

BAZE DE DATE

Proiectarea bazelor de date relaționale

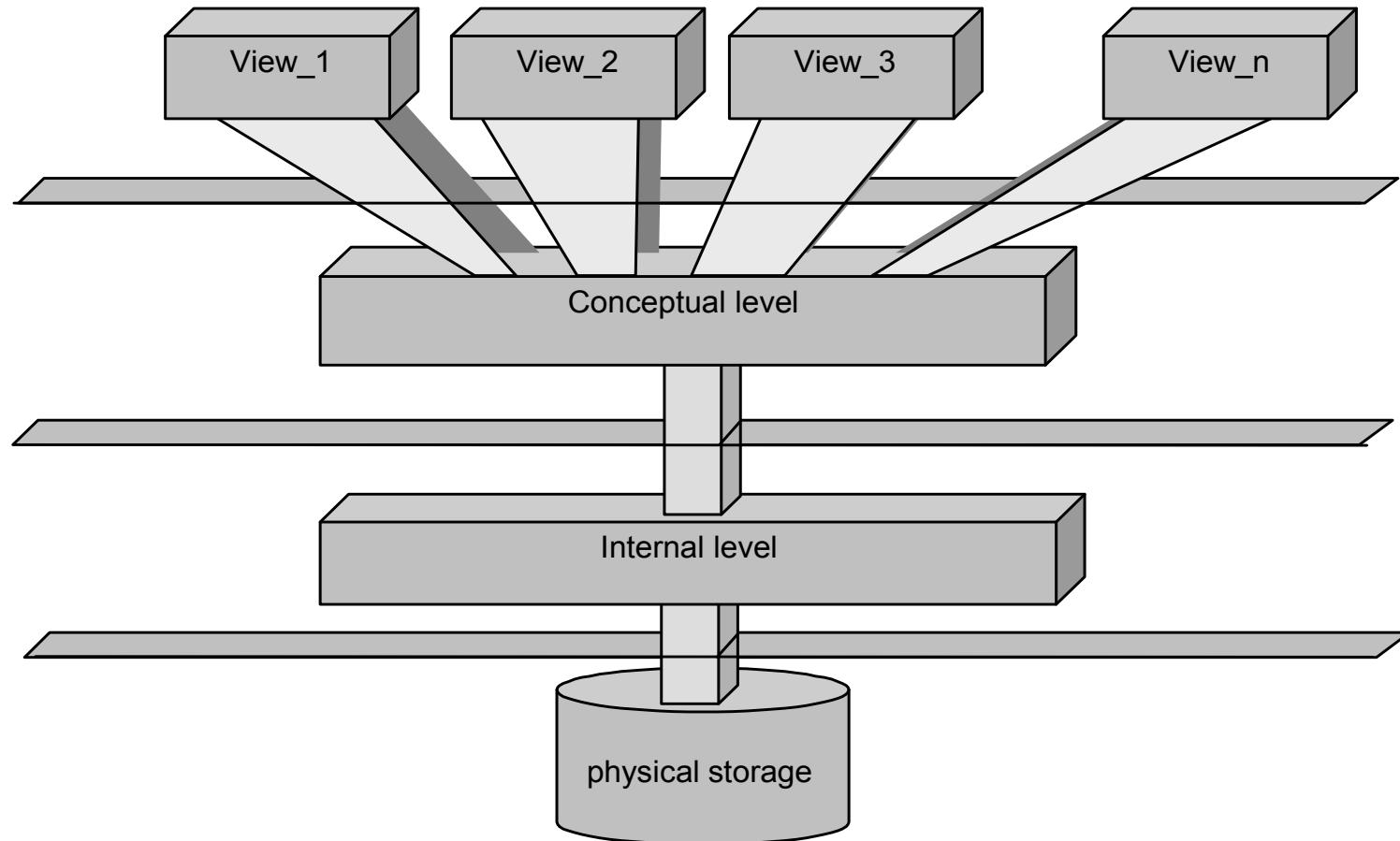
Normalizare
@FII (2011-2012)

prezentat de Mihaela Elena Breabă

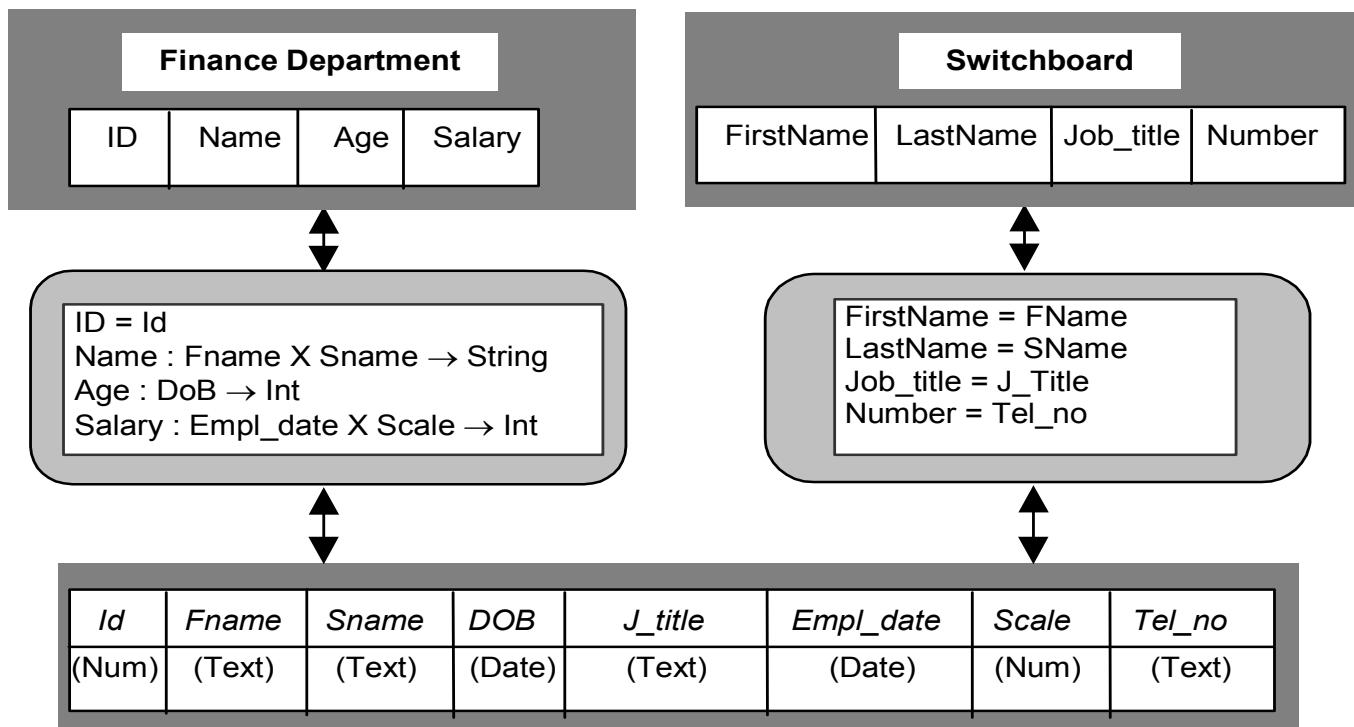
Tematică curs

- ▶ **Proiectarea bazelor de date relaționale**
 - ▶ Normalizare și denormalizare
 - ▶ Modelul entitate-asociere, diagrame UML
 - ▶ Constrângeri și declanșatoare
 - ▶ Indecși
- ▶ View-uri
- ▶ Procesarea interogărilor
- ▶ Managementul tranzacțiilor
- ▶ OLAP, Baze de date distribuite, NoSQL, Data Mining

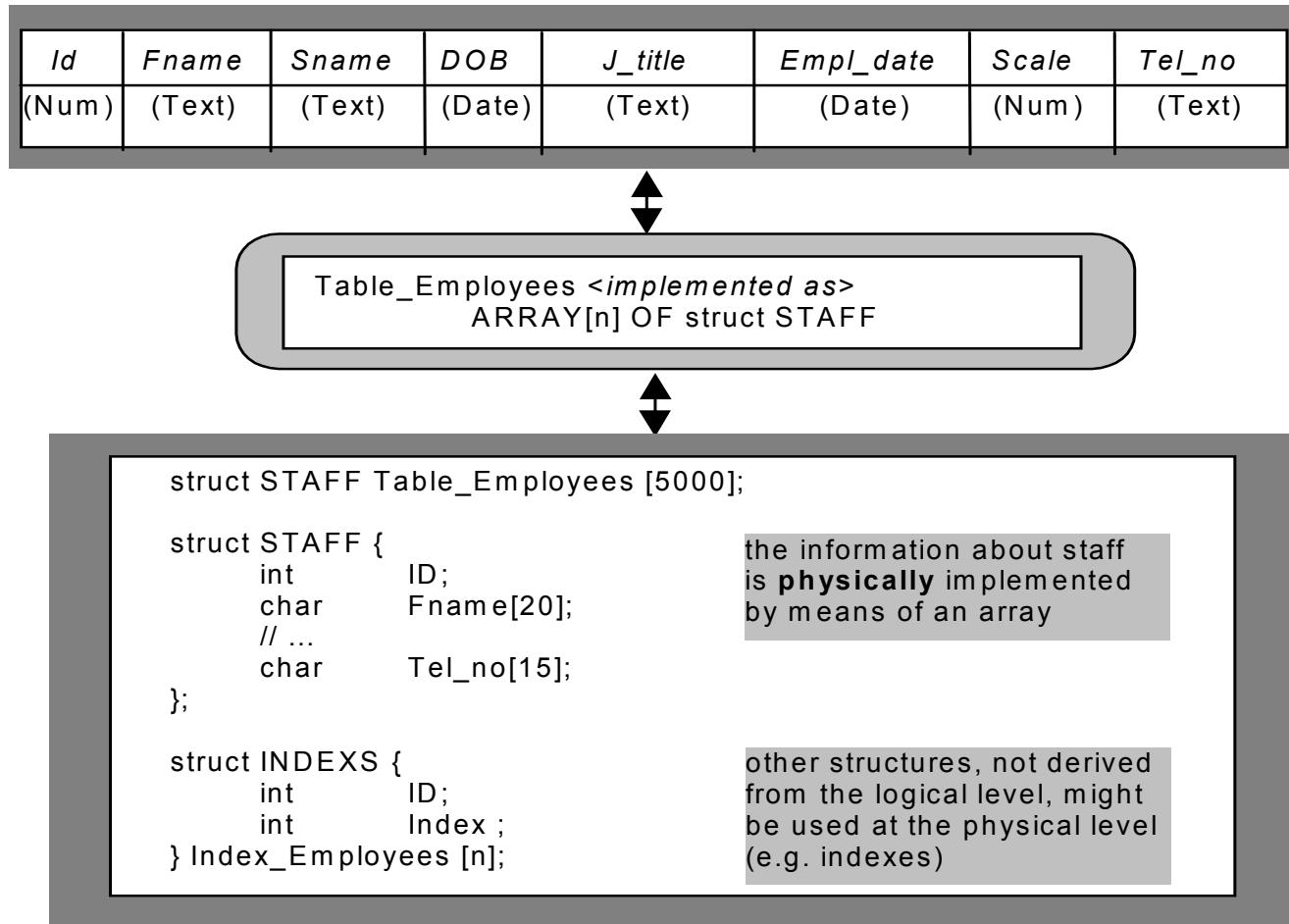
Arhitectura pe 3 nivele ANSI-SPARC



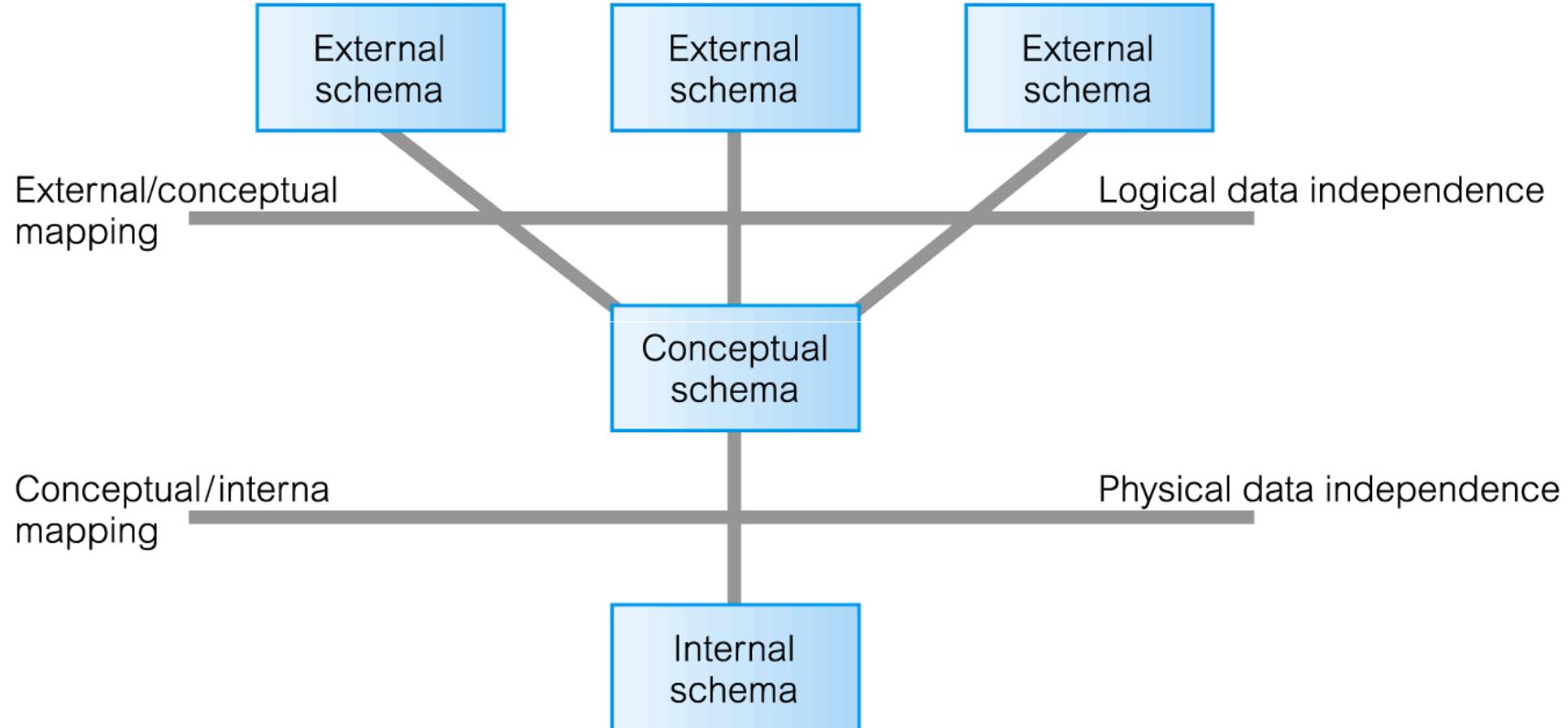
Mapare nivel extern/conceptual



Mapare nivel conceptual/intern

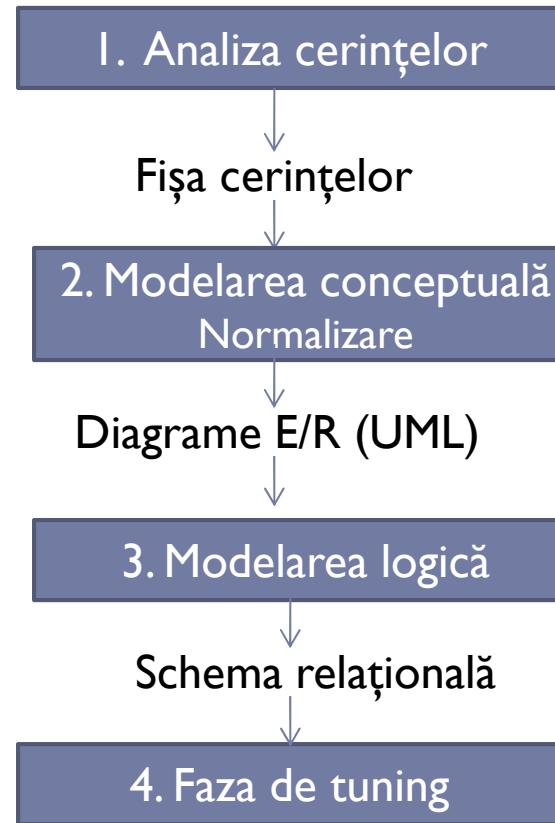


Arhitectura pe 3 nivale Scheme



Proiectarea unei BD

Metodologie



Normalizare

- ▶ Dependențe funcționale (revizitat)
- ▶ 1NF, 2NF, 3NF
- ▶ Forma normală Boyce-Codd (BCNF)
- ▶ Dependențe multivaluate (revizitat)
- ▶ Forma normală 4 (4NF)

Proiectarea schemei

- ▶ De obicei mai multe variante de proiectare
 - ▶ Unele sunt (mult) mai bune decât altele
 - ▶ Cum alegem?
-
- ▶ Teorie pentru proiectarea bazelor de date relationale cu fundamente în algebra relațională

Exemplu

Schemă cu anomalii

- ▶ Informații cu privire la aplicațiile de admitere
 - ▶ CNP și nume
 - ▶ Universitatea la care s-a aplicat
 - ▶ Liceele de la care provin candidații (și orașele)
 - ▶ Hobby-urile candidaților

Aplicatie(CNP, sNume, uNume, liceu, loraş, hobby)

Ioana cu CNP-ul **2810605222111** a studiat la **Negruzzi** în Iași, candidează la **Cuza, Asachi** și la **Babes-Bolyai**, îi place să joace **tenis** și să cânte la **chitară**

Câteuple sunt necesare a fi inserate în relația Aplicatie pentru a păstra toate informațiile despre Ioana?

Anomalii de proiectare

- ▶ Redundanță
- ▶ Anomalii de actualizare
- ▶ Anomalii la ștergere



Exemplu

Schemă fără anomalii

► Informații cu privire la aplicațiile de admitere

- CNP și nume
- Universitatea la care s-a aplicat
- Liceele de la care provin candidații (și orașele)
- Hobby-urile candidaților

Student(CNP, sNume)

Aplicatie(CNP, uNume)

Liceu(CNP, codLiceu)

LocatieLiceu(codLiceu, lNume, lOras)

Hobbies(CNP, hobby)

Quiz

- ▶ **Informații cu privire la cursurile luate de studenti**
 - ▶ Studenții au id-uri unice și nume (nu sunt unice)
 - ▶ Cursurile au număr de identificare unic și titlu (nu unic)
 - ▶ Studenții iau un curs într-un anumit an și primesc o notă
- ▶ **Care e schema recomandată?**
 - ▶ Studiază(sID, nume, cID, titlu, an, notă)
 - ▶ Curs(cID, titlu, an), Studiază(sID, cID, notă)
 - ▶ Student(sID, nume), Curs(cID, titlu), Studiază(sID, cID, an, notă)
 - ▶ Student(sID, nume), Curs(cID, titlu), Studiază(nume, titlu, an, notă)

Proiectarea prin descompunere

- ▶ Se pleacă de la “mega-relații” ce conțin tot
- ▶ Se descompune în relații mai mici ce păstrează toate informațiile

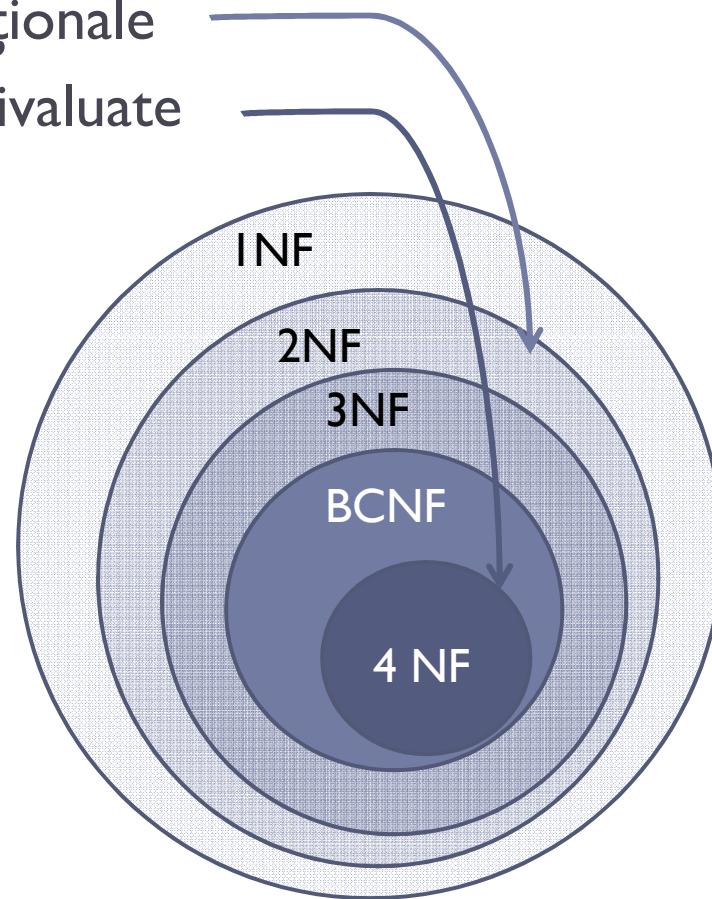
- ▶ Se poate realiza automat
 - ▶ Mega-relații + *proprietăți ale datelor*
 - ▶ Descompunerea se realizează pe baza proprietăților
 - ▶ Setul final de relații satisface anumite *forme normale*
 - ▶ **Fără anomalii**
 - ▶ **Fără pierdere de informații**

Proprietăți și forme normale

- ▶ Proprietăți

- ▶ Dependențe funcționale
- ▶ Dependențe multivaluate

- ▶ Forme normale



Dependențe funcționale

- ▶ Concepte folositoare pentru
 - ▶ Stocarea datelor – compresie
 - ▶ Optimizarea interogărilor

$X \rightarrow Y$ dacă

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

r – relație peste mulțimea de attribute U

X, Y – submulțimi ale lui U

De ce *funcțional*?

