sunt selectate pentru tabela virtuală trebuie să NU aibă constrângerea 'NOT NULL'
- NU există clauze de agregare (group by, having, count, avg, min, max, stddev, variance etc)

TABEL VIRTUAL MATERIALIZAT = se crează un nou tabel

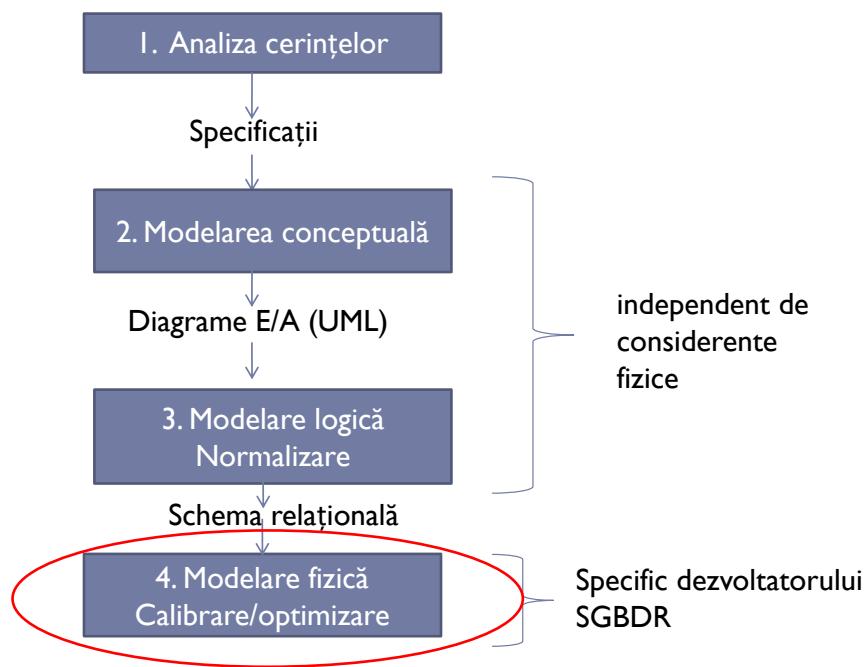


BAZE DE DATE

Implementarea constrângerilor
Declanșatoare (Triggers)
Tabele virtuale (Views)

Mihaela Elena Breabă
© FII 2020-2021

Proiectarea Bazelor de date Relaționale Metodologie



Conținut

▶ Constrângeri de integritate

▶ Declanșatoare

▶ Tabele virtuale

▶ 3

Constrângeri de integritate (statice) (1)

▶ Restricționează stările posibile ale bazei de date

- ▶ Pentru a elimina posibilitatea introducerii eronate de valori la operația de inserare
- ▶ Pentru a satisface corectitudinea la actualizare/ștergere
- ▶ Forțează consistență
- ▶ Transmit sistemului informații utile stocării, procesării interogărilor

▶ Tipuri

- ▶ Non-null
- ▶ Chei
- ▶ Integritate referențială
- ▶ Bazate pe atribut și bazate pe tuplu
- ▶ Aserții generale

▶ 4

Constrângeri de integritate (2)

► Declarare

- ▶ Odată cu schema (comanda CREATE)
- ▶ După crearea schemei (comanda ALTER)

► Realizare

- ▶ Verificare la fiecare comandă de modificare a datelor
- ▶ Verificare la final de tranzacție

► 5

Constrângeri de integritate peste 1 variabilă Implementare *inline*

CREATE TABLE tabel (

a1 tip **not null**, -- acceptă doar valori nenule

a2 tip **unique**, --cheie candidat formată dintr-un singur atribut

a3 tip **primary key**, -- cheie primară formată dintr-un singur atribut, implicit {not null, unique}

a4 tip **references** tabel2 (*b1*), --cheie străină formată dintr-un singur atribut

a5 tip **check** (*condiție*) -- condiția e o expresie booleană formulată peste atributul *a5*:
(*a5<11 and a5>4*), (*a5 between 5 and 10*), (*a5 in (5,6,7,8,9,10)*)...

)

► 6

Constrângeri de integritate peste n variabile

Implementare *out-of-line*

CREATE TABLE tabel (

$a1$ tip,
 $a2$ tip,
 $a3$ tip,
 $a4$ tip,

primary key ($a1,a2$), --cheie primară formată din 2 (sau mai multe) atribute

unique($a2,a3$), -- cheie candidat formată din 2 (sau mai multe) atribute

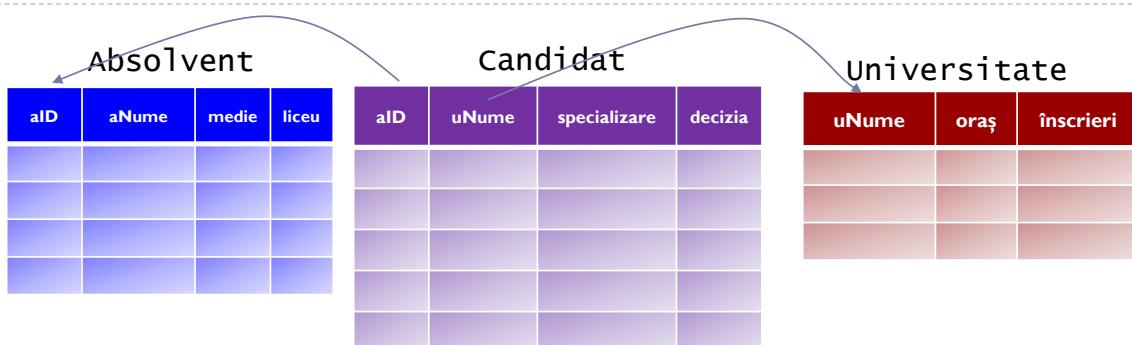
check (*condiție*), -- expresie booleană peste variabile declarate anterior:
 $((a1+a3)/2>=5)$

foreign key ($a3,a4$) **references** tabel2($b1,b2$) -- cheie străină multi-atribut
)

▶ 7

Integritate referențială

Definiții



► Integritate referențială de la R.A la S.B:

- ▶ fiecare valoare din coloana A a tabelului R trebuie să apară în coloana B a tabelului S
- ▶ A se numește cheie străină
- ▶ B trebuie să fie cheie primară pentru S sau măcar declarat unic
- ▶ sunt permise chei străine multi-atribut

▶ 8

Integritate referențială Realizare

► Comenzi ce pot genera încălcarea restricțiilor:

- ▶ inserări în R
- ▶ ștergeri în S
- ▶ actualizări pe R.A sau S.B

► Acțiuni speciale:

- ▶ la ștergere din S:

ON DELETE RESTRICT (implicit) | **SET NULL** | **CASCADE**

- ▶ la actualizări pe S.B:

ON UPDATE RESTRICT (implicit) | **SET NULL** | **CASCADE**

► 9

Integritate referențială oul sau găina?

```
CREATE TABLE chicken (cID INT PRIMARY KEY,  
                     eID INT REFERENCES egg(eID));  
  
CREATE TABLE egg(eID INT PRIMARY KEY,  
                 cID INT REFERENCES chicken(cID));
```

```
CREATE TABLE chicken(cID INT PRIMARY KEY, eID INT);  
CREATE TABLE egg(eID INT PRIMARY KEY, cID INT);
```

```
ALTER TABLE chicken ADD CONSTRAINT chickenREFegg  
    FOREIGN KEY (eID) REFERENCES egg(eID)  
    DEFERRABLE INITIALLY DEFERRED; -- Oracle  
  
ALTER TABLE egg ADD CONSTRAINT eggREFchicken  
    FOREIGN KEY (cID) REFERENCES chicken(cID)  
    DEFERRABLE INITIALLY DEFERRED; -- Oracle
```

```
INSERT INTO chicken VALUES(1, 2);  
INSERT INTO egg VALUES(2, 1);  
COMMIT;
```

Cum rezolvați problema inserării dacă verificarea constrângerii se efectuează imediat după fiecare inserare?
Dar problema ștergerii tabelelor?

► 10

Aserțiuni

create assertion Key

```
check ((select count(distinct A) from T) =  
       (select count(*) from T));
```

create assertion ReferentialIntegrity

```
check (not exists (select * from Candidat  
                   where aID not in (select aID from Student)));
```



Constrângeri de integritate Abateri de la standardul SQL

- ▶ Postgres, SQLite, Oracle, MySQL(innodb) implementează și validează toate constrângerile anterioare
- ▶ Standardul SQL permite utilizarea de interogări în clauza check însă nici un SGBD nu le suportă
- ▶ Nici un SGBD nu a implementat aserțiunile din standardul SQL, funcționalitatea lor fiind furnizată de declanșatoare



...DEMO...

(fișierul *constrângeri.sql*)

▶ 13

Declansatoare (constrângeri dinamice)

- ▶ Monitorizează schimbările în baza de date, verifică anumite condiții și inițiază acțiuni
- ▶ Reguli eveniment-condiție-acțiune
 - ▶ Introduc elemente din logica aplicației în SGBD
 - ▶ Forțează constrângeri care nu pot fi exprimate altfel
 - ▶ Sunt expresive
 - ▶ Pot întreprinde acțiuni de reparare
 - ▶ implementarea variază în funcție de SGBD, exemplele de aici urmăresc standardul SQL

▶ 14

Declansătoare Implementare

Create Trigger nume

Before|After|Instead Of evenimente

[variabile-referențiate]

[**For Each Row**] -- acțiune se execută pt fiecare linie modificată (tip row vs. statement)

[**When (condiție)**] -- ca o condiție WHERE din SQL

actiune -- în standardul SQL e o comandă SQL, în SGBD-uri poate fi bloc procedural

- ▶ evenimente:
 - ▶ **INSERT ON tabel**
 - ▶ **DELETE ON tabel**
 - ▶ **UPDATE [OF a1,a2,...] ON tabel**
- ▶ variabile-referențiate (după declarare pot fi utilizate în condiție și acțiune):
 - ▶ **OLD TABLE AS var**
 - ▶ **NEW TABLE AS var**
 - ▶ **OLD ROW AS var** – pentru ev. **DELETE, UPDATE**
 - ▶ **NEW ROW AS var** – pentru ev. **INSERT, UPDATE**

} doar pentru declansătoare de tip
linie/row

▶ 15

Declansătoare Exemplu (1)

- ▶ integritate referențială de la R.A la S.B cu ștergere în cascadă

Create Trigger Cascade

After Delete On S

Referencing Old Row As O

For Each Row

[fără condiții]

Delete From R Where A = O.B

Create Trigger Cascade

After Delete On S

Referencing Old Table As OT

[**For Each Row**]

[fără condiții]

Delete From R Where

A in (select B from OT)

▶ 16

Declansătoare Probleme potențiale

- ▶ mai multe declansătoare activate în același timp: care se execută primul?
- ▶ acțiunea declansatorului activează alte declansătoare: înlănțuire sau auto-declanșare ce poate duce la ciclare

▶ 17

Declansătoare Abateri de la standardul SQL

- ▶ **Postgres**
 - ▶ cel mai apropiat de standard
 - ▶ implementează row+statement, old/new+row/table
 - ▶ sintaxa suferă abateri de la standard
- ▶ **SQLite**
 - ▶ doar tip row (fără old/new table)
 - ▶ se execută imediat, după modificarea fiecărei linii (abatere comportamentală de la standard)
- ▶ **MySQL**
 - ▶ doar tip row (fără old/new table)
 - ▶ se execută imediat, după modificarea fiecărei linii (abatere comportamentală de la standard)
 - ▶ permite definirea unui singur declansator / eveniment asociat unui tabel
- ▶ **Oracle**
 - ▶ implementează standardul: row+statement cu modificări ușoare de sintaxă
 - ▶ tipul instead-of este permis numai pt. view-uri
 - ▶ permite inserarea de blocuri procedurale
 - ▶ introduce restricții pentru a evita ciclarea
 - ▶ **aprofundate la laborator**

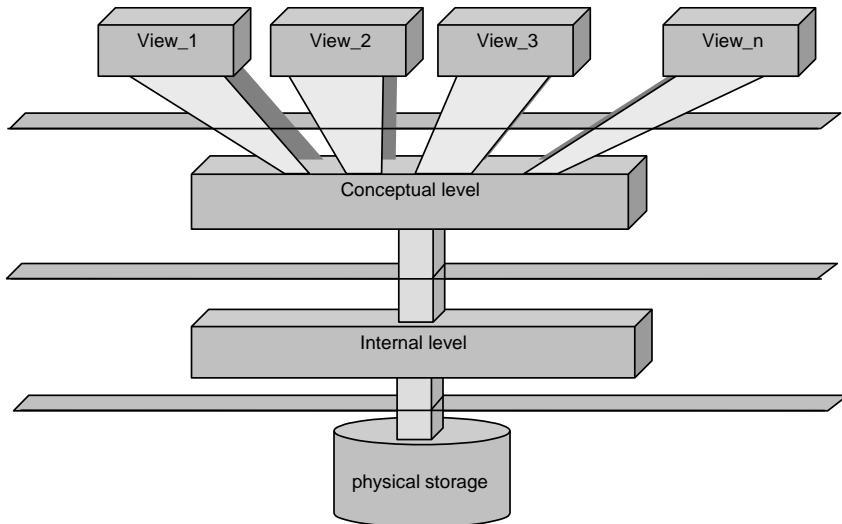
▶ 18

...DEMO...

(fișierul *declansatoare.sql*)

▶ 19

View-uri – Tabele virtuale



Cele 12 reguli ale lui Codd

1. Information Rule
2. Guaranteed Access Rule
3. Comprehensive Data Sub-language Rule
4. View Update Rule
5. High Level Insert, Update and Delete
6. Physical Data Independence
7. Logical Data Independence
8. Integrity Independence
9. Non Subversion Rule
10. Systematic Treatment of Null Values
11. Database Description Rule
12. Distribution Independence

▶ 20

Motivație

- ▶ acces modular la baza de date
- ▶ ascunderea unor date față de unii utilizatori
- ▶ ușurarea formulării unor interogări

- ▶ aplicațiile reale tend să utilizeze foarte multe view-uri

▶ 21

Definire și utilizare

- ▶ Un view este în esență o interogare stocată formulată peste tabele sau alte view-uri
- ▶ Schema view-ului este cea a rezultatului interogării

- ▶ Conceptual, un view este interogat la fel ca orice tabel
- ▶ În realitate, interogarea unui view este rescrisă prin inserarea interogării ce definește view-ul urmată de un proces de optimizare specific fiecărui SGBD

- ▶ Sintaxa

Create View numeView [a1,a2,...] As <frază_select>

▶ 22

Modificarea view-urilor

- ▶ View-urile sunt în general utilizate doar în interogări însă pentru utilizatorii externi ele sunt tabele: trebuie să poată suporta comenzi de manipulare/modificare a datelor
- ▶ Soluția: modificări asupra view-ului trebuie să fie rescrise în comenzi de modificare a datelor în tabelele de bază
 - ▶ de obicei este posibil
 - ▶ uneori există mai multe variante
- ▶ **Exemplu**
 - ▶ $R(A,B), V(A)=R[A]$, Insert into V values(3)
 - ▶ $R(N), V(A)=\text{avg}(N)$, update V set A=7

▶ 23

Modificarea view-urilor Abordări

- ▶ creatorul view-ului rescrie toate comenziile de modificare posibile cu ajutorul declanșatorului de tip **INSTEAD OF**
 - ▶ acoperă toate cazurile
 - ▶ garantează corectitudinea?
- ▶ standardul SQL prevede existența de view-uri inherent actualizabile (**updatable views**) dacă:
 - ▶ view-ul e creat cu comanda select fără clauza DISTINCT pe o singură tabelă T
 - ▶ atributele din T care nu fac parte din definiția view-ului pot fi NULL sau iau valoare default
 - ▶ subinterrogările nu fac referire la T
 - ▶ nu există clauza GROUP BY sau altă formă de agregare

▶ 24

View-uri materializate

Create Materialized View $V[a_1, a_2, \dots]$ As <frăză_select>

- ▶ are loc crearea unui nou tabel V cu schema dată de rezultatul interogării
- ▶ tuplele rezultat al interogării sunt inserate în V
- ▶ interogările asupra lui V se execută ca pe orice alt tabel
- ▶ **Avantaje:**
 - ▶ specifice view-urilor virtuale + crește viteza interogărilor
- ▶ **Dezavantaje:**
 - ▶ V poate avea dimensiuni foarte mari
 - ▶ orice modificare asupra tabelelor de bază necesită refacerea lui V
 - ▶ problema modificării tabelelor de bază la modificarea view-ului rămâne

▶ 25

Cum alegem ce materializăm

- ▶ dimensiunea datelor
- ▶ complexitatea interogării
- ▶ numărul de interogări asupra view-ului
- ▶ numărul de modificări asupra tabelelor de bază ce afectează view-ul și posibilitatea actualizării incrementale a view-ului
- ▶ punem în balanță timpul necesar execuției interogărilor și timpul necesar actualizării view-ului

▶ 26

...DEMO...

(fișierul `views.sql`)

▶ 27

Bibliografie

- ▶ **Hector Garcia-Molina, Jeff Ullman, Jennifer Widom:** *Database Systems: The Complete Book (2nd edition)*, Prentice Hall; (June 15, 2008)
- ▶ **Oracle:**
 - ▶ http://docs.oracle.com/cd/B28359_01/server.111/b28310/general005.htm
 - ▶ <http://www.oracle-base.com/articles/9i/MutatingTableExceptions.php>
 - ▶ http://www.dba-oracle.com/t_avoiding_mutating_table_error.htm

▶ 28

3 CURS 12 & 13: tipuri de indecsi, B^+ arbori (formule, nr nivele etc)

- daca datele sunt sortate dupa cheia de cautare (adica dupa ce atribut cautam ceva in tabel) \Rightarrow complexitate timp: $O(\log_2(\text{numarul de elemente din tabela}))$

- tipuri de indecsi: slide-urile 9, 10, 11, 12, 13, 15
- index B^+ arbore= interval
- index hash= numar fix
- index multilevel / multi-cheie= mai multe date din tabel
- index bitmap = valori puține (ex note: max 10 valori)

B^+ ARBORI:

- m valori si m+1 pointeri per nod

► Nu e obligatoriu ca nodurile să fie complet ocupate:

- Rădăcina are cel puțin 2 și cel mult $m + l$ pointeri/descendenți (respectiv, cel puțin 1 și cel mult m valori, ordonate crescător)
- Fiecare nod de pe un nivel intern are cel puțin $\lceil \frac{(m+l)}{2} \rceil$ și cel mult $m + l$ pointeri/descendenți (echivalent, cel puțin $\lfloor \frac{m}{2} \rfloor$ și cel mult m valori ordonate crescător)
- Fiecare nod frunză are cel puțin $\lceil \frac{m}{2} \rceil$ și cel mult m valori; toți pointerii trimit către fișierul de date, exceptând ultimul pointer care trimit către următorul nod frunză (cu valori mai mari).

- numar de blocuri transferate la cautare/stergere/inserare:

$\lceil \log_{\lceil \frac{m+1}{2} \rceil}(K) \rceil$ pentru K valori posibile a cheii de cautare

- numar nivele:

- Cel mult $\lceil \log_{\lceil \frac{m+1}{2} \rceil}(K) \rceil$ pentru K valori a cheii de căutare
 - Nivelul 2: cel puțin 2 noduri
 - Nivelul 3: cel puțin $2 * \lceil \frac{m+1}{2} \rceil$ noduri
 - Nivelul 4: cel puțin $2 * \lceil \frac{m+1}{2} \rceil * \lceil \frac{m+1}{2} \rceil$ noduri
 - etc...

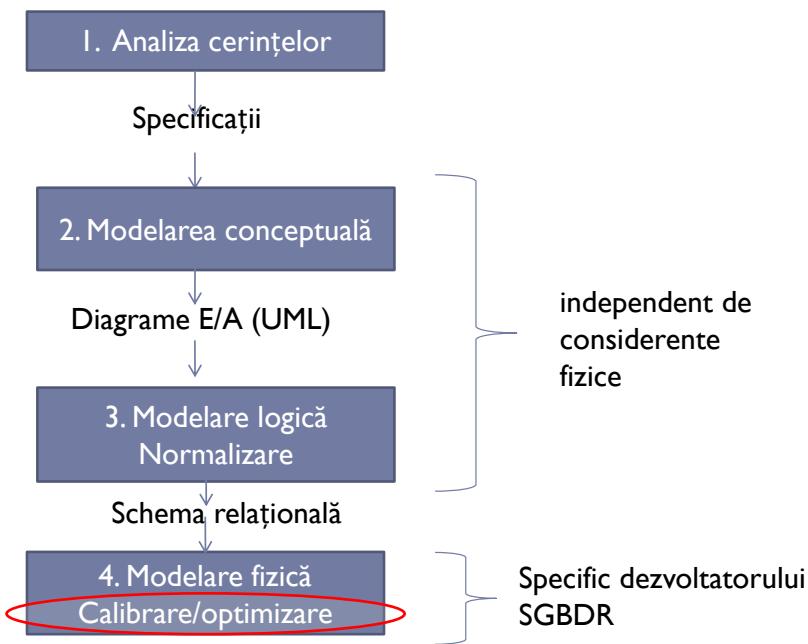


CREATE BITMAP INDEX *idx* ON
BAZE_DE_DATE (*topic*);

SELECT * FROM BAZE_DE_DATE WHERE *topic* = 'INDECSI';

Mihaela Elena Breabă
© FII 2020-2021

Proiectarea Bazelor de date Relaționale Metodologie



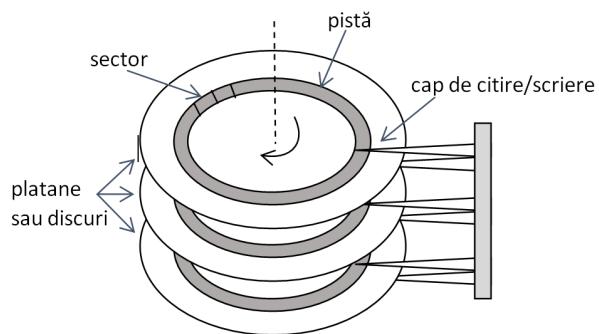
Cuprins

- ▶ Stocare fizică și acces
- ▶ Indexare – motivație și concepte de bază
- ▶ Structuri ordonate
 - ▶ Indecși secvențiali
 - ▶ B⁺-arbori
 - ▶ Indecși multi-cheie
- ▶ Hashing
 - ▶ Hashing static
 - ▶ Hashing dinamic
- ▶ Acces multi-cheie și Indecși bitmap
- ▶ Suportul SQL pentru indexare
- ▶ Indexarea în Oracle

▶ 3

Stocarea datelor și acces

| ID | prenume | nota |
|----|---------|------|
| 20 | Ioana | 9.5 |
| 40 | Andrei | 8.66 |
| 10 | Tudor | 8.55 |
| 30 | Maria | 8.33 |
| 70 | Alex | 9.33 |



- ▶ Timpul necesar pentru a aduce un **bloc** de date în memorie este determinat de:
 - ▶ *Timpul de localizare* (timpul necesar poziționării capului de citire pe pistă)
 - ▶ *Latența rotațională* (timpul de rotație a pistei/discului sub capul de citire)
 - ▶ *Timpul de transfer* (timpul necesar transferului datelor către memoria de lucru)
- ▶ Pentru a optimiza timpul de acces, o bază de date relațională stochează datele în mod secvențial, înregistrare după înregistrare

▶ 4

Indexare – Motivație (1)

- ▶ De obicei, SGBD-ul petrece majoritatea timpului rezolvând interogări (căutând)

SELECT * FROM Student

WHERE ID=40;

Cheie de căutare

Cum găsim ânregistrările dorite?

a) Ordonare aleatorie

| ID | prenume | nota |
|----|---------|------|
| 20 | Ioana | 9.5 |
| 40 | Andrei | 8.66 |
| 10 | Tudor | 8.55 |
| 30 | Maria | 8.33 |
| 70 | Alex | 9.33 |

Cheie de sortare

b) Ordonare crescătoare după ID

| ID | prenume | nota |
|----|---------|------|
| 10 | Tudor | 8.55 |
| 20 | Ioana | 9.5 |
| 30 | Maria | 8.33 |
| 40 | Andrei | 8.66 |
| 70 | Alex | 9.33 |

▶ 5

Indexare – Motivație (2)

- ▶ Când datele sunt sortate după cheia de căutare devine posibilă căutarea binară
 - ▶ Complexitate timp: $O(\log_2(N))$
 $(\log_2(100\ 000)=17)$

SELECT * FROM student

WHERE prenume='Ioana';

Cum putem rezolva interogarea de mai sus eficient (datele nu sunt sortate după prenume)?

Soluția: construim un fișier **index**

▶ 6

Concepte de bază în indexare

- ▶ **Fișier de date** – secvența de blocuri ce conțin înregistrările unui tabel
- ▶ **Cheie de căutare** – un atribut (sau o mulțime de atribut) care constituie criteriu de selecție/căutare
- ▶ **Cheie de sortare** – un atribut care decide ordonarea înregistrărilor în fișierul cu date
- ▶ **Fișier index** – este asociat unei chei de căutare într-un fișier cu date și conține **înregistrări index** de forma

| | |
|----------------------------|---------|
| Valoare a cheii de căutare | pointer |
|----------------------------|---------|

- ▶ **Index dens** – stochează câte o intrare pentru fiecare valoare existentă în fișierul cu date a cheii de căutare
- ▶ **Index rar** – nu stochează toate valorile cheii de căutare

Observații:

- ▶ Un fișier cu date poate avea asociate mai multe fișiere index
- ▶ Fișierele index sunt de obicei de dimensiuni mai mici comparativ cu fișierul de date

▶ 7

De reflectat asupra...

- ▶ CÂȚI indecsi ar trebui să folosim în practică ?
- ▶ CÂND trebuie creați indecsi și când nu trebuie creați indecsi?
- ▶ CUM ar trebui să indexăm (ca structuri de date utilizate)?

- ▶ Considerente:
 - ▶ Spațiul de stocare necesar
 - ▶ Timpul de acces
 - ▶ Timpul de inserare
 - ▶ Timpul de ștergere
 - ▶ Tipul interogărilor adresate

▶ 8

Tipuri de indecsi

- ▶ **Indecsi ordonați:** valorile cheii de căutare sunt ordonate
 - ▶ Indecsi secvențiali
 - ▶ B^+ -arbori
- ▶ **Indecsi hash:** valorile cheii de căutare sunt uniform distribuite în grupuri denumite *buckets* cu ajutorul unei funcții *hash*
 - ▶ **Bucket:** unitate de stocare ce poate conține una sau mai multe înregistrări
 - ▶ **Funcție hash** = funcție de *dispersie* – mapează date de dimensiune variabilă la o mulțime fixă de valori
- ▶ **Indecsi bitmap:** asociați atributelor categoriale/discrete, codifică distribuția valorilor ca o matrice binară

▶ 9



Indecsi ordonați:
fișiere secvențiale

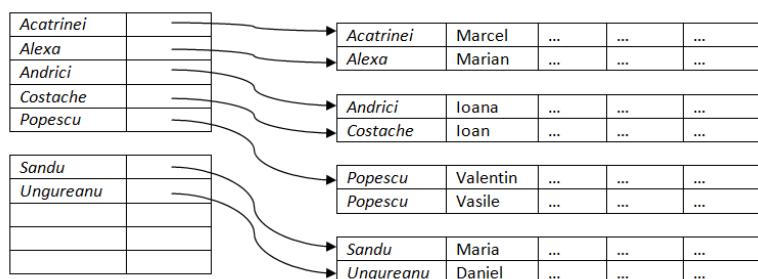
Indecși secvențiali

- ▶ Intrările index sunt sortate pe baza cheii de căutare
 - ▶ Ex: catalogul cu autori dintr-o bibliotecă
- ▶ **Index primar:** cheia de căutare (cheia indexului) este și cheie de sortare a fișierului cu date
 - ▶ Cheia de căutare/sortare în acest caz constituie de obicei (nu obligatoriu) chiar cheia primară a tabelului
 - ▶ Un tabel poate avea cel mult un index primar. **DE CE?**
- ▶ **Index secundar:** cheia de căutare dă o altă ordonare a înregistrărilor decât cea din fișierul de date



Indecși denși

- ▶ **Index dens:** conține intrări pentru fiecare valoare a cheii de căutare existentă în fișierul cu date.
- ▶ Dacă indexul este primar, conține un singur pointer pentru toate înregistrările cu aceeași valoare a cheii de căutare (un pointer către prima înregistrare din serie). **DE CE?**
- ▶ Dacă indexul este secundar, mai multe intrări pot fi necesare pentru o aceeași valoare. **DE CE?**



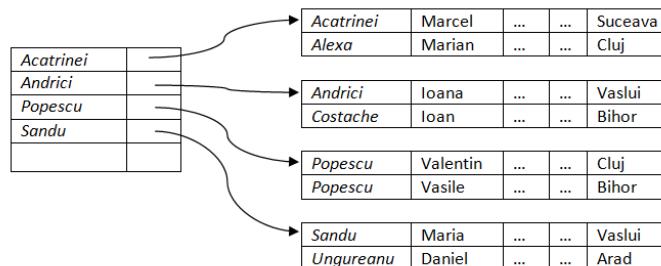
Index dens primar – cheia de căutare este aceeași cu cheia de sortare a fișierului cu date



Indecși rari

► Indecșii rari NU conțin ca intrări toate valorile cheii de căutare

- ▶ Aplicabili doar pentru înregistrări care sunt ordonate pe baza cheii de căutare (când atributul cheie de căutare este și cheie de sortare a fișierului cu date)
- ▶ De obicei o intrare în index trimite către un bloc din fișierul cu date
- ▶ Pentru a localiza o intrare cu valoarea k a cheii de căutare în fișierul cu date:
 - ▶ Găsim intrarea din index care corespunde la cea mai mare valoare mai mică decât k
 - ▶ Ne uităm secvențial în fișierul cu date începând cu înregistrarea la care ne trimite indexul

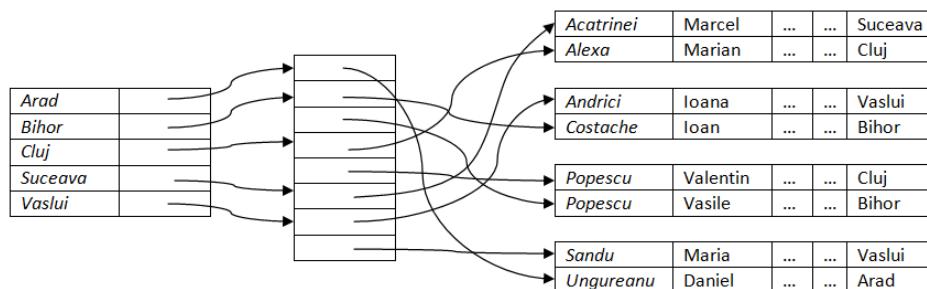


Index rar: cheia de căutare este întotdeauna și cheie de sortare a fișierului cu date

► 13

Index secundar

- ▶ Interogare: *Găsiți toți studenții care locuiesc în Cluj (fișierul cu date este sortat pe baza numelor studentilor!)*
- ▶ Soluția: indexul secundar (dens!)
- ▶ Pentru a implementa eficient asocierea de tip unu-la-mulți dintre index și fișierul cu date, blocuri cu pointeri sunt utilizate



► 14

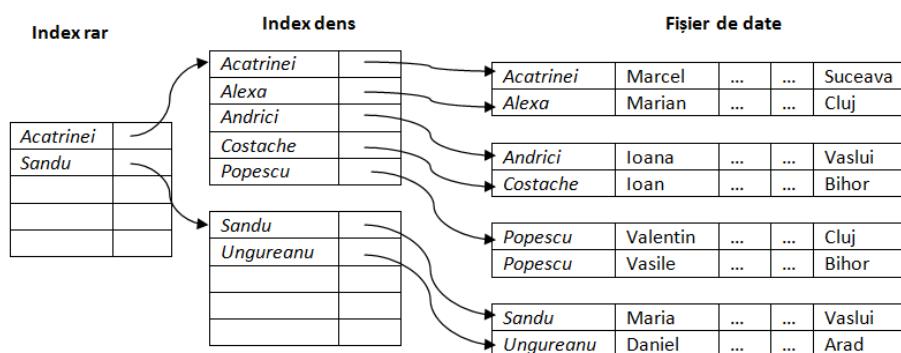
Indecși multi-nivel

- ▶ **Index multi-nivel:** un index poate fi asociat altui index și nu direct fișierului de date; acesta este un index rar
 - ▶ Necesar când fișierul index asociat fișierului cu date are dimensiuni considerabile sau nu începe în memoria de lucru
 - ▶ **Index intern** – indexul construit peste fisierul de date
 - ▶ **Index extern** – indexul rar construit peste indexul intern
- ▶ Când indexul extern este prea mare un alt index rar poate fi construit peste acesta, etc...
- ▶ Indecșii de pe toate nivelele trebuie actualizați când fișierul cu date suferă modificări prin operații DML

▶ 15

Indecși multi-nivel

Exemplu



▶ 16

Actualizarea indecșilor secvențiali Ștergeri în date

1. Se găsește înregistrarea ce trebuie ștersă – se poate apela la index;
2. Se șterge înregistrarea din fișierul cu date;
3. Se actualizează indecșii asociați tabelului:
 1. Dacă mai există și alte înregistrări cu aceeași valoare a cheii de căutare, se șterge doar pointerul
 2. Dacă înregistrarea ștersă este singura cu valoarea cheii **k**, aceasta trebuie ștersă din index
 - ▶ Ștergerea din indexul dens: e similară ștergerii dintr-un fișierul de date
 - ▶ Ștergerea dintr-un index rar:
 - Dacă există intrarea **k** în index, aceasta este înlocuită de următoarea valoare a cheii de căutare (din ordonarea valorilor cheii de căutare existente în fișierul cu date)
 - Dacă următoare valoare există deja în index, intrarea corespunzătoare lui **k** este ștersă.