Exemplu Dependențe funcționale

`Student(CNP, sNume, adresa,
lCod, lNume, lOras, medie,
prioritate)`

`Aplicatie(CNP, uNume, uOras, data,
specializare)`

- ▶ Valorile atributului **prioritate** sunt determinate de valorile atributului **medie**

medie → prioritate

Care sunt dependențele funcționale pentru relația Student?

Dar pentru relația Aplicatie?

Ce constrângere este impusă de $\{CNP, uNume \rightarrow data\}$

Quiz

- ▶ $R(A,B,C,D,E)$
- ▶ $AB \rightarrow C$
- ▶ $CD \rightarrow E$
- ▶ Fiecare din attributele A,B,D are cel mult 3 valori diferite.
- ▶ Care este numărul maxim de valori diferite pe care îl poate lua E? (3,9,27,81?)

Dependențe funcționale

Reguli de inferență

- ▶ **Reflexivitatea (FD1)**
 - ▶ (A1)
 - ▶ dependențe triviale
- ▶ **Descompunerea (FD6)**
 - ▶ A(21)
 - ▶ Se poate descompune și membrul stâng?
- ▶ **Uniunea (FD5)**
 - ▶ (A22)
- ▶ **Tranzitivitatea (FD3)**
 - ▶ (A3)

- ▶ **Teorema de completitudine**
 - ▶ o dependență funcțională este consecință a unei multimi de dependențe funcționale d.d. are demonstrație utilizând regulile de mai sus (Axiomele lui Armstrong)

Dependențe funcționale și chei

- ▶ **Dependențe funcționale (d.f.)**
 - ▶ Valorile unei submulțimi de atrbute determină valorile unei alte submulțimi de atrbute
 - ▶ Formulate pe baza cunoașterii lumii reale
 - ▶ Toate instanțele relației trebuie să le satisfacă
 - ▶ Se specifică un set minimal netrivial a.î. toate dependențele satisfăcute de relație se obțin ca și consecințe a acestei mulțimi
- ▶ **Chei**
 - ▶ Valorile unei submulțimi de atrbute determină valorile tuturor atrbutelor - supercheie
 - ▶ O cheie este submulțime minimală cu proprietatea de mai sus
 - ▶ Relație fără duplicate
- ▶ **Dependențele funcționale sunt o generalizare a noțiunii de cheie**

Închideri

- ▶ Închiderea unei multimi de d.f. Σ notată Σ^+
 - ▶ Multimea d.f. Σ împreună cu toate d.f. consecințe din Σ
- ▶ Închiderea unei multimi de attribute X notată X^+ relativ la un set de d.f. Σ
 - ▶ Multimea tuturor atributelor B pentru care există $X \rightarrow B \in \Sigma^+$
- ▶ O dependentă funcțională $X \rightarrow B$ este consecință a unei multimi de dependențe funcționale d.d. $B \in X^+$

Algoritm de calcul a înciderii lui X ?

Închideri și chei

- ▶ Date relația R definită peste mulțimea de atrbute U și un set de d.f. Σ satisfăcute de R , submulțimea de atrbute X este cheie pentru R d.d. $X^+ = U$ și $\forall X' \subset X, X'^+ \neq U$

- ▶ Exemplu

`Student(CNP, sNume, adresa,
 lCod, lNume, loras, medie, prioritate)`

$CNP \rightarrow sNume, adresa, medie$

$medie \rightarrow prioritate$

$lCod \rightarrow lNume, loras$

Perechea $\{CNP, lCod\}$ este cheie

Dată o mulțime de d.f. cum putem determina toate cheile?

Quiz

- ▶ $R(A,B,C,D,E)$
- ▶ $AB \rightarrow C$
- ▶ $AE \rightarrow D$
- ▶ $D \rightarrow B$
- ▶ Care sunt cheile pt. R?

Atribute (ne)prime

- ▶ **Atribut prim**
 - ▶ Există o cheie care să-l conțină
- ▶ **Atribut neprim**
 - ▶ Nu aparține nici unei chei
- ▶ **Exemplu**
 - ▶ $RI(A, B, C, D)$
 - ▶ $F = \{AB \rightarrow C, B \rightarrow D, BC \rightarrow A\}$.
 - ▶ *AB și BC sunt singurele chei cu privire la F, deci A, B, C sunt atrbute prime*
 - ▶ *D este atrbut neprim.*

Dependențe pline

- ▶ Fie dată o schemă de relatie R cu multimea de atribute U și F o multime de dependențe funktionale. O dependență funcțională $X \rightarrow B \in F^+$ ($X \subset U, B \in U, B \notin X$) se numește o dependență plină a lui R (sau B este dependent plin de X sub F), dacă nu există nici o submultime proprie $X' \subset X$, astfel încât $X' \rightarrow B \in F^+$.

- ▶ Exemplu
 - ▶ $R(A, B, C, D)$
 - ▶ $F = \{AB \rightarrow C, B \rightarrow D, BC \rightarrow A\}$.
 - ▶ *Toate dependențele din F sunt pline.*
 - ▶ *$AB \rightarrow D \in F^+$ nu este dependență plină*

Atribut tranzitiv dependent

- ▶ Fie R o schema cu multimea de atribute U si F o multime de dependente functionale. Un atribut B din U se numeste *tranzitiv dependent* de X ($X \subset U$, $B \notin X$), daca exista $Y \subset U$, astfel incat:
 - ▶ $B \in U - Y$,
 - ▶ $X \rightarrow Y \in F^+$,
 - ▶ $Y \rightarrow B \in F^+$,
 - ▶ $Y \rightarrow X \notin F^+$.

1NF

- ▶ O schemă de relație este în 1NF dacă domeniile de valori ale tuturor atributelor sunt elementare (indivizibile) deci diferite de multimi, de uple de valori dintr-un anumit domeniu. În general numim valoare elementară o valoare pentru care în aplicații nu se utilizează părți ale ei

2NF

- ▶ O schema de relatie R situata in 1NF, impreuna cu o multime de dependente functionale F este in a doua forma normala daca orice atribut neprim din R este dependent plin de orice cheie a lui R.
- ▶ Obs: Orice relatie ce nu are chei multivaluate este in 2NF.
- ▶ Exemplu
 - ▶ $R(A, B, C, D)$
 - ▶ $F = \{AB \rightarrow C, B \rightarrow D, BC \rightarrow A\}$.
 - ▶ AB si BC sunt singurele chei
 - ▶ D este atribut neprim
 - ▶ $B \rightarrow D \in F^+$, deci D nu este dependent plin nici de AB, nici de BC. In concluzie, aceasta schema impreuna cu F nu este in 2NF.

Quiz

- ▶ Studenti(CNP,sNume, hobby)
 - ▶ Un student are un singur nume însă mai multe hobbyuri
 - ▶ Studenti nu este în 2 NF. Ce anomalii apar din nerespectarea 2NF?
-
- ▶ Olimpici(concurs,an,CNP,nume)
 - ▶ Într-un anumit an există un singur câștigător (olimpic) la un anumit concurs. Câștigătorul e identificat prin CNP și are asociat numele.
 - ▶ Este Olimpici in 2NF?

3NF

- ▶ Schema de relatie R impreuna cu F se spune ca este in forma a treia normala (notata 3NF) daca este in a doua forma normala si orice atribut neprim din R nu este tranzitiv dependent de nici o cheie a lui R.
- ▶ Exemplu
 - ▶ $R(O, S, C)$
 - ▶ $F = \{OS \rightarrow C, C \rightarrow O\}$
 - ▶ *OS si SC sunt chei.*
 - ▶ *toate atributele sunt prime, deci schema este in 2NF si 3NF.*

Quiz

- ▶ **Olimpici(concurs,an,CNP,nume)**
- ▶ Într-un anumit an există un singur câștigător (olimpic) la un anumit concurs. Câștigătorul e identificat prin CNP și are asociat numele.
- ▶ **Olimpici nu este in 3NF. Ce probleme de inconsistență a datelor pot să apară?**

Quiz

- ▶ Aplicație(CNP,uNume,data,specializare)
- ▶ Un student poate aplica la o universitate o singură dată și la o singură specializare
- ▶ Universitățile au date de aplicație care nu se suprapun
- ▶ Este Aplicație în 3NF relativ la regulile specificate mai sus?

BCNF

- ▶ O schemă de relație R împreună cu o mulțime de dependențe funcționale D este în BCNF dacă pentru orice dependență funcțională netrivială $X \rightarrow A \in D^+$ X este (super)cheie pentru R
- ▶ Orice schemă de relație în BCNF este în 3NF
- ▶ Proiectarea unei scheme de BD în BCNF are la bază descompunerea:
 - ▶ Intrare: o mega-relație împreună cu un set de dependențe funcționale
 - ▶ Ieșire: un set de relații în BCNF care în urma reasamblării produc informațiile originale

Quiz

- ▶ Aplicație(CNP,uNume,data,specializare)
- ▶ Un student poate aplica la o universitate o singură dată și la o singură specializare
- ▶ Universitățile au date de aplicație care nu se suprapun
- ▶ Este Aplicație în BCNF?

Descompunerea schemelor de relatie

- ▶ Fie schema de relatie $R[A_1, A_2, \dots, A_n]$.
- ▶ $\rho = \{R_1, \dots, R_k\}$, $R_i[A_{i1}, \dots, A_{ih_i}]$ este o *descompunere* a lui R dacă
$$\bigcup_{i=1}^k \bigcup_{j=1}^{h_i} A_{ij} = \{A_1, \dots, A_n\}$$
- ▶ ρ este o *descompunere de tip join fără pierdere* a lui R cu privire la o mulțime de d.f. D , dacă pentru orice relatie r peste R ce satisfacă D , avem $r = r[R_1] * \dots * r[R_k]$ – deci r se obține în urma joinului natural peste descompunerea ρ .

Exemplu

Descompunere

- ▶ Student(CNP,sNume,adresa,ICod,INume,Oras,medie,prioritate)
- ▶ $\rho_1 = \{S_1(CNP, sNume, adresa, \underline{ICod}, medie, prioritate), S_2(\underline{ICod}, \underline{INume}, \underline{Oras})\}$
- ▶ $\rho_2 = \{S_1(CNP, \underline{sNume}, adresa, \underline{ICod}, \underline{INume}, \underline{Oras}), S_2(\underline{sNume}, \underline{INume}, medie, prioritate)\}$

- ▶ p1 - de tip join fără pierdere
- ▶ p2 - NU e de tip join fără pierdere

Descompuneri de tip join fără pierdere

▶ Teoremă

- ▶ Daca $\rho = (R_1, R_2)$ este o descompunere a lui R si F este o multime de d.f., atunci ρ este o descompunere join fara pierdere cu privire la F d.d. $R_1 \cap R_2 \rightarrow R_1 - R_2 \in F^+$ sau $R_1 \cap R_2 \rightarrow R_2 - R_1 \in F^+$.

▶ Exemplu

- ▶ $R(A, B, C)$
- ▶ $F = \{A \rightarrow B\}$.
- ▶ $\rho_1 = (R_1(A, B), R_2(A, C))$
- ▶ $AB \cap AC = A$, $AB - AC = B$, $A \rightarrow B \in F^+$
- ▶ ρ_1 este de tip join fara pierdere

- ▶ $\rho_2 = (R_1(A, B), R_2(B, C))$.
- ▶ $AB \cap BC = B$, $AB - BC = A$, $B \rightarrow A \notin F^+$,
- ▶ $AB \cap BC = B$, $BC - AB = C$, $B \rightarrow C \notin F^+$,
- ▶ ρ_2 nu este de tip join fara pierdere cu privire la F .

Descompunere de tip join fara pierdere in BCNF

- ▶ Intrare:
 - ▶ Schema de relatie R cu dependentele functionale F.
- ▶ Iesire:
 - ▶ Descompunerea lui $\rho = (R_1, \dots, R_k)$, astfel incat ρ este de tip join fara pierdere cu privire la F si (R_i, F_i) este in BCNF $\forall i = 1, k$.
- ▶ Pasul 1.
 - ▶ $\rho = R = R_1$
 - ▶ Calculăm F^+ și cheile necesare verificării formei BCNF
- ▶ Pasul 2.
 - ▶ Fie R_i o schema de relatie din ρ , pentru care (R_i, F_i) nu este in BCNF.
 - ▶ Exista $X \rightarrow A \in F_i^+, A \notin X$ si X nu include o cheie.
 - ▶ Construim $S_1 = X \cup \{A\}$, $S_2 = R_i - A$
 - ▶ Înlocuim R_i in ρ prin $S_1, S_2, k = k + 1$.
 - ▶ Calculăm $F_{S1}^+ F_{S2}^+$ și cheile pt. S_1, S_2 necesare verificării formei BCNF
- ▶ Pasul 3.
 - ▶ Repetam pasul 2, pana cand obtinem toate $(R_i, F_i), i = 1, k$ in BCNF.

Exemplu

Descompunere în BCNF

`Student(CNP, sNume, adresa,
lCod, lNume, lOras, medie, prioritate)`

$\text{CNP} \rightarrow \text{sNume, adresa, medie}$

$\text{medie} \rightarrow \text{prioritate}$

$\text{lCod} \rightarrow \text{lNume, lOras}$

$\{R_1(lCod, lNume, lOras),$
 $R_2(\text{medie}, \text{prioritate}),$
 $R_3(\text{CNP}, \text{sNume}, \text{adresa}, \text{medie}),$
 $R_4(\text{CNP}, lCod)\}$

este o descompunere de tip join fără pierdere în BCNF

- ▶ Pentru o schemă de relație R pot exista mai multe descompuneri de tip join fără pierdere în BCNF?

Garantează desc. în BCNF o schemă bună?

- ▶ Poate fi reconstruită relația originală?
- ▶ Elimină redundanță?
 - ▶ Aplicatie(CNP, uName, hobby)
 - ▶ d.f.? NU
 - ▶ Chei? Toate atrbutele
 - ▶ BCNF? DA
 - ▶ Schemă bună? ...

Dependențe multivariate

► Reguli generatoare de uple

$X \rightarrow\rightarrow Y$ dacă

$\forall t_1, t_2 \in r, t_1[X] = t_2[X]$, există $t_3, t_4 \in r$ astfel încât

(i) $t_3[X] = t_1[X], t_3[Y] = t_1[Y]$ și $t_3[Z] = t_2[Z]$

(ii) $t_4[X] = t_2[X], t_4[Y] = t_2[Y]$ și $t_4[Z] = t_1[Z]$

r – relație peste mulțimea de attribute U

X, Y – submulțimi ale lui U

$Z = U - XY$

► Orice d.f. este d.mv.

Exemplu

Dependențe multivaluate

- ▶ **Aplicatie(CNP, uNume, hobby)**
- ▶ Cerințe:
 - ▶ Aceleași hobbyuri la toate univ
- ▶ **Regula corespunzătoare:**
 - ▶ $\text{CNP} \rightarrow\!\!\! \rightarrow \text{uNume}$

- ▶ **Exemplu extins**
 - ▶ **Aplicatie(CNP, uNume, data, specializare, hobby)**
 - ▶ Cerințe:
 - ▶ Hobbyurile sunt introduse selectiv în funcție de universitate
 - ▶ Un student aplică într-o singură zi la o anumită universitate
 - ▶ Un student poate aplica la mai multe specializări
 - ▶ **Regulile corespunzătoare:**
 - ▶ $\text{CNP}, \text{uNume} \rightarrow \text{data}$
 - ▶ $\text{CNP}, \text{uNume}, \text{data} \rightarrow\!\!\! \rightarrow \text{specializare}$

Quiz

- ▶ Fie $R(A,B,C)$ și $A \rightarrow\!\!\!\rightarrow B$
- ▶ A ia cel puțin 3 valori diferite iar fiecare valoare a lui A este asociată cu cel puțin 4 valori diferite pentru B și cel puțin 5 valori diferite pentru C.
- ▶ Care este numărul minim deuple în R?

Dependențe multivaluate

Reguli

- ▶ Dependențe triviale
 - ▶ Reflexivitate (MVD1)
 - ▶ $X \rightarrow\!\!\! \rightarrow Y$ unde $XY=U$
- ▶ Complementariere (MVD0)
- ▶ Tranzitivitatea (\neq d.f.)
- ▶ Intersecția

4NF

- ▶ O schemă de relație R și o mulțime de dependențe multivaluate D este în 4NF dacă pentru orice dependență multivaluată netrivială $X \rightarrow\!\!\! \rightarrow A \in D^+$ X este (super)cheie pentru R
- ▶ Orice schemă de relație în 4NF este în BCNF
- ▶ Proiectarea unei scheme de BD în 4NF are la bază descompunerea:
 - ▶ Intrare: o mega-relație împreună cu un set de dependențe funcționale și multivaluate
 - ▶ Ieșire: un set de relații în 4NF care în urma reasamblării produc informațiile originale

Descompunere de tip join fara pierdere in 4NF

- ▶ Intrare:
 - ▶ Schema de relatie R cu dependentele functionale F și dependențele multivalue MV
- ▶ Iesire:
 - ▶ Descompunerea lui $\rho = (R_1, \dots, R_k)$, astfel incat ρ este de tip join fara pierdere cu privire la F si (R_i, F_i, MV_i) este in 4NF $\forall i = 1, k$.
- ▶ Pasul 1.
 - ▶ $\rho = R = R_1$
 - ▶ Calculăm $M = \{F^+, MV^+\}$ și cheile necesare verificării formei 4NF
- ▶ Pasul 2.
 - ▶ Fie R_i o schema de relatie din ρ , pentru care (R_i, F_i, MV_i) nu este in 4NF.
 - ▶ Exista $X \rightarrow A \in M$ netrivială și X nu include o cheie.
 - ▶ Construim $S_1 = X \cup \{A\}$, $S_2 = R_i - A$
 - ▶ Înlocuim R_i in ρ prin S_1, S_2 . $k = k + 1$.
 - ▶ Calculăm d.mv și cheile pt. S_1, S_2 necesare verificării formei 4NF
- ▶ Pasul 3.
 - ▶ Repetam pasul 2, pana cand obtinem toate $(R_i, F_i), i = 1, k$ in NF.

Exemplu

Descompunere în 4NF

- ▶ Aplicatie(CNP, uNume, hobby)
- ▶ CNP $\rightarrow\!\!\!\rightarrow$ uNume
- ▶ $\rho = \{A_1(CNP, uNume), A_2(CNP, hobby)\}$ este descompunere în 4NF de tip join fără pierdere
- ▶ u^*h , $u+h$
- ▶ **Exemplu extins**
 - ▶ Aplicatie(CNP, uNume, data, specializare, hobby)
 - ▶ CNP, uNume \rightarrow data
 - ▶ CNP, uNume, data $\rightarrow\!\!\!\rightarrow$ specializare
 - ▶ $\rho = \{A_1(CNP, uNume, data), A_2(CNP, uNume, specializare), A_3(CNP, uNume, hobby)\}$ este în 4NF de tip join fără pierdere

Neajunsuri ale normalizării

Exemplu 1

- ▶ Aplicație($\text{CNP}, \text{uNume}, \text{data}, \text{specializare}$)
- ▶ $\text{CNP}, \text{uNume} \rightarrow \text{data}, \text{specializare}$
- ▶ $\text{data} \rightarrow \text{uNume}$

- ▶ $\{\text{A1}(\text{data}, \text{uNume}), \text{A2}(\text{CNP}, \text{data}, \text{specializare})\}$ în 4NF este o schemă mai bună?

Neajunsuri ale normalizării

Exemplu 2

- ▶ **Student(CNP,INume,medie,prioritate)**
- ▶ **CNP → medie**
- ▶ **medie → prioritate**
- ▶ **CNP → prioritate**

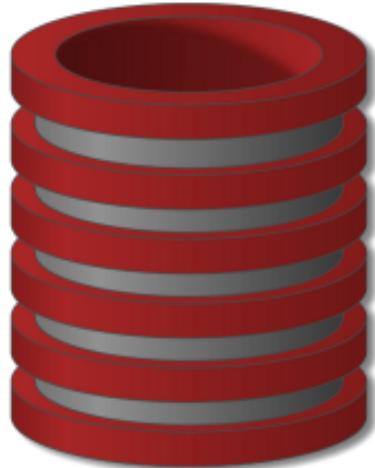
- ▶ **{S1(CNP,prioritate), S2(CNP,medie), S3(CNP,INume)}** în 4NF este o schemă bună?