▶ 17

Actualizarea indecșilor secvențiali Inserări

1. Se inserează tuplul în fișierul cu date
 2. Se caută în index valoarea cheii de căutare corespunzătoare noului tuplu și se actualizează indexul astfel:
 - ▶ Indexul dens: dacă valoarea nu apare în index, va fi inserată; altfel, dacă indexul este secundar se adaugă doar pointerul
 - ▶ Indexul rar: dacă indexul menține o intrare pentru fiecare bloc a fișierului cu date, doar când un nou bloc este creat în fișierul cu date, o nouă intrare va fi adăugată în index, trimițând la prima înregistrare a blocului.
-
- ▶ Inserările în fișierul cu date și în fișierul index poate necesita crearea unor blocuri de exces -> structura secvențială degenerăază
 - ▶ Inserarea și ștergerea în indecșii multi-nivel sunt extensii simple ale cazurilor discutate

▶ 18

Indecși ordonați: B^+ -arbori

19

Indecși bazați pe structuri B^+ -arbori Motivație

- ▶ Structurile secvențiale ordonate se degradează după multe operații DML
- ▶ Reconstruirea indecșilor este necesară dar costisitoare
- ▶ B^+ -arborii
 - ▶ Măresc viteza de găsire a datelor și elimină necesitatea de reorganizare continuă
 - ▶ Sunt utilizati extensiv pentru indexarea datelor în SGBD-urile relaționale

Structura unui B⁺-arbore (1)

- Un arbore echilibrat a.i. toate frunzele sunt pe același nivel

- Structura unui nod:

| | | | | | | | |
|----------------|----------------|----------------|----------------|-----|----------------|----------------|------------------|
| P ₁ | K ₁ | P ₂ | K ₂ | ... | P _m | K _m | P _{m+1} |
|----------------|----------------|----------------|----------------|-----|----------------|----------------|------------------|

- K_i – valori a cheii de căutare
- P_i pointeri către
 - Noduri de pe nivelul imediat inferior
 - Dacă nodul este frunză, către o înregistrare din fișierul cu date sau către blocuri de pointeri către înregistrări
- Arborele este caracterizat de o constantă m ce specifică numărul maxim de valori ce pot fi stocate într-un nod (numărul maxim de pointeri sau descendenți ai nodului este $m+1$)
 - De regulă m este calculat a.i. dimensiunea unui nod să fie egală cu cea a unui bloc de date
- Valorile cheii de căutare sunt ordonate crescător în cadrul fiecărui nod
 - $K_1 < K_2 < K_3 < \dots < K_m$

► 21

Structura unui B⁺-arbore (2)

| | | | | | | | |
|----------------|----------------|----------------|----------------|-----|----------------|----------------|------------------|
| P ₁ | K ₁ | P ₂ | K ₂ | ... | P _m | K _m | P _{m+1} |
|----------------|----------------|----------------|----------------|-----|----------------|----------------|------------------|

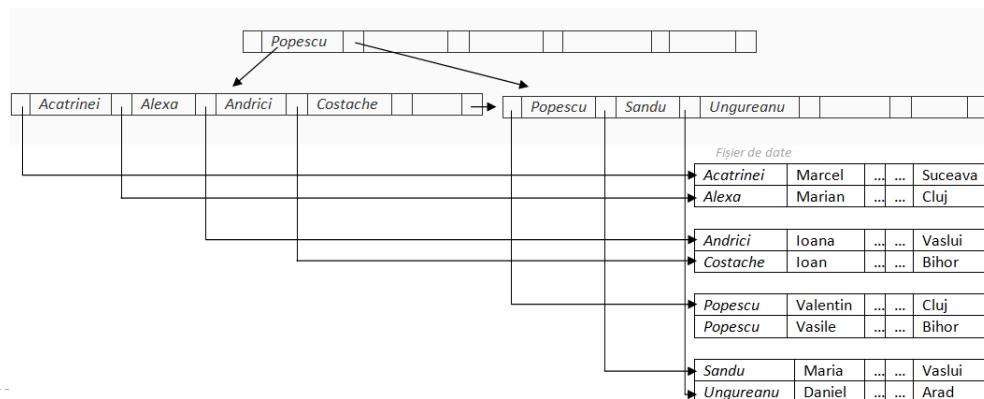
- Pentru un nod cu $m+1$ pointeri:
 - Toate valorile cheii de căutare ce apar în subarborele către care trimite P_1 sunt mai mici decât K_1
 - Pentru pointerul P_i , $2 \leq i \leq m$, toate valorile cheii de căutare din subarborele spre care acesta trimite sunt mai mari decât sau egale cu K_{i-1} și mai mici decât K_i
 - Toate valorile cheii de căutare din subarborele spre care trimite P_{m+1} sunt mai mari decât sau egale cu K_m

► 22

Reguli pentru ocuparea nodurilor

► Nu e obligatoriu ca nodurile să fie complet ocupate:

- ▶ **Rădăcina** are cel puțin 2 și cel mult $m + l$ pointeri/descendenți (respectiv, cel puțin 1 și cel mult m valori, ordonate crescător)
- ▶ Fiecare nod de pe un nivel **intern** are cel puțin $\lceil (m+l)/2 \rceil$ și cel mult $m + l$ pointeri/descendenți (echivalent, cel puțin $\lfloor m/2 \rfloor$ și cel mult m valori ordonate crescător)
- ▶ Fiecare nod **frunză** are cel puțin $\lfloor m/2 \rfloor$ și cel mult m valori; toți pointerii trimit către fișierul de date, exceptând ultimul pointer care trimit către următorul nod frunză (cu valori mai mari).



► 23

B⁺-arbori – parametrul m

Exemplu

► Presupunând că

- ▶ 1 bloc de memorie = 1024 octeți
- ▶ Cheia de căutare = un sir de maxim 20 caractere (1 caracter = 1 octet)
- ▶ 1pointer = 8 octeți

► Care este constanta arborelui, adică numărul maxim de valori într-un nod?

► Răspuns

- ▶ Identificăm cea mai mare valoare m care satisface $20m + 8(m + l) \leq 1024$.
- ▶ $m=36$

► Structura arborelui

- ▶ Rădăcina: cel puțin 2 pointeri, cel mult 36 pointeri (echivalent: intre una și 36 valori)
- ▶ Nod intern: cel puțin 19, cel mult 37 pointeri
- ▶ Nod frunză: cel puțin 18, cel mult 36 valori echivalent pointeri către fișierul cu date)

► 24

B⁺-arbori Observații

- ▶ Pentru că nodurile sunt conectate prin pointeri, blocuri apropiate logic nu trebuie să fie neapărat și apropiate fizic
- ▶ Toate nivelele exceptând nivelul frunză formează o ierarhie de indecsi rari externi peste nivelul frunză
- ▶ Nivelul frunză formează un index secvențial dens peste fișierul cu date

- ▶ B⁺-arborele conține un număr relativ mic de niveluri
 - ▶ Cel mult $\lceil \log_{(m+1)/2}(K) \rceil$ pentru K valori a cheii de căutare
 - ▶ Nivelul 2: cel puțin 2 noduri
 - ▶ Nivelul 3: cel puțin $2^*\lceil(m+1)/2\rceil$ noduri
 - ▶ Nivelul 4: cel puțin $2^*\lceil(m+1)/2\rceil * \lceil(m+1)/2\rceil$ noduri
 - ▶ etc...
- ▶ Inserările și ștergerile sunt procesate eficient: restructurarea indexului necesită timp logaritmice

▶ 25

Interogări pe B⁺- arbori

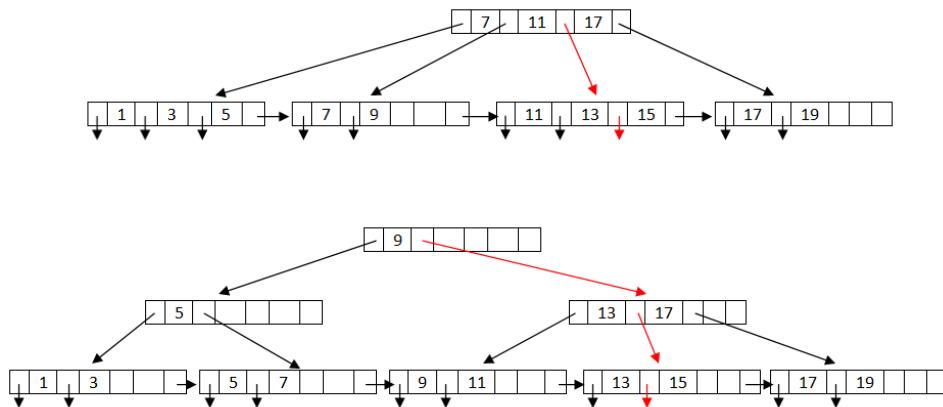
Scop: determinați toate înregistrările din fișierul de date care corespund valorii k a cheii de căutare

1. $N =$ rădăcina
2. Repetă
 - Identifică în N cea mai mică valoare a cheii de căutare care e mai mare decât k
 - Dacă o astfel de valoare K_i există, atunci $N = P_i$
 - altfel $N = P_n$ ($k \geq K_{n-1}$)până când N este frunză
3. Dacă există $K_i = k$ în frunza N, pointerul P_i trimite către înregistrarea dorită
 - Altfel, nu există nici o înregistrare cu valoarea k a cheii de căutare

▶ 26

Interogări Exemple

Cheia de căutare stochează toate valorile impare intre 1-19
Reprezentați posibil arbori pentru m=3 și efectuați o interogare pentru cheia 15



▶ 27

Interogări pe B⁺-arbori Exercițiu

- ▶ Dat m=100 (fiecare nod are dimensiunea unui bloc!)
- ▶ Pentru 1 milion de valori a cheii de căutare, cât de multe noduri (echivalență blocuri pe disc) sunt accesate la o căutare în B⁺-arbore? (R: 4)
- ▶ Dar dacă e utilizat un index secvențial? (R: 20)

▶ 28

Actualizări în B⁺-arbori

Inserarea

După inserarea unei înregistrări în fișierul de date cu valoarea k a cheii de căutare, la care trimite pointerul p :

1. Găsește nodul frunză care ar trebui să conțină k
2. Dacă valoarea există în nodul frunză:
 - ▶ Adaugă pointerul p în bucketul corespunzător valorii k a cheii
3. Dacă valoare nu există
 - ▶ Dacă este loc în nodul frunză, inserează perechea (p, k)
 - Altfel, divide nodul frunză ->

▶ 29

Actualizări în B⁺-arbori

Inserare: divizarea unui nod

- ▶ Divizarea unui nod frunză la inserarea unei perechi noi (p_i, k_i) :
 1. Se iau cele n perechi ordonate, inclusiv cea nou creată (p_i, k_i) . Se păstrează primele $\lceil n/2 \rceil$ perechi în nodul frunză existent și crează unul nou, P , care să conțină restul perechilor
 2. Fie k cea mai mică valoare din P . Inserează perechea (k, p) în nodul părinte a frunzei care s-a divizat – unde p este pointerul către noua frunză P .
 3. Dacă nodul părinte este plin, acesta trebuie să se dividă, propagând în sus divizarea până când se ajunge la un nod care nu e complet ocupat. În cel mai rău caz, nodul rădăcină este divizat, caz care crește adâncimea arborelul.
- ▶ Divizarea unui nod intern N la inserarea unei perechi (k, p)
 1. Se creează un nod temporar M care stochează $m+2$ pointeri și $m+1$ valori; se inserează perechile din N împreună cu perechea (k, p) , ordonate
 2. Se copiază $P_1, K_1, \dots, K_{\lceil (m+1)/2 \rceil - 1}, P_{\lceil (m+1)/2 \rceil}$ din M înapoi în N
 3. Se copiază $P_{\lceil (m+1)/2 \rceil + 1}, K_{\lceil (m+1)/2 \rceil + 1}, \dots, K_{m+1}, P_{m+2}$ din M într-un nod nou N'
 4. Se inserează $(K_{\lceil (m+1)/2 \rceil}, pN')$ în părintele nodului N

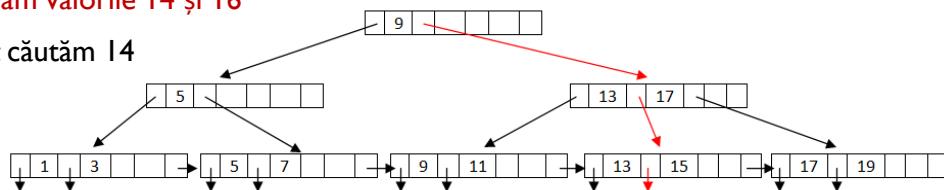
▶ 30

Actualizări în B⁺-arbori

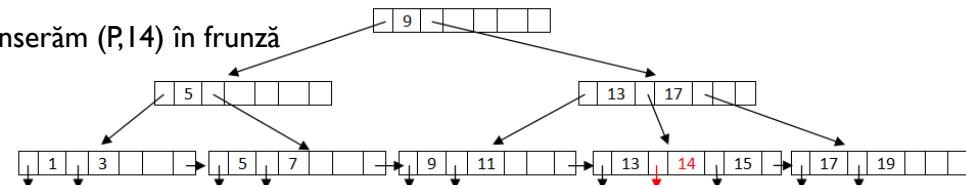
Inserare: Exemplu (1)

Inserăm valorile 14 și 16

Pas 1: căutăm 14



Pas 2: inserăm (P,14) în frunză

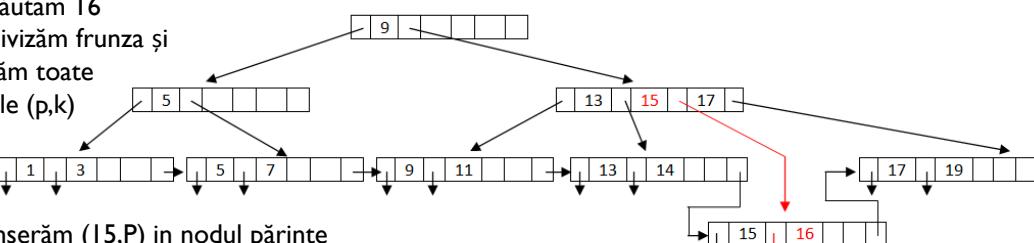


Pas 3: căutăm 16

Pas 4: divizăm frunza și

rearanjăm toate

Perechile (p,k)



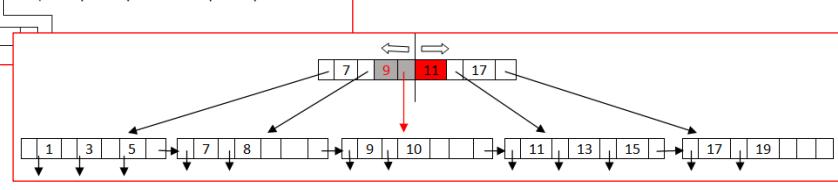
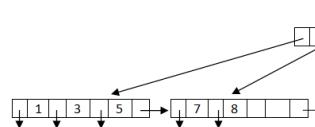
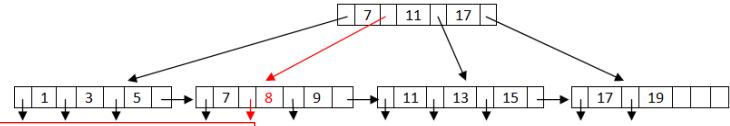
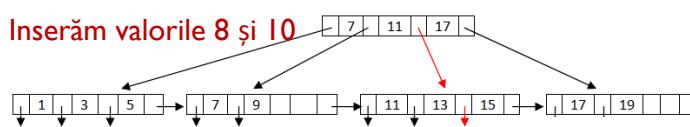
Pas 5: inserăm (15,P) în nodul părinte

▶ 31

Actualizări în B⁺-arbori

Inserare: Exemplu (2)

Inserăm valorile 8 și 10



11

7 9

17

1 3 5 7 8 9 10 11 13 15 17 19

▶ 32

Actualizări în B⁺-arbore

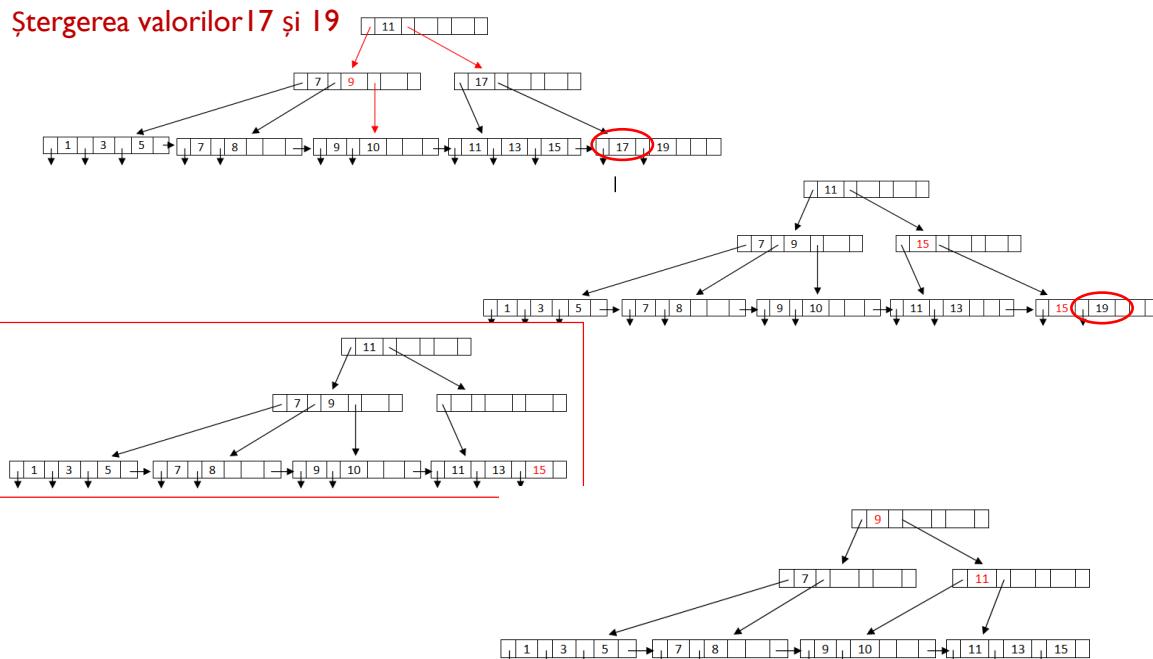
Ștergerea

1. Se șterge înregistrarea din fișierul cu date; fie k valoarea cheii și p pointerul ce ne conduce către această înregistrare
2. Se identifică în index frunza care conține valoarea k
3. Dacă pointerul p către înregistrarea ștearsă face parte dintr-un bucket în index, p este șters din bucket. Altfel (sau dacă bucketul devine gol) se șterge din nodul frunză perechea (p, k)
4. Dacă nodul frunză rămâne cu prea puține intrări și dacă există loc pentru acestea într-o frunză alăturată, un nod frunză este șters:
 1. Inserează toate intrările în frunza stângă și șterge frunza dreaptă
 2. Șterge perechea (K_{i-1}, P_i), unde P_i este pointerul către nodul frunză șters din nodul părinte. Dacă este necesar, se propagă în sus ștergerea. Dacă nodul rădăcină rămâne cu un singur pointer, acesta este șters și adâncimea arborelui scade.
5. Altfel, dacă nu este suficient spațiu într-o frunză vecină, perechile (pointer, key_value) sunt redistribuite între nodul curent și o frunză vecină:
 1. Astfel încât minimul este satisfăcut în ambele
 2. Se actualizează o pereche (key_value, pointer) în nodul părinte dacă este necesar

▶ 33

Actualizări în B⁺-arbore

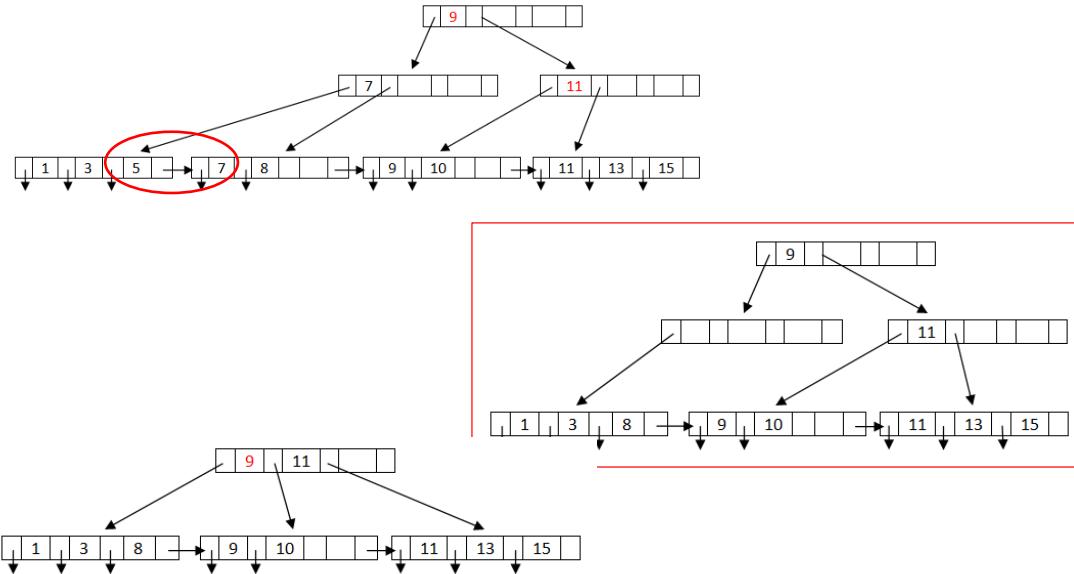
Ștergere: Exemplu (1)



▶ 34

Actualizări în B⁺-arbori Ştergere: Exemplu (2)

Ştergerea valorilor 5 și 7



▶ 35

B⁺-arbori Efficiență

- ▶ Căutare: cel mult $\lceil \log_{(m+1)/2}(K) \rceil$ blocuri transferate
- ▶ Deoarece nivelul frunză este conectat formând un index secvențial dens, interogările de tip interval sunt de asemenea rezolvate efficient
- ▶ Inserarea, ştergerea: cel mult $2 \lceil \log_{(m+1)/2}(K) \rceil$ blocuri transferate

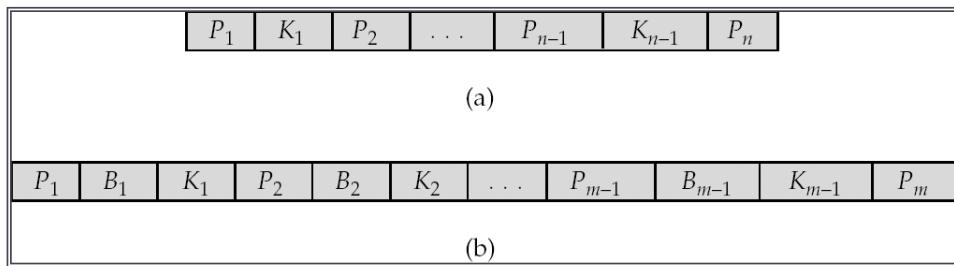
▶ 36

Indecsi ordonați: B-arbori

37

B-arbori

- ▶ Similari arborilor B^+ dar permit o singură apariție a unei valori a cheii
- ▶ Nu toate valorile cheii apar astfel pe nivelul frunză
 - ▶ Fiecare valoare vine cu un pointer în plus

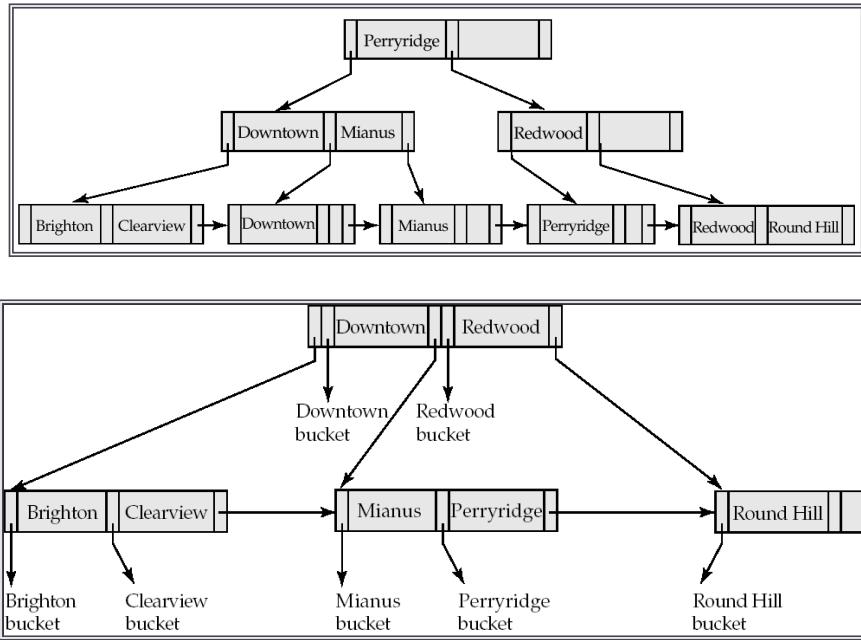


(a) $\text{Nod } \in \text{tr-un } B^+ \text{-arbore};$ (b) $\text{nod } \in \text{tr-un } B \text{-arbore}$

- ▶ Pointerii B_i conduc către înregistrări din fișierul cu date sau buckets de pointeri către înregistrări

Index B-arbore

Exemplu*



* Avi Silberschatz Henry F. Korth S. Sudarshan. "Database System Concepts".
McGraw-Hill Science/Engineering/Math; 6 edition (January 27, 2010)

Indeci B-arbore

Observații

- ▶ **Avantaje**
 - ▶ Devine posibil ca înregistrarea căutată să fie localizată înainte de ajunge la nivelul frunză
- ▶ **Dezavantaje**
 - ▶ Nodurile interne conțin mai multă informație (dar mai puține chei) rezultând în arbori cu adâncime mai mare
 - ▶ Inserările și ștergerile sunt mai complicate -> implementarea este mai dificilă
 - ▶ Nu este posibil să scanăm un tabel pe baza nivelului frunză
- ▶ **Avantajele nu cântăresc mai mult decât dezavantajele: B⁺-arborii sunt preferați în detrimentul B-arborilor de către dezvoltatorii SGBD-urilor relaționale**

Indecși ordonați: Indecși multi-cheie

41

Acces multi-cheie

```
SELECT *
FROM student
WHERE judet= 'Bihor' AND an> 2010;
```

- ▶ Sunt posibile mai multe strategii pentru a rezolva interogări cu mai multe chei de căutare:
 - ▶ Utilizarea unui index asociat atributului *judet*
 - ▶ Utilizarea unui index asociat atributului *an*
 - ▶ Utilizarea ambilor indecși de mai sus urmată de operația de intersecție a mulțimilor de pointeri
Dar dacă doar una dintre condiții este satisfăcută de un număr mare de înregistrări?

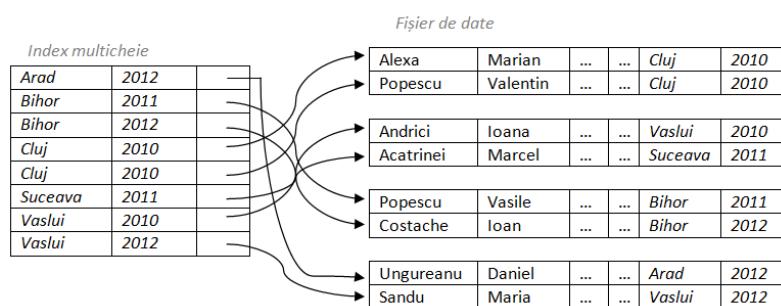
Indecși multi-cheie

- ▶ Cheia de căutare este compusă din mai mult de un atribut
- ▶ Ordinea lexicografică este utilizată: $(a_1, a_2) < (b_1, b_2)$ dacă
 - ▶ $a_1 < b_1$ sau
 - ▶ $a_1 = b_1$ și $a_2 < b_2$

Ex. Considerăm indexul multi-cheie (judet, an)

Rezolvă acesta la fel de eficient ambele interogări de mai jos?

- a) where judet = 'Bihor' AND an > 2010
- b) where judet > 'Bihor' AND an = 2010



▶ 43

Eficiență

- ▶ Ordinea atributelor într-un index multi-cheie contează!
- ▶ Eficiența depinde de selectivitatea atributelor

▶ 44

Indecsi ordonati multi-cheie: kd-arbori

45

kd-arbori

- ▶ Nu sunt utilizati de regulă în bazele de date relationale dar îi menționăm fiind o structură de căutare utilizată frecvent în bazele de date spațiale/geografice;
- ▶ Generalizare a arborelui binar de căutare:
 - ▶ k = numărul de atrbute a cheii de căutare multi-atribut
 - ▶ Fiecare nivel din arbore corespunde unuiuia dintre atrbute
 - ▶ Succesiunea de nivele reprezintă iterații peste mulțimea de atrbute
 - ▶ Pentru a obține arbori echilibrați, de obicei mediana este utilizată ca valoare în noduri interne

Organizarea de tip hash și Indecsi hash

47

Organizarea hash a fișierului cu date

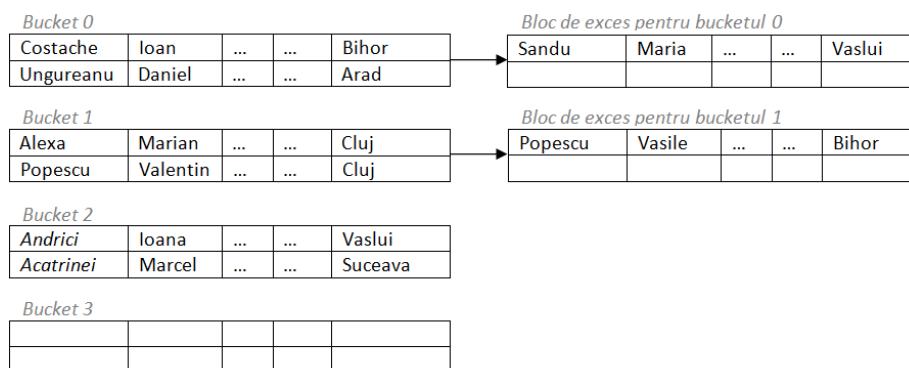
- ▶ În **organizarea de tip hash** a fișierului cu date, înregistrările nu sunt ordonate pe baza valorilor unui atribut ci sunt grupate în bucketuri cu ajutorul unei funcții hash (dimensiunea unui bucket de regulă corespunde unui bloc de memorie)
 - ▶ **Bucket:** unitate de stocare ce poate conține una sau mai multe înregistrări
 - ▶ **Funcție hash** = funcție de *dispersie* – mapează valori dintr-o mulțime de dimensiune variabilă la o mulțime fixă de valori
- ▶ **Funcția hash** $h:K \rightarrow B$ este în acest caz o funcție ce mapează valori ale cheii de căutare la o mulțime fixată de bucketuri
- ▶ Înregistrările cu valori diferite ale cheii de căutare pot fi mapate la același bucket
 - ▶ Căutarea implică parcurgerea secvențială a bucketului

Funcții hash

- ▶ Cerințe
 - ▶ Distribuție uniformă
 - ▶ Așignări aleatorii (spre deosebire de *locality sensitive hashing*)
- ▶ Funcțiile hash tipice utilizează operații pe reprezentarea binară (internă) a cheii de căutare
- ▶ Pot să apară situații în care dimensiunea bucketului e prea mică pentru a stoca toate înregistrările mapate -> blocuri de exces sunt utilizate

▶ 49

Organizarea de tip hash a fișierului cu date Exemplu

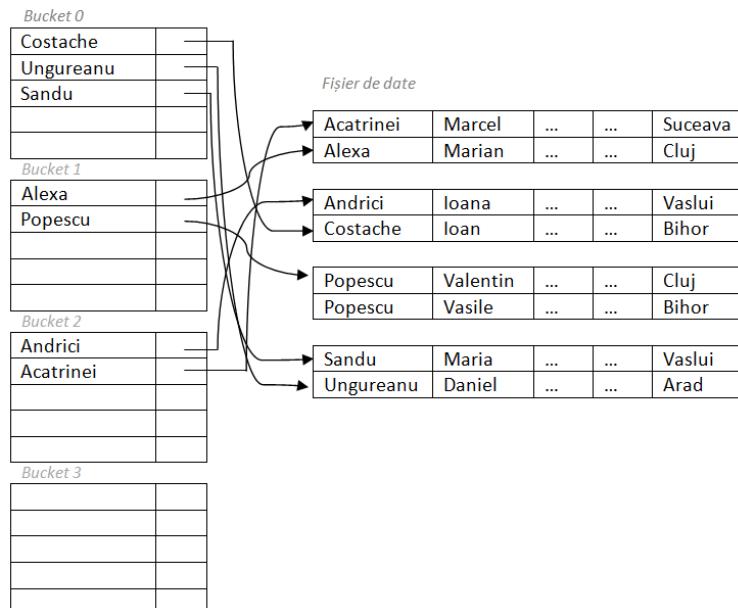


Atributul *prenume* este cheia de căutare iar numărul de bucketuri este fixat la 4:
Funcția hash: $h:\text{Dom}(\text{nume}) \rightarrow \{0, 1, 2, 3\}$ – calculează suma reprezentării binare modulo 4
Ex:
'Acatrinei':
'1000001 1100011 1100001 1110100 1110010 1101001 1101110 1100101 1101001'
 $h(\text{'Acatrinei'}) = 34 \% 4 = 2$.

▶ 50

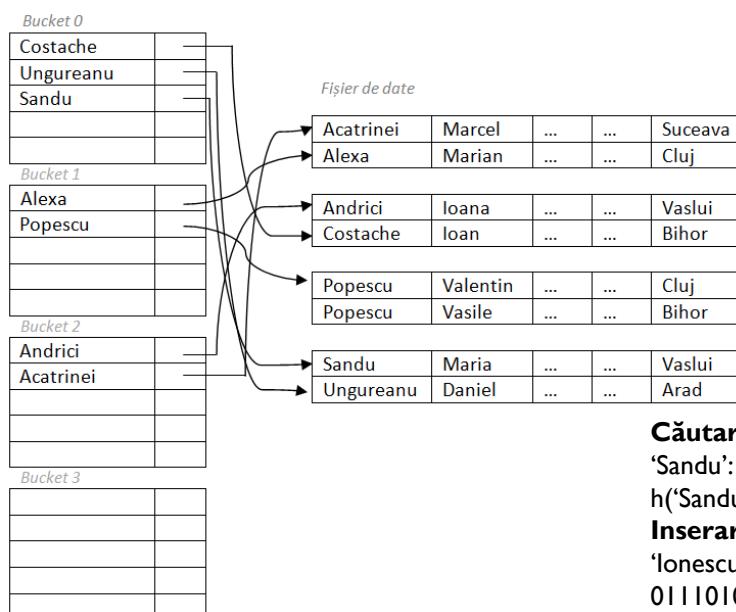
Indecși de tip hash

- Organizează valorile cheii de căutare cu pointerii asociați într-o tabelă hash



▶ 51

Indecși hash Operații



Căutare:

'Sandu': 01010011 01100001 01101110 01100100 01110101
 $h('Sandu') = 20\%4 = 0 \rightarrow$ scaneză bucketul 0

Inserare:

'Ionescu': 01001001 01101111 01101110 01100101 01110011 01100011
 01110101
 $H('Ionescu') = 32\%4 = 0 \rightarrow$ inserează mai întâi înregistrarea în fișierul de date și apoi intrarea index în bucketul 0

Stergere:

calculează hashul, scaneză bucketul, șterge

▶ 52

Indecși hash Eficiență

- ▶ În căutarea unei singure valori, în absența coliziunilor, gasirea unei înregistrări necesită citirea unui singur bloc ($O(1)$)
- ▶ Pentru interogări de tip interval, indecșii hash nu sunt eficienți. **DE CE?**

- ▶ În practică:
 - ▶ Postgres și SQLServer implementează indecșii hash
 - ▶ Oracle implementează organizarea de tip hash a fișierului cu date dar nu și indecșii hash

▶ 53

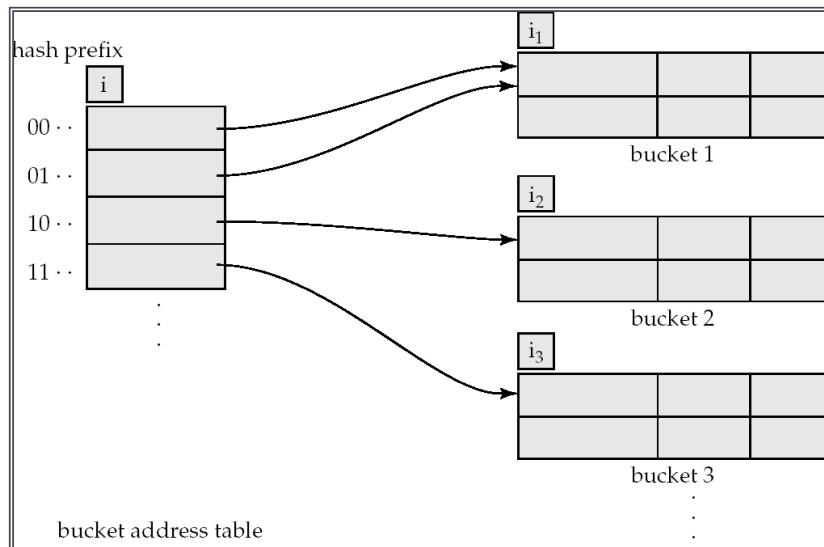
Hash dinamic Motivație

- ▶ Funcția h mapează valorile cheii de căutare la o mulțime fixă de adrese de buckets.
 - ▶ Dacă dimensiunea fișierului cu date crește, blocuri de exces sunt generate
 - ▶ Dacă dimensiunea fișierului se micșorează, spațiu este alocat inutil
- ▶ Soluții:
 - ▶ Reorganizări periodice cu o nouă funcție hash (costisitor, necesită întreruperea operațiilor bazei de date)
 - ▶ Modificarea dinamică, după necesitate, a numărului de bucketuri
- ▶ Din a doua categorie, Hashul extensibil modifică funcția hash astfel:
 - ▶ Generează valori într-o mulțime mare, de regulă întregi pe 32 biți
 - ▶ La un anumit moment utilizează doar un prefix (doar primii i biti) a cărui dimensiune crește sau descrește după necesitate

▶ 54

Organizarea de tip Hash extensibil

Structura generală



$$i=2, i_2 = i_3 = i, i_1 = i - 1$$

▶ 55

Hash extensibil

Implementare

- ▶ Fiecare bucket j are asociată o valoare i_j care specifică lungimea prefixului
 - ▶ Toate intrările din bucketul j au aceleasi valoare pe primii i_j biți
- ▶ Pentru a căuta o valoare cu cheia K_j :
 - ▶ Calculează $h(K_j) = X$
 - ▶ Utilizează doar primii i_j biți ai X , scanează tabela de adrese și umărește pointerul către bucket
- ▶ Pentru a insera o înregistrare cu cheia de căutare K_j :
 - ▶ Găsește bucketul j ca mai sus
 - ▶ Dacă este loc în bucket inserează înregistrarea
 - ▶ Altfel, divide bucketul și reîncearcă inserarea ->

▶ 56

Hash extensibil

Divizarea bucketului la inserare

Pentru a diviza bucketul j la inserarea unei noi valori K_j :

► Dacă $i > i_j$

1. Alocă un nou bucket z și initializează $i_z = i_j = (i_j + 1)$
2. Actualizează a doua jumătate a tablei de adrese ca să trimită către bucketul z
3. Elimină înregistrările din j și reinsereaza-le în j sau z conform prefixului
4. Recalculează adresa bucketului pentru K_j și execută inserarea

I. Dacă $i = i_j$

1. Dacă din anumite motive există o limită pentru i și aceasta este atinsă, se utilizează blocuri de exces
2. Altfel
 1. Incrementează i și dublează dimensiunea tablei de adrese
 2. Înlocuiește fiecare intrare din tabel cu alte două intrări, ambele trimițând la același bucket
 3. Recalculează adresa bucketului pentru K_j și realizează inserarea (acum $i > i_j$)

► 57

Hash extensibil

Ștergere

► Pentru a șterge o înregistrare

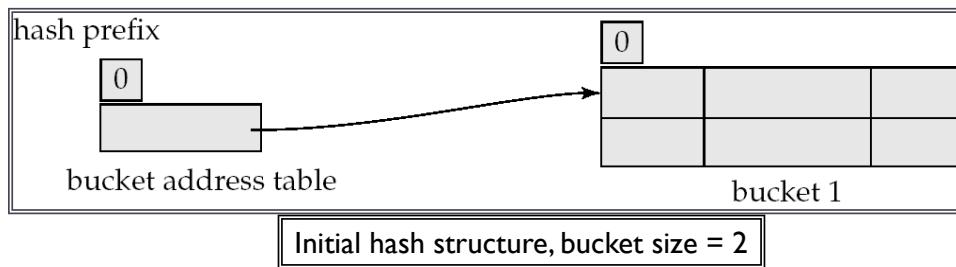
- Găsește bucketul și șterge înregistrarea din acesta
- Dacă bucketul devine gol se efectuează modificările necesare în tabela de adrese
- Bucketurile care au aceeași valoare pentru i_j și același prefix $i_j - 1$ sunt contopite
- Descrește dimensiunea (i) a tablei de adrese dacă este posibil

► 58

Hash extensibil

Exemplu*

| <i>branch_name</i> | $h(branch_name)$ |
|--------------------|---|
| Brighton | 0010 1101 1111 1011 0010 1100 0011 0000 |
| Downtown | 1010 0011 1010 0000 1100 0110 1001 1111 |
| Mianus | 1100 0111 1110 1101 1011 1111 0011 1010 |
| Perryridge | 1111 0001 0010 0100 1001 0011 0110 1101 |
| Redwood | 0011 0101 1010 0110 1100 1001 1110 1011 |
| Round Hill | 1101 1000 0011 1111 1001 1100 0000 0001 |



▶ 59

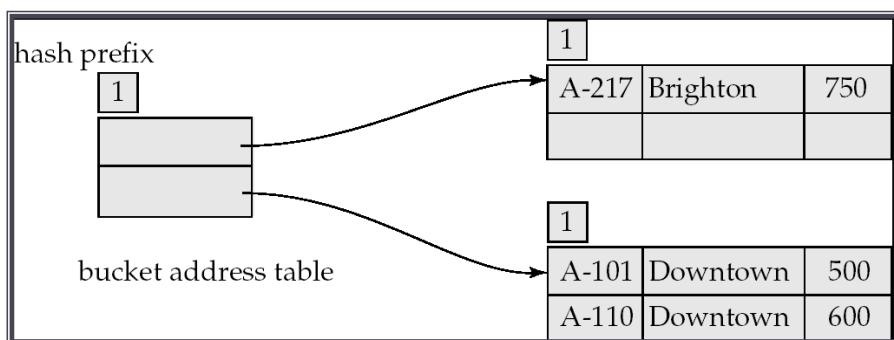
* Avi Silberschatz Henry F. Korth S. Sudarshan. "Database System Concepts".

McGraw-Hill Science/Engineering/Math; 6 edition (January 27, 2010)

Hash extensibil

Exemplu*

- ▶ După inserarea unei înregistrări Brighton și a două înregistrări Downtown



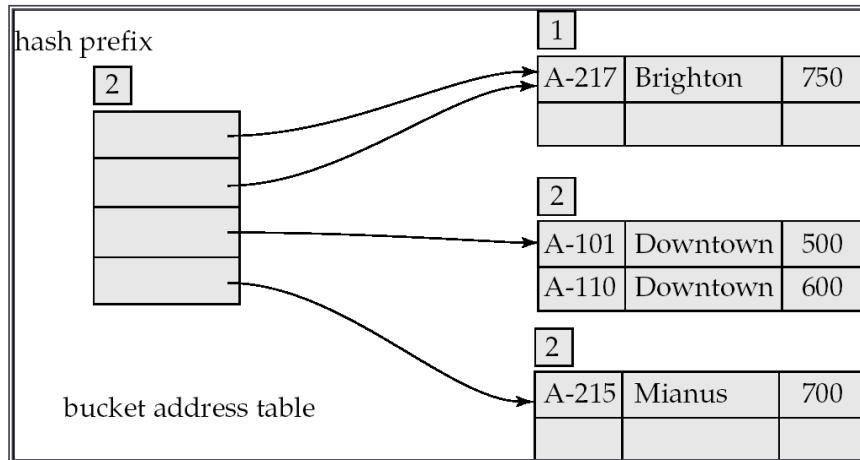
▶ 60

* Avi Silberschatz Henry F. Korth S. Sudarshan. "Database System Concepts".

McGraw-Hill Science/Engineering/Math; 6 edition (January 27, 2010)

Hash extensibil Exemplu*

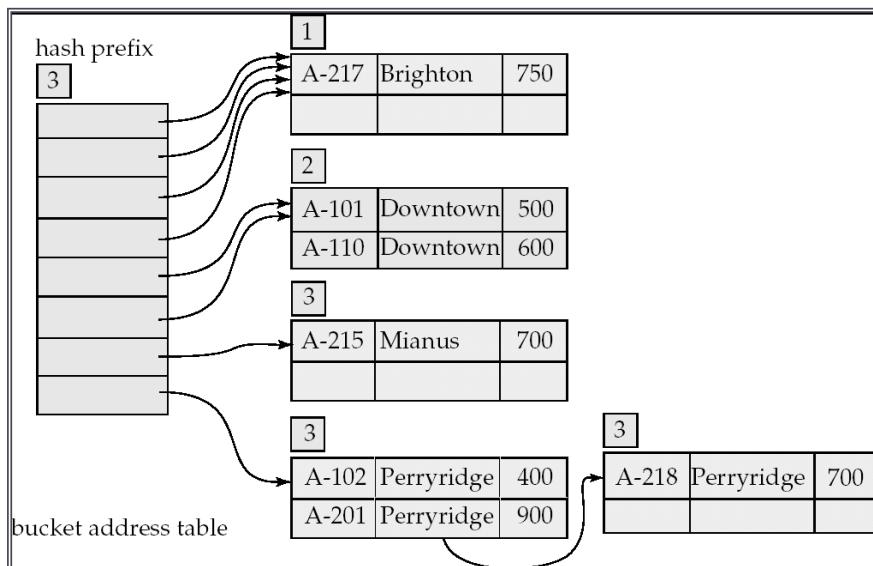
- ▶ După inserarea unei înregistrări Mianus



▶ 61 * Avi Silberschatz Henry F. Korth S. Sudarshan. "Database System Concepts". McGraw-Hill Science/Engineering/Math; 6 edition (January 27, 2010)

Hash extensibil Exemplu*

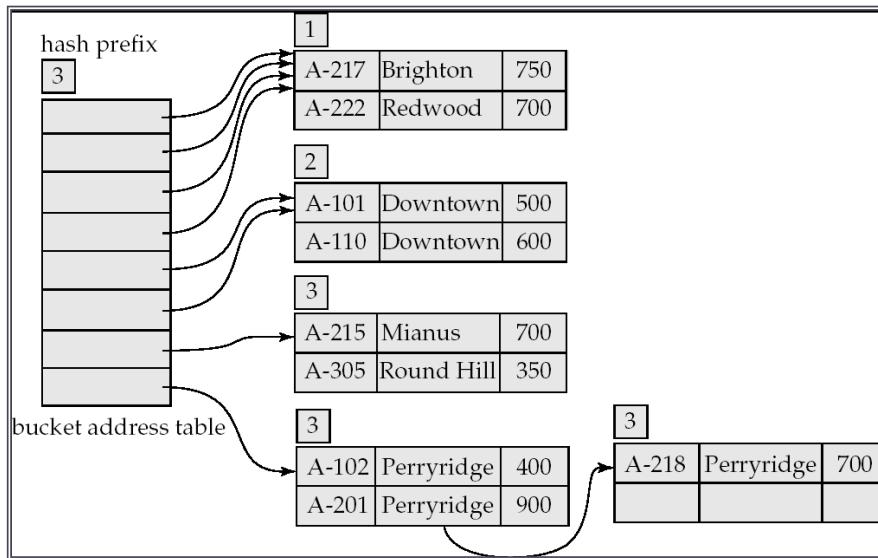
- ▶ După inserarea a trei înregistrări Perryridge



▶ 62 * Avi Silberschatz Henry F. Korth S. Sudarshan. "Database System Concepts". McGraw-Hill Science/Engineering/Math; 6 edition (January 27, 2010)

Hash extensibil Exemplu*

- ▶ După inserarea înregistrărilor Redwood și Round Hill



▶ 63

* Avi Silberschatz Henry F. Korth S. Sudarshan. "Database System Concepts".
McGraw-Hill Science/Engineering/Math; 6 edition (January 27, 2010)

Hash extensibil Eficiență

- ▶ Avantaje
 - ▶ Performanța nu scade la creșterea dimensiunii fișierului
 - ▶ Minimizează alocarea de spațiu de stocare
- ▶ Dezavantaje
 - ▶ Tabela de adrese poate deveni foarte mare
 - ▶ Soluția: utilizarea unui B⁺-arbore pentru o căutare eficientă în tabela cu adrese
 - ▶ Modificarea dimensiunii tabelei de adrese este costisitoare
- ▶ În funcție de tipul interogării:
 - ▶ Ca și hashingul static, este eficient pentru interogări punctuale dar nu și de tip interval

▶ 64

Indecsi bitmap

65

Indecsi bitmap

- ▶ Proiectați pentru a trata eficient interogri cu mai multe chei de căutare
- ▶ Aplicabili pentru atribute ce iau un număr mic de valori distincte
- ▶ Tuplele relației sunt considerate a fi numerotate
- ▶ Structura:
 - ▶ Pentru fiecare valoare a cheii un sir binar de lungime egală cu numărul de înregistrări
 - ▶ Valoarea 1 în sir indică faptul că înregistarea de la poziția dată ia valoarea la care este atașat sirul

| nume | prenume | str | loc | judet | | Arad | 0 0 0 0 0 0 1 0 |
|-----------|----------|--------------------------|-------------|---------|--|---------|-----------------|
| Alexa | Marian | Strada Florilor | Cluj Napoca | Cluj | | Bihor | 0 0 0 0 1 1 0 0 |
| Popescu | Valentin | Strada Unirii | Dej | Cluj | | Cluj | 1 1 0 0 0 0 0 0 |
| Andrici | Ioana | Bulevardul Republicii | Vaslui | Vaslui | | Suceava | 0 0 0 1 0 0 0 0 |
| Acatrinei | Marcel | | Putna | Suceava | | Vaslui | 0 0 1 0 0 0 0 1 |
| Popescu | Vasile | Bulevardul Independentei | Oradea | Bihor | | | |
| Costache | Ioan | Strada Teiului | Nucet | Bihor | | | |
| Ungureanu | Daniel | Aleea Amara | Arad | Arad | | | |
| Sandu | Maria | Strada Victoriei | Barlad | Vaslui | | | |

Indecși bitmap Observații

- ▶ Interogările cu mai multe selecții (chei de căutare) pot fi rezolvate c operatori pe biți:

- ▶ Intersecția – AND
- ▶ Reuniunea – OR
- ▶ Complementarierea – NOT

SELECT *

FROM students

WHERE county IN ('Arad', 'Cluj') AND year <> 2010;

- ▶ (Arad OR Cluj) AND NOT(2010)

| | |
|---------|-----------------|
| 2010 | 1 1 1 0 0 0 0 0 |
| | -----+----- |
| 2011 | 0 0 0 1 1 0 0 0 |
| | -----+----- |
| 2012 | 0 0 0 0 0 1 1 1 |
| | -----+----- |
| Arad | 0 0 0 0 0 0 1 0 |
| | -----+----- |
| Bihor | 0 0 0 0 1 1 0 0 |
| | -----+----- |
| Cluj | 1 1 0 0 0 0 0 0 |
| | -----+----- |
| Suceava | 0 0 0 1 0 0 0 0 |
| | -----+----- |
| Vaslui | 0 0 1 0 0 0 0 1 |

- ▶ Implementare eficientă:

- ▶ La ștergere un sir binar de existență este utilizat
- ▶ Structurile Bitmap sunt împachetate sub tipul word pe 32 sau 64 biți (operatorul AND necesită o instrucție CPU)

Definirea indecșilor în SQL

Declararea indecșilor în SQL

-
- ▶ Standardul nu reglementează mecanismele de indexare (țin de nivelul fizic), dar în practică dezvoltatorii au căzut de acord asupra sintaxei:

- ▶ Creare:

```
create index <index-name> on <relation-name>  
    (<attribute-list>)
```

E.g.: `create index c-index on student(judet)`

- ▶ Ștergere:

```
drop index <index-name>
```

- ▶ Cele mai multe SGBD-uri permit specificarea structurii pentru indexare
- ▶ Cele mai multe SGBD-uri creează implicit indecsi la declararea constrângerii unique
- ▶ Uneori indecsi sunt generați și la declararea constrângerilor referențiale

▶ 69

Indexarea în Oracle

-
- ▶ Indecși pot fi creați pe:
 - ▶ Atribute și liste de atrbute
 - ▶ Rezultatele unei funcții peste atrbute
 - ▶ Oracle offeră suport implicit pentru B⁺-arbori
 - ▶ Indecșii bitmap sunt creați cu sintaxa
`create bitmap index` <index-name> **on** <relation-name> (<attribute-list>)
 - ▶ Oracle nu oferă suport pentru indecșii hash dar implementează organizarea fișierului de date de tip hash

▶ 70

Indexarea în Oracle

Când și cum?

- ▶ Este recomandat ca crearea indecșilor să aibă loc după inserarea datelor în tabel (dar e posibil să creăm indexul în orice moment)
- ▶ Se aleg coloanele potrivite:
 - ▶ Care iau (majoritar) valori distincte
 - ▶ Pentru care selecția filtrează un număr mic de tuple dintr-un tabel de dimensiuni mari (selectivitate ridicată ~ 15%)
 - ▶ Care sunt utilizate în join
- ▶ Se alege structura de date potrivită pentru indexare:
 - ▶ Coloanele au un număr redus de valori distincte și avem condiții compuse -> bitmap
 - ▶ Interogarea este de tip interval -> B+arbori
 - ▶ Interogările punctuale sunt frecvente: -> organizarea hash
 - ▶ Selecție cu funcții -> indecși definiți peste funcții

▶ 71

Bibliografie

- ▶ Capitolul 11 în Avi Silberschatz Henry F. Korth S. Sudarshan. “Database System Concepts”. McGraw-Hill Science/Engineering/Math; 6 edition (January 27, 2010)
- ▶ Se execută scriptul `12.1create` și apoi comenziile din `12.2indexes` cu inspectarea planurilor de execuție

▶ 72

4 CURS 14: algebra relationala, estimarea costurilor

ALGEBRA RELATIONALA

π = proiectia, ce apare la *SELECT*

σ = selectia, conditiile de la *WHERE* si de la join-uri

x = produs cartezian, join cu conditie la *WHERE*

ρ = redenumirea, un alias *AS* pentru un atribut sau coloana

\bowtie = join natural

θ = conditia de la theta join \bowtie_θ (o conditie din where pentru join-uri)

REGULI DE ECHIVALENTA: slide 33-38 (in general intuitive)

Impingerea selectiilor = un select facut pe un tabel dupa join se muta pe unul din tabelele implicate in join daca aceasta conditie a selectului implica doar atribute din acel tabel (slide 39-40)

Impingerea proiectiilor = daca tabelele A si B fac join si rezulta tabela C; din tabela C se vor lua atributele strict necesare pentru un join intre tabela C si o tabela D (slide 42)

Ordonarea join-urilor = pentru join intre 3 sau mai multe tabele: se vor face 2 cate 2 si mereu se va alege combinatia de join care va produce un tabel cat mai mic (slide 43)

Estimarea costurilor

-
- ▶ l_r : dimensiunea unui tuplu din r .
 - ▶ n_r : numărul de tuple în relația r .
 - ▶ b_r : numărul de blocuri conținând tuple din r .
 - ▶ f_r : factorul de bloc al lui r — nr. de tuple din r ce intră într-un bloc
 - ▶ Dacă tuplele lui r sunt stocate împreună într-un fișier, atunci:

$$b_r = \left\lceil \frac{n_r}{f_r} \right\rceil$$

- ▶ $V(A, r)$: numărul de valori distincte care apar in r pentru atributul A ; e echivalent cu dimensiunea proiecției $\Pi_A(r)$ (pe seturi).
- ▶ Estimarea vizează numărul de tuple rezultat iar optimizarea vizează reducerea numărului și dimensiunii tuplelor cât mai devreme



BAZE DE DATE

Procesarea interogărilor

Mihaela Elena Breabă
© FII 2020-2021

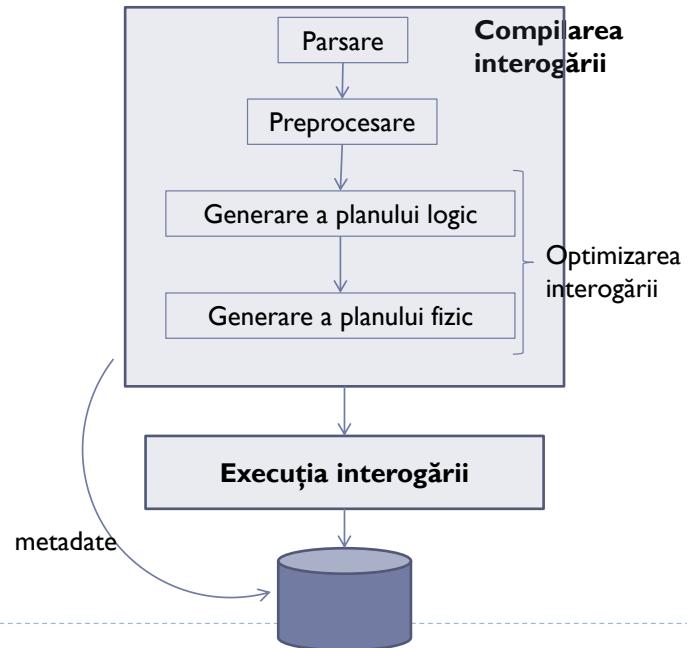
Cuprins

-
- ▶ Etapele procesării interogărilor
 - ▶ Expresii în algebra relațională
 - ▶ Operatori (revizitat)
 - ▶ Expresii
 - ▶ Echivalența expresiilor
 - ▶ Estimarea costului interogării
 - ▶ Algoritmi pentru evaluarea operatorilor/expresiilor în algebra relatională

Etapele procesării interogărilor

▶ Compilarea interogării

- ▶ Analiza sintactică
 - ▶ Parsare
 - Arbore de parsare
- ▶ Analiza semantică
 - ▶ Preprocesare si rescriere în AR
 - ▶ Selectia reprezentării algebrice
 - Plan logic
 - ▶ Selectia algoritmilor și a ordinii
 - Plan fizic



▶ 3

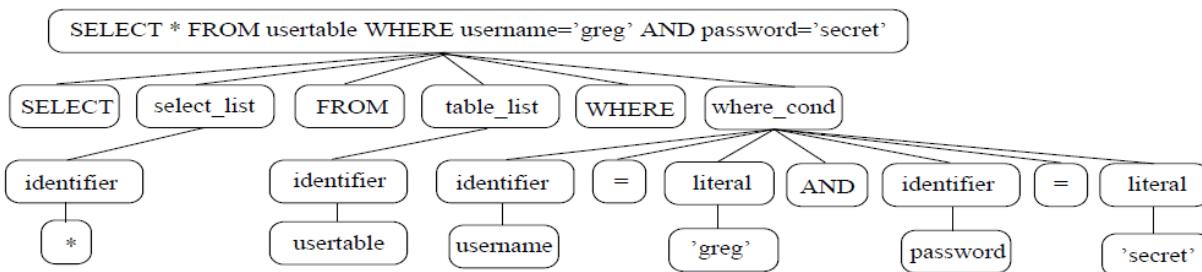
Analiza sintactică

▶ Gramatică independentă de context

```

<query> ::= <SFW> | (<query>)
<SFW> ::= SELECT <select_list> FROM <table_list> WHERE <where_cond>
<select_list> ::= <identifier>, <select_list> | <identifier>
<table_list> ::= <identifier>, <table_list> | <identifier>
...
  
```

▶ Rezultatul parsării: arbore de parsare



▶ Gramatica SQL in forma BNF: <http://savage.net.au/SQL/index.html>

▶ 4

Analiza semantică

Preprocesare

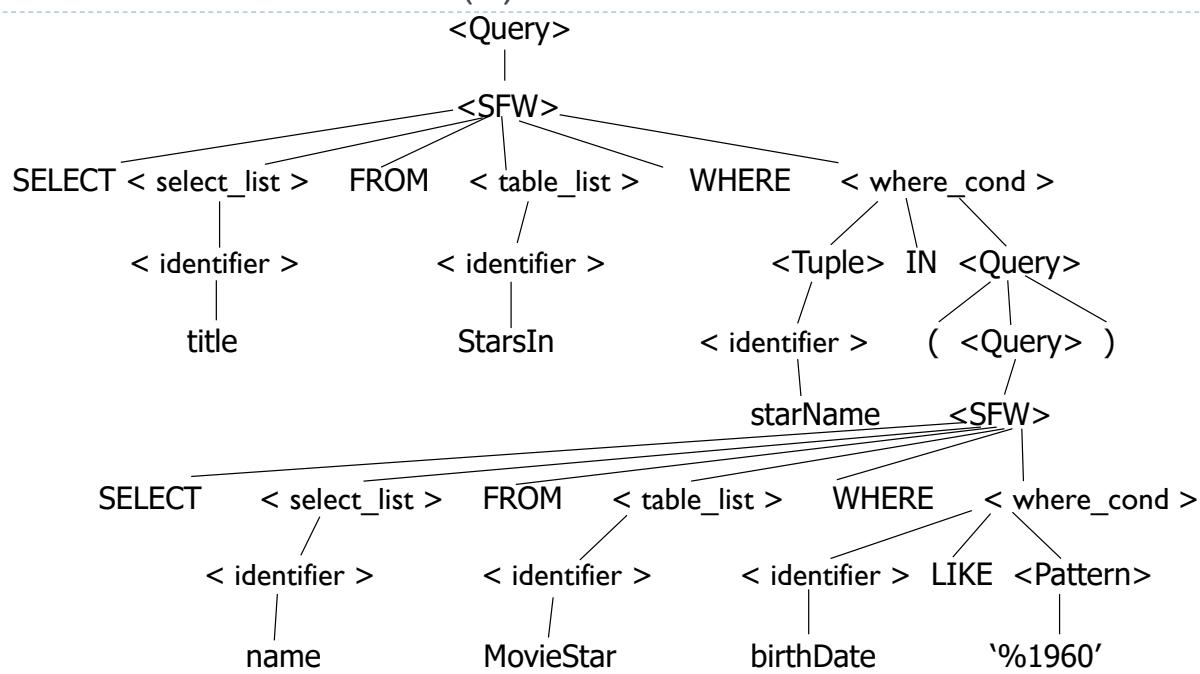
- ▶ Rescrierea apelurilor la view-uri
- ▶ Verificarea existenței relațiilor
- ▶ Verificarea existenței atributelor și a ambiguității
- ▶ Verificarea tipurilor

Dacă arborele de parsare este valid el este transformat într-o expresie cu operatori din algebra relațională

▶ 5

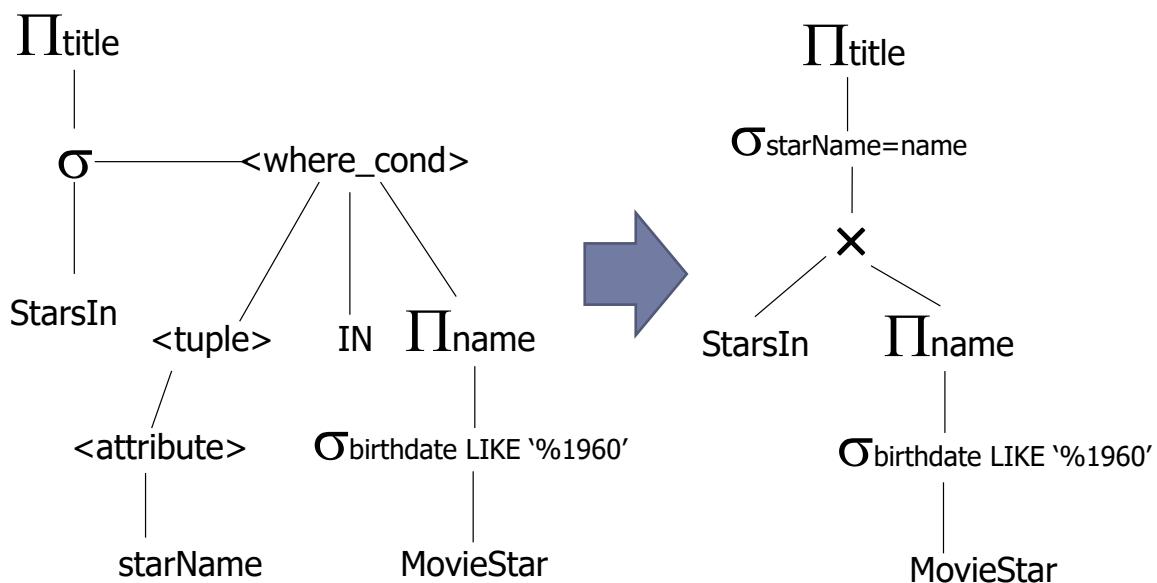
Analiza semantică

Rescriere în AR (1)



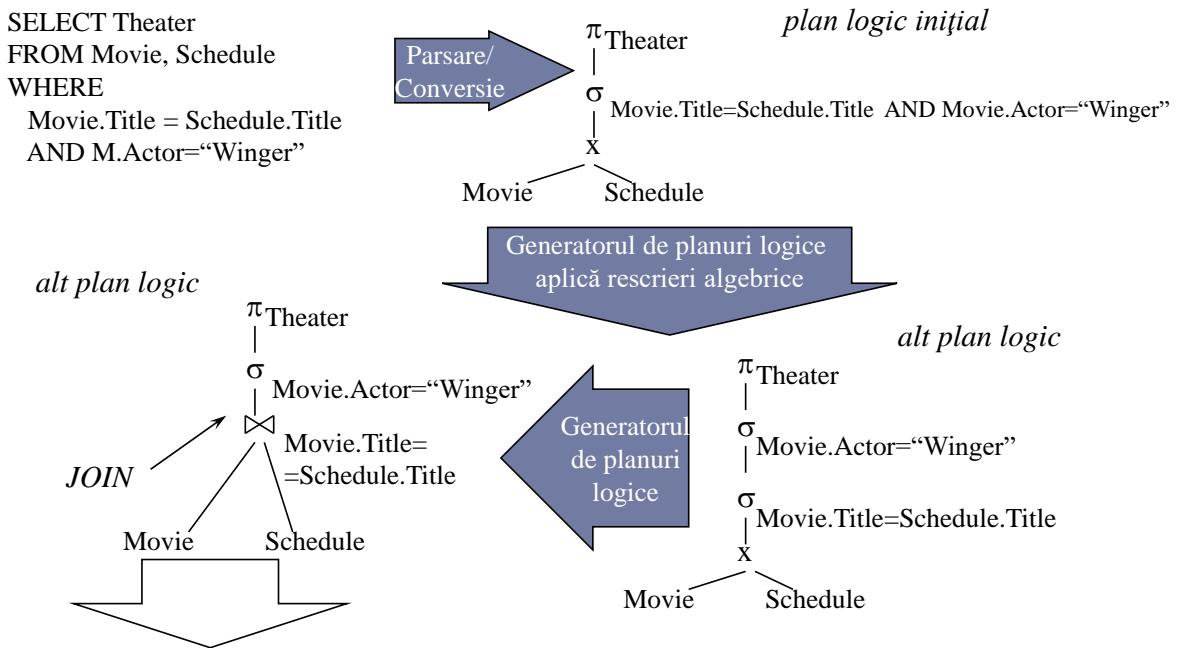
▶ 6

Analiza semantică Rescriere în AR (2)