Neajunsuri ale normalizării

- ▶ Supra-descompunere
 - ▶ Interogări supra-încărcate
-
- ▶ Ca soluție se poate aplica denormalizarea

Bibliografie

- ▶ V.Felea: *Baze de date relationale. Dependente.* Editura Universitatii “Al.I.Cuza” Iasi, 1996
- ▶ Hector Garcia-Molina, Jeff Ullman, Jennifer Widom: *Database Systems: The Complete Book*, Prentice Hall; 2nd edition (June 15, 2008)



BAZE DE DATE

Proiectarea bazelor de date relaționale

Modelul entitate-asociere
@FII (2011-2012)

prezentat de Mihaela Elena Breabă

Tematică curs

- ▶ **Proiectarea bazelor de date relaționale**
 - ▶ Normalizare și denormalizare
 - ▶ **Modelul entitate-asociere, diagrame UML**
 - ▶ Constrângeri și declanșatoare
 - ▶ Indecși
- ▶ **View-uri**
- ▶ **Procesarea interogărilor**
- ▶ **Managementul tranzacțiilor**
- ▶ **OLAP, Baze de date distribuite, NoSQL, Data Mining**

Modelul entitate-asociere (Entity/Relationship)

Obiective

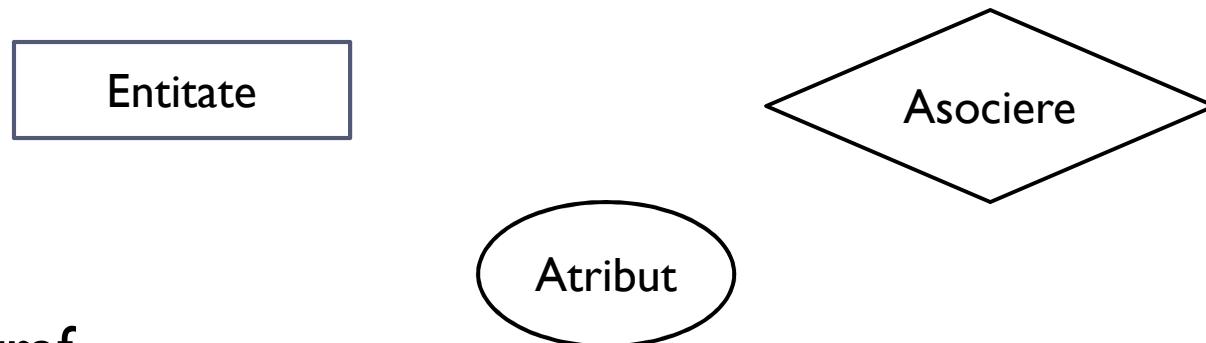
- ▶ Concepte E/R de bază
- ▶ Modelarea constrângerilor
- ▶ Capcane de conectare
- ▶ Crearea diagramelor E/R utilizând UML
- ▶ De la diagrame E/R (UML) la schema relatională

Concepțe E/R clasice (Chen 1976)

- ▶ **Entitate**
 - ▶ Un concept de sine stătător ce corespunde unui grup de obiecte de același tip
 - ▶ O instanță - entitate
 - ▶ O instanță unic identificabilă
- ▶ **Asociere (Relationship)**
 - ▶ Conexiune/asociere între două sau mai multe entități (sau chiar a unei entități cu ea însăși)
 - ▶ Gradul asocierii = nr. de entități participante
 - ▶ unare, binare, ternare...
- ▶ **Atribut**
 - ▶ Proprietate a unei entități
 - ▶ În asociere
 - ▶ Atribute ale entităților referențiate
 - ▶ Noi atribute

Diagrame E/R

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

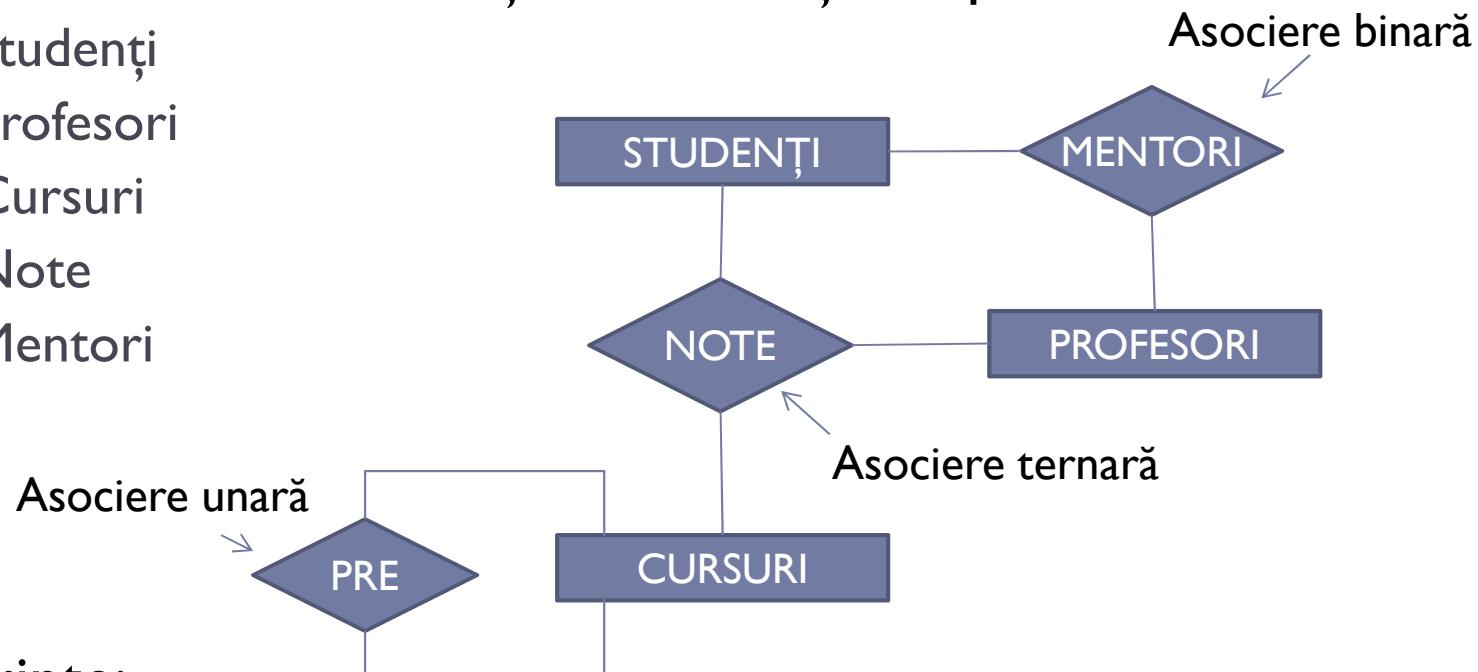


- ▶ Un graf
 - ▶ Entitățile, asocierile și attributele sunt noduri
 - ▶ Există muchii doar între
 - ▶ noduri-entitate și noduri-asociere
 - ▶ noduri-entitate și noduri-attribute
 - ▶ noduri-asociere și noduri-attribute

Exemplu

- ▶ O bază de date ce conține informații despre:

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



- ▶ Cerințe:

- ▶ Putem determina notele obținute , cursurile pe care le-a finalizat si creditele obținute de orice student
- ▶ Putem determina mentorul oricărui student

Alte concepte E/R

▶ Rol

- ▶ Explică semnificația entităților în asocieri



▶ Cheie primară

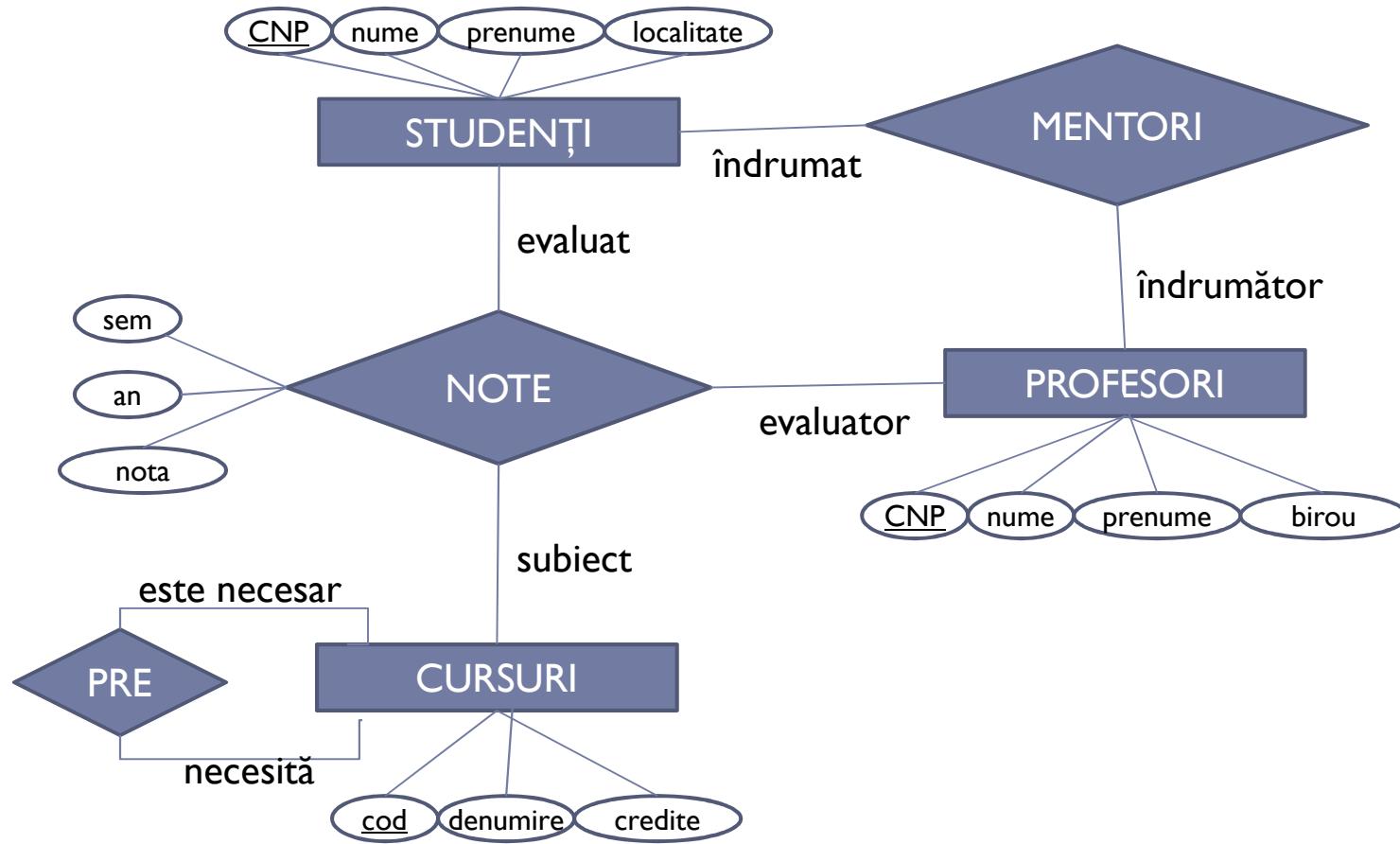
- ▶ Un atribut sau o submulțime minimală de attribute ce identifică unic o instanță a unei entități sau a unei asocieri (dependență funcțională!!!)
- ▶ Obligatorie pentru entități, pentru a indica care instanțe participă în asocieri

Cheie primară

▶ Cheie străină pentru o asociere

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

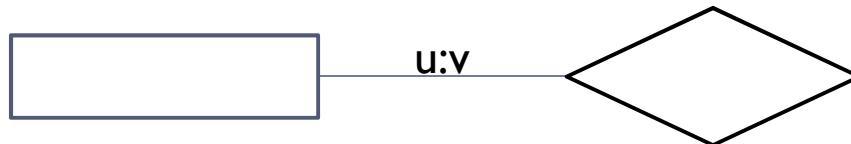
Exemplu



Care sunt cheile străine pentru cele trei asocieri?

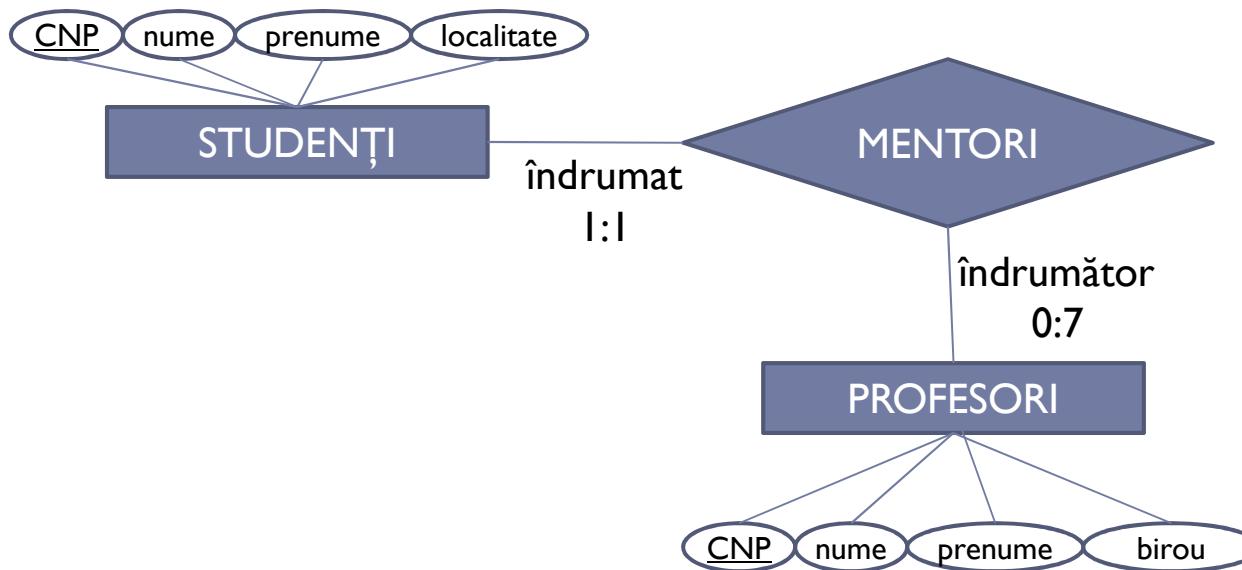
Constrângeri de conectivitate/multiplicitate

- ▶ Modelul E/R permite declararea de constrângeri asupra numărului de instanțe ale unei asocieri în care o instanță a unei entități participă
- ▶ Fie R o asociere între n entități E_i , $i=1..n$. Baza de date satisface constrângerea (E_i, u, v, R) dacă fiecare instanță a entității E_i participă în cel puțin u și cel mult v instanțe ale asocierii R.

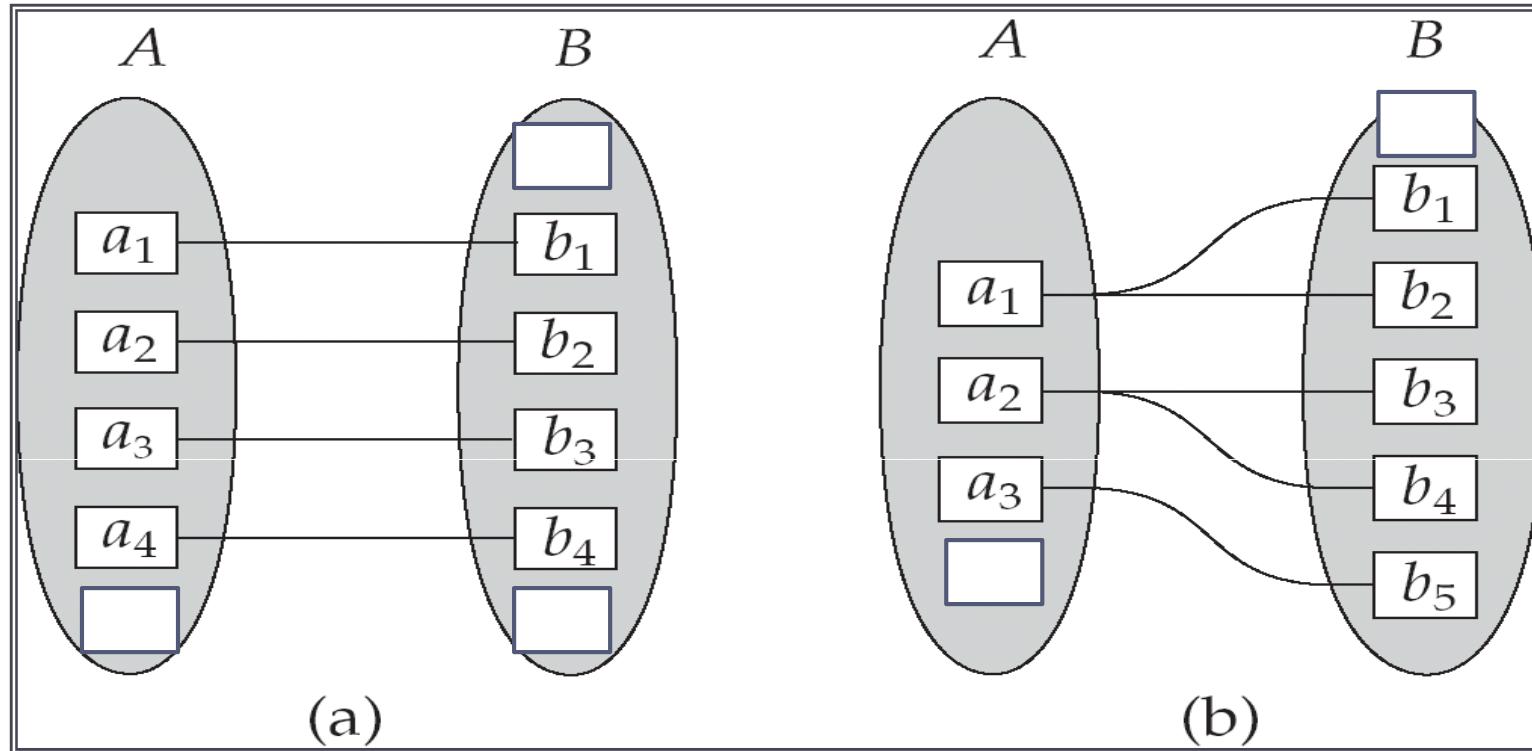


Exemplu

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



Constrângeri de conectivitate pentru asocieri binare (1)



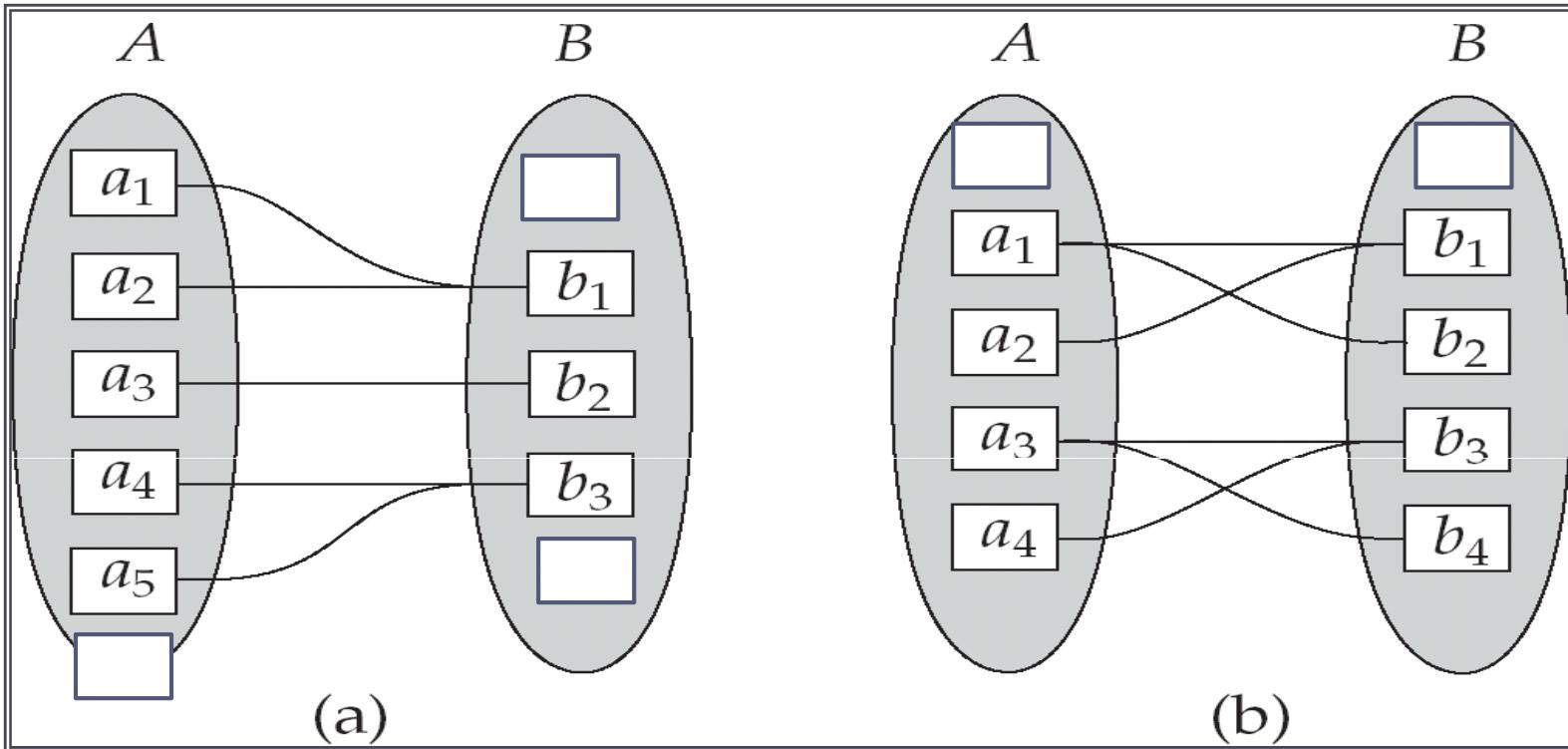
a) Asociere unu la unu
(A,0,I,R) (B,0,I,R)



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



Constrângeri de conectivitate pentru asocieri binare (2)



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

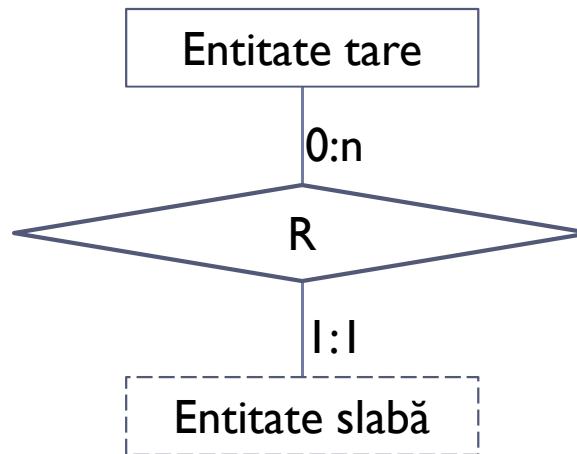


b) Asociere mulți la mulți
(A,0,m,R) (B,0,n,R), $m, n > 1$



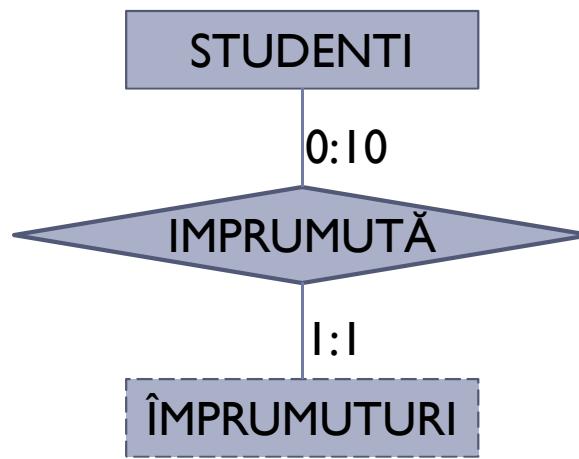
Entitate slabă

- ▶ O entitate este slabă dacă existența instanțelor sale depinde de existența instanțelor altor entități (dependentă existențială)