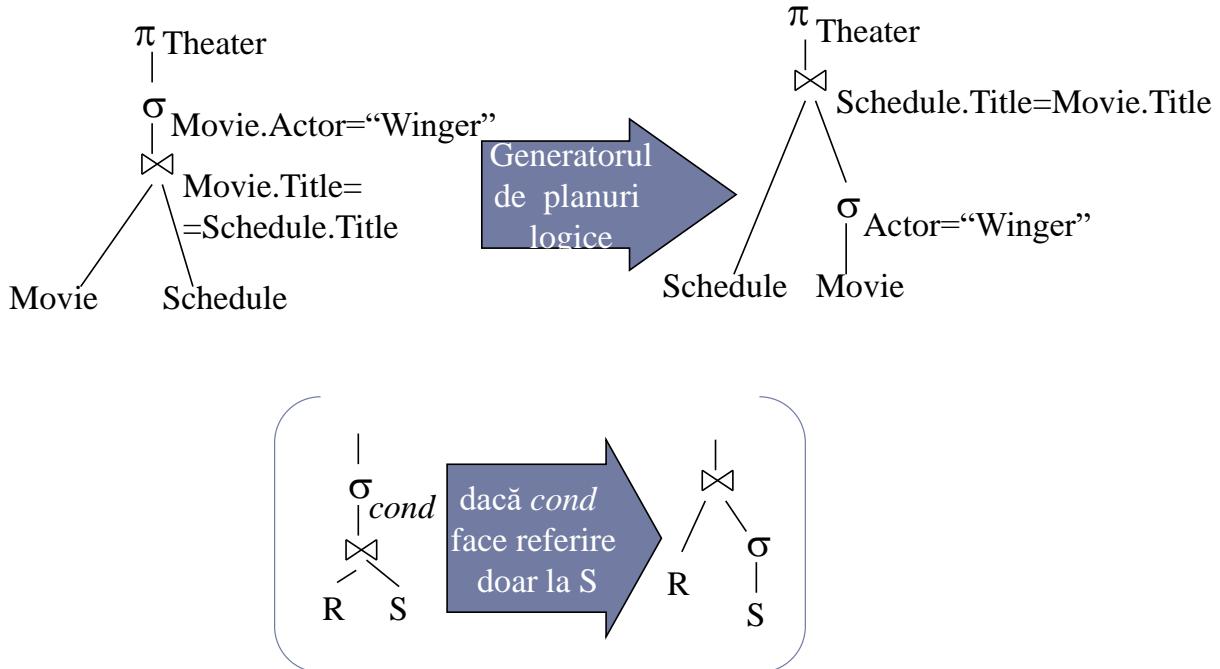
▶ 7

Analiza semantică Optimizarea planului logic



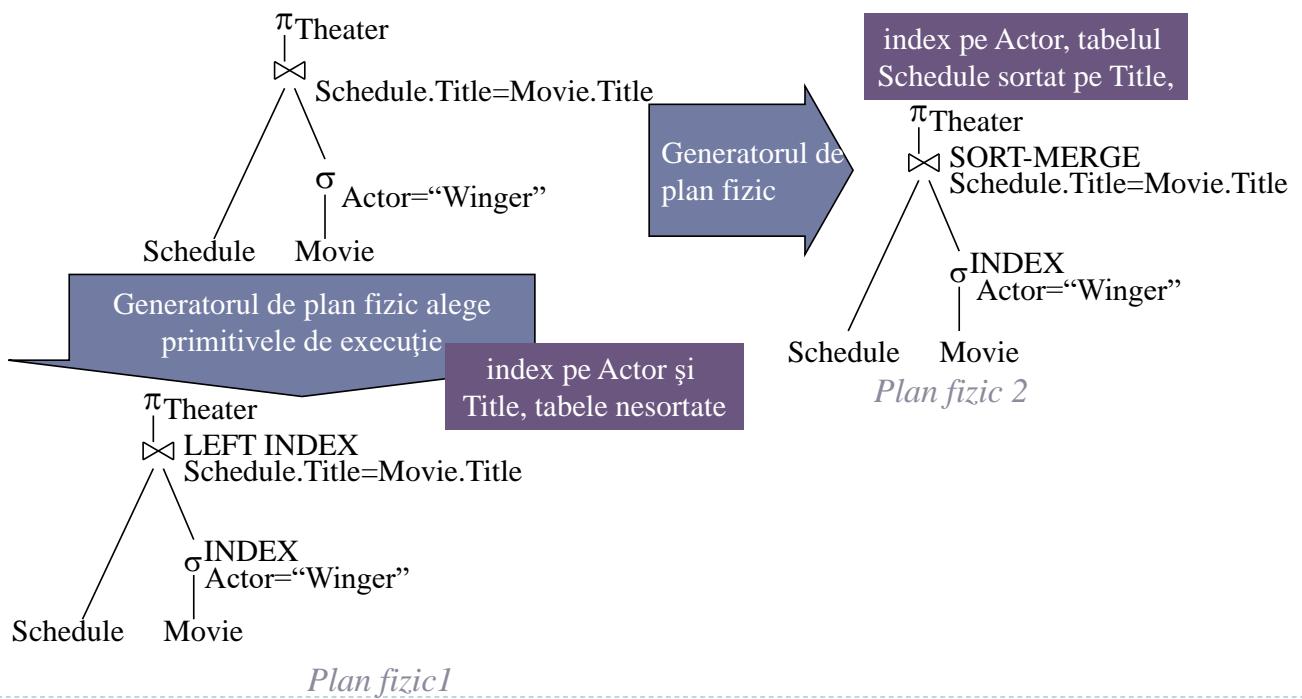
▶ 8

Analiza semantică Optimizarea planului logic



▶ 9

Analiza semantică Optimizarea planului fizic



▶ 10

Operatori în algebra relațională (revizitat)

- ▶ **Șase operatori de bază**
 - ▶ Selectia: σ
 - ▶ Proiecția: Π
 - ▶ Reuniunea: \cup
 - ▶ Diferența: $-$
 - ▶ Produsul cartezian: \times
 - ▶ Redenumirea: ρ
- ▶ **Operatorii iau ca intrare una sau două relații și generează o nouă relație**



Operatorul de selecție

- ▶ **Relația r**

| A | B | C | D |
|----------|----------|----|----|
| α | α | 1 | 7 |
| α | β | 5 | 7 |
| β | β | 12 | 3 |
| β | β | 23 | 10 |

- ▶ $\sigma_{A=B \wedge D > 5} (r)$

| A | B | C | D |
|----------|----------|----|----|
| α | α | 1 | 7 |
| β | β | 23 | 10 |



Operatorul de proiecție

- ▶ Relația r

| A | B | C |
|----------|----|---|
| α | 10 | 1 |
| α | 20 | 1 |
| β | 30 | 1 |
| β | 40 | 2 |

- ▶ $\Pi_{A,C}(r)$

| A | C | A | C |
|----------|---|----------|---|
| α | 1 | α | 1 |
| α | 1 | β | 1 |
| β | 1 | β | 2 |
| β | 2 | | |

▶ 13

Operatorul reuniune

- ▶ Relațiile r și s

| A | B | A | B |
|----------|---|----------|---|
| α | 1 | α | 2 |
| α | 2 | β | 3 |
| β | 1 | | |

r s

- ▶ $r \cup s:$

| A | B |
|----------|---|
| α | 1 |
| α | 2 |
| β | 1 |
| β | 3 |

▶ 14

Operatorul diferență

► Relațiile r și s

| | |
|----------|---|
| A | B |
| α | 1 |
| α | 2 |
| β | 1 |

r

| | |
|----------|---|
| A | B |
| α | 2 |
| β | 3 |

s

► r-s

| | |
|----------|---|
| A | B |
| α | 1 |
| β | 1 |

► 15

Produsul cartezian

► Relațiile r și s

| | |
|----------|---|
| A | B |
| α | 1 |
| β | 2 |

r

| | | |
|----------|----|---|
| C | D | E |
| α | 10 | a |
| β | 10 | a |
| β | 20 | b |

s

► $r \times s$

| A | B | C | D | E |
|----------|---|----------|----|---|
| α | 1 | α | 10 | a |
| α | 1 | β | 10 | a |
| α | 1 | β | 20 | b |
| α | 1 | γ | 10 | b |
| β | 2 | α | 10 | a |
| β | 2 | β | 10 | a |
| β | 2 | β | 20 | b |
| β | 2 | γ | 10 | b |

► 16

Operatorul de redenumire

- ▶ $\rho_x(E)$ - returnează expresia E sub numele X
- ▶ Dacă o expresie E în algebra relațională are aritate n atunci

$$\rho_{x(A_1, A_2, \dots, A_n)}(E)$$

returnează rezultatul expresiei E sub numele X și atributele redenumite în A_1, A_2, \dots, A_n

▶ 17

Compunerea operatorilor

- ▶ $\sigma_{A=C}(r \times s)$

1. $r \times s$

| A | B | C | D | E |
|----------|---|----------|----|---|
| α | 1 | α | 10 | a |
| α | 1 | β | 10 | a |
| α | 1 | β | 20 | b |
| α | 1 | γ | 10 | b |
| β | 2 | α | 10 | a |
| β | 2 | β | 10 | a |
| β | 2 | β | 20 | b |
| β | 2 | γ | 10 | b |

2. $\sigma_{A=C}(r \times s)$

| A | B | C | D | E |
|----------|---|----------|----|---|
| α | 1 | α | 10 | a |
| β | 2 | β | 10 | a |
| β | 2 | β | 20 | b |

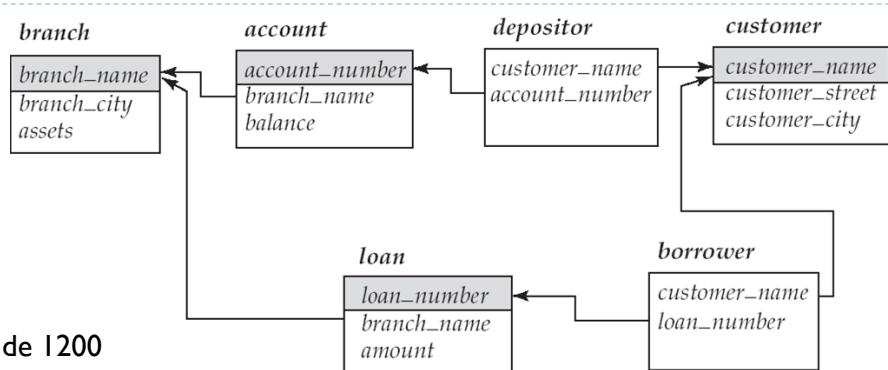
▶ 18

Expresii în algebra relațională

- ▶ Cea mai simplă expresie este o relație în baza de date
- ▶ Fie E_1 și E_2 expresii în algebra relațională; următoarele sunt expresii în algebra relatională:
 - ▶ $E_1 \cup E_2$
 - ▶ $E_1 - E_2$
 - ▶ $E_1 \times E_2$
 - ▶ $\sigma_p(E_1)$, P este un predicat peste atrbute din E_1
 - ▶ $\Pi_s(E_1)$, S este o listă de atrbute din E_1
 - ▶ $\rho_x(E_1)$, x este noul nume pentru rezultatul lui E_1

▶ 19

Exprimarea interogărilor în algebra relațională



- ▶ Împumuturile (*loan*) mai mari de 1200

$$\sigma_{amount > 1200} (loan)$$
- ▶ Numărul împrumutului (*loan_number*) pentru împrumuturi mai mari de 1200

$$\Pi_{loan_number} (\sigma_{amount > 1200} (loan))$$
- ▶ Numele clienților care au un împrumut, un depozit sau ambele la bancă

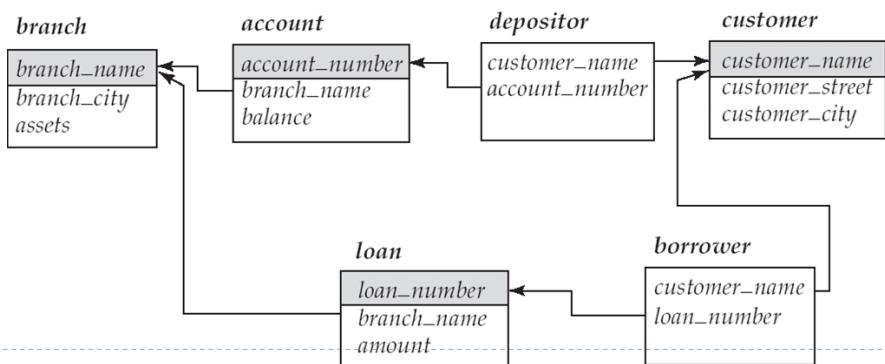
$$\Pi_{customer_name} (borrower) \cup \Pi_{customer_name} (depositor)$$

▶ 20

Interogări

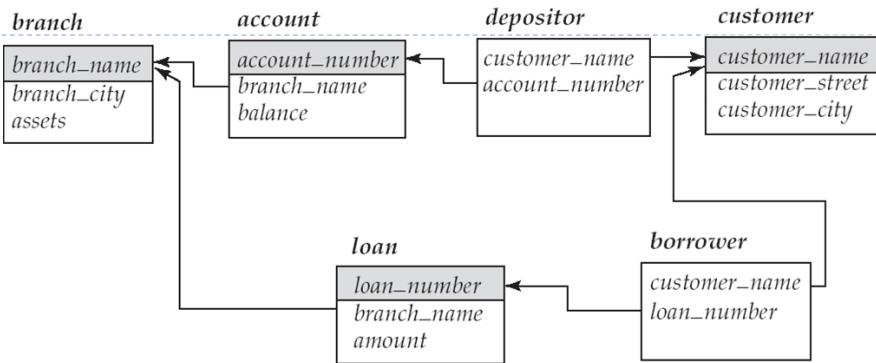
- ▶ Numele tuturor clienților care au un împrumut la filiala Perryridge

- $\Pi_{customer_name} (\sigma_{branch_name = "Perryridge"} (\sigma_{borrower.loan_number = loan.loan_number} (borrower \times loan)))$
- $\Pi_{customer_name} (\sigma_{loan.loan_number = borrower.loan_number} (\sigma_{branch_name = "Perryridge"} (loan) \times borrower))$



▶ 21

Interogări



- ▶ Numele tuturor clienților care au un împrumut la filiala Perryridge dar nu au un depozit la nici o filială a băncii

$\Pi_{customer_name} (\sigma_{branch_name = "Perryridge"} ($

$(\sigma_{borrower.loan_number = loan.loan_number} (borrower \times loan))) -$

$\Pi_{customer_name} (depositor)$

▶ 22

Operatori adiționali

- ▶ Intersecția pe mulțimi
- ▶ Joinul natural
- ▶ Agregarea
- ▶ Joinul extern
- ▶ Teta-joinul

- ▶ Toți cu excepția agregării pot fi exprimați utilizând operatori de bază

▶ 23

Intersecția pe mulțimi

- ▶ Relațiile r și s

| A | B |
|----------|---|
| α | 1 |
| α | 2 |
| β | 1 |

r

| A | B |
|----------|---|
| α | 2 |
| β | 3 |

s

- ▶ $r \cap s$

| A | B |
|----------|---|
| α | 2 |

Joinul natural

- ▶ Relațiile r și s

| A | B | C | D |
|---|---|---|---|
| α | I | α | a |
| β | 2 | γ | a |
| γ | 4 | β | b |
| α | I | γ | a |
| δ | 2 | β | b |

r

| B | D | E |
|---|---|---|
| I | a | α |
| 3 | a | β |
| I | a | γ |
| 2 | b | δ |
| 3 | b | ε |

s

- ▶ $r \bowtie s$

| A | B | C | D | E |
|---|---|---|---|---|
| α | I | α | a | α |
| α | I | α | a | γ |
| α | I | γ | a | α |
| α | I | γ | a | γ |
| δ | 2 | β | b | δ |

- ▶ $\prod_{r.A, r.B, r.C, r.D, s.E} (\sigma_{r.B = s.B \wedge r.D = s.D} (r \times s))$

▶ 25

Agregare Exemplu

- ▶ Cea mai mare balanță din tabela account

| account | | |
|----------------|--|--|
| account_number | | |
| branch_name | | |
| balance | | |

$$\prod_{balance}(account) - \prod_{account.balance}$$

$$(\sigma_{account.balance < d.balance} (account \times \rho_d (account)))$$

▶ 26

Functii de agregare și operatori

► Funcții de agregare:

- ▶ **avg**
- ▶ **min**
- ▶ **max**
- ▶ **sum**
- ▶ **count**
- ▶ **var**

► Operatorul de agregare în algebra relațională

$$g_{F_1(A_1), F_2(A_2), \dots, F_n(A_n)}(E)$$

- ▶ E – expresie în algebra relațională
- ▶ G_1, G_2, \dots, G_n o listă de atrbute de grupare (poate fi goală)
- ▶ Fiecare F_i este o funcție de agregare
- ▶ Fiecare A_i este un atrbut

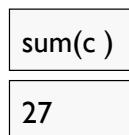
► 27

Agregare Exemplu

► relația r

| A | B | C |
|----------|----------|----|
| α | α | 7 |
| α | β | 7 |
| β | β | 3 |
| β | β | 10 |

► $g_{\text{sum}(c)}(r)$



► Care operații de agregare nu pot fi exprimate pe baza celorlalți operatori relaționali?

► 28

Join extern

relația loan

| loan_number | branch_name | amount |
|-------------|-------------|--------|
| L-170 | Downtown | 3000 |
| L-230 | Redwood | 4000 |
| L-260 | Perryridge | 1700 |

relația borrower

| customer_name | loan_number |
|---------------|-------------|
| Jones | L-170 |
| Smith | L-230 |
| Hayes | L-155 |

► $loan \bowtie borrower$ (join natural)

| loan_number | branch_name | amount | customer_name |
|-------------|-------------|--------|---------------|
| L-170 | Downtown | 3000 | Jones |
| L-230 | Redwood | 4000 | Smith |

► $loan \bowtie\leftarrow borrower$ (join extern stânga)

| loan_number | branch_name | amount | customer_name |
|-------------|-------------|--------|---------------|
| L-170 | Downtown | 3000 | Jones |
| L-230 | Redwood | 4000 | Smith |
| L-260 | Perryridge | 1700 | null |

► 29

Join extern

➤ Join extern dreapta

$loan \bowtie\leftarrow\text{borrower}$

| loan_number | branch_name | amount | customer_name |
|-------------|-------------|--------|---------------|
| L-170 | Downtown | 3000 | Jones |
| L-230 | Redwood | 4000 | Smith |
| L-155 | null | null | Hayes |

➤ Join extern plin

$loan \bowtie\leftarrow\text{borrower}$

| loan_number | branch_name | amount | customer_name |
|-------------|-------------|--------|---------------|
| L-170 | Downtown | 3000 | Jones |
| L-230 | Redwood | 4000 | Smith |
| L-260 | Perryridge | 1700 | null |
| L-155 | null | null | Hayes |

► 30

Exemple interogări

- ▶ Numele clienților care au un împrumut și un depozit la bancă

$$\Pi_{customer_name} (borrower) \cap \Pi_{customer_name} (depositor)$$

- ▶ Numele clienților care au un împrumut la bancă și suma împrumutată

$$\Pi_{customer_name, loan_number, amount} (borrower \bowtie loan)$$

- ▶ Clienții care au depozite la ambele filiale Downtown și Uptown

$$\Pi_{customer_name} (\sigma_{branch_name = "Downtown"} (depositor \bowtie account)) \cap$$
$$\Pi_{customer_name} (\sigma_{branch_name = "Uptown"} (depositor \bowtie account))$$

▶ 31

Echivalența expresiilor

- ▶ Două expresii în algebra relațională sunt echivalente dacă acestea generează același set de tuple pe orice instanță a bazei de date
 - ▶ ordinea tuplelor e irelevantă
- ▶ Obs: SQL lucrează cu multiseturi
 - ▶ în versiunea multiset a algebrei relaționale echivalența se verifică relativ la multiseturi de tuple

▶ 32

Reguli de echivalență

1. selecția pe bază de conjuncții e echivalentă cu o secvență de selecții,

$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

2. operațiile de selecție sunt comutative

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

3. într-un sir de proiecții consecutive doar ultima efectuată e necesară

$$\Pi_{L_1}(\Pi_{L_2}(\dots(\Pi_{L_n}(E))\dots)) = \Pi_{L_1}(E)$$

4. selecțiile pot fi combinate cu produsul cartezian și teta joinurile

- a. $\sigma_{\theta}(E_1 \times E_2) = E_1 \bowtie_{\theta} E_2$
- b. $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$

▶ 33

Reguli de echivalență

5. operațiile de teta-join și de join natural sunt comutative

$$E_1 \bowtie_{\theta} E_2 = E_2 \bowtie_{\theta} E_1$$

6. a) Operațiile de join natural sunt asociative

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

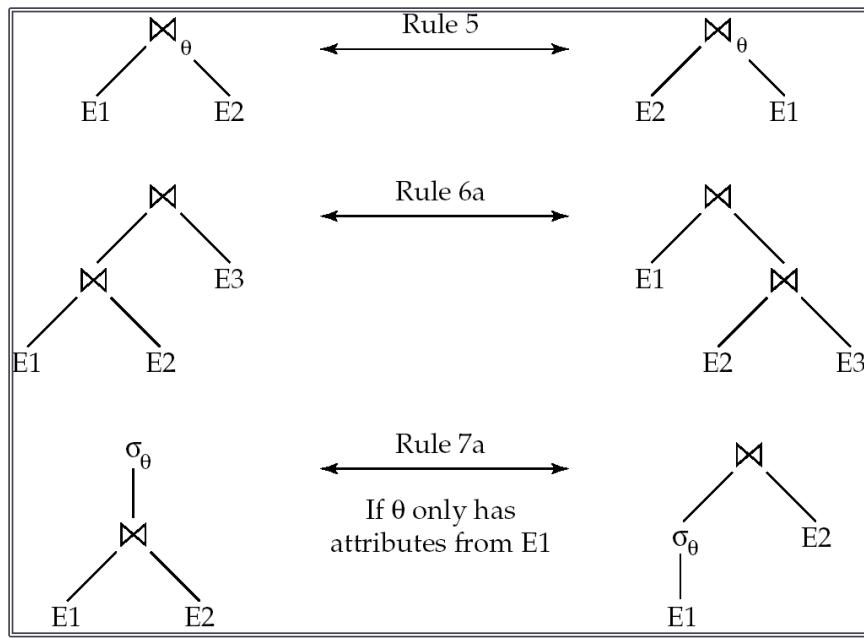
- b) Operațiile de teta-join sunt asociative astfel:

$$(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \wedge \theta_3} E_3 = E_1 \bowtie_{\theta_1 \wedge \theta_3} (E_2 \bowtie_{\theta_2} E_3)$$

unde θ_2 implică atrbute doar din E_2 și E_3

▶ 34

Reguli de echivalență



▶ 35

Reguli de echivalență

7. Distribuția selecției asupra operatorului de teta-join

- a) când θ_0 implică atribute doar din una dintre expresiile (E_1) din join:

$$\sigma_{\theta_0}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta_0}(E_1)) \bowtie_{\theta} E_2$$

- b) când θ_1 implică numai attribute din E_1 și θ_2 implică numai attribute din E_2 :

$$\sigma_{\theta_1} \wedge_{\theta_2} (E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_{\theta} (\sigma_{\theta_2}(E_2))$$

▶ 36

Reguli de echivalență

8. Distribuția proiecției asupra teta-joinului

a) dacă θ implică numai atribute din $L_1 \cup L_2$:

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = (\Pi_{L_1}(E_1)) \bowtie_\theta (\Pi_{L_2}(E_2))$$

b) Fie joinul $E_1 \bowtie_\theta E_2$

Fie L_1 și L_2 mulțimi de atribute din E_1 și respectiv E_2

Fie L_3 atribute din E_1 care sunt implicate în condiția de join θ , dar nu sunt în $L_1 \cup L_2$,

Fie L_4 atribute din E_2 care sunt implicate în condiția de join θ , dar nu sunt în $L_1 \cup L_2$

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_\theta (\Pi_{L_2 \cup L_4}(E_2)))$$

▶ 37

Reguli de echivalență

9. Operațiile de reuniune și intersecție pe mulțimi sunt comutative

$$E_1 \cup E_2 = E_2 \cup E_1$$

$$E_1 \cap E_2 = E_2 \cap E_1$$

10. Reuniunea și intersecția pe mulțimi sunt asociative

$$(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)$$

$$(E_1 \cap E_2) \cap E_3 = E_1 \cap (E_2 \cap E_3)$$

11. Selectia se distribuie peste \cup , \cap și $-$.

$$\sigma_\theta(E_1 - E_2) = \sigma_\theta(E_1) - \sigma_\theta(E_2)$$

similar pentru \cup și \cap în locul $-$

$$\sigma_\theta(E_1 - E_2) = \sigma_\theta(E_1) - E_2$$

similar pentru \cap în locul $-$, dar nu pentru \cup

12. Proiecția se distribuie peste reuniune

$$\Pi_L(E_1 \cup E_2) = (\Pi_L(E_1)) \cup (\Pi_L(E_2))$$

▶ 38

Optimizări Împingerea selecțiilor

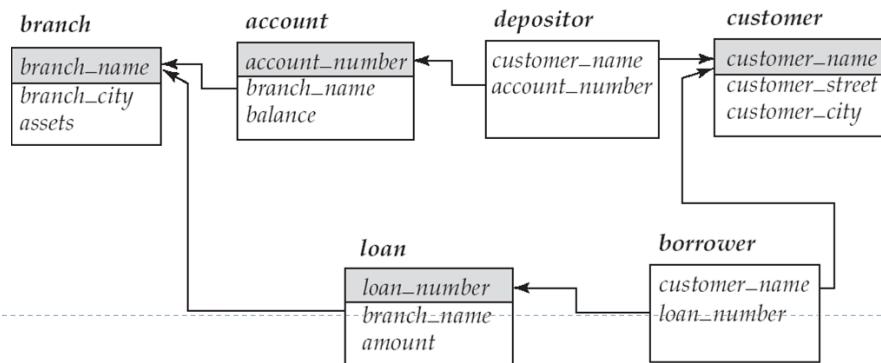
- ▶ Numele clienților care au un cont la o filială din Brooklyn

$$\Pi_{customer_name}(\sigma_{branch_city = "Brooklyn"}(branch \bowtie (account \bowtie depositor)))$$

- ▶ Pe baza regulii 7a

$$\Pi_{customer_name}((\sigma_{branch_city = "Brooklyn"}(branch)) \bowtie (account \bowtie depositor))$$

- ▶ Realizarea selecției în primele etape reduce dimensiunea relației care participă în join



▶ 39

Optimizări Împingerea selecțiilor

- ▶ Numele clienților cu un cont la o filială din Brooklyn care are balanță peste 1000

$$\Pi_{customer_name}(\sigma_{branch_city = "Brooklyn" \wedge balance > 1000}(branch \bowtie (account \bowtie depositor)))$$

- ▶ Regula 6a (asociativitatea la join)

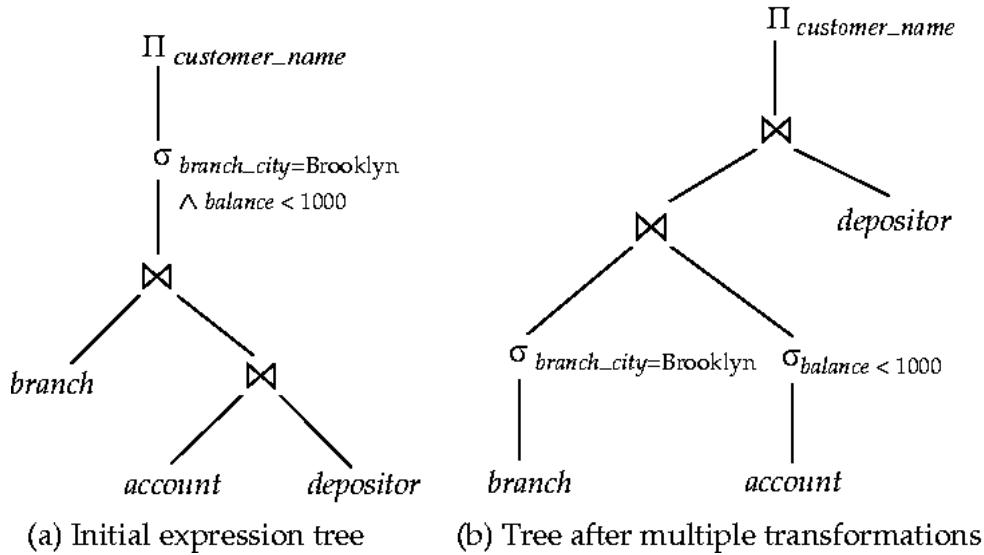
$$\Pi_{customer_name}((\sigma_{branch_city = "Brooklyn" \wedge balance > 1000}(branch \bowtie account)) \bowtie depositor)$$

- ▶ A doua formă furnizează oportunitatea de a efectua selecția devreme

$$\sigma_{branch_city = "Brooklyn"}(branch) \bowtie \sigma_{balance > 1000}(account)$$

▶ 40

Vizualizare sub formă de arbori



▶ 41

Optimizări Împingerea proiecțiilor

$$\Pi_{customer_name}((\sigma_{branch_city = "Brooklyn"} (branch) \bowtie account) \bowtie depositor)$$

- ▶ Eliminarea atributelor care nu sunt necesare din rezultatele intermediiare

$$\begin{aligned} \Pi_{customer_name} ((\\ \Pi_{account_number} (\sigma_{branch_city = "Brooklyn"} (branch) \bowtie account) \\ \bowtie depositor) \end{aligned}$$

- ▶ Realizarea devreme a proiecției reduce dimensiunea relațiilor din join

▶ 42

Optimizări Ordonarea joinurilor

- ▶ Pentru orice relații r_1, r_2 , și r_3 ,
$$(r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie (r_2 \bowtie r_3)$$
- ▶ Dacă $r_2 \bowtie r_3$ are dimensiuni mari și $r_1 \bowtie r_2$ e de dimensiuni mai mici, alegem
$$(r_1 \bowtie r_2) \bowtie r_3$$
- ▶ Exemplu

$$\Pi_{customer_name} ((\sigma_{branch_city} = "Brooklyn" (branch)) \bowtie (account \bowtie depositor))$$

Numai un mic procent din clienți au conturi în filiale din Brooklyn deci e mai bine să se execute mai întâi

$\sigma_{branch_city} = "Brooklyn" (branch) \bowtie account$

- ▶ Pentru n relații există $(2(n - 1))/(n - 1)!$ ordonări diferite pentru join.
- ▶ $n = 7 \rightarrow 665280$, $n = 10 \rightarrow 176$ miliarde!
- ▶ Pentru a reduce numărul de ordonări supuse evaluării se utilizează programarea dinamică

▶ 43

Estimarea costurilor

- ▶ I_r : dimensiunea unui tuplu din r .
- ▶ n_r : numărul de tuple în relația r .
- ▶ b_r : numărul de blocuri conținând tuple din r .
- ▶ f_r : factorul de bloc al lui r — nr. de tuple din r ce intră într-un bloc
- ▶ Dacă tuplele lui r sunt stocate împreună într-un fișier, atunci:

$$b_r = \left\lceil \frac{n_r}{f_r} \right\rceil$$

- ▶ $V(A, r)$: numărul de valori distincte care apar în r pentru atributul A ; e echivalent cu dimensiunea proiecției $\Pi_A(r)$ (pe seturi).
- ▶ Estimarea vizează numărul de tuple rezultat iar optimizarea vizează reducerea numărului și dimensiunii tuplelor cât mai devreme

▶ 44

Estimarea dimensiunii selecției

- ▶ $\sigma_{A=v}(r)$
 - ▶ $n_r / V(A,r)$: numărul de înregistrări ce satisfac selecția
 - ▶ pentru atribut cheie: I
- ▶ $\sigma_{A \leq v}(r)$ (cazul $\sigma_{A \geq v}(r)$ este simetric)
 - ▶ dacă sunt disponibile $\min(A,r)$ și $\max(A,r)$
 - ▶ 0 dacă $v < \min(A,r)$
 - ▶ $n_r \cdot \frac{v - \min(A,r)}{\max(A,r) - \min(A,r)}$ altfel
 - ▶ dacă sunt disponibile histograme se poate rafina estimarea anterioară
 - ▶ în lipsa oricărei informații statistice dimensiunea se consideră a fi $n_r / 2$.

▶ 45

Estimarea dimensiunii selecțiilor complexe

- ▶ Selectivitatea unei condiții θ_i este probabilitatea ca un tuplu în relația r să satisfacă θ_i
 - ▶ dacă numărul de tuple ce satisfac θ_i este s_i , selectivitatea e s_i / n_r
- ▶ Conjuncția (în ipoteza independenței)

$$\sigma_{\theta_1 \wedge \theta_2 \wedge \dots \wedge \theta_n}(r): \quad n_r * \frac{s_1 * s_2 * \dots * s_n}{n_r^n}$$

▶ Disjuncția

$$\sigma_{\theta_1 \vee \theta_2 \vee \dots \vee \theta_n}(r): \quad n_r * \left(1 - \left(1 - \frac{s_1}{n_r}\right) * \left(1 - \frac{s_2}{n_r}\right) * \dots * \left(1 - \frac{s_n}{n_r}\right)\right)$$

▶ Negația

$$\sigma_{\neg \theta}(r): \quad n_r - \text{size}(\sigma_\theta(r))$$

▶ 46

Estimarea dimensiunii joinului

-
- ▶ pentru produsul cartezian $r \times s: n_r * n_s$ tuple, fiecare tuplu ocupă $s_r + s_s$ octeți,
 - ▶ pentru $r \bowtie s$
 - ▶ $R \cap S = \emptyset: n_r * n_s$
 - ▶ $R \cap S$ este o (super)cheie pentru $R: \leq n_s$
 - ▶ $R \cap S = \{A\}$ nu e cheie pentru R sau S :
 - ▶ minimul este considerat de acuratețe mai mare
 - ▶ dacă sunt disponibile histograme se calculează formulele anterioare pe fiecare celulă pentru cele două relații

▶ 47

Estimarea dimensiunii pentru alte operații

-
- ▶ Proiecția $\Pi_A(r) : V(A,r)$
 - ▶ Agregarea: ${}_A g_F(r) : V(A,r)$
 - ▶ Operații pe mulțimi
 - ▶ $r \cup s : n_r + n_s$
 - ▶ $r \cap s : \min(n_r, n_s)$
 - ▶ $r - s : n_r$
 - ▶ Join extern
 - ▶ $r \bowtie s: \dim(r \bowtie s) + n_r$
 - ▶ $r \bowtie s = \dim(r \bowtie s) + n_r + n_s$
 - ▶ $\sigma_{\theta_1}(r) \cap \sigma_{\theta_2}(r)$ echivalent cu $\sigma_{\theta_1} \sigma_{\theta_2}(r)$
 - ▶ Estimatorii furnizează în general margini superioare

▶ 48

Optimizarea planului fizic

▶ 49

Estimarea costului la nivelul planului fizic

-
- ▶ Costul e în general măsurat ca durata de timp necesară pentru returnarea răspunsului
 - ▶ Accesul la disc este costul predominant
 - ▶ Numărul de căutări * t_S (timpul pentru o localizare a unui bloc pe disc)
 - ▶ Numărul de blocuri citite/scrise * t_T (timpul de transfer)
 - ▶ costul CPU e ignorat pentru simplitate
 - ▶ Costul pentru transferul a b blocuri plus S căutări pe disc:
$$b * t_T + S * t_S$$

▶ 50

Algoritmi pentru selectie

- ▶ Căutare liniară (full scan)
 - ▶ cost: $b_r * t_T + t_S$
 - ▶ dacă selecția e pe un atribut cheie, costul estimativ: $b_r/2 * t_T + t_S$
 - ▶ poate fi aplicată indiferent de condiția de selecție, ordonarea înregistrărilor în fișier, existența indecsilor
- ▶ Căutarea binară
 - ▶ aplicabilă pentru condiții de selecție de tip egalitate pe atributul după care e ordonat fișierul
 - ▶ costul găsirii primului tuplu ce satisface condiția: $\lceil \log_2(b_r) \rceil * (t_T + t_S)$; dacă există mai multe tuple se adaugă timpul de transfer al blocurilor
- ▶ Scanarea indexului – condiția de selecție = cheia de căutare a indexului
 - ▶ index primar pe cheie candidat, egalitate: $(h_i + 1) * (t_T + t_S)$
 - ▶ index primar pe non-cheie, egalitate: $h_i * (t_T + t_S) + t_S + t_T * b$
 - ▶ index secundar, egalitate, n tuple returnate: $(h_i + n) * (t_T + t_S)$
 - ▶ index primar, comparație: $h_i * (t_T + t_S) + t_S + t_T * b$

▶ 51

Algoritmi pentru selecții complexe

- ▶ Conjunction: $\sigma_{\theta_1} \wedge \theta_2 \wedge \dots \wedge \theta_n(r)$
 - ▶ utilizarea unui index pentru θ_1 și verificarea celorlalte condiții pe măsură ce tuplele sunt aduse în memorie
 - ▶ utilizarea unui index multi-cheie
 - ▶ intersecția identificatorilor (pointerilor la înregistrări) returnați de indecsii asociați condițiilor urmată de citirea înregistrărilor
- ▶ Disjunction: $\sigma_{\theta_1} \vee \theta_2 \vee \dots \vee \theta_n(r)$
 - ▶ reunirea identificatorilor

▶ 52

Algoritmi pentru join

- ▶ Algoritmi:
 - ▶ join cu bucle imbricate (nested-loop join)
 - ▶ join indexat cu bucle imbricate
 - ▶ join cu fuziune (merge join)
 - ▶ join hash
- ▶ Alegerea se face pe baza estimării costului
- ▶ Sunt necesare estimări realizate la nivelul planului logic

▶ 53

Join cu bucle imbricate

- ▶ Pentru teta-join: $r \bowtie_{\theta} s$

```
for each tuplu  $t_r$  in  $r$  do begin
    for each tuplu  $t_s$  in  $s$  do begin
        if  $(t_r, t_s)$  satisfac  $\theta$ 
            adaugă  $t_r \cdot t_s$  la rezultat
    end
end
```
- ▶ relația interioară – s
- ▶ relația exterioară – r
- ▶ Costul estimat: $(n_r * b_s + b_r) * t_T + (n_r + b_r) * t_s$

▶ 54

Join indexat cu bucle imbricate

- ▶ Căutările în index pot înlocui scanarea fișierelor dacă:
 - ▶ e un echi-join sau join natural
 - ▶ există un index pe atributul de join al relației interioare
- ▶ pentru fiecare tuplu t_r în relația exterioară r se utilizează indexul pentru localizarea tuplelor din s care satisfac condiția de join cu uplul t_r
- ▶ costul: $b_r (t_T + t_S) + n_r * c$
 - ▶ c este costul parcurgerii indexului pentru a returna tuple din s care se potrivesc pentru un tuplu din r (echivalent cu selecția pe s cu condiția de join)
 - ▶ dacă există indecsi pentru ambele relații, relația cu mai puține tuple va fi preferată drept relație exterioară în join
- ▶ Exemplu
 - ▶ depositor \bowtie customer, depositor relație exterioară
 - ▶ customer are asociat un index primar de tip B⁺-arbore pe atributul de join *customer-name*, cu 20 intrări pe nod
 - ▶ *customer*: 10,000 tuple ($f=25$), *depositor*: 5000 tuple ($f=50$)
 - ▶ costul: $100 + 5000 * 5 = 25,100$ blocuri transferate și căutări (corespondentul în joinul neindexat: 2,000, 100 blocuri transferate și = 5100 căutări)

▶ 55

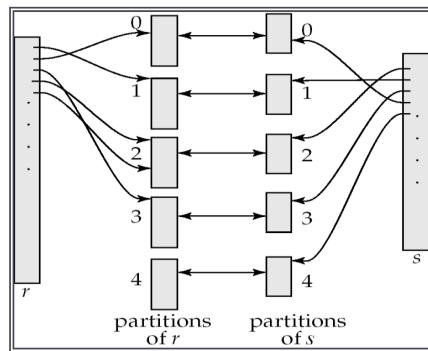
Join cu fuziune

- ▶ Algoritm
 1. se sortează ambele relații în funcție de atributul de join
 2. are loc fuziunea relațiilor
- ▶ Poate fi utilizat doar pentru echi-joinuri
- ▶ Costul:
 - ▶ $b_r + b_s$ blocuri transferate
 - ▶ + costul sortării relațiilor
- ▶ Join cu fuziune hibrid: o relație este sortată iar a doua are un index secundar pe atributul de join de tip B⁺-arbore
 - ▶ relația sortată fuzionează cu intrările de pe nivelul frunză al arborelui

▶ 56

Join hash

- ▶ aplicabil pentru echi-join
- ▶ o funcție hash h ce ia la intrare atributele de join partăționează tuplele ambelor relații în blocuri ce încap în memorie
 - ▶ r_1, r_2, \dots, r_n
 - ▶ s_1, s_2, \dots, s_n
- ▶ tuplele din r_i sunt comparate doar cu tuplele din s_i



▶ 57

Joinuri complexe

- ▶ Condiție de tip conjuncție: $r \bowtie_{\theta_1 \wedge \theta_2 \wedge \dots \wedge \theta_n} s$
 - ▶ bucle imbricate cu verificarea tuturor condițiilor sau
 - ▶ se calculează un join mai simplu $r \bowtie_{\theta_i} s$ și se realizează selecția pentru celelalte condiții
- ▶ Condiție de tip disjuncție: $r \bowtie_{\theta_1 \vee \theta_2 \vee \dots \vee \theta_n} s$
 - ▶ bucle imbricate cu verificarea condițiilor sau
 - ▶ calculul reunii joinurilor individuale (aplicabil numai versiunii set a reunii)
 $(r \bowtie_{\theta_1} s) \cup (r \bowtie_{\theta_2} s) \cup \dots \cup (r \bowtie_{\theta_n} s)$

▶ 58

Eliminarea duplielor

- ▶ Sortarea tuplelor sau hashing
- ▶ Fiindca e costisitoare, SGBD-urile nu elimina duplicatele decat la cerere

▶ 59

Evaluare expresiilor

- ▶ Alternative:
 - ▶ Materializarea: (sub)expresiile sunt materializate sub forma unor relații stocate pe disc pentru a fi date ca intrare operatorilor de pe nivele superioare
 - ▶ Pipelining: tuple sunt date ca intrare operațiilor de pe nivele superioare imediat ce acestea sunt returnate în timpul procesării unui operator
 - nu e întotdeauna posibil (sortare, join hash)
 - ▶ varianta la cerere: nivelul superior solicită noi tuple
 - ▶ varianta la producător: operatorul scrie în buffer tuple iar părintele scoate din buffer (la umplerea bufferului există timpi de așteptare)

▶ 60

Planuri de executie Oracle

- ▶ Inregistreaza planul:

EXPLAIN PLAN

```
[SET STATEMENT_ID = <id>]  
[INTO <table_name>]  
FOR <sql_statement>;
```

- ▶ Pentru orice comanda DML

- ▶ Vizualizeaza planul:

```
SELECT * FROM table(dbms_xplan.display);
```

sau

```
select * from plan_table [where statement_id = <id>];
```

<http://www.oracle.com/technetwork/database/bi-datawarehousing/twp-explain-the-explain-plan-052011-393674.pdf>

▶ 61

Planuri de executie Oracle-statistici

- ▶ Table statistics

- ▶ Number of rows
- ▶ Number of blocks
- ▶ Average row length

- ▶ Column statistics

- ▶ Number of distinct values (NDV) in column
- ▶ Number of nulls in column
- ▶ Data distribution (histogram)

- ▶ Index statistics

- ▶ Number of leaf blocks
- ▶ Levels
- ▶ Clustering factor

- ▶ System statistics

- ▶ I/O performance and utilization
- ▶ CPU performance and utilization

▶ 62

Planuri de executie Colectarea statisticilor

- ▶ Proceduri Oracle din pachetul DBMS_STATS:
 - ▶ GATHER_INDEX_STATS
 - ▶ Index statistics
 - ▶ GATHER_TABLE_STATS
 - ▶ Table, column, and index statistics
 - ▶ GATHER_SCHEMA_STATS
 - ▶ Statistics for all objects in a schema
 - ▶ GATHER_DATABASE_STATS
 - ▶ Statistics for all objects in a database
 - ▶ GATHER_SYSTEM_STATS
 - ▶ CPU and I/O statistics for the system
- ▶ http://docs.oracle.com/cd/B10500_01/server.920/a96533/stats.htm

▶ 63

Planuri de executie Hints

- ▶ In cadrul unei comenzi DML este posibil a instrui optimizatorul Oracle asupra planului de executie:

```
SELECT /*+ USE_MERGE(employees departments) */ * FROM employees, departments WHERE employees.department_id =  
departments.department_id;
```

http://docs.oracle.com/cd/B19306_01/server.102/b14200/sql_elements006.htm

▶ 64

Bibliografie

-
- ▶ Capitolele 13 și 14 în *Avi Silberschatz Henry F. Korth S. Sudarshan. “Database System Concepts”*. McGraw-Hill Science/Engineering/Math; 4th edition

BD partea 1

—

February 2021

Cuprins

| | | |
|---|--|----|
| 1 | How to use: | 2 |
| 2 | CURS 1 *istoric* | 2 |
| 3 | CURS 2 | 32 |
| 4 | CURS 3 | 52 |
| 5 | CURS 4 | 62 |
| 6 | CURS 5-6: NF-1, NF-2, BCNF, JOIN-uri, 4-NF | 72 |

1 How to use:

La cursul 5, la NF-uri sunt notate niste pagini in bold; CTRL + F introduceti numarul paginii pentru a gasi pagina respectiva si se gasesc exemple din curs + explicatiile profului.

2 CURS 1 *istoric*

Baze de date Introducere

Nicolae-Cosmin Vârlan

October 1, 2020

Prima bază de date:





© Corbis

Cal (15.000 î.H. - 13.000 î.H.) - Lascaux, Franța

A set of small, semi-transparent navigation icons used for navigating through the presentation slides.

3 / 57

Nicolae-Cosmin Vârlan

Baze de date Introducere

Istoric și Motivatie

Sisteme de gestiune a bazelor de date (SGBD)

Modele de baze de date

Modelul relațional



© Corbis

4

Câte animale am (7.000 î.H.) - Rio Pinturas, Argentina

A set of small, semi-transparent navigation icons used for navigating through the presentation slides.

4 / 57



Bazorelief (3200 î.H. - 400) - Egipt

A set of small, light-blue navigation icons typically used in presentation software like Beamer for navigating between slides and sections.

5 / 57

Nicolae-Cosmin Vârlan

Baze de date Introducere

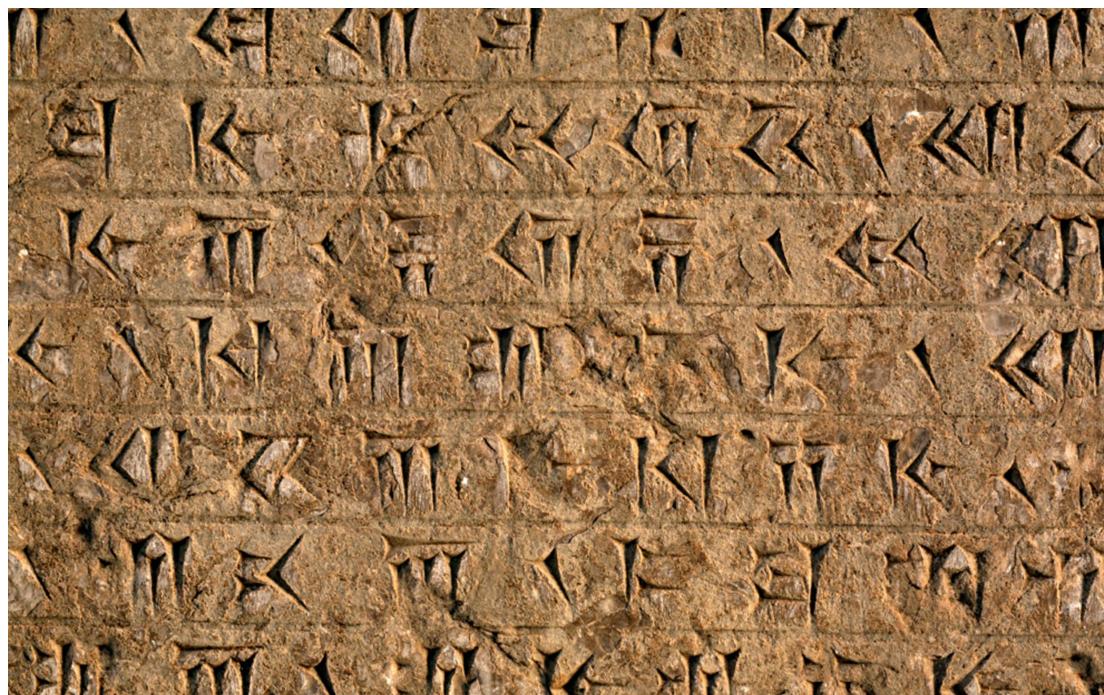
Istoric și Motivatie

Sisteme de gestiune a bazelor de date (SGBD)

Modele de baze de date

Modelul relațional

Baze de date - Introducere

Scrierea cuneiformă (600 î.H. - 300 î.H.) - Persia⁵

A set of small, light-blue navigation icons typically used in presentation software like Beamer for navigating between slides and sections.

6 / 57

Nicolae-Cosmin Vârlan

Baze de date Introducere



Mai recent...



Stocarea datelor: foarte interesant este faptul că datele sunt "citite" în paralel - stocarea melodii pare foarte tentantă.



A set of small, light-blue navigation icons typically used in presentation software for navigating through slides.

9 / 57

Nicolae-Cosmin Vârlan

Baze de date Introducere

Istoric și Motivatie

Sisteme de gestiune a bazelor de date (SGBD)

Modele de baze de date

Modelul relațional



Ideea de cilindru a fost păstrată, dar datele au fost scrise mult mai dens. Se observă și citarea "serială"

A set of small, light-blue navigation icons typically used in presentation software for navigating through slides.

10 / 57

Nicolae-Cosmin Vârlan

Baze de date Introducere



Citirea "serială", de această dată pe un disc.



Discurile pe care le folosim azi au o densitate foarte mare și pot fi "citite" cu aprox. 200Mb/s (atenție la unele site-uri care zic că:)

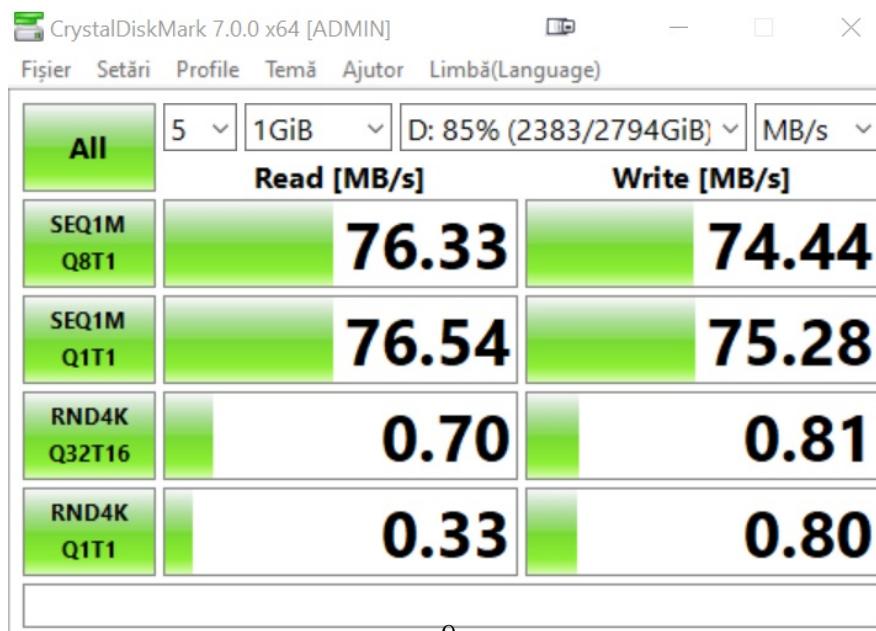
Specificatii

CARACTERISTICI GENERALE

| | |
|-----------------------|----------|
| Capacitate | 2 TB |
| Viteza de rotatie | 7200 rpm |
| Buffer | 64 MB |
| Interfata | SATA III |
| Rata de transfer SATA | 600 MB/s |
| Format | 3.5 inch |

De fapt ei nu zic nimic greșit, portul SATA III chiar merge cu 600MB/s, dar nu și HDD-urile clasice (cu platane).
 [imagine preluată de pe situl eMAG]

Realitatea este puțin diferită (aceasta este viteza HDD-ului meu de acasă):





Este adevarat că SSD-urile ajung la viteze mai mari... DAR... 15 / 57

Nicolae-Cosmin Vârlan

Baze de date Introducere

Istoric și Motivatie

Sisteme de gestiune a bazelor de date (SGBD)

Modele de baze de date

Modelul relațional

Evident, de această dată se prezintă (și) realitatea despre SSD:

Specificatii

CARACTERISTICI GENERALE

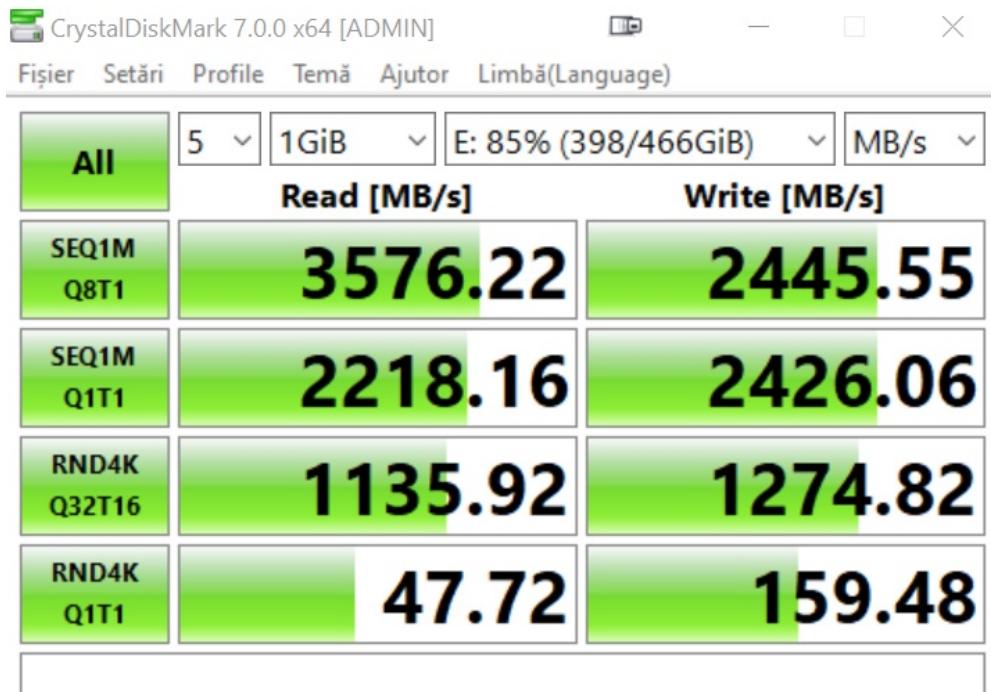
| | |
|------------------------------------|--------|
| Form Factor | 2.5" |
| Capacitate | 500 GB |
| Interfata | SATA 3 |
| Rata de transfer la citire (MB/s) | 550 |
| Rata de transfer la scriere (MB/s) | 520 |
| Rata de transfer SATA (MB/s) | 600 |

Se pare că viteza SATA III este prezentată ca o limitare (ceva de genu' "de asta nu se poate mai mult")

Cu toate acestea, există SSDuri pe interfețe mai “noi” decât SATA. Este cazul SSD-urilor M.2 care folosesc standardul NVMe (Non Volatile Memory Express)



Primii care au reușit să atingă viteze de 3600MB/s la citire sunt cei de la Samsung.



Ați făcut aplicații care să utilizeze o bază de date ? [ce limbaj ?]



De ce credeți că sunt importante ?

De ce nu aş scrie într-un fișier informațiile ?



De ce nu aş scrie într-un fișier informațiile ?

Iată câteva posibile răspunsuri:

- ▶ separarea și izolarea datelor (no joins);
- ▶ posibilitatea duplicării datelor în mai multe fișiere (data integrity ?);
- ▶ inter-dependența datelor (chei primare / străine);
- ▶ formatele incompatibile ale diferitelor aplicații (logica stocării datelor este hardcodată în aplicație și datele sunt dependente de aplicație);
- ▶ interogări fixe specifice fiecărui fișier (nu există un limbaj de obținere a datelor);

- ▶ un fișier este stocat pe un singur calculator, nu au acces mai mulți utilizatori; chiar dacă ar fi distribuit (e.g. Samba) tot nu poate fi editat simultan de mai mulți utilizatori.
- ▶ ACID trebuie construit la nivel manual de fiecare dată pentru a permite accesul mai multor utilizatori (simultan) la informație;

Dezavantaje DBMS ?

- ▶ fișiere de dimensiuni mari;
- ▶ aplicații complexe ce depind de menenanța exterioară;
- ▶ costuri (uneori mari);
- ▶ costuri pentru hardware;
- ▶ fișierele s-ar putea să fie accesate mai rapid dacă știu ce vreau;
- ▶ toți se bazează pe același DBMS - dacă acesta este închis...

Sisteme de gestiune a bazelor de date (SGBD)

Un sistem de gestiune de baze de date este alcătuit din:

- ▶ Hardware
- ▶ Software
- ▶ Utilizatori
- ▶ Date

Scopul său este de a deservi **rapid** foarte **mulți** utilizatori care fac interogări diferite într-o **cantitate foarte mare de date**.

Securitate * Acces controlat la baza de date * Stocarea, regăsirea, actualizarea datelor * Integritate * Suport pentru tranzacții * Control concurrent * Recuperarea datelor * Catalog (dicționar de date)

- ▶ **Hardware**
- ▶ Software
- ▶ Utilizatori
- ▶ Date

Hardware-ul poate varia de la un simplu PC până la rețele de calculatoare și trebuie să asigure:

- ▶ Persistența datelor (chiar în cazuri critice).
- ▶ Stocarea unui volum mare de date.
- ▶ Accesul rapid la date (vezi discuția despre HDD-uri).

- ▶ Hardware
- ▶ Software
- ▶ Utilizatori
- ▶ Date

Hardware-ul poate varia de la un simplu PC până la rețele de calculatoare și trebuie să asigure:

- ▶ Persistența datelor (chiar în cazuri critice).
- ▶ Stocarea unui volum mare de date.
- ▶ Accesul rapid la date (vezi discuția despre HDD-uri).

- ▶ Hardware
- ▶ Software
- ▶ Utilizatori
- ▶ Date

Partea de software trebuie să ofere (măcar) o metodă de definire a datelor și una de manipulare a acestora [fox]:

- ▶ DDL (Data Definition Language)
- ▶ DML (Data Manipulation Language)

Dar are și rolul de management a utilizatorilor (Data Control Language - grant, revoke), realizează conexiuni cu software extern ce dorește să acceseze informațiile din BD, control al tranzacțiilor (Transaction Control Language -¹⁵ savepoint, commit, rollback), etc.

- ▶ Hardware
- ▶ Software
- ▶ Utilizatori
- ▶ Date

Persoanele care interacționează cu baza de date:

- ▶ Administratorul
- ▶ Proiectantul
- ▶ Programatorii de aplicații
- ▶ Utilizatorii ce folosesc aplicațiile

- ▶ Hardware
- ▶ Software
- ▶ Utilizatori
- ▶ Date

Datele vor fi stocate în fișiere având formate specifice.

De-a lungul timpului au existat mai multe modele de baze de date. Modelul relațional, pe care îl studiem, este în prezent cel utilizat de SGBD-uri precum MySQL, MariaDB, Oracle, Postgresql, SQL Server, SQL Lite, etc.

Același limbaj: SQL.

Modele de baze de date

În trecut:

- ▶ Modelul ierarhic (IBM's IMS, sf. '60)
- ▶ Modelul rețea (CODASYL 1971)
- ▶ Modelul relațional (Codd, '70)
- ▶ Modelul obiect-relațional ('90)

În prezent:

- ▶ Modelul relațional (cel al lui Codd)
- ▶ Diverse modele nerelaționale (2000s) - peste 225 variante
(despre care puteți citi la <http://nosql-database.org/>) ce nu constituie subiectul acestui curs.

Modele de baze de date

În trecut:

- ▶ **Modelul ierarhic (IBM's IMS, sf. '60)**
- ▶ Modelul rețea (CODASYL 1971)
- ▶ Modelul relațional (Codd, '70)
- ▶ Modelul obiect-relațional ('90)

În prezent:

- ▶ Modelul relațional (cel al lui Codd)
- ▶ Diverse modele nerelaționale (2000s) -
<http://nosql-database.org/>

Modele de baze de date - modelul ierarhic

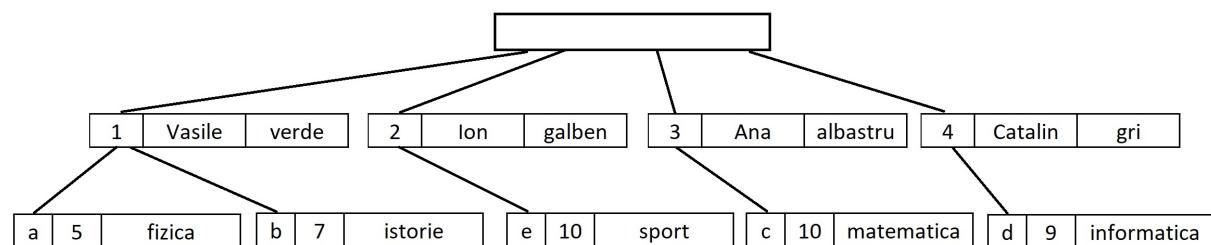
Datele sunt organizate într-un arbore, fiecare nod reprezentând o linie din tabel. Baza de date are și un nod rădăcina care nu conține nimic.

Exemplu:

$$S = \{1\text{-Vasile-verde}, 2\text{-Ion-galben}, 3\text{-Ana-albastru}, 4\text{-Catalin-gri}\}$$

$$N = \{a\text{-5-fizica}, b\text{-7-istorie}, c\text{-10-matematica}, d\text{-9-informatica}, e\text{-10-sport}\}$$

$$\text{Muchii} = \{(\text{null}, 1), (\text{null}, 2), (\text{null}, 3), (\text{null}, 4), (1, a), (1, b), (2, e), (3, c), (4, d)\}$$



Modelul ierarhic permite unui nod să aibă doar un singur părinte.

Modele de baze de date

În trecut:

- ▶ Modelul ierarhic (IBM's IMS, sf. '60)
- ▶ **Modelul rețea (CODASYL 1971)**
- ▶ Modelul relațional (Codd, '70)
- ▶ Modelul obiect-relațional ('90)

În prezent:

- ▶ Modelul relațional (cel al lui Codd)
- ▶ Diverse modele nerelaționale (2000s) -
<http://nosql-database.org/>

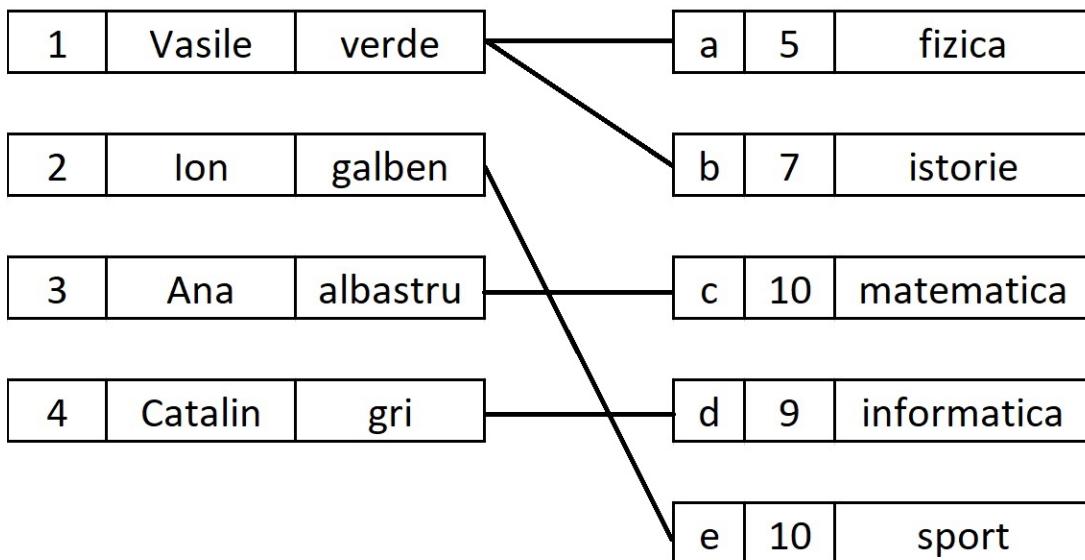
Modele de baze de date - modelul rețea

Considerăm o mulțime de studenți și o mulțime de note. În modelul ierarhic se construiește o a treia mulțime de cupluri din conținând identificatori ai elementelor primei mulțimi respectiv ai elementelor celei de-a doua mulțimi.

Exemplu:

$$\begin{aligned} S &= \{1\text{-Vasile-verde}, 2\text{-Ion-galben}, 3\text{-Ana-albastru}, 4\text{-Catalin-gri}\} \\ N &= \{a\text{-5-fizica}, b\text{-7-istorie}, c\text{-10-matematica}, d\text{-9-informatica}, \\ &\quad e\text{-10-sport}\} \end{aligned}$$

$$C = \{(1,a), (1,b), (2,e), (3,c), (4,d)\}$$



Modele de baze de date

În trecut:

- ▶ Modelul ierarhic (IBM's IMS, sf. '60)
- ▶ Modelul rețea (CODASYL 1971)
- ▶ **Modelul relațional (Codd, '70)**
- ▶ Modelul obiect-relațional ('90)

În prezent:

- ▶ Modelul relațional (cel al lui Codd)
- ▶ Diverse modele nerelaționale (2000s) -
<http://nosql-database.org/>

Modele de baze de date - modelul relațional

În modelul relațional se stabilesc care sunt entitățile ce trebuie să fie memorate în baza de date și se construiesc asocieri între tabele. În exemplul nostru, entitățile sunt elevii și notele sunt doar asociate acestora. În continuare informațiile sunt stocate ca mulțimi, relațiile dintre elementele acestora este realizată în baza unor chei primare / străine.

Exemplu:

$$S = \{1\text{-Vasile-verde}, 2\text{-Ion-galben}, 3\text{-Ana-albastru}, 4\text{-Catalin-gri}\}$$

$$N = \{a\text{-5-fizica-1}, b\text{-7-istorie-1}, c\text{-10-matematica-3}, d\text{-9-informatica-4}, e\text{-10-sport-2}\}$$

| id | nume | culoare |
|----|---------|----------|
| 1 | Vasile | verde |
| 2 | Ion | galben |
| 3 | Ana | albastru |
| 4 | Catalin | gri |

| id | nota | materie | ref_id |
|----|------|-------------|--------|
| a | 5 | fizica | 1 |
| b | 7 | istorie | 1 |
| c | 10 | matematica | 3 |
| d | 9 | informatica | 4 |
| e | 10 | sport | 2 |

Informația din coloana "id" din primul tabel este cea pe baza careia se face legătura dintre cele două tabele. Valorile din coloana "ref_id" din cel de-al doilea tabel fac referință către această colană.

Modele de baze de date

În trecut:

- ▶ Modelul ierarhic (IBM's IMS, sf. '60)
- ▶ Modelul rețea (CODASYL 1971)
- ▶ Modelul relațional (Codd, '70)
- ▶ **Modelul obiect-relațional ('90)** - impedance mismatch problem

În prezent:

- ▶ Modelul relațional (cel al lui Codd)
- ▶ Diverse modele nerelaționale (2000s) -
<http://nosql-database.org/>

În prezent:

- ▶ Modelul relațional (cel al lui Codd)
- ▶ Diverse modele nerelaționale (2000s) -
<http://nosql-database.org/>

Relațional vs Nerelațional - Oare cine câștigă ?

În prezent:

- ▶ Modelul relațional (cel al lui Codd)
- ▶ Diverse modele nerelaționale (2000s) -
<http://nosql-database.org/>

Relațional vs Nerelațional - Oare cine câștigă ?

Dar mai întâi, o poveste: cea a dublei rezervări a unei camere de hotel...

În prezent:

- ▶ Modelul relațional (cel al lui Codd)
- ▶ Diverse modele nerelaționale (2000s) -
<http://nosql-database.org/>

Relațional vs Nerelațional - Oare cine câștigă ?

Proprietățile ce s-ar dori pentru un sistem (distribuit):

- ▶ Consistență (Consistency)
- ▶ Disponibilitate (Availability)
- ▶ Distribuirea pe mai multe calculatoare (Partition Tolerance)

Teorema CAP afirma că un sistem distribuit poate satisface doar două dintre aceste proprietăți.

CAP (continuare / demonstrație)

Sistemele distribuite (presupunem formate din două calculatoare C_1 și C_2) se bazează pe rețea care poate eșua (fail). Atunci:

- A) Păstrarea disponibilității: C_1 poate continua operațiile fără a se sincroniza cu C_2 și poate obține rezultate care din cauză că depindeau de C_2 sunt eronate (pentru că nu a putut obține informații de la C_2) - de exemplu poate vinde stocul de produse ce a fost deja vândut de C_2 (dar va fi disponibil).
- B) Păstrarea consistenței: C_1 poate să NU servească nici un client pentru a nu obține date eronate (dar va fi consistent).

Evident, dacă rețeaua funcționează, C_1 poate să se sincronizeze cu C_2 și sincronizarea va duce atât la disponibilitate cât și la consistență. Pot să fiu sigur că rețeaua funcționează doar dacă nu depind de ea; deci sistemul nu mai e distribuit / partaționat.