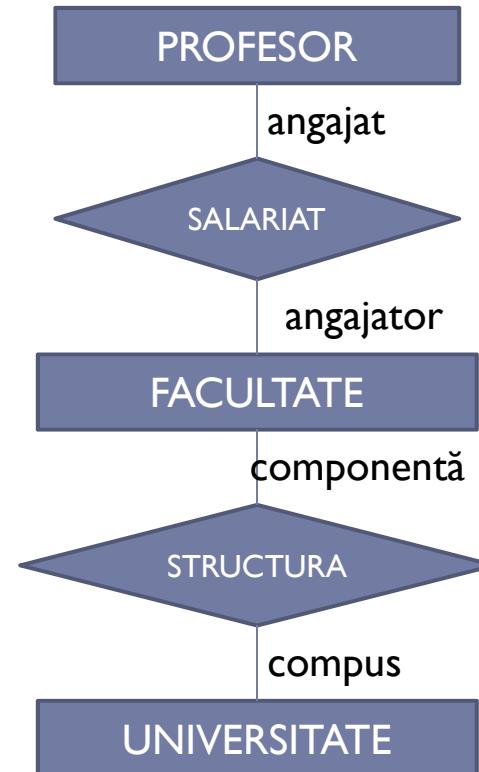
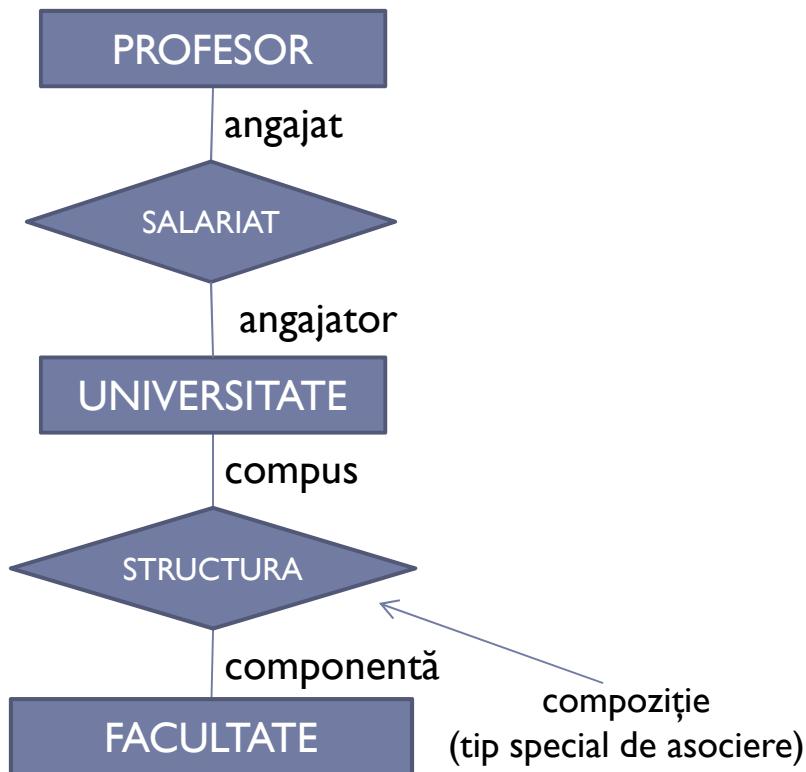
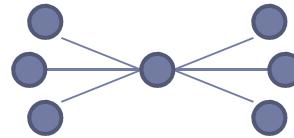


- ▶ Nu are cheie
- ▶ Satisfac constrângerea de conectivitate
(Entitate_slabă, I,I,R), deci participă într-o asociere de tip unu la mulți relativ la entitatea tare

Exemplu



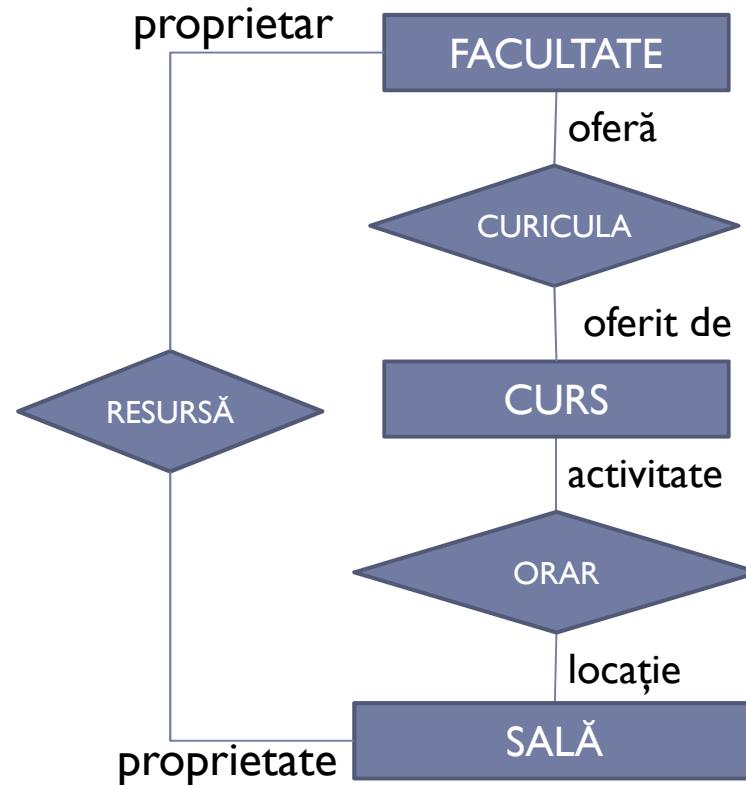
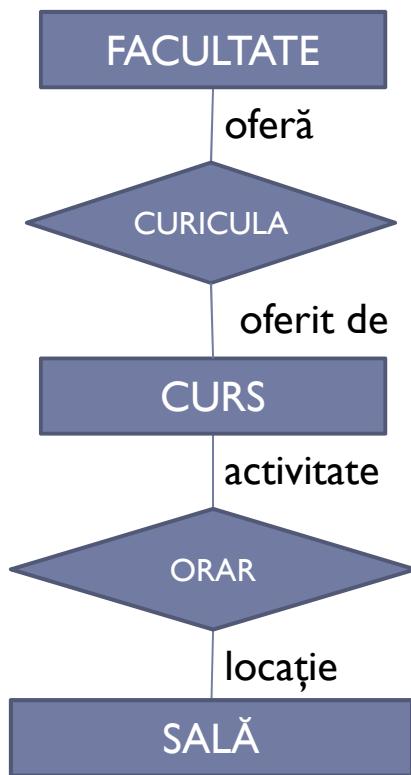
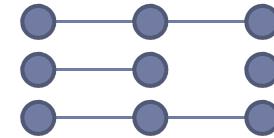
Capcane de conectare (Fan traps)



Problema:
La ce departament aparține profesorul X?

Soluția:
Model restructurat

Capcane de conectare (Chasm traps)



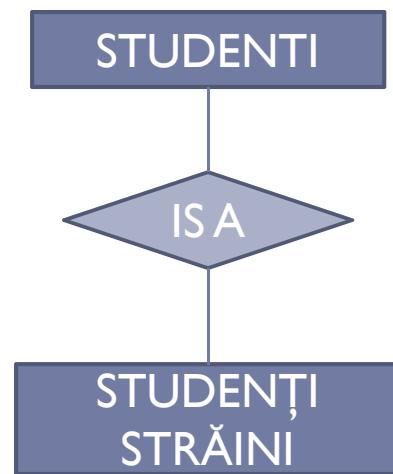
Problema:
Care sunt toate sălile ce aparțin unei facultăți?

Soluția:
Noi asocieri

Modelul E/R extins

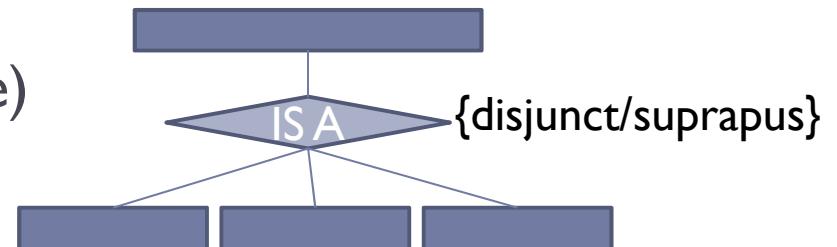
Specializare

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

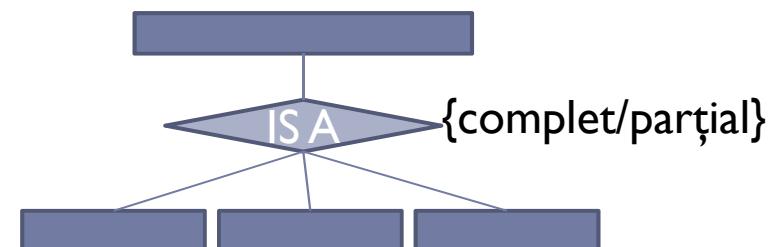


Constrângeri specifice specializării

- ▶ Instanțele specializării moștenesc toate atributele și asocierile entității de bază, inclusiv cheia
- ▶ O instanță a unei entități poate apartine numai la una sau la mai multe specializări
 - ▶ Specializări disjuncte (exclusive)
 - ▶ Specializări cu suprapunere



- ▶ O instanță a unei entități trebuie sau nu să aparțină la cel puțin o specializare
 - ▶ Complet
 - ▶ Incomplet (partial)



Modelare UML

► Unified Modeling Language

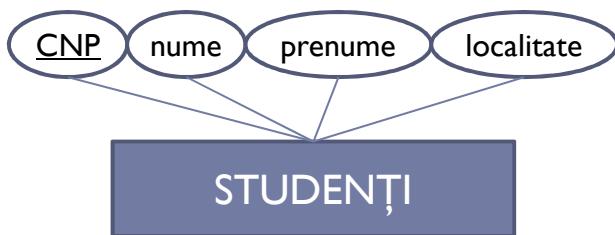
- Utilizat în ingineria software
- Bazat pe concepte orientate obiect
- Unealtă de comunicare cu clientul în termenii utilizării în companie
- Un limbaj foarte mare, utilizăm un set restrâns (diagrama de clase) de elemente pentru a modela o bază de date.

Mapare E/R – UML

E/R	UML
Entitate + atribute	Clasă
Asociere fără atribute proprii	Asociere
Asociere cu atribute proprii	Clasă de asociere
Specializare	Subclasă
	Compoziție și agregare

Clase

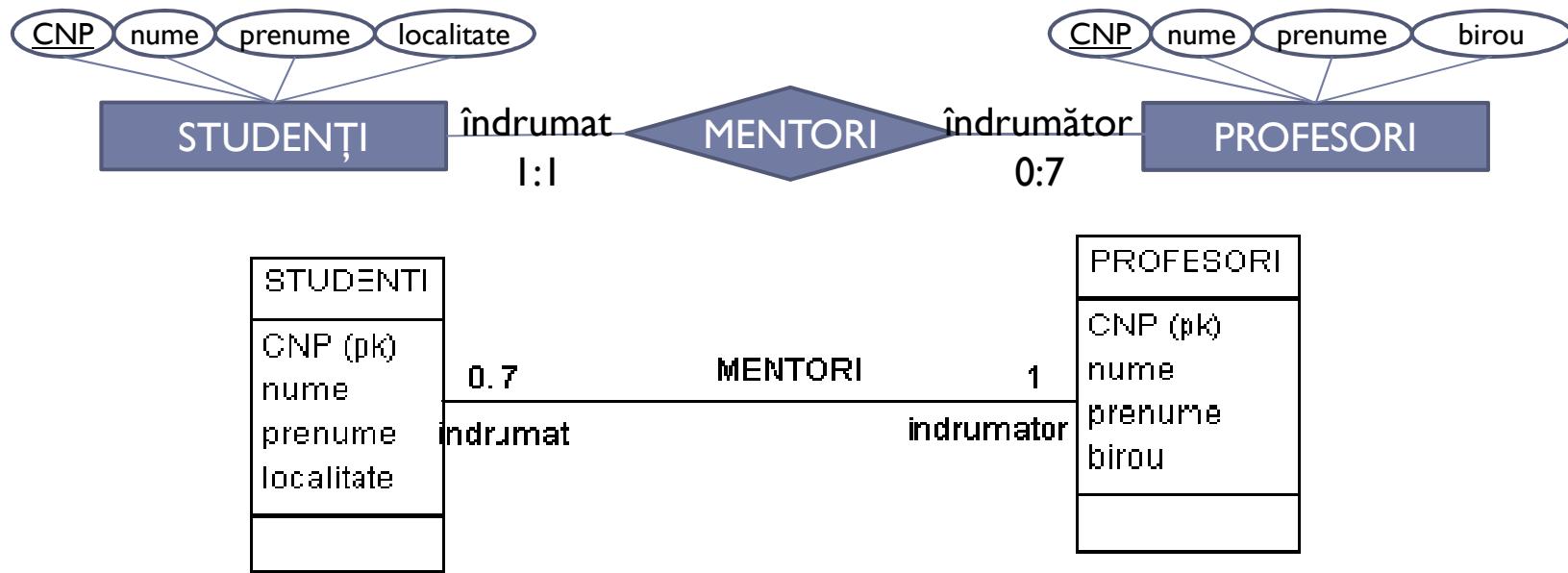
- ▶ Componente: nume, atribute, metode
- ▶ BD: nume, atribute (cheia primară)



STUDENTI
CNP (pk)
nume
prenume
localitate

Asocieri

- ▶ Exprimă asocierea dintre obiectele aparținând la **2 clase**
- ▶ BD: asocierea dintre instanțele a două entități



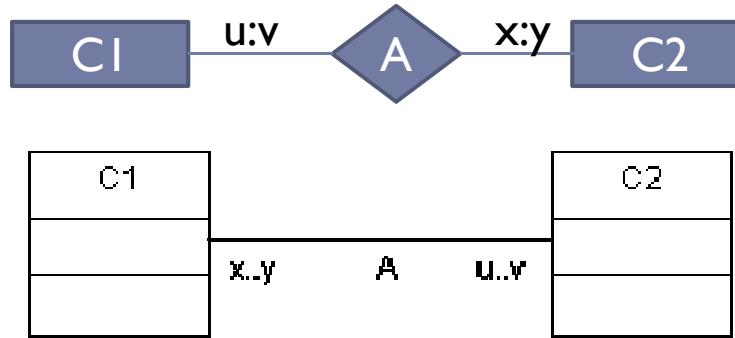
- ▶ Obs: constrângerile de cardinalitate se specifică invers decât în diagramele E/R

Asocieri

Constrângeri de conectivitate/multiplicitate

- ▶ Restricții

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

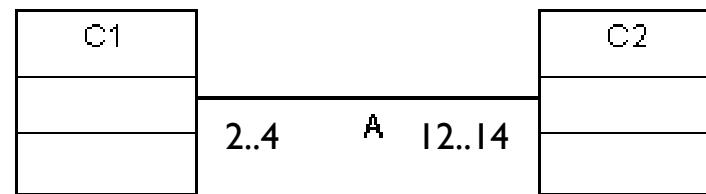


- ▶ Fiecare obiect din (instanță a entității) C1 este asociat cu cel puțin u și cel mult v obiecte din (instanțe ale entității) C2
- ▶ Fiecare obiect din (instanță a entității) C2 este asociat cu cel puțin x și cel mult y obiecte din (instanțe ale entității) C2

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

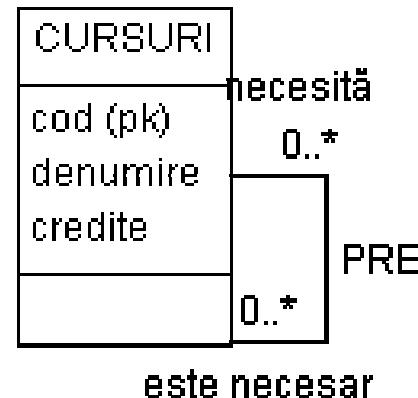
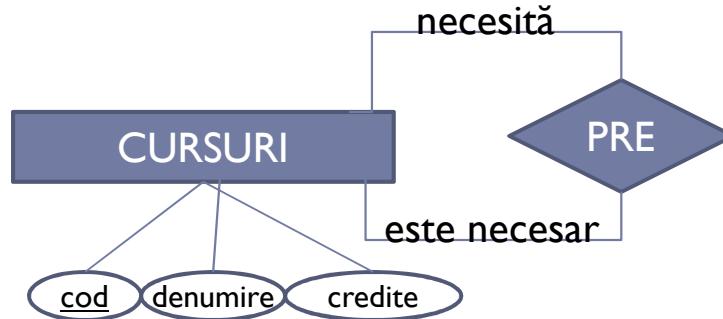
Quiz

-
- ▶ Modelați asocierea dintre STUDENTI și UNIVERSITĂȚI. Un student poate aplica la cel mult 5 universități și e necesar să aplice la cel puțin una. O universitate primește cel mult 10.000 aplicații.
 - ▶ Fie asocierea

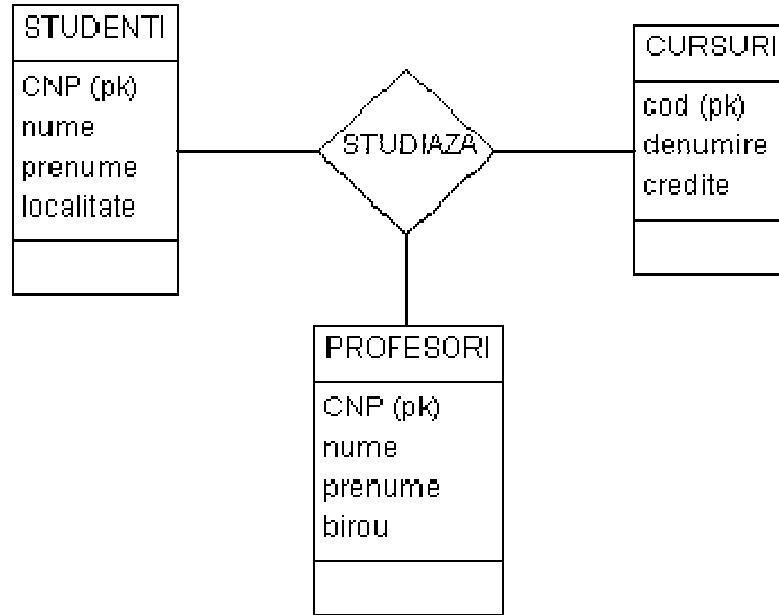


Care e numărul minim de instanțe pentru entitatea C1 și pentru C2?