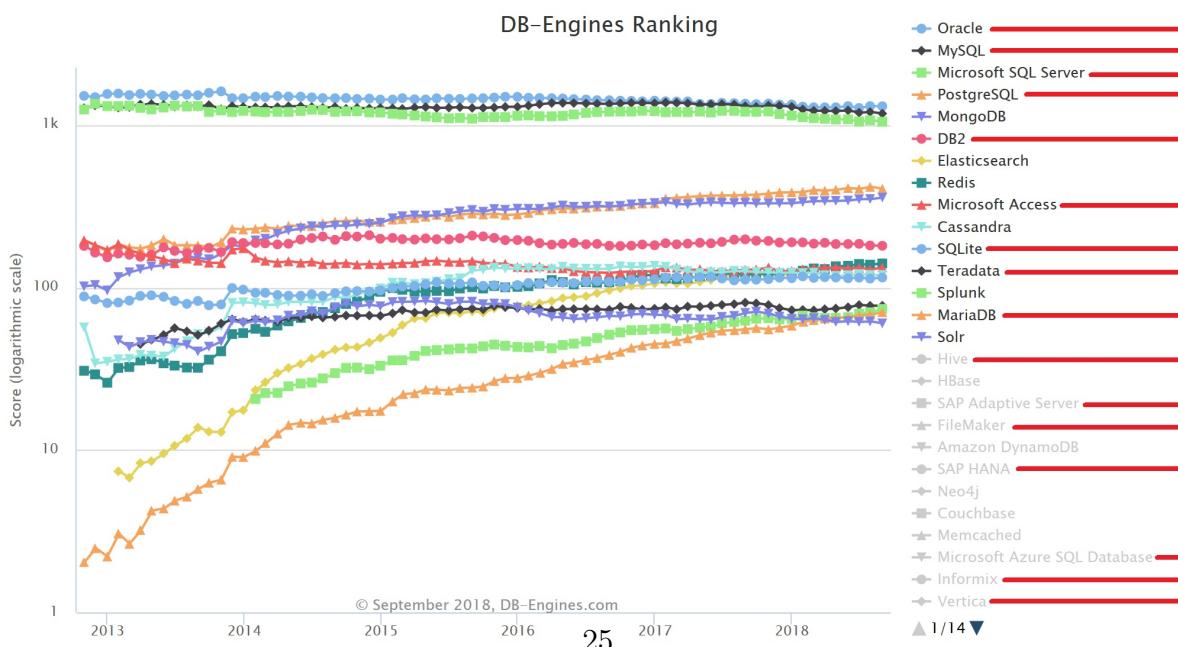
Tranzacția în BD relaționale este ACID

- ▶ **Atomică** - Toate operațiile din tranzacție sunt executate (ca o singură entitate atomică) sau, în caz contrar, toate sunt ignorate și BD revine la starea inițială.
- ▶ **Consistență** - La sfârșitul tranzacției, datele sunt aşa cum ar trebui să fie conform logicii tranzacției.
- ▶ **Izolarea** - Tranzacțiile nu știu una de alta și nu se influențează, chiar dacă sunt executate simultan. Fiecare tranzacție crede că în momentul în care ea se execută, nimeni altcineva nu mai interacționează cu BD.
- ▶ **Durabilitate** - Dacă tranzacția s-a terminat și s-a facut commit, nimeni și nimic nu o mai pot aduce într-o stare anterioară.

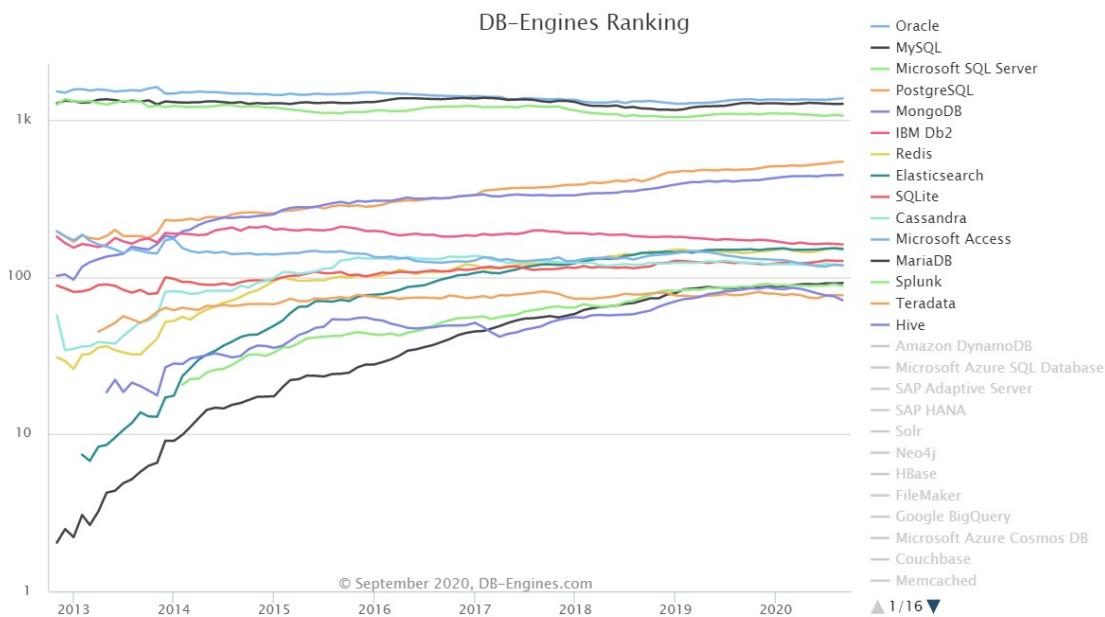
Tranzacția în BD nerelaționale este BASE

- ▶ **Basic Availability** - Baza de date pare că este funcțională de cele mai multe ori (adică, “în principiu este disponibilă”).
- ▶ **Soft-state** - Locurile din rețea unde este stocată baza de date nu trebuie neapărat să fie consistente unele cu celălalte (deși se încearcă).
- ▶ **Eventual consistency** - Uneori datele vor fi consistente (când se sincronizează)

Relational vs Nerelational; De ce Oracle ?



Relațional vs Nerelațional; De ce Oracle ?



<https://db-engines.com> (01-oct.-2020)

47 / 57

Nicolae-Cosmin Vârlan

Baze de date - Introducere

Baze de date Introducere

Istoric și Motivăție
 Sisteme de gestiune a bazelor de date (SGBD)
 Modele de baze de date
 Modelul relațional

| Rank | Sep 2020 | Aug 2020 | Sep 2019 | DBMS | Database Model | Score | | |
|------|----------|----------|----------|------------------------------|----------------------------|----------|----------|----------|
| | | | | | | Sep 2020 | Aug 2020 | Sep 2019 |
| 1. | 1. | 1. | 1. | Oracle + | Relational, Multi-model | 1369.36 | +14.21 | +22.71 |
| 2. | 2. | 2. | 2. | MySQL + | Relational, Multi-model | 1264.25 | +2.67 | -14.83 |
| 3. | 3. | 3. | 3. | Microsoft SQL Server + | Relational, Multi-model | 1062.76 | -13.12 | -22.30 |
| 4. | 4. | 4. | 4. | PostgreSQL + | Relational, Multi-model | 542.29 | +5.52 | +60.04 |
| 5. | 5. | 5. | 5. | MongoDB + | Document, Multi-model | 446.48 | +2.92 | +36.42 |
| 6. | 6. | 6. | 6. | IBM Db2 + | Relational, Multi-model | 161.24 | -1.21 | -10.32 |
| 7. | 7. | ↑ 8. | 8. | Redis + | Key-value, Multi-model | 151.86 | -1.02 | +9.95 |
| 8. | 8. | ↓ 7. | 7. | Elasticsearch + | Search engine, Multi-model | 150.50 | -1.82 | +1.23 |
| 9. | 9. | ↑ 11. | 11. | SQLite + | Relational | 126.68 | -0.14 | +3.31 |
| 10. | ↑ 11. | 10. | 10. | Cassandra + | Wide column | 119.18 | -0.66 | -4.22 |
| 11. | ↓ 10. | ↓ 9. | 9. | Microsoft Access | Relational | 118.45 | -1.41 | -14.26 |
| 12. | 12. | ↑ 13. | 13. | MariaDB + | Relational, Multi-model | 91.61 | +0.69 | +5.54 |
| 13. | 13. | ↓ 12. | 12. | Splunk | Search engine | 87.90 | -2.01 | +0.89 |
| 14. | 14. | ↑ 15. | 15. | Teradata + | Relational, Multi-model | 76.39 | -0.39 | -0.57 |
| 15. | 15. | ↓ 14. | 14. | Hive | Relational | 71.17 | -4.12 | -11.93 |
| 16. | 16. | ↑ 18. | 18. | Amazon DynamoDB + | Multi-model | 66.18 | +1.43 | +8.36 |
| 17. | 17. | ↑ 25. | 25. | Microsoft Azure SQL Database | Relational, Multi-model | 60.45 | +3.60 | +32.91 |
| 18. | 18. | ↑ 19. | 19. | SAP Adaptive Server | Relational | 54.01 | +0.05 | -2.09 |
| 19. | 19. | ↑ 21. | 21. | SAP HANA + | Relational, Multi-model | 52.86 | -0.26 | -2.53 |
| 20. | 20. | ↓ 16. | 16. | Solr | Search engine | 51.62 | -0.08 | -7.35 |

<https://db-engines.com> (01-oct.-2020)

48 / 57

Nicolae-Cosmin Vârlan

Baze de date Introducere

Concluzia ?

Bazele de date relaționale și cele nerelaționale vor coexista, fiecare având avantajele lor.

Cu siguranță cele relaționale nu vor dispărea în următoarele două secole.

Dovada:

Enterprise still uses SQL in 23th century [Star Trek Discovery]

Modelul relațional - concepte (intuitiv)

Momentan vom da câteva concepte la nivel intuitiv, dar vom formaliza în cursul viitor.

| id | nume | culoare |
|----|---------|----------|
| 1 | Vasile | verde |
| 2 | Ion | galben |
| 3 | Ana | albastru |
| 4 | Catalin | gri |

Relație = Tabel (cu roșu)

Modelul relațional - concepte (intuitiv)

| id | nume | culoare |
|----|---------|----------|
| 1 | Vasile | verde |
| 2 | Ion | galben |
| 3 | Ana | albastru |
| 4 | Catalin | gri |

Atribute (cu roșu)

Modelul relațional - concepte (intuitiv)

| id | nume | culoare |
|----|---------|----------|
| 1 | Vasile | verde |
| 2 | Ion | galben |
| 3 | Ana | albastru |
| 4 | Catalin | gri |

Tuplu sau linie

Modelul relațional - concepte (intuitiv)

| id | nume | culoare |
|----|---------|----------|
| 1 | Vasile | verde |
| 2 | Ion | galben |
| 3 | Ana | albastru |
| 4 | Catalin | gri |

Valoare (a atributului nume din tuplul t_3)

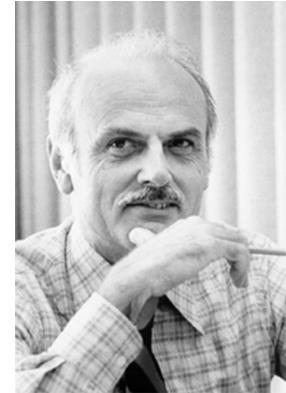
Modelul relațional - concepte (intuitiv)

| id | nume | culoare |
|----|---------|----------|
| 1 | Vasile | verde |
| 2 | Ion | galben |
| 3 | Ana | albastru |
| 4 | Catalin | gri |

Domeniul unui atribut = mulțimea de elemente din care se poate selecta valoarea unui atribut (e.g. {Vasile, Ion, Ana, Catalin})

Modelul relațional

- ▶ Fiecare element al relației conține informație (null este informație).
- ▶ Fiecare atribut este unic.
- ▶ Valorile unui atribut sunt din același domeniu.
- ▶ Nu există linii identice în tabel.
- ▶ Ordinea rândurilor și coloanelor este arbitrară.
- ▶ Are la bază algebra relațională introdusă de Edgar Frank Codd.



E.F. Codd

Modelul relațional - chei

- ▶ **Supercheie** - un atribut sau o mulțime de attribute care identifică unic un tuplu într-o relație
- ▶ **Cheie candidat** - o supercheie cu proprietatea că nici o submulțime proprie a sa nu este supercheie
- ▶ **Cheie primară** - o cheie candidat selectată pentru a identifica în mod unic tuplele într-o relație
- ▶ **Cheie alternativă** - Chei candidat care nu au fost selectate pentru a juca rolul de cheie primară
- ▶ **Cheie străină** - un atribut sau o submulțime de attribute dintr-o relație care face referință la o cheie candidat a altrei relații

Bibliografie

- ▶ Database Systems - A Practical Approach to Design, Implementation, and Management, *Thomas Connolly, Carolyn Begg*; Pearson, 2015
- ▶ Database System Concepts, *Abraham Silberschatz. Henry F. Korth, S. Sudarshan*; McGrawHill, 2011

3 CURS 2

Baze de date Algebra relațională

Nicolae-Cosmin Vârlan

October 8, 2020

Elemente ale modelului relațional

- ▶ U mulțime de atribute: $U = \{A_1, A_2, \dots, A_n\}$;
- ▶ $\text{dom}(A_i)$ - domeniul valorilor atributului A_i ;

Definim ***uplu*** peste U ca fiind funcția:

$$\varphi : U \rightarrow \bigcup_{1 \leq i \leq n} \text{dom}(A_i) \quad \text{a.i. } \varphi(A_i) \in \text{dom}(A_i), 1 \leq i \leq n$$

Fie valorile v_i astfel încât $v_i = \varphi(A_i)$.

Notăm cu $\{A_1 : v_1, A_2 : v_2, \dots, A_n : v_n\}$ asocierea dintre atributele existente în U și valorile acestora. În cazul în care sunt considerate mulțimi ordonate (de forma (A_1, A_2, \dots, A_n)), notația va fi de forma: (v_1, v_2, \dots, v_n) . 33

Elemente ale modelului relațional

Considerăm mulțimea ordonată (A_1, A_2, \dots, A_n) . Pentru orice uplu φ , există vectorul (v_1, v_2, \dots, v_n) a.î. $\varphi(A_i) = v_i$, $1 \leq i \leq n$.

Pentru un vector (v_1, v_2, \dots, v_n) cu $v_i \in \text{dom}(A_i)$, $1 \leq i \leq n$ există un uplu φ a.î. $\varphi(A_i) = v_i$.

În practică este considerată o anumită ordonare a atributelor.

Elemente ale modelului relațional

O mulțime de uple peste U se numește **relație** și se notează cu r .

r poate varia în timp dar nu și în structură.

Exemplu:

$$r = \{(v_{11}, v_{12}, \dots, v_{1n}), (v_{21}, v_{22}, \dots, v_{2n}), \dots, (v_{m1}, v_{m2}, \dots, v_{mn})\}.$$

Structura relației se va nota cu $R[U]$ unde R se numește **numele relației** iar U este mulțimea de **attribute** corespunzătoare.

Notații echivalente $R(U)$, $R(A_1, A_2, \dots, A_n)$, $R[A_1, A_2, \dots, A_n]$.

$R[U]$ se mai numește și **schemă de relație**.

Elemente ale modelului relațional

În practică, o relație r poate fi reprezentată printr-o matrice:

$$r : \begin{array}{cccc} & A_1 & A_2 & \dots & A_n \\ \hline & v_{11} & v_{12} & \dots & v_{1n} \\ & \dots & \dots & \dots & \dots \\ & v_{m1} & v_{m2} & \dots & v_{mn} \end{array}$$

unde $(v_{i1}, v_{i2}, \dots, v_{in})$ este un uplu din r , $1 \leq i \leq m$ și $v_{ij} \in \text{dom}(A_j)$, $1 \leq j \leq n$, $1 \leq i \leq m$

Vom nota cu t_i linia (tplul) cu numărul i din matrice:

$$t_i = (v_{i1}, v_{i2}, \dots, v_{in}), \forall i \in [1, m]$$

Elemente ale modelului relațional

O mulțime finită D de scheme de relație se numește **schemă de baze de date**. Formal, $D = \{R_1[U_1], \dots, R_h[U_h]\}$ unde $R_i[U_i]$ este o schemă de relație, $1 \leq i \leq h$.

O **bază de date peste D** este o corespondență ce asociază fiecărei scheme de relație din D o relație.

Exemplu:

r_1, r_2, \dots, r_h este o bază de date peste $D = \{R_1[U_1], \dots, R_h[U_h]\}$.

Considerând D ca fiind ordonată $D = (R_1[U_1], \dots, R_h[U_h])$, putem nota baza de date sub forma (r_1, r_2, \dots, r_h)

Corespondența cu terminologia din practică

- ▶ atribut (A_i) = denumirea unei coloane dintr-un tabel;
- ▶ valoarea atributului A_i ($\varphi(A_i)$ sau v_i) = valoarea dintr-o celulă a tabelului
- ▶ relație (r) = tabel
- ▶ schema de relație ($R[U]$) = schema tabelei
- ▶ tuplu (t_i) = linie din tabel

Operații

Asupra unei mulțimi de relații putem efectua o serie de operații.

Există două categorii de operatori:

- ▶ Operatori din teoria mulțimilor: Reuniunea(\cup), Intersecția (\cap), Diferența($-$), Produsul Cartezian(\times)
- ▶ Operatori specifici algebrei relaționale: Proiecția (π), Selecția(σ), Redenumirea(ρ), Joinul Natural(\bowtie), θ -Joinul, equijoinul, Semijoinul(\bowtie și $\bowtie\bowtie$), Antijoinul(\triangleright), Divizarea(\div), Joinul la Stânga ($\bowtie\bowtie\bowtie$), Joinul la Dreapta($\bowtie\bowtie\bowtie\bowtie$), Joinul Exterior($\bowtie\bowtie\bowtie\bowtie\bowtie$)

Operații pe mulțimi de tuple - *Reuniunea*: \cup

În cazul operațiilor pe mulțimi (cu excepția Produsului Cartezian), acestea se realizează între două relații r_1 și r_2 care sunt NEAPĂRAT construite peste aceeași mulțime de atrbute.

Reuniunea a două relații r_1 și r_2 , ambele peste o aceeași mulțime de atrbute U (sau peste aceeași schemă de relație $R[U]$), este o relație notată cu $r_1 \cup r_2$ definită astfel:

$$r_1 \cup r_2 = \{t \mid t = \text{uplu}, \quad t \in r_1 \text{ sau } t \in r_2\}$$

În practică, acest lucru se realizează utilizând cuvântul cheie UNION. Studenții din anii 1 și 3 sunt selectați de interogarea:

```
SELECT * FROM studenti WHERE an=1
UNION
SELECT * FROM studenti WHERE an=3;
```

Operații pe mulțimi de tuple - *Diferența*: $-$

Diferența a două relații r_1 și r_2 , ambele peste o aceeași mulțime de atrbute U (sau peste aceeași schemă de relație $R[U]$), este o relație notată cu $r_1 - r_2$ definită astfel:

$$r_1 - r_2 = \{t \mid t = \text{uplu}, \quad t \in r_1 \text{ si } t \notin r_2\}$$

În practică, acest lucru se realizează utilizând cuvântul cheie MINUS. Pentru a-i selecta pe studenții din anul 2 fără bursă, putem să îi selectăm pe toți studenții din anul 2 și apoi să îi eliminăm pe cei cu bursă:

```
SELECT * FROM studenti WHERE an=2
MINUS
SELECT * FROM studenti WHERE bursa IS NOT NULL;
```

Operații pe mulțimi de tuple - *Intersecția*: \cap

Intersecția a două relații r_1 și r_2 , ambele peste o aceeași mulțime de atribut U (sau peste aceeași schemă de relație $R[U]$), este o relație notată cu $r_1 \cap r_2$ definită astfel:

$$r_1 \cap r_2 = \{t \mid t = uplu, \quad t \in r_1 \text{ și } t \in r_2\}$$

În practică, acest lucru se realizează utilizând cuvântul cheie **INTERSECT**. Putem afla care studenți din anul 2 au bursă rulând:

SELECT * FROM studenti WHERE an=2

INTERSECT

SELECT * FROM studenti WHERE bursa IS NOT NULL;

Operatorul de intersecție poate fi obținut din ceilalți doi:

$$r_1 \cap r_2 = r_1 - (r_1 - r_2).$$

Operații pe mulțimi de tuple - *Produsul Cartezian*: \times

Produsul cartezian a două relații r_1 definită peste $R_1[U_1]$ și r_2 definită peste $R_2[U_2]$ cu $U_1 \cap U_2 = \emptyset$ este o relație notată cu $r_1 \times r_2$ definită astfel:

$$r_1 \times r_2 = \{t \mid t = uplu \text{ peste } U_1 \cup U_2, \quad t[U_1] \in r_1 \text{ și } t[U_2] \in r_2\}$$

De aceasta dată, cele două relații nu trebuie să fie peste aceeași mulțime de atribut. Rezultatul va fi o nouă relație peste o mulțime de atribut formată din atributele relațiilor inițiale.

Operații pe mulțimi de tuple - *Produsul Cartezian*:

Dacă un atribut s-ar repeta, el va fi identificat diferit. Spre exemplu, chiar dacă tabelele note și cursuri au un același atribut (id_curs), nu se face nici o sincronizare după acesta ci se vor crea două attribute diferite: note.id_curs respectiv cursuri.id_curs.

Produsul cartezian între aceste tabele, în practică, se obține executând interogarea:

```
SELECT * FROM cursuri, note;
```

Operații specifice algebrei relaționale

Operațiile pe mulțimi aveau ca elemente tuplele. Uneori aceste tuple nu sunt compatibile (de exemplu nu putem reuni o relație peste $R_1[U_1]$ cu una peste $R_2[U_2]$ dacă $U_1 \neq U_2$).

Pentru a opera asupra atributelor ce definesc tuplele din rezultat, avem nevoie de o serie de operatori specifici algebrei relaționale.

Operații în algebra relațională - Proiecția: π

Considerăm:

- ▶ $R[U]$ = schemă de relație;
- ▶ $X \subseteq U$;
- ▶ t = tuplu peste $R[U]$ ($t \in r$).

Se numește *proiecția lui t relativă la X* și notată cu $\pi_X[t]$, restricția lui t la mulțimea de atrbute X . (Uneori vom scrie $t[X]$)

Exemplu:

Dacă $U = (A_1, A_2, \dots, A_n)$ atunci $t = (v_1, v_2, \dots, v_n)$.

Considerăm $X = (A_{i_1}, A_{i_2}, \dots, A_{i_k})$, $1 \leq i_1 < i_2 < \dots < i_k \leq n$.

atunci $\pi_X[t] = (v_{i_1}, v_{i_2}, \dots, v_{i_k})$;

Operații în algebra relațională - Proiecția: π

Dacă r este o relație peste $R[U]$ și $X \subseteq U$, atunci *proiecția lui r relativă la X* este $\pi_X[r] = \{\pi_X[t] \mid t \in r\}$

Exemplu:

Dacă $U = (A_1, A_2, \dots, A_n)$ atunci

$r = \{(v_{11}, v_{12}, \dots, v_{1n}), (v_{21}, v_{22}, \dots, v_{2n}), \dots, (v_{m1}, v_{m2}, \dots, v_{mn})\}$.

Considerăm $X = (A_{i_1}, A_{i_2}, \dots, A_{i_k})$, $1 \leq i_1 < i_2 < \dots < i_k \leq n$.

atunci

$\pi_X[r] = \{(v_{1i_1}, v_{1i_2}, \dots, v_{1i_k}), (v_{2i_1}, \dots, v_{2i_k}), \dots, (v_{mi_1}, \dots, v_{mi_k})\}$

În practică, proiecția se realizează selectând doar anumite câmpuri ale tabelei (anumite atrbute):

`SELECT nume, prenume FROM studenti;`

Operații în algebra relațională - Proiecția: π

Ca și exemplu, vom scrie o interogare care să returneze toate persoanele care trec pragul Facultății (studenți și profesori):

SELECT nume, prenume FROM studenti

UNION

SELECT nume, prenume FROM profesori;

În cazul în care cele două câmpuri (nume, prenume) din cele două tabele au același tip (de exemplu nume este de tip VARCHAR2(10) în ambele tabele), interogarea va afișa toate persoanele ce "trec pragul Facultății".

Observație: Pentru a modifica tipul nume din tabela profesori la VARCHAR2(10) execuți comanda:

ALTER TABLE profesori MODIFY nume VARCHAR2(10);

Operații în algebra relațională - Selectia: σ

Fie r o relație peste $R[U]$, $A, B \in U$ și c este o constantă

O **expresie elementară de selecție** este definită prin următoarea formulă (forma Backus-Naur):

$$e = A \varphi B \mid A \varphi c \mid c \varphi B$$

Unde φ este o relație booleană între operanzi.

Se numește **expresie de selecție** (forma Backus-Naur):

$$\theta = e \mid \theta \wedge \theta \mid \theta \vee \theta \mid (\theta)$$

Operații în algebra relațională - Selectia: σ

Fie θ o expresie de selecție. Atunci:

- ▶ când $\theta = A \varphi B$, t satisfacă θ dacă are loc $\pi_A[t] \varphi \pi_B[t]$,
- ▶ când $\theta = A \varphi c$, t satisfacă θ dacă are loc $\pi_A[t] \varphi c$,
- ▶ când $\theta = c \varphi B$, t satisfacă θ dacă are loc $c \varphi \pi_B[t]$,
- ▶ când $\theta = \theta_1 \wedge \theta_2$, t satisfacă θ dacă t satisfacă atât pe θ_1 cât și pe θ_2 ,
- ▶ când $\theta = \theta_1 \vee \theta_2$, t satisfacă θ dacă t satisfacă măcar pe unul dintre θ_1 și θ_2 .

Dacă θ este o expresie de selecție atunci **selecția** se notează cu $\sigma_\theta(r)$ și este definită ca:

$$\sigma_\theta(r) = \{t | t \in r, t \text{ satisfacă } \theta\}$$

Operații în algebra relațională - Selectia: σ

În SQL, selecția se obține utilizând o formulă logică ce are rolul de a selecta doar anumite rânduri.

Exemplu:

```
SELECT * FROM studenti
WHERE ((an=2) AND (bursa IS NULL));
```

În acest exemplu, θ_1 este $an = 2$, θ_2 este $bursa IS NULL$, $\theta = \theta_1 \wedge \theta_2$ și r este mulțimea de rânduri din tabela studenți.

Rezultatul este mulțimea studenților din anul 2 care nu au bursă.

Operații în algebra relațională - *Redenumirea*: ρ

Operatorul de **redenumire** are rolul de a schimba numele unui atribut cu alt nume. Formal, dacă dorim să schimbăm atributul A_1 în A'_1 vom utiliza scrierea $\rho_{A_1/A'_1}(r)$. Restul atributelor peste care a fost construit r vor rămâne neschimbate.

În SQL, redenumirea se realizează prin utilizarea cuvântului AS:

Exemplu:

`SELECT bursa * 1.25 AS "BursaNoua" FROM studenti;`

`SELECT bursa + bursa/4 AS "BursaNoua" FROM studenti;`

Dacă nu am redenumi atributul nou obținut, cele două relații ar fi considerate diferite (în prima numele atributului ar fi "bursa * 1.25", iar în a doua ar fi fost "bursa + bursa/4")

Operații în algebra relațională - *Join natural*: \bowtie

Considerăm:

- ▶ r_1 relație peste $R_1[U_1]$;
- ▶ r_2 relație peste $R_2[U_2]$;

Se numește **Join natural** a relațiilor r_1 și r_2 , relația $r_1 \bowtie r_2$ peste $U_1 \cup U_2$ definită prin:

$$r_1 \bowtie r_2 = \{t \mid t \text{ uplu peste } U_1 \cup U_2, t[U_i] \in r_i, i = 1, 2\}$$

Dacă R este un nume pentru relația peste $U_1 \cup U_2$ atunci $r_1 \bowtie r_2$ este definită peste $R[U_1 \cup U_2]$

Pentru simplitate vom nota $U_1 \cup U_2$ cu $U_1 U_2$.

Operații în algebra relațională - *Join natural*:

Exemplu:

Fie $R_1[A, B, C, D]$, $R_2[C, D, E]$ și r_1, r_2 a.i.:

| | A | B | C | D | | C | D | E |
|---------|---|---|---|---|--|---|---|---|
| $r_1 :$ | 0 | 1 | 0 | 0 | | 1 | 1 | 0 |
| | 1 | 1 | 0 | 0 | | 1 | 1 | 1 |
| | 0 | 0 | 1 | 0 | | 0 | 0 | 0 |
| | 1 | 1 | 0 | 1 | | 1 | 0 | 0 |
| | 0 | 1 | 0 | 1 | | 1 | 0 | 1 |

| | A | B | C | D | E |
|---------|---|---|---|---|---|
| $r_2 :$ | 0 | 1 | 0 | 0 | 0 |
| | 1 | 1 | 0 | 0 | 0 |
| | 0 | 0 | 1 | 0 | 0 |
| | 0 | 0 | 1 | 0 | 1 |

Atunci: $r_1 \bowtie r_2 :$

Operații în algebra relațională - *Join natural*:

Următoarea interogare identifică cui aparține fiecare nota din tabelul note. Joinul se face după câmpul nr_matricol între tabelele studenti și note:

```
SELECT nume, valoare FROM studenti
    NATURAL JOIN note;
```

```
SELECT nume, valoare FROM studenti
    JOIN note ON studenti.nr_matricol = note.nr_matricol;
```

Se poate observa că dacă din produsul cartezian am elibera acele cazuri în care câmpul "nr_matricol" nu este identic în ambele tabele, am obține, de fapt, același rezultat. Din acest motiv, joinul de mai sus poate fi scris și sub forma:

```
SELECT nume, valoare FROM studenti,note
    WHERE studenti.nr_matricol =44 note.nr_matricol;
```

Proprietăți ale Joinului natural

- ▶ $(r_1 \bowtie r_2)[U_1] \subseteq r_1$
- ▶ $(r_2 \bowtie r_1)[U_2] \subseteq r_2$

Dacă $X = U_1 \cap U_2$ și:

$r'_1 = \{t_1 | t_1 \in r_1, \exists t_2 \in r_2 \text{ a.i. } t_1[X] = t_2[X]\}$ și $r_1'' = r_1 - r'_1$,
 $r'_2 = \{t_2 | t_2 \in r_2, \exists t_1 \in r_1 \text{ a.i. } t_1[X] = t_2[X]\}$ și $r_2'' = r_2 - r'_2$,
atunci: $r_1 \bowtie r_2 = r'_1 \bowtie r'_2$, $(r_1 \bowtie r_2)[U_1] = r'_1$, $(r_2 \bowtie r_1)[U_2] = r'_2$.

Dacă $\overline{r_1} \subseteq r_1$, $\overline{r_2} \subseteq r_2$ și $\overline{r_1} \bowtie \overline{r_2} = r_1 \bowtie r_2$ atunci $r'_1 \subseteq \overline{r_1}$ și $r'_2 \subseteq \overline{r_2}$

Dacă $U_1 \cap U_2 = \emptyset$ atunci $r_1 \bowtie r_2 = r_1 \times r_2$.

Extindere Join natural

Fie r_i relație peste $R_i[U_i]$, $i = \overline{1, h}$ atunci:

$r_1 \bowtie r_2 \bowtie \dots \bowtie r_h = \{t | t \text{ uplu peste } U_1, \dots, U_h, \text{ a.i. } t[U_i] \in r_i, i = \overline{1, h}\}$

Notății echivalente:

- ▶ $r_1 \bowtie r_2 \bowtie \dots \bowtie r_h$
- ▶ $\bowtie \langle r_i, i = 1, h \rangle$
- ▶ $*\langle r_i, i = 1, h \rangle$

Operația join este asociativă.

Operații în algebra relațională - θ -join, equijoin

Fie r_i peste $R_i[U_i]$, $i = \overline{1, 2}$ cu $A_{\alpha_1}, A_{\alpha_2}, \dots, A_{\alpha_k} \in U_1$ și $B_{\beta_1}, B_{\beta_2}, \dots, B_{\beta_k} \in U_2$ și $\theta_i : \text{dom}(A_{\alpha_i}) \times \text{dom}(B_{\beta_i}) \rightarrow \{\text{true}, \text{false}\}$, $\forall i = \overline{1, k}$

θ -joinul a două relații r_1 și r_2 , notat cu $r_1 \bowtie_{\theta} r_2$, este definit prin:

$$r_1 \bowtie_{\theta} r_2 = \{(t_1, t_2) | t_1 \in r_1, t_2 \in r_2, t_1[A_{\alpha_i}] \theta_i t_2[B_{\beta_i}], i = \overline{1, k}\}$$

$$\text{unde } \theta = (A_{\alpha_1} \theta_1 B_{\beta_1}) \wedge (A_{\alpha_2} \theta_2 B_{\beta_2}) \wedge \dots \wedge (A_{\alpha_k} \theta_k B_{\beta_k})$$

Dacă θ_i este operatorul de egalitate, atunci θ -joinul se mai numește și **equijoin**.

Operații în algebra relațională - θ -join, equijoin

Observație 1: un join oarecare cu condiția TRUE pentru orice combinație de tuple este un produs cartezian: $r_1 \bowtie_{\text{true}} r_2 = r_1 \times r_2$

Observație 2: Joinul oarecare poate fi considerat ca fiind o filtrare după anumite criterii ale rezultatelor unui produs cartezian:

$$r_1 \bowtie_{\theta} r_2 = \sigma_{\theta}(r_1 \times r_2)$$

Exemplu SQL:

```
SELECT s.nume, p.nume FROM studenti s, profesori p
WHERE s.nume > p.nume;
```

Operații în algebra relațională - *Semijoin*: \bowtie și \bowtie

Operația de **semijoin stâng** selectează acele rânduri din relația aflată în partea stângă (\bowtie) care au corespondent (în sensul joinului natural) în relația din partea dreapta.

Formal, definim semijoinul stâng a două relații r_1 peste $R_1[U_1]$ și r_2 peste $R_2[U_2]$ ca fiind:

$$r_1 \bowtie r_2 = \pi_{U_1}(r_1 \bowtie r_2)$$

Deja întâlnit la proprietăile Joinului natural sub denumirea r'_1 .

Semijoinul drept este definit similar dar preluând liniile din relația aflată în dreapta (doar cele ce au corespondent în relația din stânga).

Operații în algebra relațională - *Antijoin*: \triangleright

Tuplele rămase din relația din stânga (care nu au fost preluate de semijoinul stâng), formează rezultatul operatorului **Antijoin**.

Formal, definim antijoinul stâng a două relații r_1 peste $R_1[U_1]$ și r_2 peste $R_2[U_2]$ ca fiind:

$$r_1 \triangleright r_2 = r_1 - \pi_{U_1}(r_1 \bowtie r_2)$$

$\dots r_1''$

Operații în algebra relațională - *Joinul la Stânga:*

Fie r_1 și r_2 două relații în care nu toate tuplele din r_1 au un corespondent în r_2 .

Operația **Join la Stanga** a celor două relații r_1 și r_2 este reuniunea dintre tuplele existente în $r_1 \bowtie r_2$ și tuplele din r_1 ce nu sunt utilizate în join completate cu valoarea NULL pentru attributele din U_2 .

$$r_1 \bowtie r_2 = r_1 \bowtie r_2 \text{ UNION } \pi_{U_1 U_2}(r_1 - \pi_{U_1}(r_1 \bowtie r_2))$$

Joinul la Dreapta este definit similar, de această dată preluând și liniile ce nu s-au folosit în Joinul natural din tabela din dreapta (r_2).

Operații în algebra relațională - *Joinul Extern:*

Operația de Join exterior cuprinde toate liniile din Joinul la Stânga și din Joinul la Dreapta.

$$r_1 \bowtie\bowtie r_2 = (r_1 \bowtie r_2) \cup (r_1 \bowtie\bowtie r_2)$$

Operații în algebra relațională - *Joinul Extern*:

Exemple:

`SELECT * FROM studenti LEFT JOIN profesori ON
studenti.prenume = profesori.prenume;`

(Toți studenții și asociații cu profesorii cu același prenume când este cazul)

`SELECT * FROM studenti RIGHT JOIN profesori ON
studenti.prenume = profesori.prenume;`

(Unii studenți care sunt asociați cu profesorii având același prenume împreună cu restul profesorilor)

`SELECT * FROM studenti FULL JOIN profesori ON
studenti.prenume = profesori.prenume;`

(Studenții și profesorii și asocierile între ei, dacă există)

Notății (alternative) pentru operatorii din alg. relațională

Proiecția ($\pi_U(r_1)$): $r_1[U]$

Join natural ($r_1 \bowtie r_2$): $r_1 * r_2$

Join oarecare (sau theta-join): $r_1 \bowtie_{\theta} r_2$

Selecția : $\sigma_{\theta}(r_1)$ [obs: $r_1 \bowtie_{\theta} r_2 = \sigma_{\theta}(r_1 \times r_2)$]

Join la stânga: $r_1 \triangleright \circ \triangleleft_L r_2$

Join la dreapta: $r_1 \triangleright \circ \triangleleft_R r_2$

Full outer join : $r_1 \triangleright \circ \triangleleft r_2$

Redenumirea: Dacă r este definit peste B_1, B_2, \dots, B_n și vrem să redenumim numele atributelor, vom folosi operatorul de redenumire ρ : $r' = \rho(r_1)_{A_1, A_2, \dots, A_n}$ - redenumirea atributelor lui r în A_1, A_2, \dots, A_n

Exerciții:

1. Pentru r_1, r_2 exemplificate la Joinul natural, construiți restul tipurilor de Join studiate.
2. Utilizând schema de baze de date de la laborator, scrieți în algebra relațională expresii de selecție pentru următoarele:
 - ▶ Cursurile din facultate împreună cu numele prof. ce le țin.
 - ▶ Numele și prenumele studenților din anul 1 și care au bursă mai mare de 300 ron.
 - ▶ Prenumele studenților care au același nume de familie ca măcar unul din profesori.
 - ▶ Numele și prenumele studenților, cursurile pe care le-au urmat și notele pe care le-au obținut.

Scrieți interogările SQL asociate formulelor din algebra relațională scrise mai sus.

Bibliografie

- ▶ Baze de date relationale. Dependente - *Victor Felea*; Univ. Al. I. Cuza, 1996

Software

► Relational

4 CURS 3

Baze de date Dependențe funcționale

Nicolae-Cosmin Vârlan

October 15, 2020

Egalitatea a două tuple

Considerăm $U = \{A_1, A_2 \dots A_n\}$ o mulțime de atrbute și două tuple t_1 și t_2 construite peste această mulțime de atrbute.

Spunem că *tuplele t_1 și t_2 sunt egale*, dacă și numai dacă

$$\pi_{A_i}[t_1] = \pi_{A_i}[t_2], \forall i \in \{1..n\}$$

Cu alte cuvinte, tuplele t_1 și t_2 sunt egale dacă ele sunt egale pe fiecare dintre componente lor. Considerând că $t_1 = (v_{11}, v_{12}, \dots v_{1n})$ și $t_2 = (v_{21}, v_{22}, \dots v_{2n})$, atunci $t_1 = t_2$ dacă și numai dacă $v_{11} = v_{21}$, $v_{12} = v_{22}$, ..., $v_{1n} = v_{2n}$.

În restul cursului, vom înlocui notația $\pi_X[t]$ cu $t[X]$.

[Jump to Table of contents](#)

Dependențe funcționale

Fie $X, Y \subseteq U$. Vom nota o dependență funcțională cu $X \rightarrow Y$.

O relație r peste U satisface **dependența funcțională** $X \rightarrow Y$ dacă:

$$(\forall t_1, t_2)(t_1, t_2 \in r) t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$$

$X = \emptyset$ avem $\emptyset \rightarrow Y$ dacă $(\forall t_1, t_2)(t_1, t_2 \in r)[t_1[Y] = t_2[Y]]$

$Y = \emptyset$ atunci orice $\forall r$ peste U avem că $X \rightarrow \emptyset$

Dacă r satisface $X \rightarrow Y$, atunci există o funcție $\varphi : r[X] \rightarrow r[Y]$ definită prin $\varphi(t) = t'[Y]$, unde $t' \in r$ și $t'[X] = t \in r[X]$.

Dacă r satisface $X \rightarrow Y$ spunem că X determină funcțional pe Y în r .

Exemplu

Fie relația r peste mulțimea de atrbute

$$U = \{nume, l(num), data_nastere, zodie, varsta\}$$

| | nume | $l(nume)$ | data_nastere | zodie | varsta |
|-------|--------|-----------|--------------|-------|--------|
| $r :$ | Ion | 3 | 20.02.1990 | Pesti | 28 |
| | Vasile | 6 | 24.02.1992 | Pesti | 26 |
| | Maria | 5 | 1.08.2014 | Leu | 4 |
| | Cosmin | 6 | 7.07.1978 | Rac | 40 |
| | Maria | 5 | 4.08.2010 | Leu | 8 |
| | ... | ... | ... | ... | ... |

Puteți depista dependențele funcționale ?

Exemplu

Fie relația r peste mulțimea de atrbute

$$U = \{nume, l(nume), data_nastere, zodie, varsta\}$$

| | <i>nume</i> | <i>l(nume)</i> | <i>data_nastere</i> | <i>zodie</i> | <i>varsta</i> |
|-------|-------------|----------------|---------------------|--------------|---------------|
| $r :$ | Ion | 3 | 20.02.1990 | Pesti | 28 |
| | Vasile | 6 | 24.02.1992 | Pesti | 26 |
| | Maria | 5 | 1.08.2014 | Leu | 4 |
| | Cosmin | 6 | 7.07.1978 | Rac | 40 |
| | Maria | 5 | 4.08.2010 | Leu | 8 |
| | ... | ... | ... | ... | ... |

- ▶ $nume \rightarrow l(nume)$
- ▶ $data_nastere \rightarrow varsta$
- ▶ $data_nastere \rightarrow zodie$
- ▶ $nume \rightarrow zodie$ - *discuție*

Proprietăți ale dependențelor funcționale

FD1. (**Reflexivitate**) Dacă $Y \subseteq X$, atunci r satisfacă $X \rightarrow Y$, $\forall r \in U$.

FD2. (**Extensie**) Dacă r satisfacă $X \rightarrow Y$ și $Z \subseteq W$, atunci r satisfacă $XW \rightarrow YZ$.

FD3. (**Tranzitivitate**) Dacă r satisfacă $X \rightarrow Y$ și $Y \rightarrow Z$, atunci r satisfacă $X \rightarrow Z$.

FD4. (**Pseudotranzitivitate**) Dacă r satisfacă $X \rightarrow Y$ și $YW \rightarrow Z$, atunci r satisfacă $XW \rightarrow Z$.

Proprietăți ale dependențelor funcționale

FD5. (**Uniune**) Dacă r satisface $X \rightarrow Y$ și $X \rightarrow Z$, atunci r satisface $X \rightarrow YZ$.

FD6. (**Descompunere**) Dacă r satisface $X \rightarrow YZ$, atunci r satisface $X \rightarrow Y$ și $X \rightarrow Z$.

FD7. (**Proiectabilitate**) Dacă r peste U satisface $X \rightarrow Y$ și $X \subset Z \subseteq U$, atunci $r[Z]$ satisface $X \rightarrow Y \cap Z$

FD8. (**Proiectabilitate inversă**) Dacă $X \rightarrow Y$ este satisfacută de o proiecție a lui r , atunci $X \rightarrow Y$ este satisfacută de r .

Dependențe funcționale - consecință și acoperire

Dacă Σ este o mulțime de dependențe funcționale peste U atunci spunem că $X \rightarrow Y$ este **consecință din Σ** dacă orice relație ce satisface toate dependențele din Σ satisface și $X \rightarrow Y$.

Notație: $\Sigma \models X \rightarrow Y$

Fie $\Sigma^* = \{X \rightarrow Y | \Sigma \models X \rightarrow Y\}$. Fie Σ_1 = mulțime de dependențe funcționale. Σ_1 constituie o **acoperire** pentru Σ^* dacă $\Sigma_1^* = \Sigma^*$.

Exercițiu: Fie $U = \{A, B, C, D, E, F\}$ și $\Sigma = \{A \rightarrow BD, B \rightarrow C, DE \rightarrow F\}$ găsiți cât mai multe elemente din $\Sigma^* - \Sigma$.

Proprietăți ale dependențelor funcționale

Propoziție

Pentru orice mulțime Σ de dependențe funcționale există o acoperire Σ_1 pentru Σ^* , astfel încat toate dependențele din Σ_1 sunt de forma $X \rightarrow A$, A fiind un atribut din U .

Propoziție

$\Sigma \models X \rightarrow Y$ dacă și numai dacă $\Sigma \models X \rightarrow B_j$ pentru $j = \overline{1, h}$, unde $Y = B_1 \dots B_h$.

Reguli de deducere (la nivel sintactic)

Fie \mathcal{R} o mulțime de reguli de deducere pentru dependențe funcționale și Σ o mulțime de dependențe funcționale.

Spunem că $X \rightarrow Y$ este o **demonstrație** în Σ utilizând regulile \mathcal{R} și vom nota $\Sigma \vdash_{\mathcal{R}} X \rightarrow Y$, dacă există sirul $\sigma_1, \sigma_2, \dots, \sigma_n$, astfel încât:

- ▶ $\sigma_n = X \rightarrow Y$ și
- ▶ pentru $\forall i = \overline{1, n}$, $\sigma_i \in \Sigma$ sau există în \mathcal{R} o regulă de forma $\frac{\sigma_{j_1}, \sigma_{j_2}, \dots, \sigma_{j_k}}{\sigma_i}$, unde $j_1, j_2, \dots, j_k < i$.

Reguli de deducere (la nivel sintactic)

Conform proprietăților FD1-FD6 putem defini regulile:

$$FD1f: \frac{Y \subseteq X}{X \rightarrow Y}$$

$$FD4f: \frac{X \rightarrow Y, YW \rightarrow Z}{XW \rightarrow Z}$$

$$FD2f: \frac{X \rightarrow Y, Z \subseteq W}{XW \rightarrow YZ}$$

$$FD5f: \frac{X \rightarrow Y, X \rightarrow Z}{X \rightarrow YZ}$$

$$FD3f: \frac{X \rightarrow Y, Y \rightarrow Z}{X \rightarrow Z}$$

$$FD6f: \frac{X \rightarrow YZ}{X \rightarrow Y}, \frac{X \rightarrow YZ}{X \rightarrow Z}$$

Propoziție

Regulile FD4f, FD5f, FD6f se exprimă cu ajutorul regulilor FD1f, FD2f, FD3f.

Notăm cu $\mathcal{R}_1 = \{FD1f, FD2f, FD3f\}$,
și cu $\mathcal{R}_2 = \mathcal{R}_1 \cup \{FD4f, FD5f, FD6f\}$

Propoziție

Regulile FD4f, FD5f, FD6f se exprimă cu ajutorul regulilor FD1f, FD2f, FD3f.

Idei de demonstrație:

- ▶ FD4f: Se aplică FD2f pentru $X \rightarrow Y$ și $W \subseteq Y$ iar din rezultat și din $YW \rightarrow Z$ prin FD3f se obține rezultatul;
- ▶ FD5f: Se aplică FD2f pentru $X \rightarrow Y$ și $X \subseteq Y$ și la fel pentru $X \rightarrow Z$ și $Y \subseteq Z$ apoi FD3f (tranzitivitatea) între rezultate;
- ▶ FD6f: din FD1f avem ca $YZ \rightarrow Y$ și $YZ \rightarrow Z$ și din FD3f rezulta $X \rightarrow Y$ și $X \rightarrow Z$

Axiomele lui Armstrong

Armstrong a definit (în *Dependency structures of database relationships* Proc. IFIP 74, Amsterdam, 580-583) următoarele reguli de inferență (numite *Axiomele lui Armstrong*):

$$A1: \frac{}{A_1 \dots A_n \rightarrow A_i}, i = \overline{1, n}$$

$$A2: \frac{A_1, \dots, A_m \rightarrow B_1, \dots, B_r}{A_1 \dots A_m \rightarrow B_j}, j = \overline{1, r}$$

$$\frac{A_1, \dots, A_m \rightarrow B_j, j = \overline{1, r}}{A_1 \dots A_m \rightarrow B_1, \dots, B_r}$$

$$A3: \frac{A_1, \dots, A_m \rightarrow B_1, \dots, B_r, \quad B_1, \dots, B_r \rightarrow C_1, \dots, C_p}{A_1 \dots A_m \rightarrow C_1, \dots, C_p}$$

unde A_i, B_j, C_k sunt attribute. Notăm $\mathcal{R}_A = \{A1, A2, A3\}$.

Obs: regula A3 este de fapt FD3f (tranzitivitatea).

Propoziție

Regulile din \mathcal{R}_1 se exprimă prin cele din \mathcal{R}_A și invers.

Notație:

$$\Sigma_{\mathcal{R}}^+ = \{X \rightarrow Y | \Sigma \vdash_{\mathcal{R}} X \rightarrow Y\}$$

Propoziție

Fie \mathcal{R}'_1 și \mathcal{R}'_2 două multimi de reguli astfel incat \mathcal{R}'_1 se exprima prin \mathcal{R}'_2 și invers. Atunci $\Sigma_{\mathcal{R}'_1}^+ = \Sigma_{\mathcal{R}'_2}^+$ pentru orice multime Σ de dependente funcționale.

Consecinta: $\Sigma_{\mathcal{R}_1}^+ = \Sigma_{\mathcal{R}_A}^+$

~~Fie $X \subseteq U$ și \mathcal{R} o multime de reguli de inferenta. Notam cu~~

$$X_{\mathcal{R}}^+ = \{A \mid \Sigma \vdash_{\mathcal{R}} X \rightarrow A\}$$

Lema

$\Sigma \vdash_{\mathcal{R}} X \rightarrow Y$ daca si numai daca $Y \subseteq X_{\mathcal{R}_1}^+$.

Lema

Fie Σ o multime de dependente functionale si $\sigma : X \rightarrow Y$ o dependenta functionala astfel incat $\Sigma \not\vdash_{\mathcal{R}_1} X \rightarrow Y$. Atunci exista o relatie r_σ ce satisface toate dependentele functionale din Σ si r_σ nu satisface $X \rightarrow Y$.

Theorem

Fie Σ o multime de dependente functionale. Atunci exista o relatie r_0 ce satisface exact elementele lui $\Sigma_{\mathcal{R}_1}^+$, adica:

- ▶ r_0 satisface τ , $\forall \tau \in \Sigma_{\mathcal{R}_1}^+$ si
- ▶ r_0 nu satisface γ , $\forall \gamma \notin \Sigma_{\mathcal{R}_1}^+$

[Jump to Table of contents](#)

Bibliografie

- ▶ Baze de date relaționale. Dependențe - *Victor Felea*; Univ. Al. I. Cuza, 1996

5 CURS 4

Baze de date relaționale

Dependențe multivaluate

Nicolae-Cosmin Vârlan

October 22, 2019

Exemplu

Presupunem că persoana cu CNP = 1 a fost admisă la două facultăți și are permis de conducere pentru categoriile A și B :

| | CNP | Admis la facult. | Are permis categ. |
|-------|-----|------------------|-------------------|
| $r :$ | 1 | Informatică | A |
| | 1 | Matematică | B |

Deși anumite rânduri nu sunt scrise în tabelă, putem să intuim că persoana cu CNP = 1 a dat la Facultatea de Informatică și are permis de conducerea categoria B . Deci, deși în r nu există t -uplu $\langle 1, \text{Informatica}, B \rangle$, ar trebui să existe și el (pentru că poate fi dedus din cele existente).

Care alt t -uplu mai poate fi dedus ?

Exemplu

| | CNP | Admis la facult. | Are permis categ. |
|-------|-----|------------------|-------------------|
| $r :$ | 1 | Informatică | A |
| | 1 | Matematică | B |
| | 1 | Informatică | B |
| | 1 | Matematică | A |

t -uplele marcate cu roșu ar putea lipsi, ele fiind redundante deoarece pot fi obținute din primele două t -uple.

Prin intermediul dependențelor funcționale pot afla la care coloane pot renunța astfel încât să le pot reface ulterior.

Prin intermediul dependențelor multivaluate pot afla la care linii pot renunța astfel încât să le pot reface ulterior.

Dependențe multivaluate - definiție

Fie $X, Y \subseteq U$. O dependență multivaluată este notată cu $X \twoheadrightarrow Y$.

Definition

Relația r peste U satisfacă dependența multivaluată $X \twoheadrightarrow Y$ dacă pentru oricare două tuple $t_1, t_2 \in r$ satisfăcând $t_1[X] = t_2[X]$, există tuplele t_3 și t_4 din r , astfel încât:

- ▶ $t_3[X] = t_1[X], t_3[Y] = t_1[Y], t_3[Z] = t_2[Z];$
- ▶ $t_4[X] = t_2[X], t_4[Y] = t_2[Y], t_4[Z] = t_1[Z]$

unde $Z = U - XY$ (Z mai este denumită și *rest*).

Exemplul 2 (mai formal)

| A | B | C | D | | | |
|-------|-------|-------|-------|-------|-------|--------------|
| a_1 | b_1 | c_1 | d_1 | t_1 | | t_1'' |
| a_1 | b_2 | c_2 | d_2 | t_2 | | |
| $r :$ | a_1 | b_1 | c_1 | d_2 | t_3 | t_2'' |
| | a_1 | b_2 | c_2 | d_1 | t_4 | |
| | a_2 | b_3 | c_1 | d_1 | | t'_1, t'_4 |
| | a_2 | b_3 | c_1 | d_2 | | t'_2, t'_3 |

Intrebare: cum alegem t_3'' , t_4'' ?

Deoarece atunci când $t_1[A] = t_2[A]$ avem că:

$t_3[A] = t_1[A], t_3[BC] = t_1[BC], t_3[D] = t_2[D]$ și

$t_4[A] = t_2[A], t_4[BC] = t_2[BC], t_4[D] = t_1[D]$

Definiție echivalentă

Relația r peste U *satisfacă dependența multivaluată* $X \twoheadrightarrow Y$, dacă pentru orice $t_1, t_2 \in r$ cu $t_1[X] = t_2[X]$ avem că $M_Y(t_1[XZ]) = M_Y(t_2[XZ])$

unde $M_Y(t[XZ]) = \{t'[Y] | t' \in r, t'[XZ] = t[XZ]\}$ = valorile lui Y din diferite tuple în care XZ sunt egale (cu XZ -ul din parametru).

| A | B | C | D | |
|-------|-------|-------|-------|---------|
| a_1 | b_1 | c_1 | d_1 | = t_1 |
| a_1 | b_2 | c_2 | d_2 | = t_2 |
| $r :$ | a_1 | b_1 | c_1 | d_2 |
| | a_1 | b_2 | c_2 | d_1 |
| | a_2 | b_3 | c_1 | d_1 |
| | a_2 | b_3 | c_1 | d_2 |

$M_Y(t1[AD]) = M_Y(t2[AD]) = \{(b_1, c_1), (b_2, c_2)\}$

Observații

- ▶ Dacă r satisface dependența funcțională $X \rightarrow Y$, atunci pentru orice $t \in r$, avem $M_Y(t[XZ]) = \{t[Y]\}$.
- ▶ Dacă r satisface dependența funcțională $X \rightarrow Y$, atunci r satisface și dependența multivaluată $X \rightarrow Y$.
- ▶ Dacă r satisface dependența multivaluată $X \rightarrow Y$, atunci putem defini o funcție $\psi : r[X] \rightarrow \mathcal{P}(r[Y])$, prin $\psi(t[X]) = M_Y(t[XZ])$, $\forall t \in r$ (returnează valorile diferite din proiecția pe Y). Când r satisface $X \rightarrow Y$, atunci $\psi : r[X] \rightarrow r[Y]$ (deoarece valorile pe Y nu sunt diferite în cadrul dependenței funcționale).

Dependențe multivaluate (exercițiu)

Arătați că $AC \rightarrow BD$:

| | A | B | C | D | E |
|-------|---|---|---|---|---|
| $r :$ | 8 | 1 | 2 | 0 | 4 |
| | 8 | 9 | 2 | 2 | 9 |
| | 9 | 3 | 2 | 4 | 9 |
| | 8 | 1 | 2 | 0 | 9 |
| | 8 | 9 | 2 | 2 | 4 |
| | 9 | 3 | 2 | 4 | 4 |

Când $r[AC] = \{(8, 2)\}$ avem $r[BD] = \{(1, 0), (9, 2)\}$ și $r[E] = \{(4), (9)\}$. Găsim toate produsele carteziene dintre cele 3 ?

Când $r[AC] = \{(9, 2)\}$ avem $r[BD] = \{(3, 4)\}$ și $r[E] = \{(4), (9)\}$. Găsim toate produsele carteziene ?

DA (este MVD)

Proprietăți ale dependențelor multivaluate

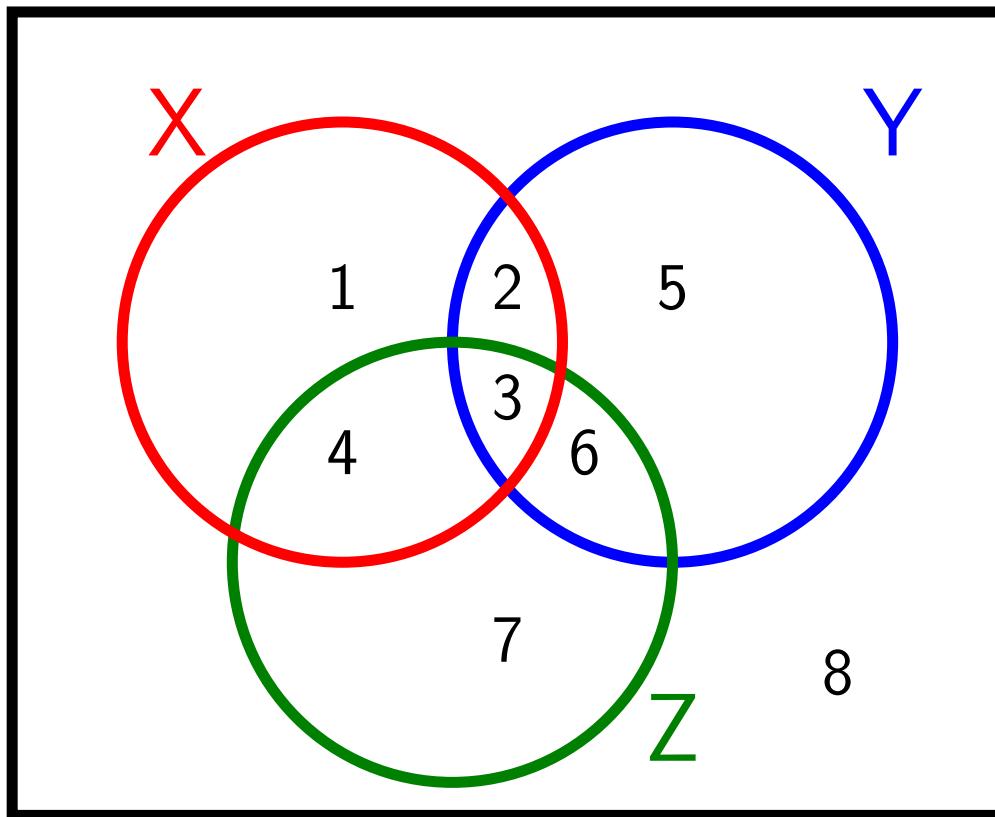
MVD0 ([Complementariere](#)) Fie $X, Y, Z \subseteq U$, astfel încât $XYZ = U$ și $Y \cap Z \subseteq X$. Dacă r satisface $X \twoheadrightarrow Y$, atunci r satisface $X \twoheadrightarrow Z$.

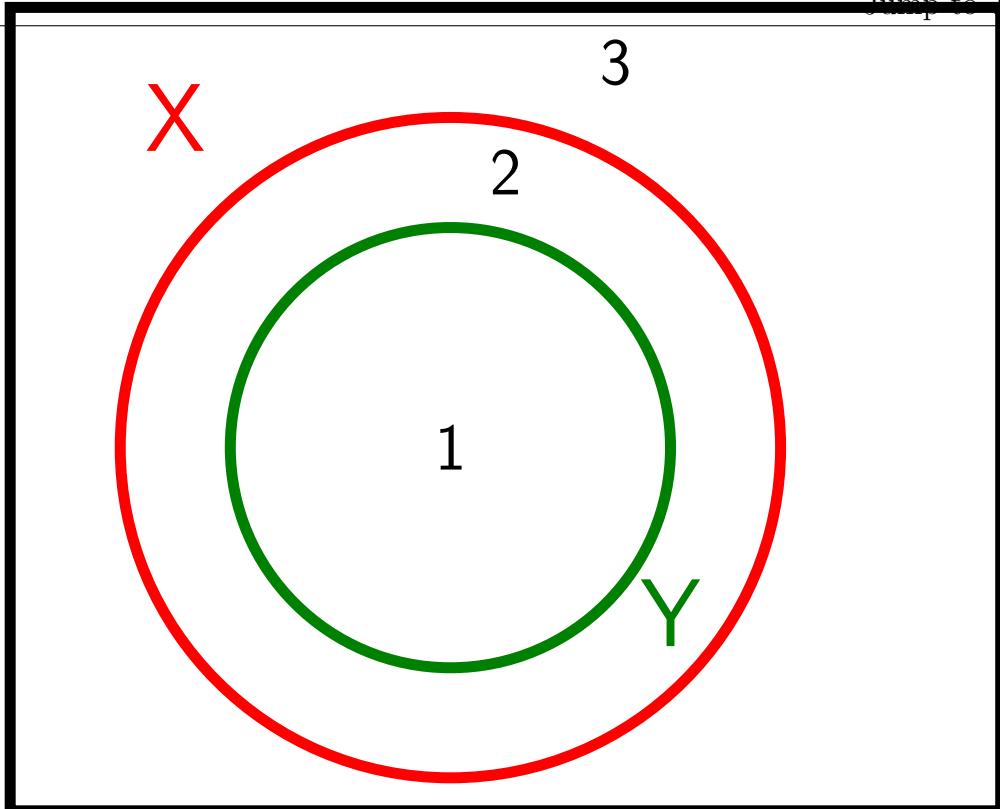
MVD1 ([Reflexivitate](#)) Dacă $Y \subseteq X$, atunci orice relație r satisface $X \twoheadrightarrow Y$.

MVD2 ([Extensie](#)) Fie $Z \subseteq W$ și r satisface $X \twoheadrightarrow Y$. Atunci r satisface $XW \twoheadrightarrow YZ$

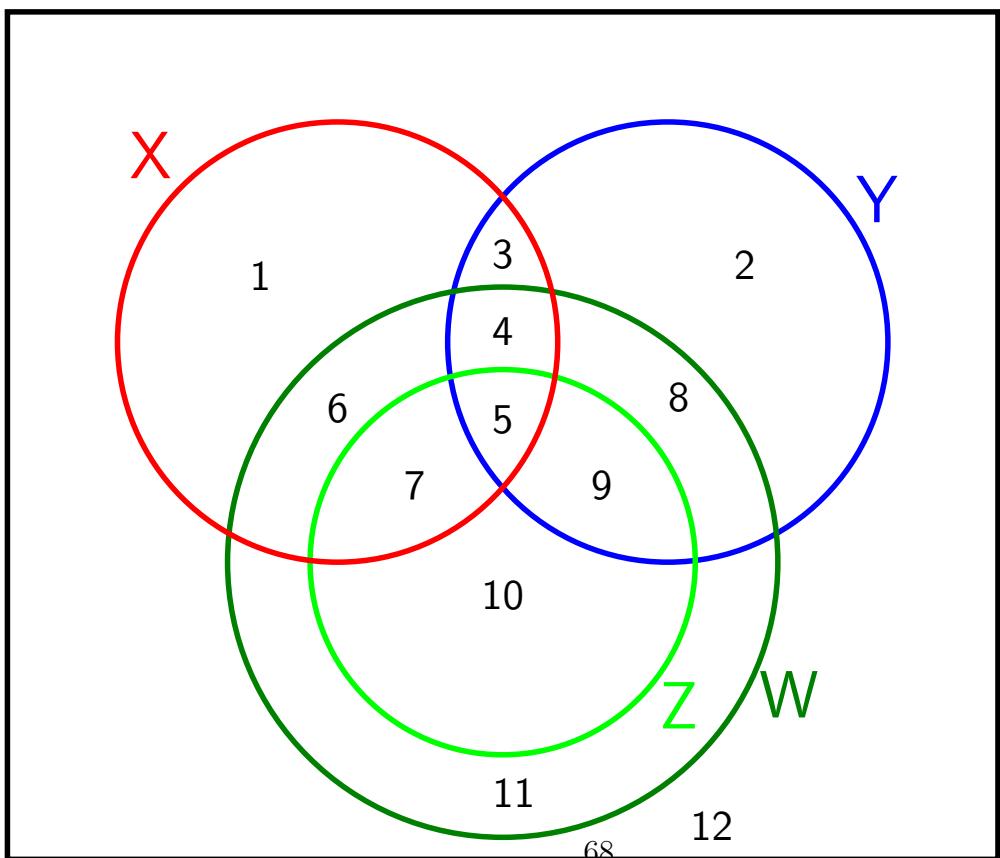
MVD3 ([Tranzitivitate](#)) Dacă r satisface $X \twoheadrightarrow Y$ și $Y \twoheadrightarrow Z$, atunci r satisface $X \twoheadrightarrow Z - Y$

MVD0





MVD2



Proprietăți ale dependențelor multivaluate

MVD4 ([Pseudotranzitivitate](#)) Dacă r satisfacă $X \twoheadrightarrow Y$ și $YW \twoheadrightarrow Z$, atunci r satisfacă și $XW \twoheadrightarrow Z - YW$.

MVD5 ([Uniune](#)) Dacă r satisfacă $X \twoheadrightarrow Y$ și $X \twoheadrightarrow Z$ atunci r satisfacă $X \twoheadrightarrow YZ$.

MVD6 ([Descompunere](#)) Dacă r satisfacă $X \twoheadrightarrow Y$ și $X \twoheadrightarrow Z$, atunci r satisfacă $X \twoheadrightarrow Y \cap Z$, $X \twoheadrightarrow Y - Z$, $X \twoheadrightarrow Z - Y$

Proprietăți mixte ale dependențelor multivaluate

FD-MVD1. Dacă r satisfacă $X \rightarrow Y$, atunci r satisfacă și $X \twoheadrightarrow Y$.

FD-MVD2. Dacă r satisfacă $X \twoheadrightarrow Z$ și $Y \rightarrow Z'$, cu $Z' \subseteq Z$ și $Y \cap Z = \emptyset$, atunci r satisfacă $X \rightarrow Z'$.

FD-MVD3. Dacă r satisfacă $X \twoheadrightarrow Y$ și $XY \twoheadrightarrow Z$, atunci r satisfacă $X \rightarrow Z - Y$.

[Jump to Table of contents](#)

Reguli de inferență

$$\text{MVD0f: } \frac{XYZ=U, Y \cap Z \subseteq X, X \twoheadrightarrow Y}{X \twoheadrightarrow Z}$$

$$\text{MVD1f: } \frac{Y \subseteq X}{X \twoheadrightarrow Y}$$

$$\text{MVD2f: } \frac{Z \subseteq W, X \twoheadrightarrow Y}{XW \twoheadrightarrow YZ}$$

$$\text{MVD3f: } \frac{X \twoheadrightarrow Y, Y \twoheadrightarrow Z}{X \twoheadrightarrow Z - Y}$$

$$\text{MVD4f: } \frac{X \twoheadrightarrow Y, YW \twoheadrightarrow Z}{XW \twoheadrightarrow Z - YW}$$

Reguli de inferență

$$\text{MVD5f: } \frac{X \twoheadrightarrow Y, X \twoheadrightarrow Z}{X \twoheadrightarrow YZ}$$

$$\text{MVD6f: } \frac{X \twoheadrightarrow Y, X \twoheadrightarrow Z}{X \twoheadrightarrow Y \cap Z, X \twoheadrightarrow Y - Z, X \twoheadrightarrow Z - Y}$$

$$\text{FD-MVD1f: } \frac{X \rightarrow Y}{X \twoheadrightarrow Y}$$

$$\text{FD-MVD2f: } \frac{X \twoheadrightarrow Z, Y \rightarrow Z', Z' \subseteq Z, Y \cap Z = \emptyset}{X \rightarrow Z'}$$

$$\text{FD-MVD3f: } \frac{X \twoheadrightarrow Y, XY \twoheadrightarrow Z}{X \twoheadrightarrow Z - Y}$$

Propoziție

Fie \mathcal{R} o multime de reguli valide si γ o regula $\frac{\alpha_1, \alpha_2, \dots, \alpha_k}{\beta}$, astfel incat $\{\alpha_1, \dots, \alpha_k\} \vdash_{\mathcal{R}} \beta$, atunci si regula γ este valida.

Propoziție

Fie $\mathcal{R}_{FM} = \{FD1f - FD3f^1, MVD0f - MVD3f, FD - MVD1f - FD - MVD3f\}$. Avem:

- ▶ $FD - MVD3f$ se exprima cu celelalte reguli din \mathcal{R}_{FM} si FD
- ▶ $MVD2f$ se exprima prin celelalte reguli din \mathcal{R}_{FM} .