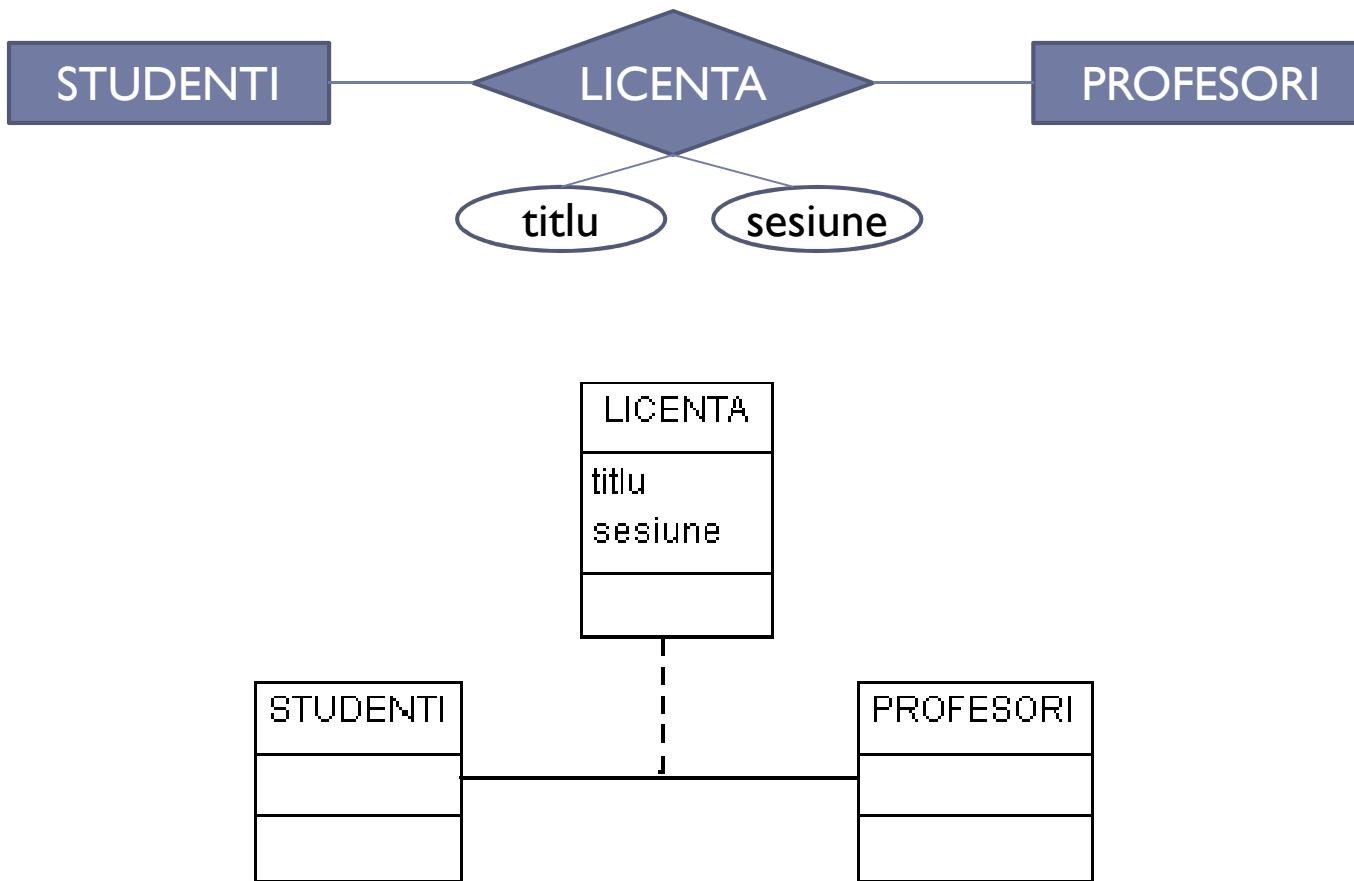
Asocieri recursive



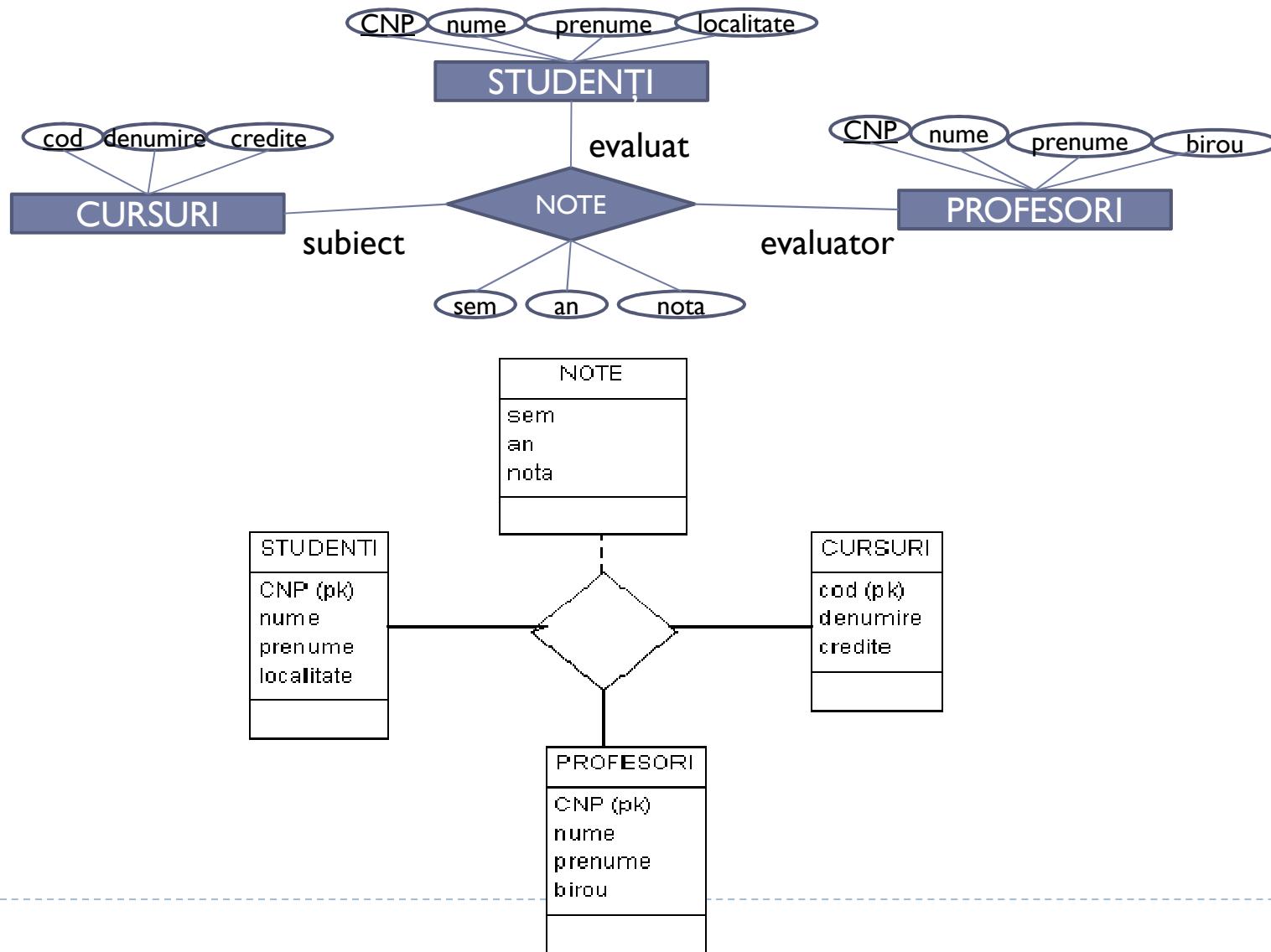
Asocieri n-are



Clase de asociere

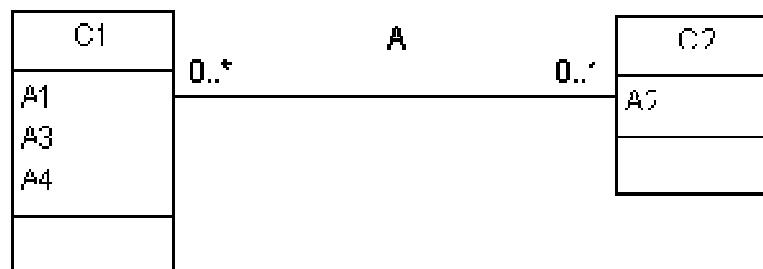
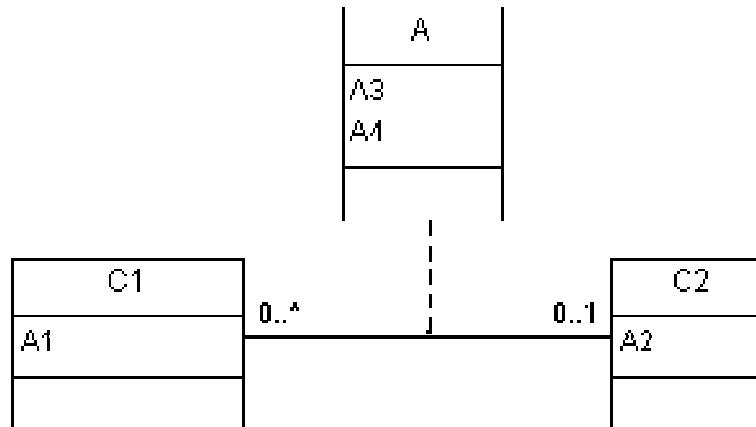


Clase de asociere



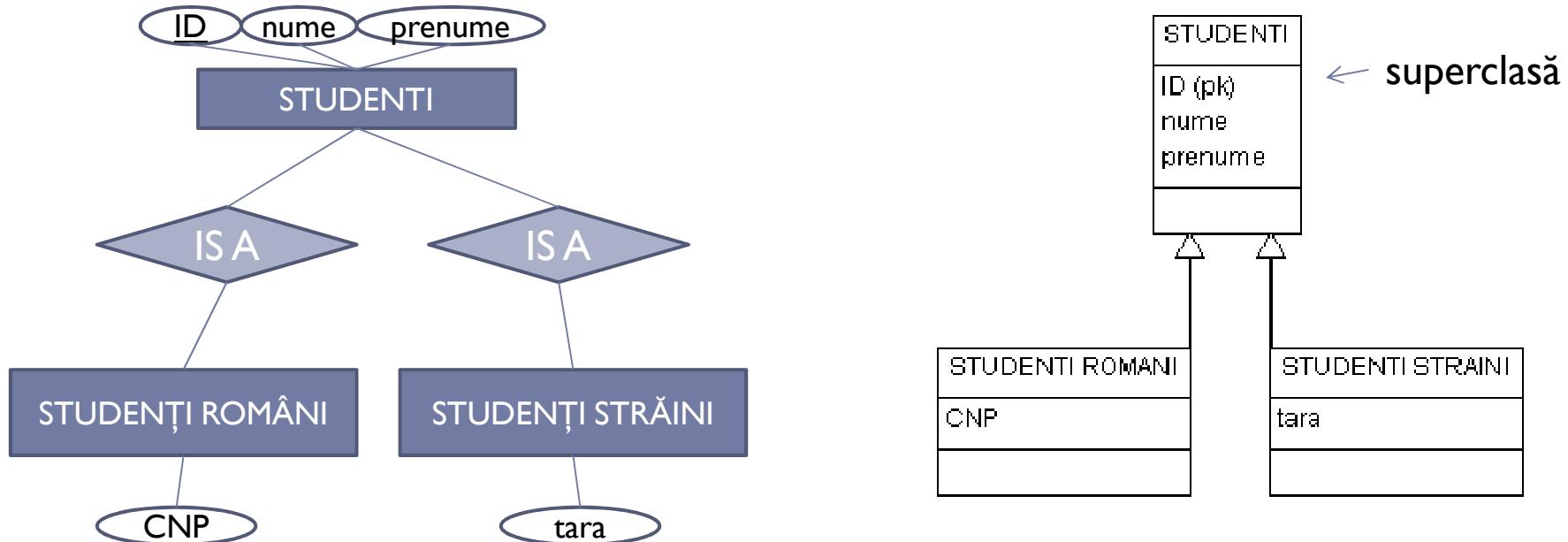
Eliminarea claselor de asociere

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



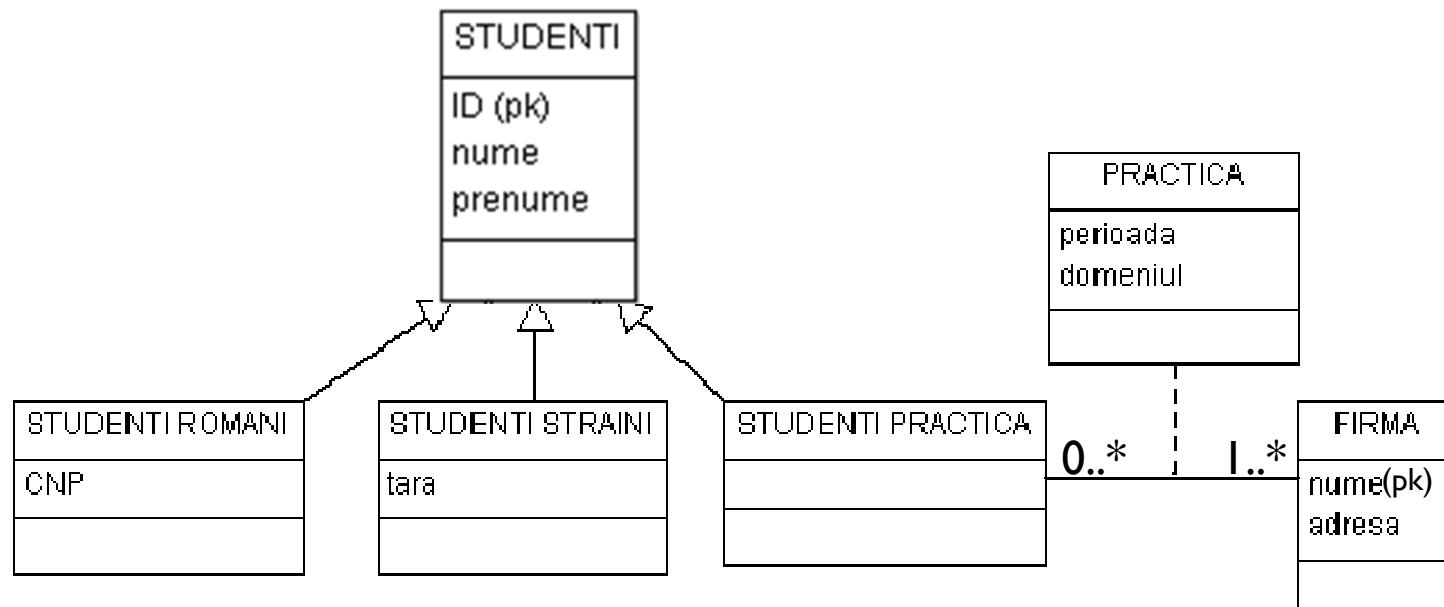
Subclasă

(1)



Specializare completă, disjunctă

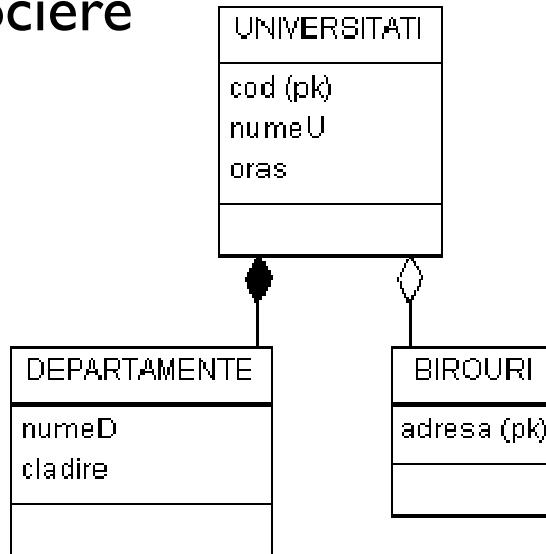
Subclasă (2)



Specializare completă, cu suprapunere

Compoziție și agregare

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

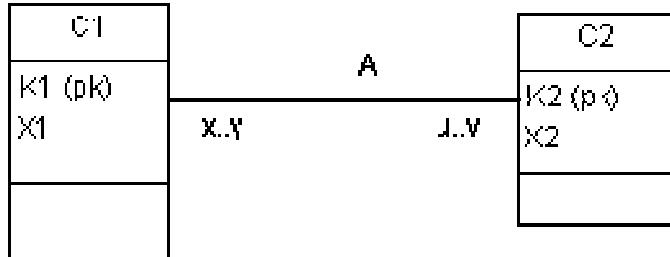


- Compoziția: **toate** obiectele unei clase **părți** aparțin obiectelor dintr-o clasă **compusă**; clasei **părți** îi corespunde de obicei o entitate slabă (multiplicitate **1..1**; fără cheie primară);
- Agregarea: **unele** obiecte dintr-o clasă aparțin obiectelor din altă clasă (multiplicitate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

Mapare E/R, UML -> schema BD relațională

E/R	UML	Schema relațională
Entitate + atrbute	Clasă	Relație cu cheie primară
Asociere fără atrbute proprii	Asociere	Relație cu chei străine
Asociere cu atrbute proprii	Clasă de asociere	Relație cu chei străine și alte atrbute
Specializare	Subclasă	Relație cu cheie primară (cea a superclasei) și atrbute particulare/specializate
	Compoziție și agregare	Relație cu cheie străină și atrbute particulare

Entități/clase și asocieri



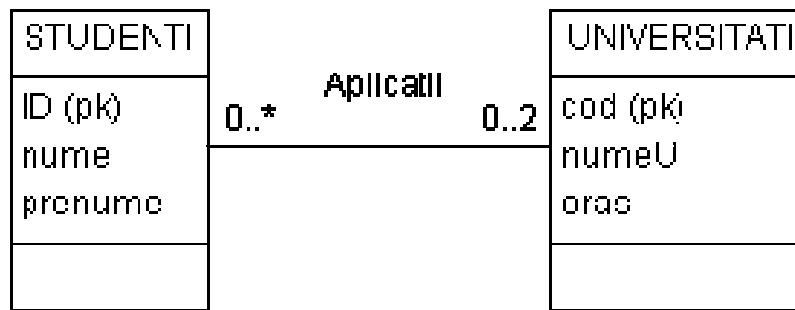
{C1(K1, X1), C2(K2, X2), A(K1, K2)}

- ▶ 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)	

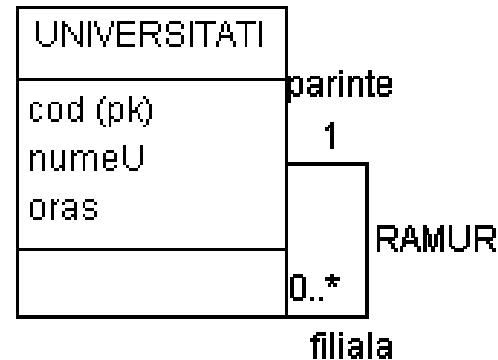
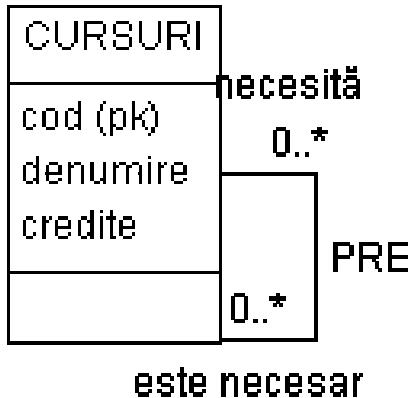
Quiz

- ▶ Fie diagrama



- ▶ Mai este posibilă renuntarea la relația corespunzătoare asocierii?

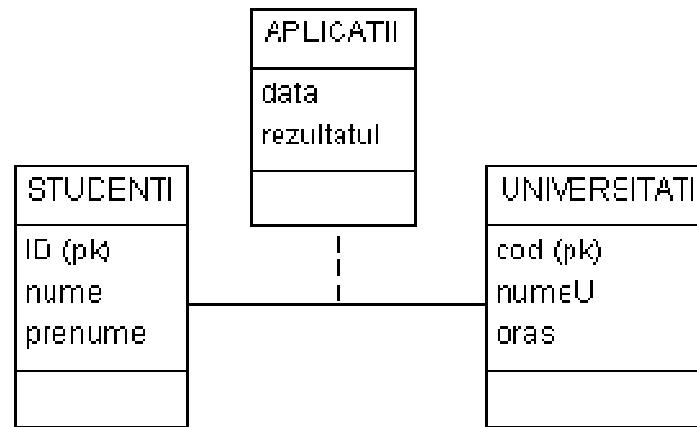
Asocieri recursive



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

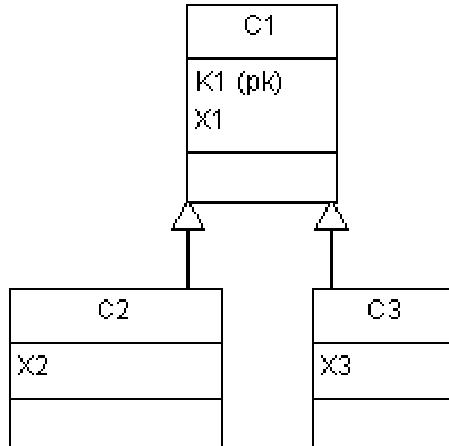
{UNIVERSITATI (cod, numeU, oras)
RAMURI (codFiliala, codParinte)}

Clase de asociere



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

Specializare/Subclase



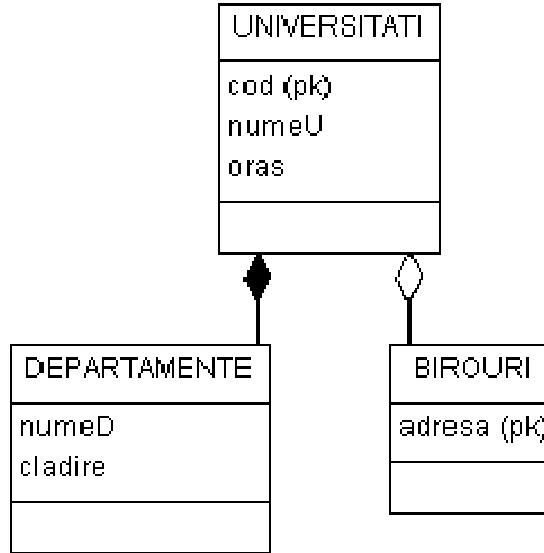
▶ Posibilități

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

Quiz

- Fie superclasa S cu un număr de subclase. Considerăm că relația de specializare este incompletă și cu suprapunere. Dacă n_1 , n_2 și n_3 reprezintă numărul total deuple necesare fiecărei scheme de decodificare din cele 3 date anterior (în ordinea dată), care este relația dintre cele 3 valori?
- $n_1 < n_2 < n_3$
 - $n_1 \leq n_2 \leq n_3$
 - $n_3 < n_2 < n_1$
 - $n_3 \leq n_2 \leq n_1$

Compoziție și agregare



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

NU acceptă NULL
acceptă NULL

Modelare ER/UML

Sumar

▶ PROS

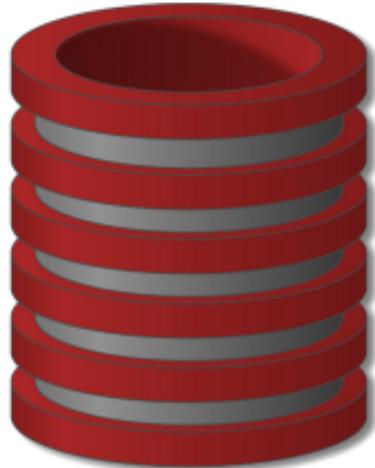
- ▶ Tehnică populară de modelare conceptuală
- ▶ Construcții expresive, descriu punctul de vedere personal asupra aplicației
- ▶ Permite exprimarea unor tipuri de constrângeri (chei primare, străine, multiplicitate, exclusivitate...)

▶ CONS

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

Bibliografie

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



BAZE DE DATE

Implementarea constrângerilor
View-uri

@FII (2011-2012)

prezentat de Mihaela Elena Breabă

Tematică curs

- ▶ **Proiectarea bazelor de date relaționale**
 - ▶ Normalizare și denormalizare
 - ▶ Modelul entitate-asociere, diagrame UML
 - ▶ **Constrângeri și declanșatoare**
 - ▶ View-uri
- ▶ Indecși
- ▶ **Procesarea interogărilor**
- ▶ **Managementul tranzacțiilor**
- ▶ **OLAP, Baze de date distribuite, NoSQL, Data Mining**

Obiective

- ▶ **Implementarea constrângerilor**
 - ▶ Constrângeri de integritate
 - ▶ Declansatoare

- ▶ **View-uri**
 - ▶ Rol și definire
 - ▶ Tratarea comenziilor de modificare

Constrângeri de integritate (statice)

(1)

- ▶ Restricționează stările posibile ale bazei de date
 - ▶ Pentru a elimina posibilitatea introducerii eronate de valori la inserare
 - ▶ Pentru a satisface corectitudinea la actualizare
 - ▶ Forțează consistență
 - ▶ Transmit sistemului informații utile stocării, procesării interogărilor
- ▶ Tipuri
 - ▶ Non-null
 - ▶ Chei
 - ▶ Integritate referențială
 - ▶ Bazate pe atribut și bazate pe uplu
 - ▶ Aserțiuni generale

Constrângeri de integritate

(2)

- ▶ **Declarare**
 - ▶ Odată cu schema
 - ▶ După crearea schemei
- ▶ **Realizare**
 - ▶ Verificare după fiecare modificare
 - ▶ Verificare la final de tranzacție

Constrângeri de integritate peste 1 variabilă

Implementare

`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 a5: (a5<11 and a5>4), (a5 between 5 and 10), (a5 in (5,6,7,8,9,10))...*

`)`

Constrângeri de integritate peste n variabile

Implementare

CREATE TABLE *tabel* (

a1 tip,

a2 tip,

a3 tip,

a4 tip,

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

unique(*a2,a3*), -- cheie candidat formată din 2 (sau mai multe) atributе

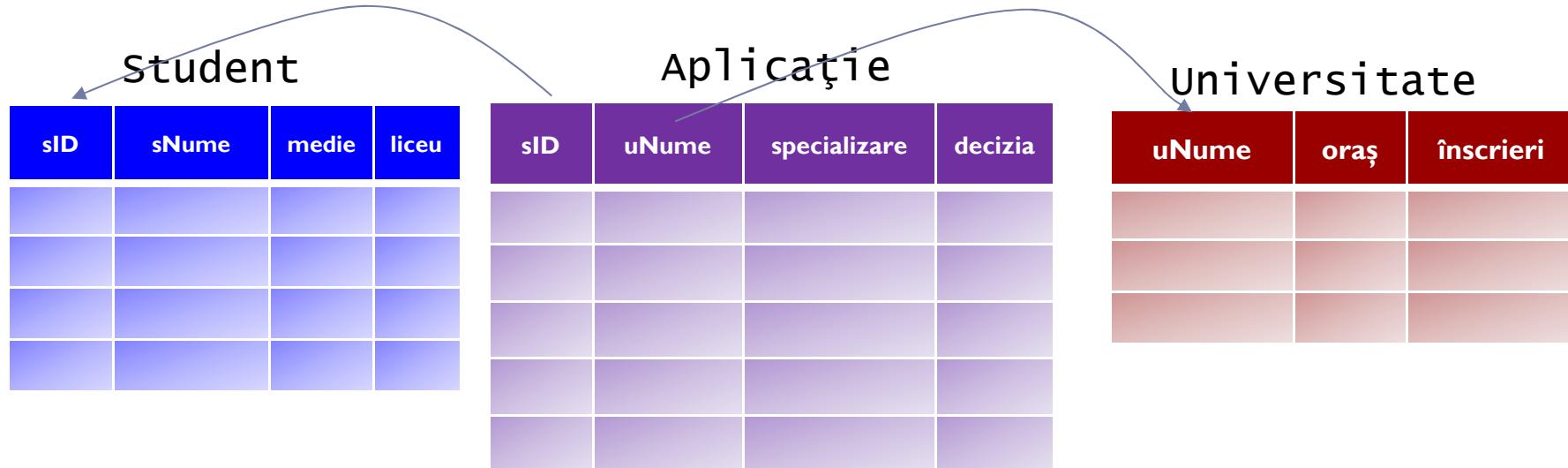
check (*condiție*), -- expresie booleană peste variabile declarate anterior: $((a1+a3)/2>=5)$

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

)

Integritate referențială

Definiții



- ▶ *Integritate referențială de la R.A la S.B:*
 - ▶ fiecare valoare în 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

Integritate referențială

Realizare

- ▶ Comenzi ce pot genera violări:
 - ▶ 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**

Integritate referențială

Problema “chicken-egg”

```
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)
```

INITIALLY DEFERRED DEFERRABLE; -- Oracle

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

Asertări

create assertion Key

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

create assertion ReferentialIntegrity

```
check (not exists (select * from Aplica  
                  where sID not in (select sID from Student)));
```

create assertion AvgAccept

```
check (3.0 < (select avg(medie) from Student  
                  where sID in  
                  (select SID from Aplica where decizie = 'DA')));
```

Constrângeri de integritate

Abateri de la standardul SQL

- ▶ Postgres, SQLite, Oracle forțează toate constrângerile anterioare, MySQL permite declararea constrângerilor de tip check dar nu le forțează
- ▶ Standardul SQL permite utilizarea de interogări în clauza check însă nici un SGBD nu le suportă
- ▶ Nici un SGBD nu a implementat asertiunile din standardul SQL, funcționalitatea lor fiind furnizată de declanșatoare

...DEMO...

(fișierul *constrângeri.sql*)

Declanșatoare (dynamice)

- ▶ 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

Declanșatoare Implementare

Create Trigger *nume*

Before|After|Instead Of *evenimente*

[*variabile-referențiate*]

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

[**When (conditie)**] -- 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 ROW AS** *var* – pentru ev. *DELETE, UPDATE*
 - ▶ **NEW ROW AS** *var* – pentru ev. *INSERT, UPDATE*
 - ▶ **OLD TABLE AS** *var*
 - ▶ **NEW TABLE AS** *var*

} doar pentru triggere de
tip row

Declanșatoare

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)

Declanșatoare

Exemplu (2)

Create Trigger *IncreaseInserts*

After Insert On *T*

Referencing New Row As *NR*, New Table As *NT*

For Each Row

When (Select Avg(*V*) From *T*) < (Select Avg(*V*) From *NT*)

Update *T* set *V*=*V*+10 where *K*=*NR.K*

- ▶ nu e posibilă definirea unui declanșator echivalent de tip statement
- ▶ are un comportament nedeterminist

Declanșatoare Capcane

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

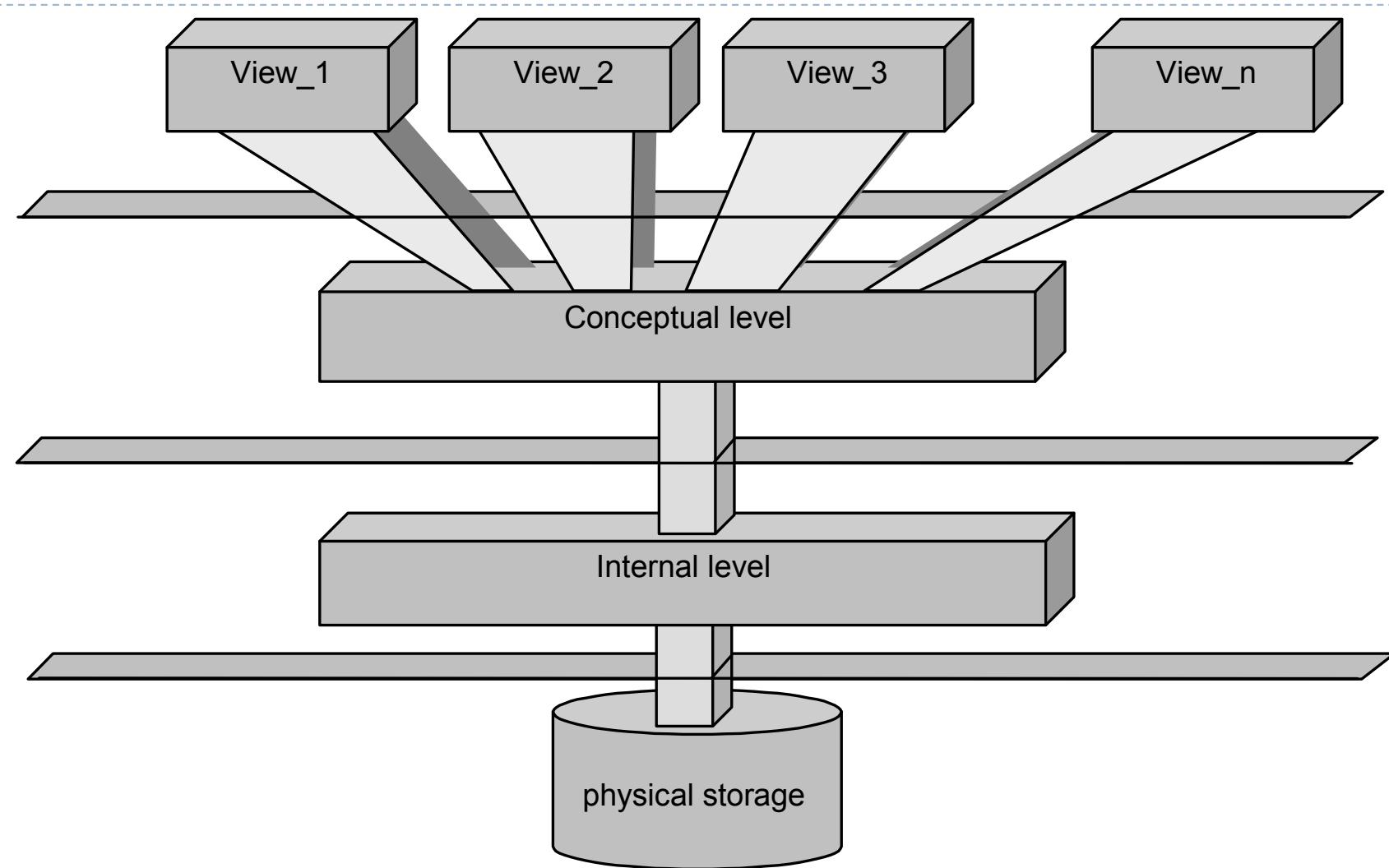
Declanșatoare Abateri de la standardul SQL

- ▶ **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**

...DEMO...

(fișierul *declansatoare.sql*)

View-uri



Motivație

- ▶ ascunderea unor date față de unii utilizatori
 - ▶ ușurarea formulării unor interogări
 - ▶ acces modular la baza de date
-
- ▶ aplicațiile reale tind să utilizeze foarte multe view-uri

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*>

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
 - ▶ $R(A,B), V(A) = \Pi_A(R)$, Insert into V values(3)
 - ▶ $R(N), V(A) = \text{avg}(N)$, update V set A=7

Modificarea view-urilor

Abordări

- ▶ creatorul view-ului rescrie toate comenziile de modificare posibile cu ajutorul declanșatorului de tip **INSTEAD OF**
 - ▶ acoperă toate cazurile
 - ▶ nu garantează corectitudinea
- ▶ standardul SQL prevede existența de view-uri inherent actualizabile (**updatable views**) dacă:
 - ▶ view-ul e creat cu comandă select fără clauza DISTINCT pe o singură tabelă T
 - ▶ atrbutele 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

View-uri materializate

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

- ▶ are loc crearea unui nou tabel V cu schema dată de rezultatul interogării
- ▶ upile 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

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

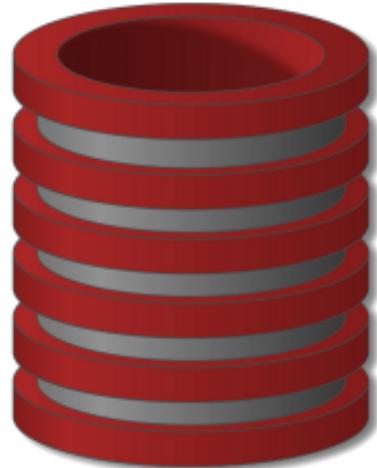
...DEMO...
(fișierul views.sql)

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/general05.htm#i1006732

 - ▶ <http://www.oracle-base.com/articles/9i/MutatingTableExceptions.php>



BAZE DE DATE

Indecsi

@FII (2011-2012)

prezentat de Mihaela Elena Breabăñ

Tematică curs

▶ Proiectarea bazelor de date relaționale

- ▶ Normalizare și denormalizare
- ▶ Modelul entitate-asociere, diagrame UML
- ▶ Constrângeri și declanșatoare
- ▶ View-uri
- ▶ Indecși

- ▶ Procesarea interogărilor
- ▶ Managementul tranzacțiilor
- ▶ OLAP, Baze de date distribuite, NoSQL, Data Mining

Indecși

Cuprins

- ▶ Concepte de bază
- ▶ Indecși ordonați
 - ▶ Indecși secvențiali
 - ▶ B⁺-Arbore
- ▶ Hashing
 - ▶ Hashing static
 - ▶ Hashing dinamic
- ▶ Acces multi-cheie și Indecși Bitmap
- ▶ Definirea indecșilor în standardul SQL
- ▶ Indecși în Oracle

Motivație

- ▶ Bazele de date consumă mult timp căutând

```
SELECT * FROM Student  
WHERE sID=40
```

Cum putem regăsi rezultatul în următoarele situații:

a) Ordine aleatoare a datelor

sID	sNume	medie
20	Ioana	9.5
40	Andrei	8.66
10	Tudor	8.55
30	Maria	8.33
70	Alex	9.33

b) Date secvențiale/ordonate

sID	sNume	medie
10	Tudor	8.55
20	Ioana	9.5
30	Maria	8.33
40	Andrei	8.66
70	Alex	9.33

Ideea de bază

- ▶ Căutarea binară
 - ▶ Complexitate: $\log_2(N)$
 $(\log_2(100\ 000)=17)$
- ▶ Construirea unei structuri auxiliare care să ajute la localizarea unei înregistrări
 - ▶ Mecanismele de indexare sunt utilizate pentru a mări viteza de acces la datele dorite

Concepte de bază

- ▶ Un index este asociat cu o **cheie de căutare** – atribut sau set de atrbute dintr-un fișier/tabel/relație utilizate pentru a căuta înregistrări în fișier
- ▶ **Fișier index** – constă din **înregistrări index** de forma

cheie de căutare	pointer
------------------	---------
- ▶ Sortare:
 - ▶ a indexului pe baza cheii de căutare
 - ▶ a fișierului/tabelei/relației stocate
- ▶ Un fișier/tabel/relație poate avea asociați mai mulți indecsi
- ▶ Fișierele index sunt de obicei de dimensiuni mult mai mici decât fișierul original cu date

Metrici de evaluare a indecșilor

- ▶ Timpul de acces
- ▶ Timpul de inserare
- ▶ Timpul de ștergere
- ▶ Spațiul necesar
- ▶ Tipurile de acces suportate eficient – influențează alegerea indexului
 - ▶ Înregistrări cu o valoare specificată a atributului
 - ▶ Înregistrări cu valoarea atributului inclusă într-un interval specificat

Tipuri de indecși

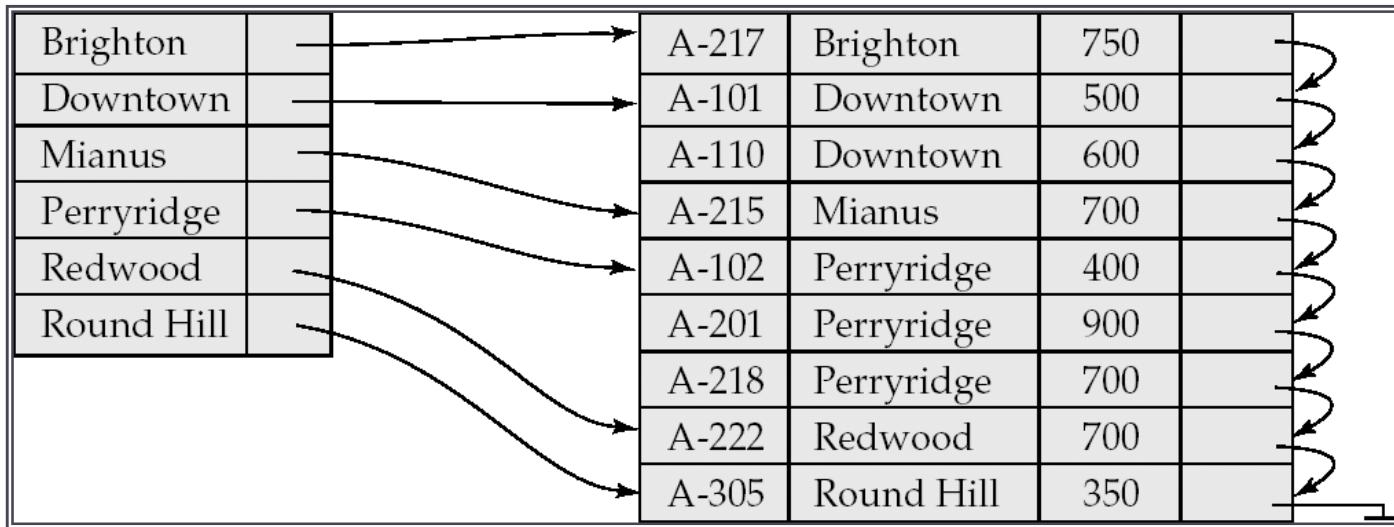
- ▶ **Indecși ordonați:** cheile de căutare sunt stocate într-o anumită ordine
- ▶ **Indecși hash:** cheile de căutare sunt distribuite uniform în bucket-uri cu ajutorul unei funcții hash
 - ▶ **Bucket:** o unitate de stocare conținând una sau mai multe înregistrări
- ▶ **Indecși bitmap:** asociați cheilor de căutare de tip attribute discrete, reprezintă distribuția valorilor sub formă de matrice binară

Indecși secvențiali

- ▶ Intrările index sunt sortate după cheia de căutare
 - ▶ Ex: catalogul cu autori într-o bibliotecă
- ▶ **Index primar:** indexul a cărui cheie de căutare definește și ordonarea secvențială a fișierului/tabelei/relației
 - ▶ denumit și **index de grupare** (clustering index)
 - ▶ cheia de căutare a unui index primar este de obicei cheia primară dar nu e obligatoriu!!!
 - ▶ o tabelă poate avea cel mult un index primar
- ▶ **Index secundar (nonclustering):** index a cărui cheie de căutare specifică o ordonare diferită de ordonarea secvențială a fișierului cu date
- ▶ **Fișier index-secvențial:** combinația fișier ordonat secvențial cu un index primar

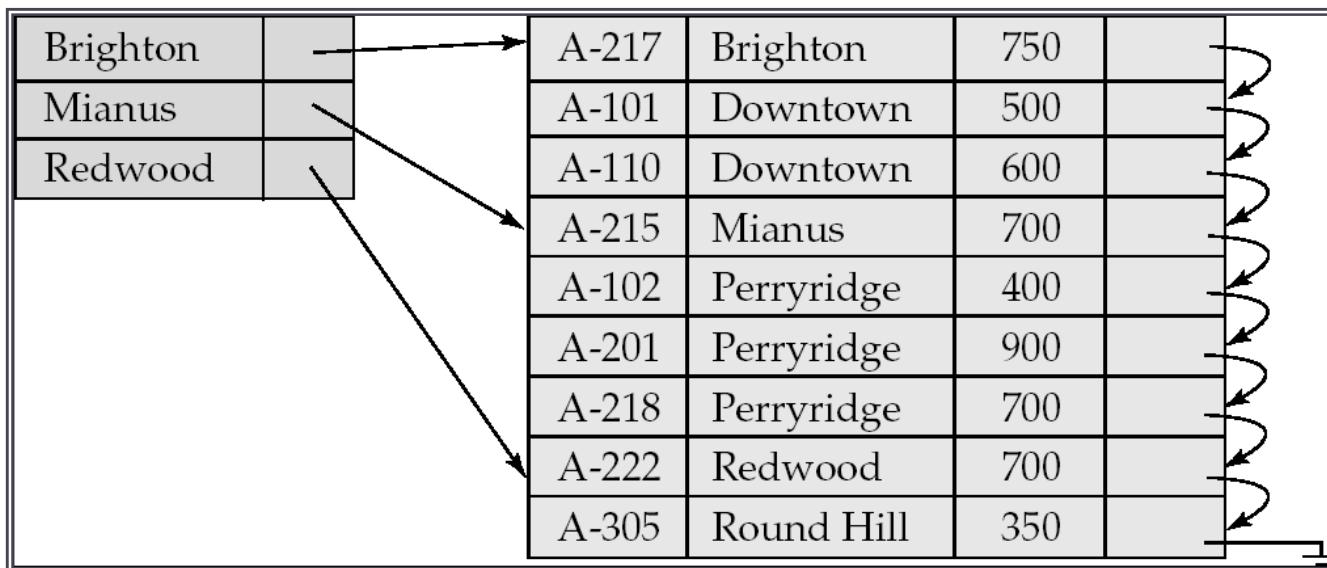
Fișiere index dense

- ▶ **Index dens:** există înregistrări index pentru fiecare valoare a cheii de căutare în fișier/tabel/relație
- ▶ Dacă indexul e primar va păstra câte un singur pointer-doar la prima intrare cu valoarea respectivă
- ▶ Dacă indexul e secundar vor fi necesari mai mulți pointeri la o singură valoare a cheii de căutare