Propoziție

Regulile $MVD4f - MVD6f$ se exprima cu ajutorul regulilor $MVD0f - MVD3f$

¹cele de la dependente funktionale

Bibliografie

- ▶ Baze de date relaționale. Dependențe - Victor Felea; Univ. Al. I. Cuza, 1996

6 CURS 5-6: NF-1, NF-2, BCNF, JOIN-uri, 4-NF

1-NF ... pagina 82

NU este in 1-NF daca intr-o casuta din relatia R sunt enumerate mai multe valori

PASI (ca sa aduci tabelul in 1-NF):

1. se gaseste o cheie primara
2. se separa tabelul in mai multe tabele cu anumite atribute:
 - x = cate atribute cu elemente repetitive sunt in R
 - separam in $x + 1$ tabele
 - in x tabele avem cheia primara si cate unul din atributele cu elemente repetitive
 - intr-un tabel va fi cheia primara + retul atributelor fara elemente repetitive

2-NF ... pagina 84

- trebuie sa fie in 1-NF
- orice atribut neprim din R este dependent plin de orice cheie candidat a lui R.

VERIFICARE (daca este in 2-NF):

1. verificare 1-NF
 2. se gasesc cheile candidat
 3. se scriu atributele prime si cele neprime
 4. se verifica daca TOATE atributele NEPRIME sunt dependente pline de cheile candidat
- PASI** (aducere in 2NF):

1. pentru fiecare din atributele neprime A identificam care sunt atributele dintr-o cheie de care depinde A ($B \rightarrow A$, B inclus intr-o cheie candidat)
2. se ceaza o noua relatie R_0 peste acele atribute identificate la pasul anterior impreuna cu atributul neprim pentru care s-a gasit dependenta

3. se repeta algoritmul pentru relatiile R_0, R_1, \dots gasite pana cand toate sunt in 2-NF
obs :

1. daca atribute prime = multimea vida \Rightarrow R este in 2-NF si 3-NF
2. daca nu avem chei multivaluate \Rightarrow R este in 2-NF

3-NF ... pagina 87

- trebuie sa fie in 2-NF
- orice atribut neprim din R *NU* este tranzitiv dependent de nici o cheie a lui R
- (sau) orice atribut neprim depinde de chei si nu de un alt atribut neprim sau grupare de atribute neprime

VERIFICARE (daca este in 3-NF):

1. verificare 2-NF
 2. se gasesc cheile candidat
 3. se scriu atributele prime si cele neprime
 4. se verifica daca sunt (grupari de) atribute neprime dependente tranzitiv de (una sau mai multe) chei candidat; daca nu sunt \Rightarrow este in 3-NF
- PASI** (aducere in 3-NF):

1. fie A = multimea de atribute tranzitiv dependente; se desparte tabelul in 2 relatii, una cu atributele din A si una cu atributele ramase
2. se verifica relatiile create daca sunt si ele in 3-NF; daca nu, se repeta algoritmul

BCNF ... pagina 88

- trebuie sa fie in 1-NF
- orice dependenta functionala netriviala X , $X \in \Sigma^+$, X este supercheie in R

VERIFICARE (daca este in BCNF):

1. verificare 1-NF
2. se gasesc cheile candidat
3. se scriu atributele prime si cele neprime
4. toate dependentele sunt de forma stanga \rightarrow dreapta; stanga=supercheie=contine una din cheile candidat

PASI (aducere in BCNF):

1. se face o relatie cu atributele din dependentele care nu respecta (4) si o relatie cu toate atributele (in afara de dreapta de la dependentele care nu respecta (4))
2. se verifica relatiile create daca sunt si ele in BCNF; daca nu, se repeta algoritmul
obs :

- daca este in BCFN \Rightarrow este si in 3-NF

Descompunerea de tip join fara pierdere ... pagina 89

- asemanator cu descompunerea de la 3-NF
- conditie:

$$R_1 \cap R_2 \rightarrow R_1 - R_2 \in \Sigma^+ \text{ sau } R_1 \cap R_2 \rightarrow R_2 - R_1 \in \Sigma^+$$

- se noteaza cu ρ

PASI:

1. se gasesc cheile candidat
2. toate dependentele sunt de forma stanga \rightarrow dreapta; stanga=supercheie=contine una din cheile candidat
3. se face o relatie cu atributele din dependentele care nu respecta (2) si o relatie cu toate atributele (in afara de dreapta de la dependentele care nu respecta (2))
4. se verifica conditia pe cele 2 relatii
5. daca nu se respecta conditia , se revine la pasul (3) si se incearca alta dependenta care nu respecta (2)

4-NF ... pagina 93

- trebuie sa fie in 1-NF

- orice dependenta multivaluata netriviala $X \rightarrow\rightarrow A \in \Delta^+$, X este supercheie pentru R.

VERIFICARE (daca este in 4-NF):

1. verificare 1-NF
2. se gasesc cheile candidat (din sigma)
3. calculam Σ^+ si Δ^+
 - $\Sigma^+ = \Sigma \cup \Sigma^* \cup$ triviale
 - $\Delta^+ = \Delta \cup \Delta^* \cup$ triviale
 - $\Delta^* =$ dependentele din Σ cu sageata dubla + complementarele lor + ce se mai poate deduce din Δ

4. toate dependentele sunt de forma stanga $\rightarrow\rightarrow$ dreapta; stanga=supercheie=contine una din cheile candidat

PASI (aducere in 4NF):

1. se iau pe rand dependentele multivaluate din Δ^+
2. in R_i se pun atributele din dependenta dupa care se face descompunerea si in R_{i+1} se pun atributele din multimea initiala R pentru care s-a facut descompunerea MAI PUTIN atributele din dreapta multivaluatei
3. se noteaza Δ_i si Σ_i pentru retaliile R_i notate mai sus
4. daca $\Delta_i = \emptyset$ SAU in stanga multivaluatelor din Δ_i sunt superchei ale cheii candidat (adica cheia candidat sau cheia candidat + alte atrbute) atunci R_i este in $4 - NF$
5. daca pasul 4 nu este indeplinit se marcheaza relatia R_i si se revine asupra ei pentru descompunere (de la pasul 1)

Baze de date relaționale

Normalizarea bazelor de date

Nicolae-Cosmin Vârlan

November 14, 2019

Modelul relațional - chei (recapitulare de la primul curs)

- ▶ **Supercheie** - un atribut sau o mulțime de attribute care identifică unic un tuplu într-o relație
- ▶ **Chei candidat** - o supercheie cu proprietatea că nici o submulțime proprie a sa nu este supercheie
- ▶ **Chei primară** - o cheie candidat selectată pentru a identifica în mod unic tuplele într-o relație
- ▶ **Chei alternativă** - Chei candidat care nu au fost selectate pentru a juca rolul de cheie primară
- ▶ **Chei străină** - un atribut sau o submulțime de attribute dintr-o relație care face referință la o cheie candidat a altrei relații

Găsirea cheilor candidat utilizând dependențele funcționale

Intuitiv: Cheia candidat, este de fapt formată dintr-o combinație de atribut care pot determina unic linia. Dacă pot determina unic linia (deci oricare dintre valorile celorlalte atribut), atunci putem considera că avem o dependență funcțională de la $X \subseteq U$ către toate atributele din U atunci X este cheie în orice relație r construită peste $R[U]$.

Formal: Fie $R[U]$ o schemă de relație și Σ o mulțime de dependențe funcționale satisfăcute de $R[U]$. $X \subseteq U$ este cheie candidat dacă $X^+ = U$ și $\forall X' \subseteq X, X'^+ \neq U$

Un atribut se numește **prim** dacă face parte dintr-o cheie candidat.

Un atribut este **neprim** dacă nu este parte din nicio cheie candidat.

... reminder (din cursul 2)

Fie $X \subseteq U$ și \mathcal{R}_A regulile de inferență ale lui Armstrong. Notăm cu

$$X_{\mathcal{R}_A}^+ = \{A | \Sigma \vdash_{\mathcal{R}_A} X \rightarrow A\}$$

Regulile de inferență ale lui Armstrong:

$$A1: \frac{}{A_1 \dots A_n \rightarrow A_i}, i = \overline{1, n}$$

$$A2: \frac{A_1, \dots, A_m \rightarrow B_1, \dots, B_r}{A_1 \dots A_m \rightarrow B_j}, j = \overline{1, r}$$

$$\frac{A_1, \dots, A_m \rightarrow B_j, j = \overline{1, r}}{A_1 \dots A_m \rightarrow B_1, \dots, B_r}$$

$$A3: \frac{A_1, \dots, A_m \rightarrow B_1, \dots, B_r, \quad B_1, \dots, B_r \rightarrow C_1, \dots, C_p}{A_1 \dots A_m \rightarrow C_1, \dots, C_p}$$

Revenim - Exemplul 1:

Să considerăm $U = \{A, B, C\}$ și $\Sigma = \{A \rightarrow B, B \rightarrow C\}$. Vom construi mulțimea $X_{\mathcal{R}_A}^+$ pentru fiecare dintre atrbute:

$$\begin{aligned} A_{\mathcal{R}_A}^+ &= \{A, B, C\} & (A \text{ din } A_1, B \text{ din } A \rightarrow B, \\ && C \text{ din } A \rightarrow B, B \rightarrow C \text{ și folosind } A_3) \\ B_{\mathcal{R}_A}^+ &= \{B, C\} & (B \text{ din } A_1, C \text{ din } B \rightarrow C) \\ C_{\mathcal{R}_A}^+ &= \{C\} & (C \text{ din } A_1) \end{aligned}$$

Se observă că A este cheie candidat pentru că de el depind (funcțional) celelalte atrbute.

Atribute prime: $\{A\}$

Atribute neprime: $\{B, C\}$

Să considerăm $U = \{A, B, C\}$ și $\Sigma = \{A \rightarrow B, B \rightarrow C\}$. Vom construi mulțimea $X_{\mathcal{R}_A}^+$ pentru fiecare dintre atrbute:

Putem organiza atrbutele ținând cont de locul unde apar ele în cadrul dependențelor din Σ :

- ▶ Stânga: Apar numai în partea stangă a dependențelor din Σ .
- ▶ Mijloc: Apar și în stânga și în dreapta dependențelor din Σ .
- ▶ Dreapta: Apar numai în partea dreaptă în dependențele din Σ .

| Stânga | Mijloc | Dreapta |
|--------|--------|---------|
| A | B | C |

Regulă: întotdeauna atrbutele din Stânga sunt atrbute prime, cele din Dreapta sunt neprime. Cele din Mijloc pot fi în oricare dintre categorii.

Exemplul 2:

Fie $U = \{A, B, C, D, E, F\}$ și

$\Sigma = \{A \rightarrow BD, B \rightarrow C, DE \rightarrow F\}$. Care sunt cheile candidat ?

Exemplul 2:

Fie $U = \{A, B, C, D, E, F\}$ și

$\Sigma = \{A \rightarrow BD, B \rightarrow C, DE \rightarrow F\}$. Care sunt cheile candidat ?

| Stânga | Mijloc | Dreapta |
|--------|--------|---------|
| A, E | B, D | C, F |

$A_{\mathcal{R}_A}^+ = \{A, B, C, D\}$ - nu e cheie (nu conține F), dar cu siguranță apare în orice cheie.

De fapt, am stabilit că fiecare cheie candidat va conține toate atributele din "Stânga" - în cazul nostru pe A și pe E:

$$AE_{\mathcal{R}_A}^+ = \{A, B, C, D, E, F\}$$

AE = **cheie multivaluată** (este compusă din mai multe attribute).

Exemplul 3:

Fie $U = \{A, B\}$ și $\Sigma = \{A \rightarrow B, B \rightarrow A\}$. Care sunt cheile candidat?

Exemplul 3:

Fie $U = \{A, B\}$ și $\Sigma = \{A \rightarrow B, B \rightarrow A\}$. Care sunt cheile candidat?

| | | |
|--------|--------|---------|
| Stânga | Mijloc | Dreapta |
| A, B | | |

$$A_{R_A}^+ = \{A, B\}$$

$$B_{R_A}^+ = \{A, B\}$$

Ambele sunt chei candidat.

Atribute prime: $\{A, B\}$

Atribute neprime: \emptyset

Dependențe pline

Fie $R[U]$ o schema de relație peste o mulțime de atrbute U și Σ o mulțime de dependențe funcționale ce au loc în $R[U]$. O dependență $X \rightarrow A \in \Sigma^+$ se numește **plină** dacă $\nexists X' \subset X$ astfel încât $X' \rightarrow A \in \Sigma^+$.

Exemplu:

$R[A, B, C, D]$

$\Sigma = \{AB \rightarrow C, B \rightarrow D, BC \rightarrow A\}$

Toate dependențele din Σ sunt pline (e.g. nu avem în Σ^+ nici una dintre dependențele: $A \rightarrow C, B \rightarrow C, B \rightarrow A, C \rightarrow A$).

$AB \rightarrow D$ nu este plină pentru că $B \subset AB$ și avem $B \rightarrow D \in \Sigma^+$

Atribut tranzitiv dependent

Fie $R[U]$ o schemă de relație peste o mulțime de atrbute U și Σ o mulțime de dependențe funcționale ce au loc în $R[U]$. Un atrbut $A \in U$ se numește **tranzitiv dependent** de X ($X \subset U, A \notin X$) dacă $\exists Y \subset U$ astfel încât:

- ▶ $A \in U - Y$
- ▶ $X \rightarrow Y \in \Sigma^+$
- ▶ $Y \rightarrow A \in \Sigma^+$
- ▶ $Y \rightarrow X \notin \Sigma^+$

Exemplu:

$R[A, B, C, D, E]$

$\Sigma = \{AB \rightarrow C, AB \rightarrow D, CD \rightarrow E\}$

E este tranzitiv dependent de AB ($X = AB, Y = CD$).⁸⁰

Dependențe triviale

O dependență funcțională $X \rightarrow Y$ este trivială dacă și numai dacă $Y \subseteq X$

O dependență multivaluată $X \twoheadrightarrow Y$ este trivială dacă $Y \subseteq X$ sau dacă $Z = \emptyset$ ($X \cup Y = U$)

Normalizarea schemelor bazelor de date

Normalizarea este necesară din două motive:

- ▶ Pentru eliminarea redundanțelor.
- ▶ Pentru a păstra datele într-o manieră consistentă.

Normalizarea trebuie făcută încă din faza de proiectare a bazei de date și din acest motiv este firesc să vorbim despre **NORMALIZAREA SCHEMEI** bazei de date și nu despre normalizarea anumitor relații.

Există mai multe forme normale (clasice), fiecare aducând ceva în plus față de forma precedentă: 1NF, 2NF, 3NF, BCNF, 4NF.

Să vedem în ce constau...

1NF (1971)

Fie U o mulțime de attribute și $R[U]$ o schema de relație.

Spunem că schema de relație $R[U]$ este în Forma Normală 1 (**1NF**) dacă și numai dacă domeniile atributelor sunt indivizibile și fiecare valoare a fiecărui atribut este atomică.

Pentru a avea o relație în 1NF, trebuie efectuate următoarele operații:

- ▶ Eliminarea grupurilor repetitive din fiecare relație.
- ▶ Identificarea tuplelor ce ar putea avea duplicate în coloană printr-o cheie.
- ▶ Crearea unei noi scheme de relație având ca attribute: identifierul de la punctul precedent și valoarea repetată ca atribut atomic.

1NF

Exemplu:

Avem schema: Studenti[nr_matricol, nume, prenume, an] în care studenții ar putea avea câte două prenume:

| $r :$ | nr_matricol | nume | prenume | an |
|-------|-------------|-----------|-------------|----|
| | 1 | Ionescu | Maria Ioana | 1 |
| | 2 | Popescu | Vasile | 1 |
| | 3 | Vasilescu | Vali Cristi | 2 |

Pasul 1: Eliminăm grupul repetitiv:

| $r :$ | nr_matricol | nume | an |
|-------|-------------|--------------|----|
| | 1 | Ionescu | 1 |
| | 2 | Popescu | 1 |
| | 3 | 82 Vasilescu | 2 |

1NF

Pasul 2: Identificam cheia:

| | nr_matricol | nume | an |
|-------|-------------|-----------|----|
| $r :$ | 1 | Ionescu | 1 |
| | 2 | Popescu | 1 |
| | 3 | Vasilescu | 2 |

Pasul 3: Creare relație în care fiecare valoare apare pe un rând nou și e legată de relația principală prin intermediul cheii:

| | nr_matricol | prenume |
|--------|-------------|---------|
| $r' :$ | 1 | Maria |
| | 1 | Ioana |
| | 2 | Vasile |
| | 3 | Vali |
| | 3 | Cristi |

1NF

Deși ar putea părea că am utilizat efectiv relația care este construită peste $R[U]$, în fapt nu am facut decât să modificăm schema de relație $R[U]$ și să construim o nouă schemă de relație în schema bazei noastre de date denumita $R'[U]$.

Pasul 1: de fapt a eliminat atributul “prenume” din mulțimea U .

Pasul 2: de fapt a identificat o cheie candidat (ce se poate face direct din schemă - vezi cum am gasit o cheie candidat din dependențele funcționale).

Pasul 3: de fapt a construit o nouă schemă de relație $R'[U]$

S-ar putea că în unele locuri să întâlniți ideea că relația este într-o anumită formă normală. Aceasta idee este incorectă datorită faptului că normalizarea se realizează înainte de a crea baza de date, în stadiul de proiectare a acesteia !

2NF (1971)

O schema de relație $R[U]$ care este în 1NF, împreună cu o mulțime de dependențe funcționale Σ este în a doua formă normală (2NF) dacă orice atribut neprim din $R[U]$ este dependent plin de orice cheie a lui $R[U]$.

Să reluăm exemplul de la dependențe pline:

$R[A, B, C, D]$

$\Sigma = \{AB \rightarrow C, B \rightarrow D, BC \rightarrow A\}$

1. Găsiți cheile
2. Găsiți atributele neprime
3. Atributele neprime sunt dependente plin de cheile găsite ?

2NF

Să reluam exemplul de la dependențe pline:

$R[A, B, C, D]$

$\Sigma = \{AB \rightarrow C, B \rightarrow D, BC \rightarrow A\}$

Posibilele chei: AB, BC

Atribute prime: A, B, C

Atribute neprime: D

$B \rightarrow D \in \Sigma^+$. D nu este dependent plin de AB pentru că, deși avem $AB \rightarrow D \in \Sigma^+$, avem și $B \rightarrow D \in \Sigma^+$ rezultă că $R[U]$ împreună cu Σ nu este în 2NF.

Observație: dacă nu avem chei multivalue, cu siguranță $R[U]$ este în 2NF (pentru că avem numai dependențe pline de la chei - nu putem găsi submulțimi de attribute atunci când cheia este formată dintr-un singur attribut).

2NF

Având o schemă de relație $R[U]$ care nu este în 2NF, putem să o ducem în 2NF urmând următorii pași:

Pasul 1: Identificăm cheile candidat.

Pasul 2: Găsim attributele neprime.

Pasul 3: Pentru fiecare din attributele neprime A identificăm care sunt attributele dintr-o cheie de care depinde A .

Pasul 4: Creăm o nouă relație R' peste acele attribute identificate la pasul anterior împreună cu atributul neprim pentru care s-a găsit dependența.

Urmați cei patru pași pentru exemplul anterior care nu era în 2NF.

2NF

Exemplu: Dacă avem schema de relație
 $OS[ume, versiune, an, companie]$

| ume | versiune | an | companie |
|---------|---------------|------|-----------|
| Windows | XP | 2001 | Microsoft |
| MacOS | Sierra | 2017 | Apple |
| Ubuntu | Bionic Beaver | 2018 | Ubuntu |
| Windows | 7 | 2009 | Microsoft |
| MacOS | Mojave | 2018 | Apple |

De obicei, cand ne exprimăm, spunem că Windows XP este facut de Microsoft. Avem dependență: $ume, versiune \rightarrow companie$. În același timp, știm că Windows este făcut numai de Microsoft și MacOS numai de Apple. Deci avem și $ume \rightarrow companie$. Fiecare versiune este făcută într-un anumit an. Avem: $versiune \rightarrow an$.

2NF

dependențe:

- ▶ **nume,versiune → companie**
- ▶ **nume → companie**
- ▶ **versiune → an**

Cheie: (nume, versiune) (apar în stânga dependentelor)

Atribute neprime: companie

companie nu este dependent plin pentru că, deși avem **nume,versiune → companie**, totodată avem și **nume → companie**. Vom elimina din schema OS atributul companie și vom adăuga o schemă de relație $R'[nume, companie]$.

Vom avea aşadar...

2NF

| nume | versiune | an |
|---------|---------------|------|
| Windows | XP | 2001 |
| MacOS | Sierra | 2017 |
| Ubuntu | Bionic Beaver | 2018 |
| Windows | 7 | 2009 |
| MacOS | Mojave | 2018 |

și

| nume | companie |
|---------|-----------|
| Windows | Microsoft |
| MacOS | Apple |
| Ubuntu | Ubuntu |

Este în 2NF ? Mai avem de verificat dacă an este dependent plin sau nu... să vedem cum faceți asta voi :D

3NF (1971)

Schema de relație $R[U]$ împreună cu mulțimea de dependențe funcționale Σ este în forma a treia normală (**3NF**) dacă este în 2NF și orice atribut neprim din R **NU** este tranzitiv dependent de nici o cheie a lui R.

(Adică e în 2NF și orice atribut neprim depinde de chei și nu de un alt atribut neprim sau grupare de atribute neprime)

Exemplu:

Considerăm schema de relație $R[A, B, C]$ și

$$\Sigma = \{AB \rightarrow C, C \rightarrow A\}$$

Chei: AB, BC

Atribute neprime: \emptyset - deci este în 2NF și în 3NF.

3NF - glumița de pe wikipedia....

An approximation of Codd's definition of 3NF, paralleling the traditional pledge to give true evidence in a court of law, was given by Bill Kent: *[Every] non-key [attribute] must provide a fact about the key, the whole key, and nothing but the key.*¹

A common variation supplements this definition with the oath: *so help me Codd*².

¹Kent, William - A Simple Guide to Five Normal Forms in Relational Database Theory, Communications of the ACM 26 (2), Feb. 1983, pp. 120 – 125.
87

²Diehr, George. Database Management (Scott, Foresman, 1989), p. 331.

3NF

Exemplu de normalizare 3NF

Fie schema: *Concursuri[materie, an, castigator, IQ]*.

Avem dependențele funcționale:

$$\Sigma = \{(materie, an) \rightarrow castigator, castigator \rightarrow IQ\}$$

Cheie primara: *(materie, an)*

Atribute neprime: *castigator, IQ* (*IQ* este tranzitiv dependent)

Se observă că $(materie, an) \rightarrow IQ \in \Sigma^+$, și în același timp $materie \rightarrow IQ \notin \Sigma^+$ respectiv $an \rightarrow IQ \notin \Sigma^+$. Deci schema de relație se află în 2NF (atributele neprime fiind dependente plin de chei).

Vom normaliza schema pentru

Concursuri[materie, an, castigator, IQ] prin construirea a două scheme diferite: *Concursuri[materie, an, castigator]* și *Inteligenta[castigator, IQ]*.

BCNF (Boyce-Codd Normal Form) \equiv 3.5NF (1975)

O schemă de relație $R[U]$ împreună cu o mulțime de dependențe Σ este în BCNF dacă este în 1NF și pentru orice dependență funcțională netrivială $X \rightarrow A \in \Sigma^+$, X este supercheie în $R[U]$.

Observație: O schemă de relație ce este în BCNF este în 3NF.

Obs: O schemă ce este în BCNF este și în 3NF.

[Jump to Table of contents](#)

Consideram (R, Σ) în BCNF - în 1NF este sigur din definiție.

a) **PP (RA) că nu e în 2NF**, adică există un atribut neprim A și o cheie K și A nu este dep. plin de K . Adică există $X \subset K$ a.i. $X \rightarrow A \in \Sigma^+$. Deoarece nu considerăm dep. triviale avem că $A \notin K$. Atunci X este cheie (pentru că (R, Σ) este în BCNF). Ceea ce înseamnă că K nu este cheie (pentru că trebuia să fie minimală).

b) Deci (R, Σ) ce este în BCNF trebuie să fie în 2NF. **PP (RA) că nu ar fi în 3NF**: adică există un atribut A tranzitiv dependent de o cheie X . Adică există Y a.î. $A \notin X$, $A \notin Y$ și avem că: $X \rightarrow Y \in \Sigma^+, Y \rightarrow A \in \Sigma^+$ și $Y \rightarrow X \notin \Sigma^+$. Y nu conține nicio cheie pentru că altfel am avea $Y \rightarrow \forall \in \Sigma^+$ deci și $Y \rightarrow X \in \Sigma^+$. Deoarece Y nu este cheie și totusi am $Y \rightarrow A \in \Sigma^+$ avem că (R, Σ) NU este în BCNF - fals. Deci este și 3NF

29 / 41

Descompunerea de tip join fără pierdere

Considerăm o schemă de relație $R[A_1, A_2, \dots, A_n]$. Spunem despre o mulțime $\rho = \{R_1[A_{i_1^1}, A_{i_2^1}, \dots, A_{i_{k_1}^1}], R_2[A_{i_1^2}, A_{i_2^2}, \dots, A_{i_{k_2}^2}], \dots, R_t[A_{i_1^t}, A_{i_2^t}, \dots, A_{i_{k_t}^t}]\}$ că este o **descompunere de tip join** a lui $R[A_1, A_2, \dots, A_n]$ dacă:

$$\bigcup_{p=1}^t \bigcup_{r=1}^{k_p} A_{i_r^p} = \{A_1 \dots A_n\}$$

ρ este o **descompunere de tip join fără pierdere cu privire la o multime de dependente functionale Σ** dacă $\forall r$ peste R ce satisface mulțimea de dependențe funcționale Σ , avem că

$$r = r[A_{i_1^1}, A_{i_2^1}, \dots, A_{i_{k_1}^1}] \bowtie r[A_{i_1^2}, A_{i_2^2}, \dots, A_{i_{k_2}^2}] \bowtie \dots r[A_{i_1^t}, A_{i_2^t}, \dots, A_{i_{k_t}^t}].$$

Descompunerea de tip join fără pierdere

Teorema: Dacă $\rho = \{R_1, R_2\}$ este o descompunere a lui R și Σ o mulțime de dependențe funcționale, atunci ρ este o descompunere de tip join fără pierdere cu privire la Σ dacă și numai dacă $R_1 \cap R_2 \rightarrow R_1 - R_2 \in \Sigma^+$ sau $R_1 \cap R_2 \rightarrow R_2 - R_1 \in \Sigma^+$ (operațiile sunt de fapt pe atributele peste care sunt construite schemele)

Exemplu: Considerăm $R[A, B, C]$ și $\Sigma = \{A \rightarrow B\}$.

$\rho_1 = \{R_1[A, B], R_2[A, C]\}$ este fără pierdere deoarece:
 $AB \cap AC = A$, $AB - AC = B$ și $A \rightarrow B \in \Sigma^+$

$\rho_2 = \{R_1[A, B], R_2[B, C]\}$ este cu pierdere deoarece:
 $AB \cap BC = B$, $AB - BC = A$ și $B \rightarrow A \notin \Sigma^+$
 $AB \cap BC = B$, $BC - AB = C$ și $B \rightarrow C \notin \Sigma^+$

Testați cele două desc. pentru $r = \{(1, 1, 2), (1, 1, 3), (2, 1, 2)\}$.

Descompunerea de tip join fără pierdere

Putem calcula Σ_i pentru $R_i[U_i]$ și continua procesul de descompunere până când ajungem la scheme de relație ce sunt în BCNF. $\Sigma_i = \{X \rightarrow Y | X, Y \in U_i\}$ - adică, pentru $R_i[U_i]$ ce este un element al descompunerii ρ luăm acele dependențe din Σ care sunt peste atributele ce sunt în U_i .

Pentru exemplul precedent:

Considerăm $R[A, B, C]$ și $\Sigma = \{A \rightarrow B\}$.

$\rho_1 = \{R_1[A, B], R_2[A, C]\}$ este fără pierdere deoarece:
 $AB \cap AC = A$, $AB - AC = B$ și $A \rightarrow B \in \Sigma^+$

avem că $\Sigma_1 = \{A \rightarrow B\}$ și $\Sigma_2 = \emptyset$

Alg. pt. descompunerea în join fără pierdere de tip BCNF

Intrare: (R, Σ)

Ieșire: $\rho = \{(R_1, \Sigma_1), \dots, (R_t, \Sigma_t)\}$ = descompunere fără pierdere a lui R cu privire la Σ . Unde (R_i, Σ_i) în BCNF, $\forall i \in \{1..t\}$

Pas 1: $\rho = R = R_1$; se caculează Σ^+ și cheile din R (necesare verificării formei BCNF).

Pas 2: Cât timp există în ρ un cuplu (R_i, Σ_i) ce nu e în BCNF (daca nu e în BCNF atunci există $X \rightarrow A$ și X nu e supercheie):

Pas 2.1: Alege $X \rightarrow A$ din Σ_i a.i. $A \notin X$ și X nu conține cheie

Pas 2.2: $S_1 = X \cup \{A\}$; $S_2 = R_i - A$;

Pas 2.3: $\rho = \rho - R_i$; $\rho = \rho \cup S_1 \cup S_2$;

Pas 2.4: Se caculează $\Sigma_{S_1}^+$, $\Sigma_{S_2}^+$ și cheile pentru S_1 și S_2 .

Exemplu

Schema de relație:

Absolvent(CNP, aNume, adresa, lCod, lNume, lOras, medie, prioritate)

$\Sigma = \{ \text{CNP} \rightarrow \text{aNume}, \text{adresa}, \text{medie} \quad \text{medie} \rightarrow \text{prioritate}$
 $\text{lCod} \rightarrow \text{lNume}, \text{lOras} \}$

Se descompune în:

... calculati

Exemplu

Schema de relație:

Absolvent(CNP, aNume, adresa, ICod, INume, IOras, medie, prioritate)

$\Sigma = \{ CNP \rightarrow aNume, adresa, medie \quad medie \rightarrow prioritate$
 $ICod \rightarrow INume, IOras \}$

Se descompune în:

$\rho = \{ R1[ICod, INume, IOras], R2[medie, prioritate],$
 $R3[CNP, aNume, adresa, medie], R4[CNP, ICol] \}$

Dependente multivaleute (quick reminder)

Definition

Relația r peste U *satisfacă dependența multivaluată* $X \twoheadrightarrow Y$ dacă pentru oricare două tuple $t_1, t_2 \in r$ satisfăcând $t_1[X] = t_2[X]$, există tuplele t_3 și t_4 din r , astfel încât:

- ▶ $t_3[X] = t_1[X], t_3[Y] = t_1[Y], t_3[Z] = t_2[Z];$
- ▶ $t_4[X] = t_2[X], t_4[Y] = t_2[Y], t_4[Z] = t_1[Z]$

unde $Z = U - XY$ (Z mai este denumită și *rest*).

Relația r peste U *satisfacă dependența multivaluată* $X \twoheadrightarrow Y$, dacă pentru orice $t_1, t_2 \in r$ cu $t_1[X] = t_2[X]$ avem că $M_Y(t_1[XZ]) = M_Y(t_2[XZ])$

unde $M_Y(t[XZ]) = \{t'[Y] | t' \in r, t'[XZ] = t[XZ]\}$.

Dependențe multivaluate (exercitiu)

Arătați că $AC \twoheadrightarrow BD$:

| | A | B | C | D | E |
|-------|---|---|---|---|---|
| | 8 | 1 | 2 | 0 | 4 |
| | 8 | 9 | 2 | 2 | 9 |
| $r :$ | 9 | 3 | 2 | 4 | 9 |
| | 8 | 1 | 2 | 0 | 9 |
| | 8 | 9 | 2 | 2 | 4 |
| | 9 | 3 | 2 | 4 | 4 |

Cand $r[AC] = \{(8, 2)\}$ avem $r[BD] = \{(1, 0), (9, 2)\}$ și $r[E] = \{(4), (9)\}$. Gasim toate produsele carteziene dintre cele 3 ?

Cand $r[AC] = \{(9, 2)\}$ avem $r[BD] = \{(3, 4)\}$ și $r[E] = \{(4), (9)\}$. Gasim toate produsele carteziene ?

DA (este MVD)

4NF (Ronald Fagin) (1977)

O schemă de relație R împreună cu o mulțime de dependențe multivaluate Δ (delta) este în 4NF dacă este în 1NF și pentru orice dependență multivaluată netrivială $X \twoheadrightarrow A \in \Delta^+$, X este supercheie pentru R .

Observație: O schemă de relație ce este în 4NF este în BCNF.

Putem presupune ca nu este în BCNF ceea ce inseamnă ca există d.f. a.i. $X \rightarrow A \in \Sigma^+$ și X nu este cheie. Atunci există aceeași dependență multivaluată ($X \twoheadrightarrow A \in \Delta^+$) și X nu este cheie: deci nu este 4NF

Descompunere in 4NF

Intrare: (R, Σ, Δ)

Ieșire: $\rho = \{R_1, \dots, R_t\} =$ descompunere fără pierdere a lui R cu privire la Σ . Unde $(R_i, \Sigma_i, \Delta_i)$ în 4NF, $\forall i \in \{1..t\}$

Pas 1: $\rho = R = R_1$; se caculează $\Sigma^+ \cdot \Delta^+$ și cheile din R (necesare verificării formei 4NF).

Pas 2: Cât timp există în ρ un triplet $(R_i, \Sigma_i, \Delta_i)$ ce nu e în 4NF (daca nu e în 4NF atunci există $X \rightarrow A$ și X nu e supercheie):

Pas 2.1: Alege $X \rightarrow A$ din Σ_i a.i. $A \notin X$ și X nu e supercheie

Pas 2.2: $S_1 = X \cup \{A\}$; $S_2 = R_i - A$;

Pas 2.3: $\rho = \rho - R_i$; $\rho = \rho \cup S_1 \cup S_2$;

Pas 2.4: Caculăm $\Sigma_{S_1}^+, \Delta_{S_1}^+, \Sigma_{S_2}^+, \Delta_{S_2}^+$ și cheile pentru S_1 și S_2 .

Exemplu de descompunere in 4NF

Consideram schema: **Student[cnp, nume, facultate, pasiune]**

Studentul poate fi la două facultăți și să aibă mai multe pasinui:

cnp → nume;

cnp, nume → facultate;

Se descompune în:

$\rho = \{S1[cnp, nume], S2[cnp, facultate], S3[cnp, pasiune]\}$.

[Jump to Table of contents](#)

Bibliografie

- ▶ Further Normalization of the Data Base Relational Model. - *Frank Edgar Codd*; IBM Research Report RJ909 (August, 1971)
- ▶ Baze de date relaționale. Dependențe - *Victor Felea*; Univ. Al. I. Cuza, 1996