Fișiere index rare

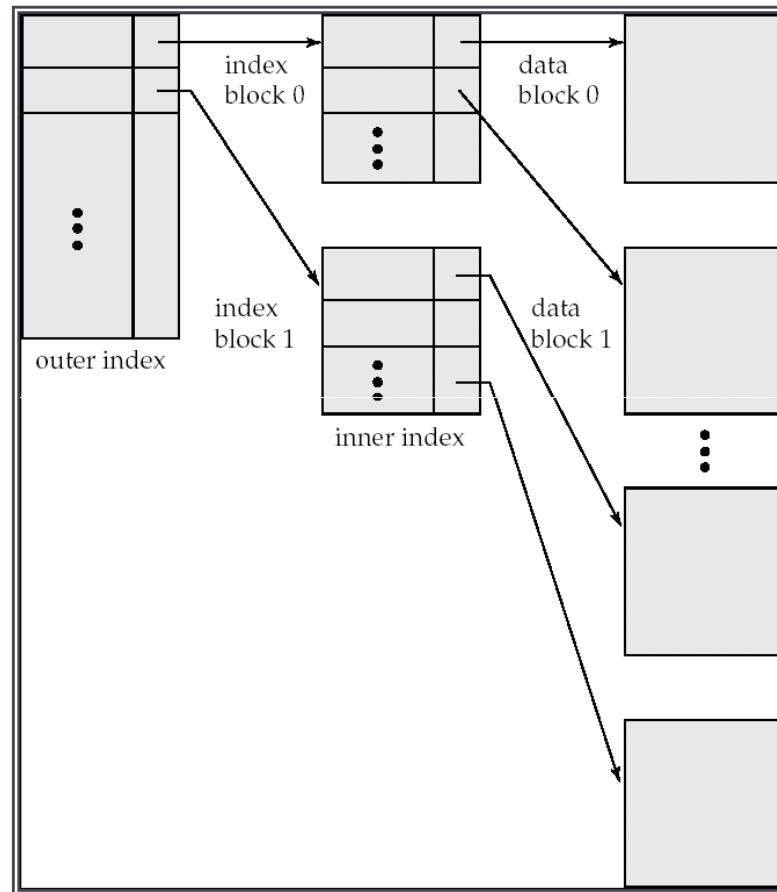
- ▶ **Index rar:** conține valori doar pentru unele valori a cheii de căutare
 - ▶ Aplicabil doar când înregistrările sunt ordonate secvențial după cheia de căutare
 - ▶ Balansul timp-spațiu
- ▶ Pentru a localiza o înregistrare cu valoarea k a cheii de căutare:
 - ▶ Se determină înregistrarea index cu cea mai mare valoare a cheii de căutare <k
 - ▶ Se caută secvențial în fișier începând cu înregistrarea spre care indică înregistrarea index



Indecși multi-nivel

- ▶ **Index multi-nivel:** index asociat unui alt index
 - ▶ Dacă indexul primar nu încape în memorie accesul devine costisitor
 - ▶ Soluția: indexul primar păstrat pe disc este tratat ca un fișier secvențial și se construiește un index rar pentru el
- ▶ Indexul extern – un index rar al indexului primar
- ▶ Index intern – fișierul index primar
- ▶ Dacă și indexul extern este prea mare pentru a încăpea în memorie, se creează un index pe un nou nivel, etc...
- ▶ Indecșii de pe toate nivelele trebuie actualizați la inserare și ștergere în fișierul cu date

Indeçsi multi-nivel



Actualizarea indecșilor secvențiali

Ștergere

- ▶ Dacă înregistrarea ștearsă este singura care conține o valoare particulară a cheii de căutare, aceasta este ștearsă și din index

- ▶ Ștergerea în
 - ▶ Indecși denși: similară ștergerii din fișierul cu date
 - ▶ Indecși rari:
 - ▶ dacă există o intrare a cheii de căutare în index aceasta este înlocuită cu următoarea valoare a cheii de căutare din fișier (în ordinea cheii de căutare)
 - ▶ dacă următoarea valoare a cheii de căutare deja are o intrare în index, este efectuată ștergerea.

Actualizarea indecșilor secvențiali

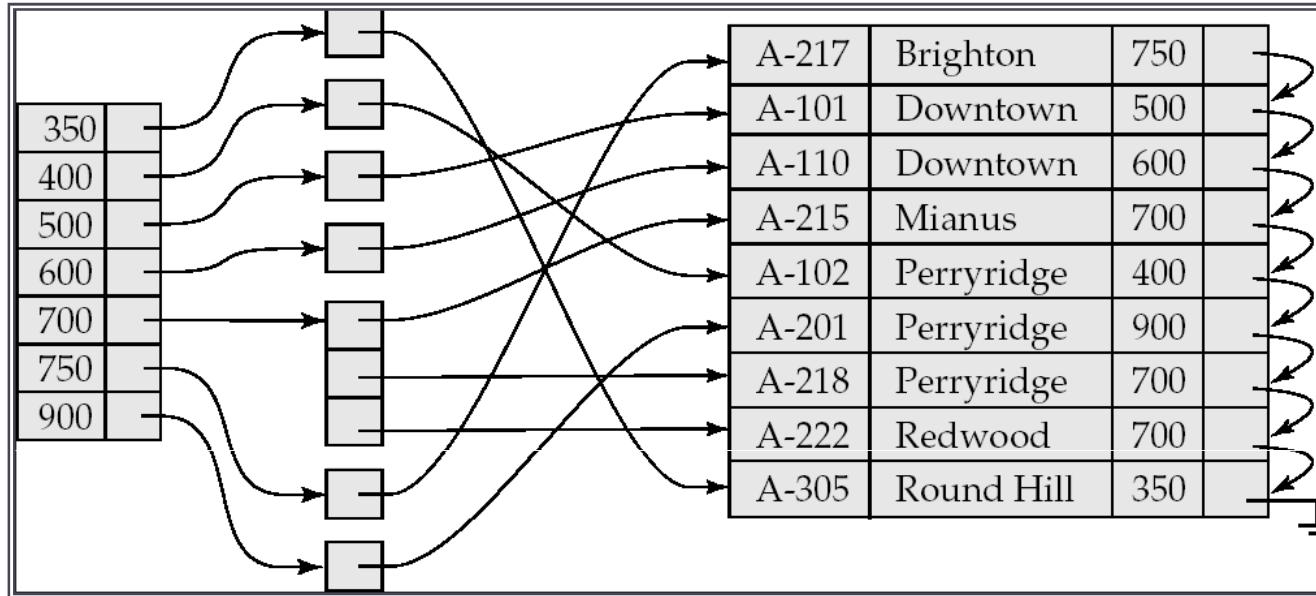
Inserare

- ▶ Este necesară localizarea valorii cheii de căutare ce apare în uplul inserat
- ▶ Indecși denși: dacă valoarea nu apare în index se va inseră
- ▶ Indecși rari: dacă indexul păstrează o intrare pentru fiecare bloc al fișierului nu sunt necesare modificări decât dacă un nou bloc este creat (prima valoare a cheii de căutare ce apare în noul bloc este inserată în index)

- ▶ Inserarea (și ștergerea) pentru indecși multi-nivel sunt extensii simple a algoritmilor uni-nivel prezentate

Indecși secundari

Exemplu



- ▶ În relația conturi sortată după filiale, care sunt conturile cu o balanță specificată?
- ▶ Soluția: index secundar (dens!)
- ▶ Pentru a implementa relația de tip multi-la-unu dintre index și datele destinație se utilizează referințe la bucket-uri de pointeri

Fișiere index de tip B⁺-arbori

Motivație

- ▶ Organizarea secvențială a indecsilor se degradează pe măsură ce dimensiunea crește
- ▶ Reconstruirea indecsilor la intervale de timp e necesară însă costisitoare
- ▶ Indexul B⁺-arbore
 - ▶ mărește viteza de localizare și elimină necesitatea constantă de reorganizare
 - ▶ utilizați extensiv

Structura B⁺-arbore

(1)

- ▶ Un arbore balansat astfel încât fiecare drum de la rădăcină la frunze are aceeași lungime
- ▶ Un nod tipic



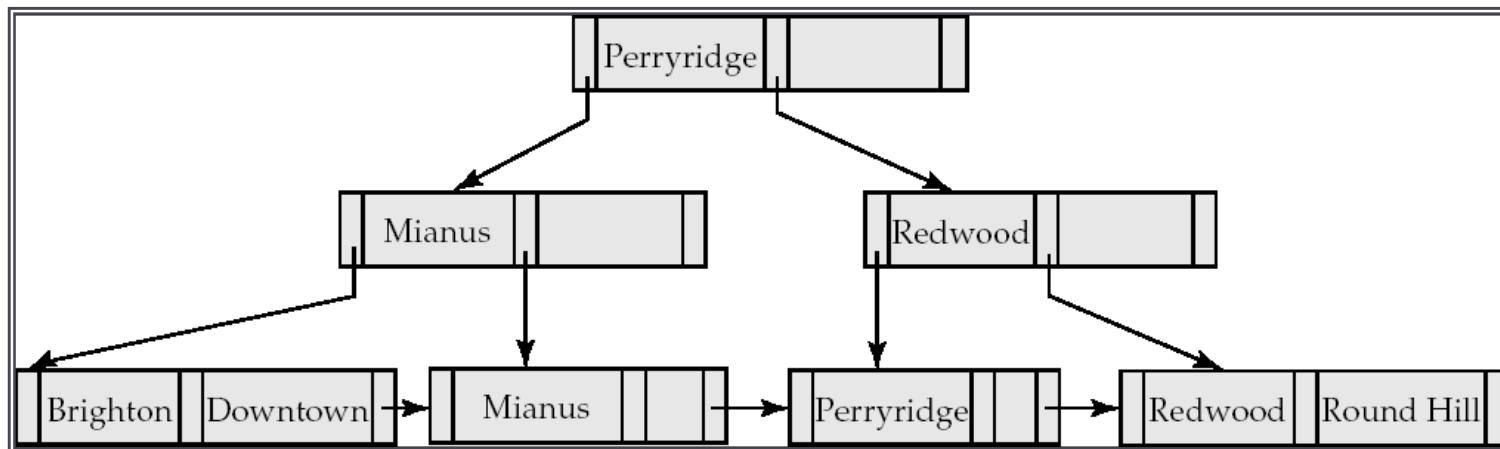
- ▶ K_i sunt valori a cheii de căutare
- ▶ P_i sunt pointeri către
 - ▶ noduri copil/descendente (noduri care nu sunt frunze)
 - ▶ înregistrări sau bucket-uri de înregistrări (noduri frunză)
- ▶ Arborele este definit de o constantă n care specifică numărul maxim de valori dintr-un nod ca fiind $(n-1)$ și numarul maxim de pointeri egal cu n
 - ▶ De obicei dimensiunea unui nod e cea a unui bloc
- ▶ Valorile cheii de căutare într-un nod sunt ordonate

$$K_1 < K_2 < K_3 < \dots < K_{n-1}$$

Structura B⁺-arbore

(2)

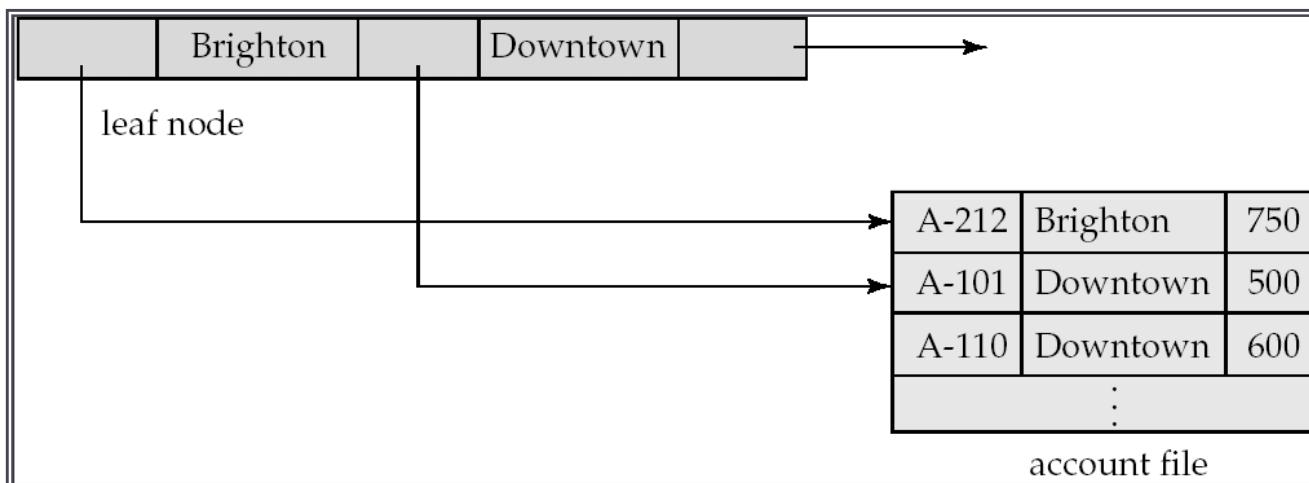
- ▶ Numărul de descendenți a unui nod este egal cu numărul de pointeri și este constrâns la o valoare între $\lceil n/2 \rceil$ și n
- ▶ Un nod frunză are între $\lceil (n-1)/2 \rceil$ și $n-1$ valori
- ▶ Cazuri speciale
 - ▶ Dacă rădăcina nu e frunză are măcar 2 descendenți
 - ▶ Dacă rădăcina e frunză poate avea între 0 și $n-1$ valori



Noduri frunză în B⁺-arbore

▶ Proprietăți a unui nod frunză

- ▶ Fiecare pointer P_i dintre P_1, \dots, P_{n-1} indică spre o înregistrare în fișier/tabel/relație cu valoarea K_i , a cheii de căutare, sau spre un bucket de pointeri către înregistrări care au aceeași cheie de căutare K_i
- ▶ Dacă nodurile N și M apar în această ordine de la stânga la dreapta, atunci valorile cheii de căutare din nodul N sunt mai mici decât cele din nodul M
- ▶ P_n indică spre următorul nod în ordinea valorilor cheii de căutare



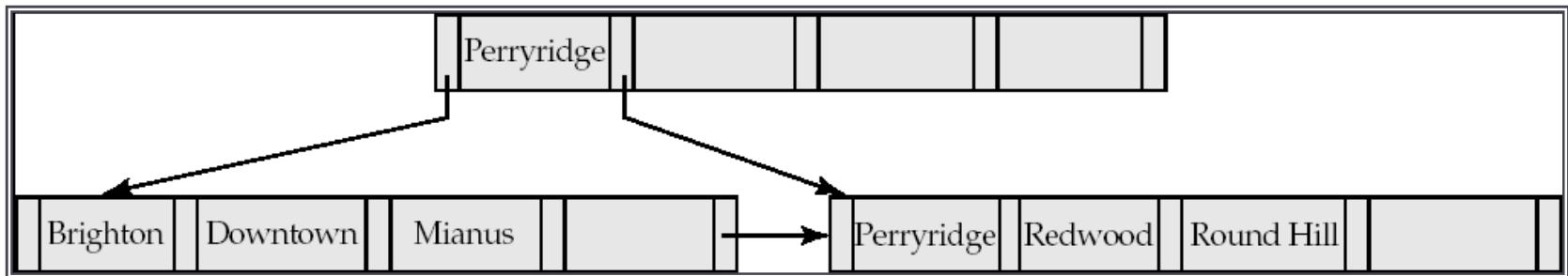
Noduri care nu sunt frunze în B^+ -arbore

- ▶ formează un index rar multi-nivel pentru nodurile frunză
- ▶ Pentru un nod cu n pointeri:
 - ▶ Toate valorile cheii de căutare din subarborele spre care P_1 indică sunt mai mici decât K_1 ,
 - ▶ Pentru P_i , $2 \leq i \leq n - 1$, toate valorile cheii de căutare din subarborele spre care indică sunt mai mari sau egale cu K_{i-1} și mai mici decât K_i
 - ▶ Toate valorile cheii de căutare din subarborele spre care indică P_n sunt mai mari sau egale cu K_{n-1}

P_1	K_1	P_2	\dots	P_{n-1}	K_{n-1}	P_n
-------	-------	-------	---------	-----------	-----------	-------

B⁺-arbore

Exemplu



- ▶ n=5
 - ▶ Nodurile frunză au între 2 și 4 valori
 - ▶ Nodurile care nu sunt frunză au între 3 și 5 noduri copil
 - ▶ Rădăcina are măcar 2 noduri copil

B⁺-arbori

Observații

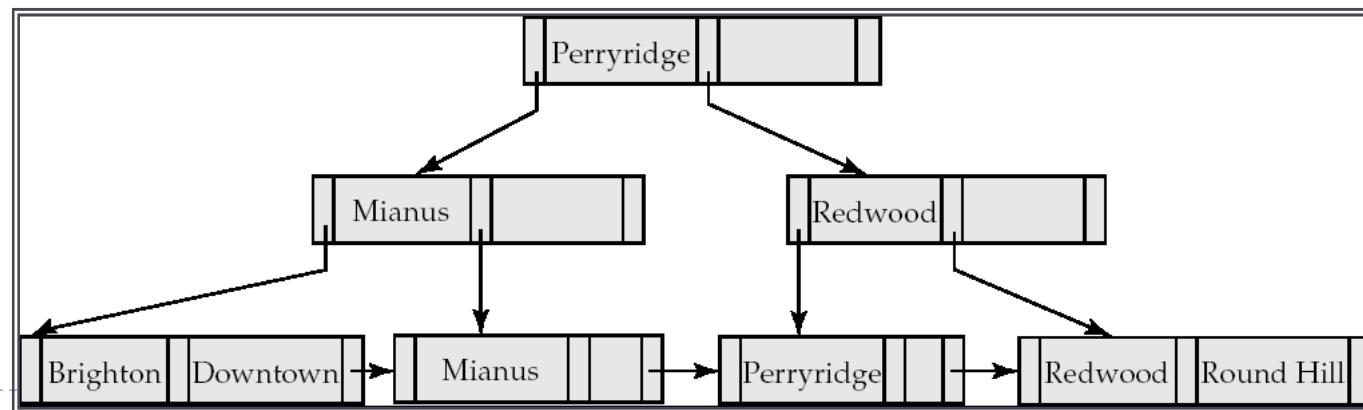
- ▶ Fiindcă conexiunile dintre noduri se realizează prin pointeri, blocuri apropiate logic nu trebuie să fie apropiate și fizic
- ▶ Nivelele diferite de nivelul frunză formează o ierarhie de indecsări rari
- ▶ B⁺-arborele conține un număr relativ mic de nivale
 - ▶ Cel mult $\lceil \log_{\lceil n/2 \rceil}(K) \rceil$ pentru k valori a cheii de căutare
 - ▶ Nivelul imediat următor rădăcinii: cel puțin $2 * \lceil n/2 \rceil$ valori
 - ▶ Următorul: cel puțin $2 * \lceil n/2 \rceil * \lceil n/2 \rceil$
 - ▶ etc...
- ▶ Inserările și ștergerile se fac eficient, restructurarea indexului necesitând timp logaritmic

Interogări pe B^+ -arbori

Algoritm

Determinarea tuturor înregistrărilor cu valoarea k a cheii de căutare

1. $N = \text{rădăcina}$
2. Repetă
 1. Caută în N cea mai mică valoare a cheii de căutare $>k$
 2. Dacă aceasta există și e egală cu K_i , atunci $N = P_i$
 3. Altfel $N = P_n$ ($k \geq K_{n-1}$)
- Până N este nod frunză
3. Dacă există $K_i = k$, pointerul P_i indică înregistrarea dorită
4. Altfel nu există înregistrarea cu cheia de căutare k



Interogări pe B^+ -arbori

Observații

- ▶ Un nod este în general de aceeași dimensiune ca unui bloc pe disc - 4Kbytes
 - ▶ Pt. fiecare intrare în index se utilizează 40 bytes ($n=100$)
 - ▶ Fiecare acces a unui nod poate necesita o citire pe disc (<20 milisecunde)
-
- ▶ *Pentru 1 milion valori a cheii de căutare și $n=100$, câte noduri (blocuri pe disc) sunt accesate la o căutare în B^+ -arbore?*
 - ▶ *Dar dacă se utilizează un index secvențial?*

Actualizări în B⁺-arbori

Inserarea

1. Determină nodul frunză în care va apărea valoarea cheii de căutare
2. Dacă valoarea e deja prezentă într-un nod frunză
 1. Adaugă înregistrarea în fișier/tabel/relație
 2. Dacă e necesar adaugă un pointer în bucket
3. Dacă nu e prezentă valoarea
 1. Adaugă înregistrarea în fișier/tabel/relație
 2. Dacă e loc în nodul frunză inserează perechea (valoare cheie, pointer)
 3. Altfel divide nodul

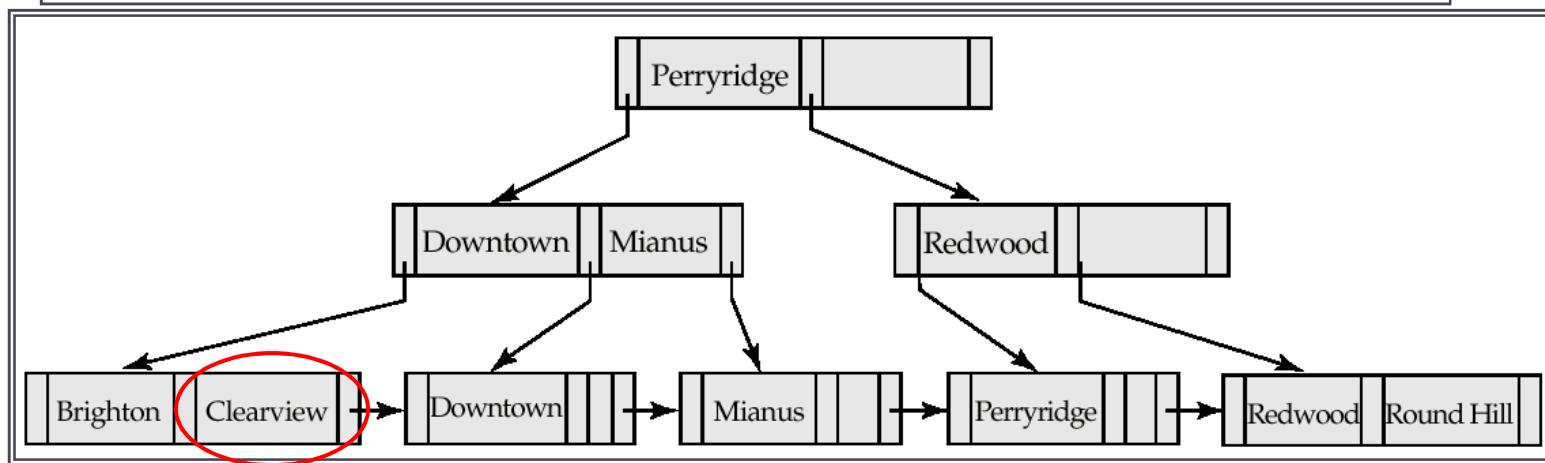
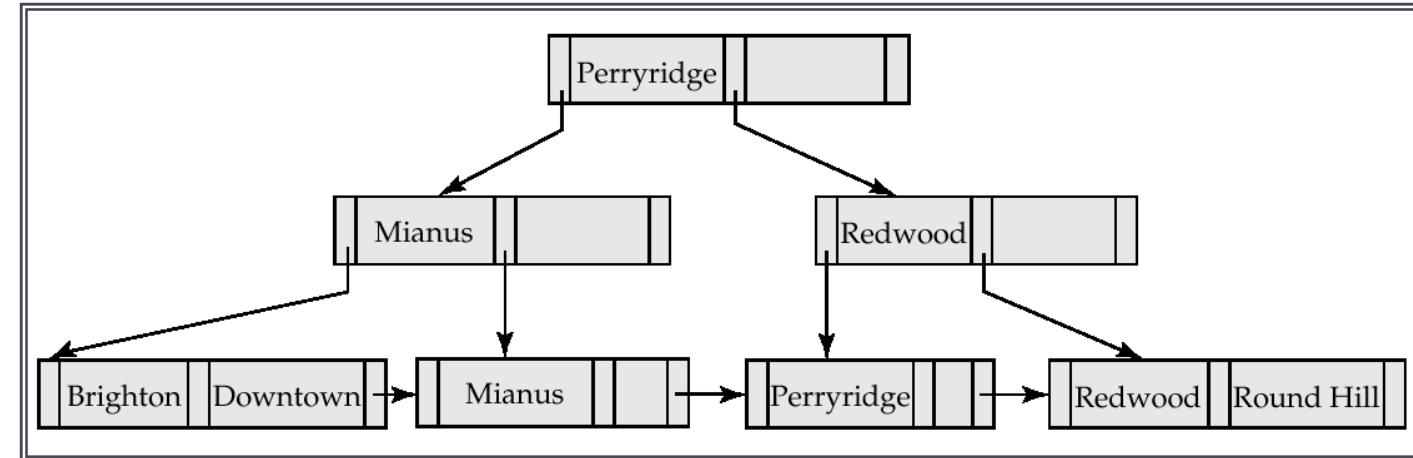
Actualizări în B⁺-arbori

Inserarea: divizarea nodurilor

- ▶ Divizarea unui nod frunză
 1. Se iau cele n perechi (inclusiv cea care urmează a fi inserată) ordonate. În nodul original se pun primele $\lceil n/2 \rceil$ perechi iar restul într-un nod nou
 2. Fie p noul nod și k cea mai mică valoare din p. Insereză (k,p) în părintele nodului care se divide
 3. Dacă părintele este plin acesta se divide la rândul său și divizarea se propagă înapoi în sus până când un nod nu este plin. În cel mai rău caz nodul rădăcină este divizat ceea ce crește înălțimea arborelui cu 1.
 - ▶ Divizarea unui nod plin intern N la inserarea unei perechi (k,p)
 1. Se crează un nod temporar M cu spațiu pentru $n+1$ pointeri și n valori în care se copiează N și perechea (k,p)
 2. Se copiază $P_1, K_1, \dots, K_{\lceil n/2 \rceil - 1}, P_{\lceil n/2 \rceil}$ din M înapoi în N
 3. Se copiază $P_{\lceil n/2 \rceil + 1}, K_{\lceil n/2 \rceil + 1}, \dots, K_n, P_{n+1}$ din M într-un nou nod N'
 4. Insereză $(K_{\lceil n/2 \rceil}, N')$ în părintele lui N
-

Actualizări în B⁺-arbori

Inserarea: Exemplu



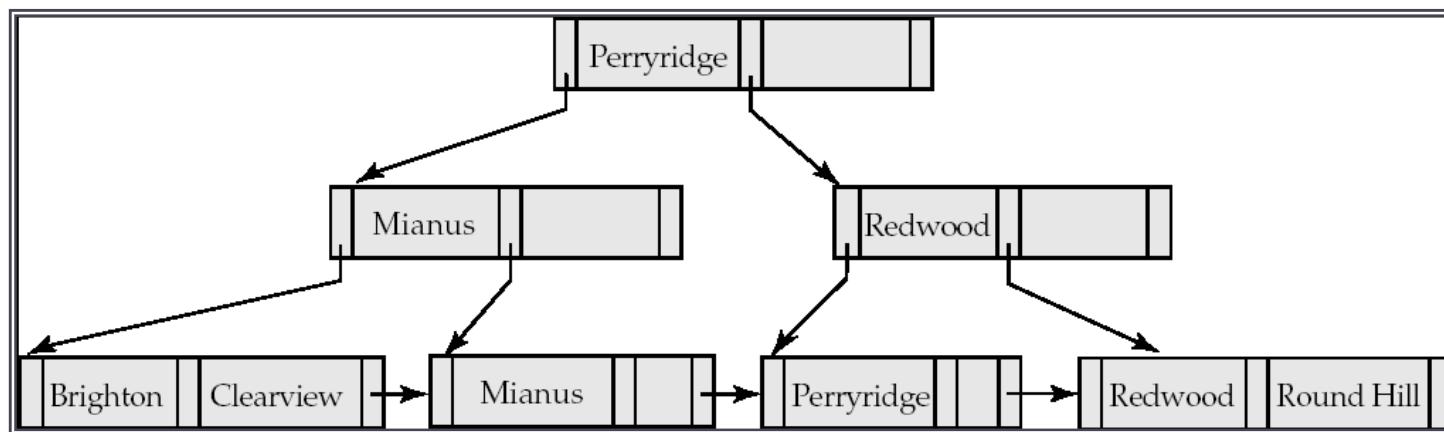
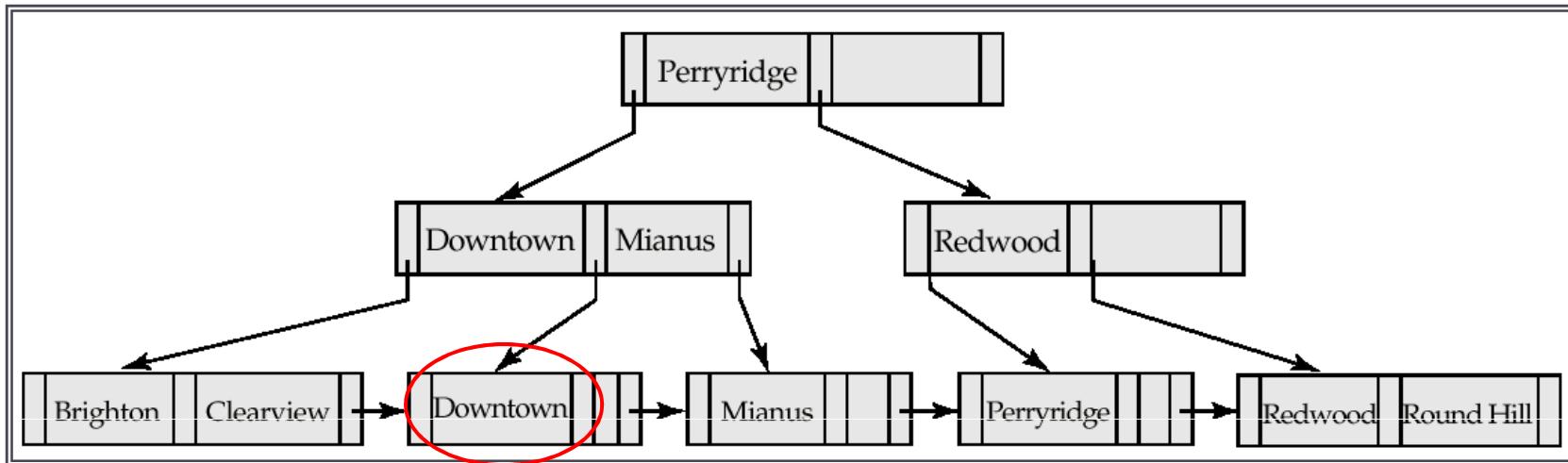
Actualizări în B⁺-arbori

Ştergerea

1. Șterge înregistrarea din fișier/tabel/relație
2. Dacă înregistrarea face parte dintr-un bucket, e ștearsă din acesta. Altfel (sau dacă bucketul devine gol) șterge din nodul frunză perechea (valoare cheie,pointer)
3. Dacă nodul va avea prea puține intrări în urma ștergerii și ele încap într-un nod vecin va avea loc unirea:
 1. Se inserează toate intrările în nodul stâng și se șterge celălalt
 2. Se șterge perechea (K_{i-1}, P_i), unde P_i este pointerul către nodul șters de la părinte. Dacă e necesar se propagă ștergerea recursiv în sus. Dacă nodul rădăcină rămâne cu un singur pointer va fi șters.
4. Altfel, dacă nodurile nu încap în vecin se redistribuie pointerii:
 1. Se redistribuie astfel încât avem satisfăcută condiția de minim în ambele noduri
 2. Se actualizează valoarea cheii de căutare corespunzătoare în părinte

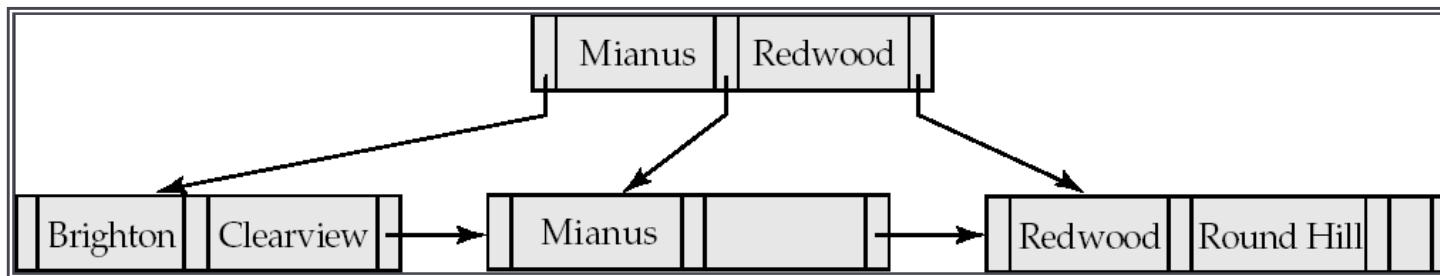
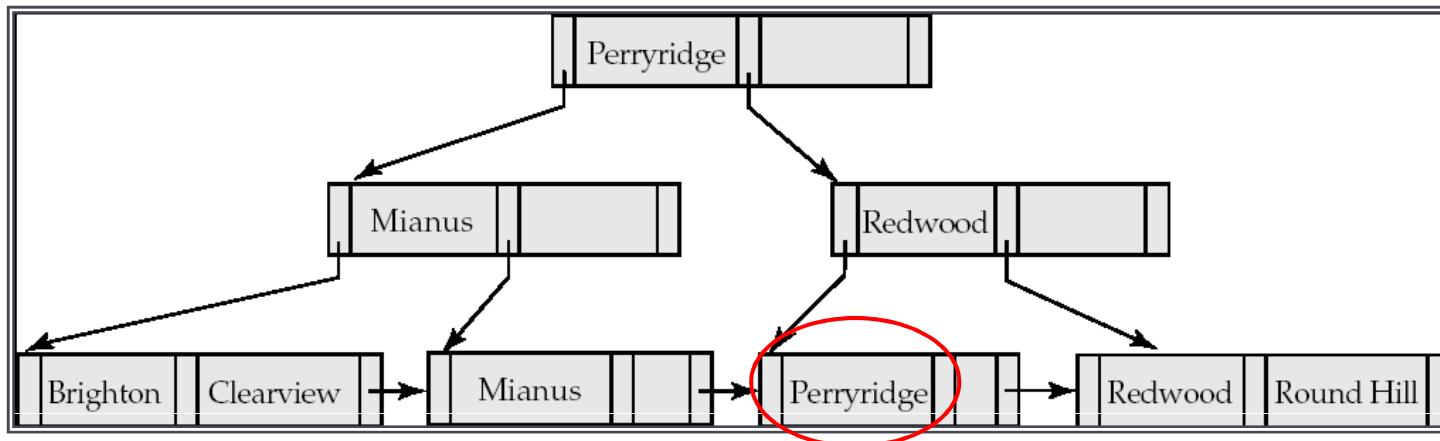
Actualizări în B⁺-arbori

Ștergerea: Exemplu (1)



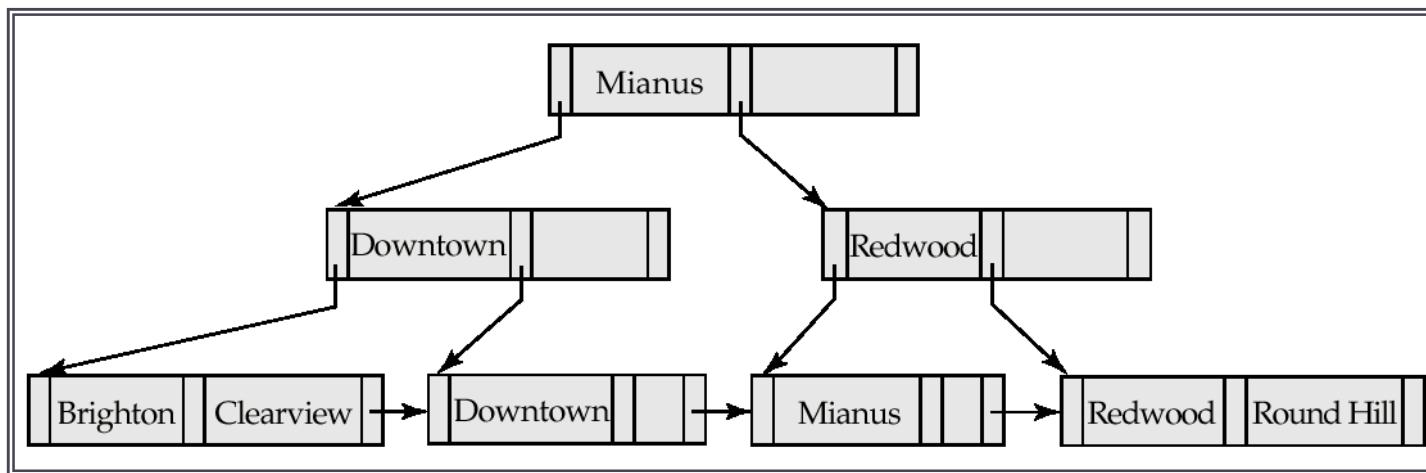
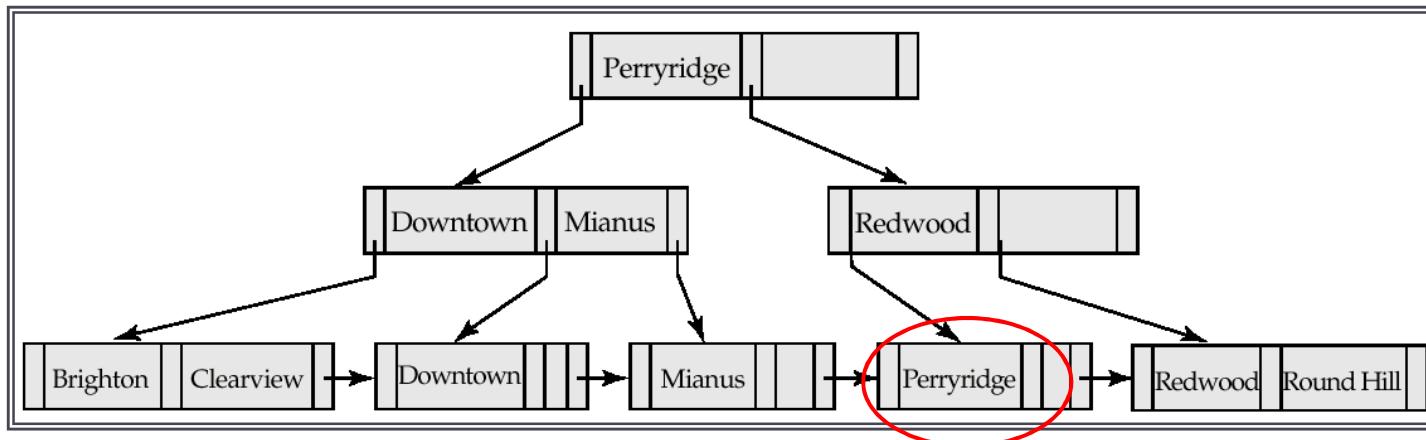
Actualizări în B⁺-arbori

Ștergerea: Exemplu (2)



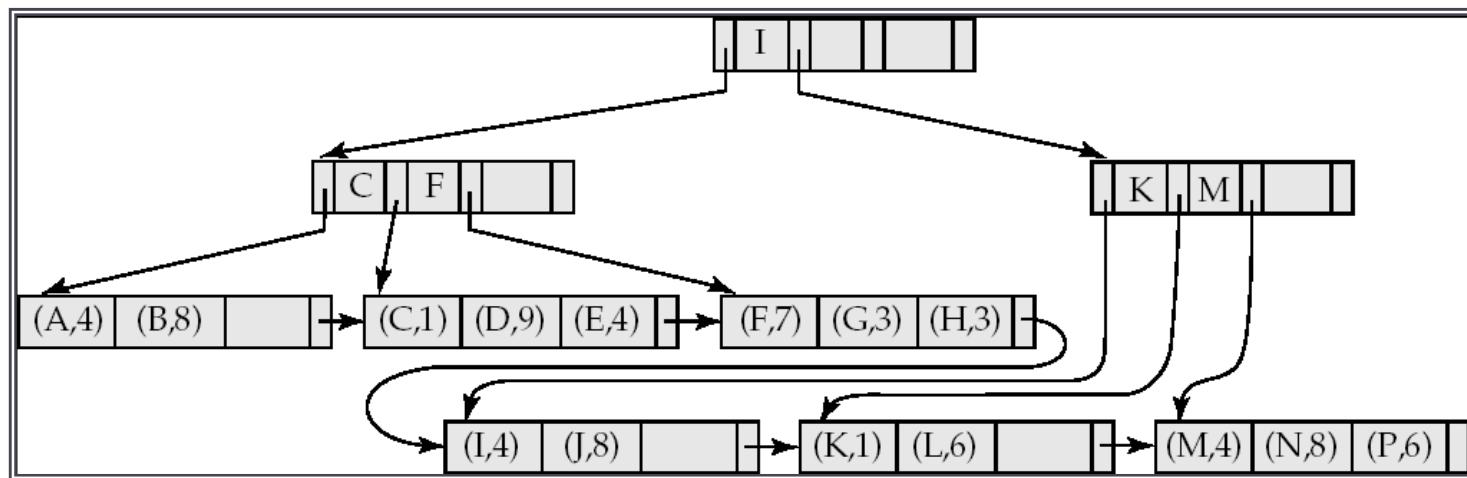
Actualizări în B⁺-arbori

Ștergerea: Exemplu (3)



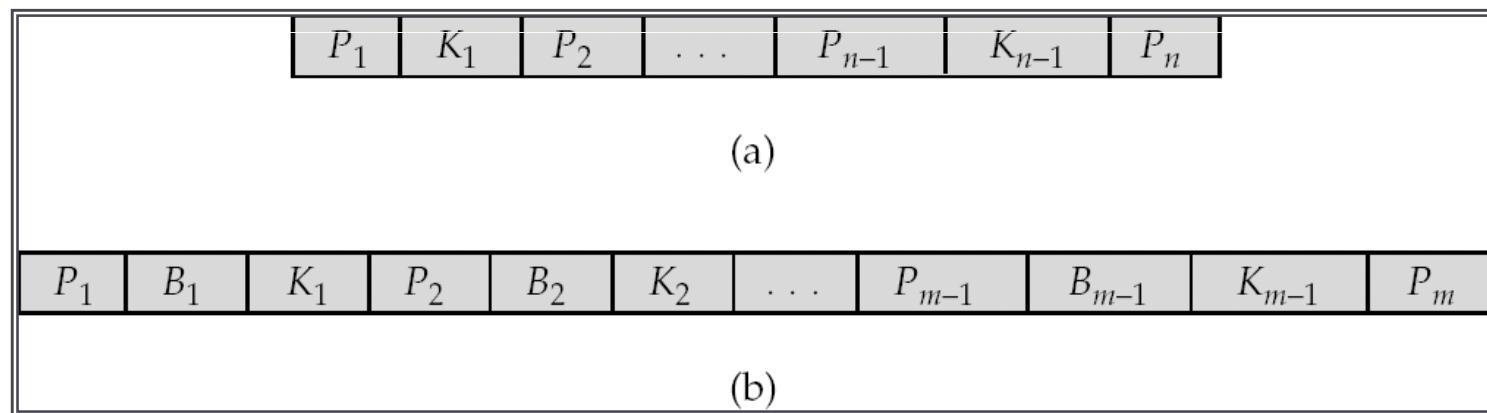
Organizarea fișierelor B⁺-arbore

- ▶ B⁺-arborii pot fi utilizați direct pentru organizarea fișierului și nu doar pentru indexare
 - ▶ Nodurile frunză stochează înregistrări și nu pointeri
 - ▶ Pentru a îmbunătăți utilizarea spațiului sunt implicați mai mulți vecini în redistribuire pentru a evita divizarea sau unirea (utilizând doi vecini la redistribuire rezultă noduri având cel puțin $\lfloor 2n/3 \rfloor$ intrări)



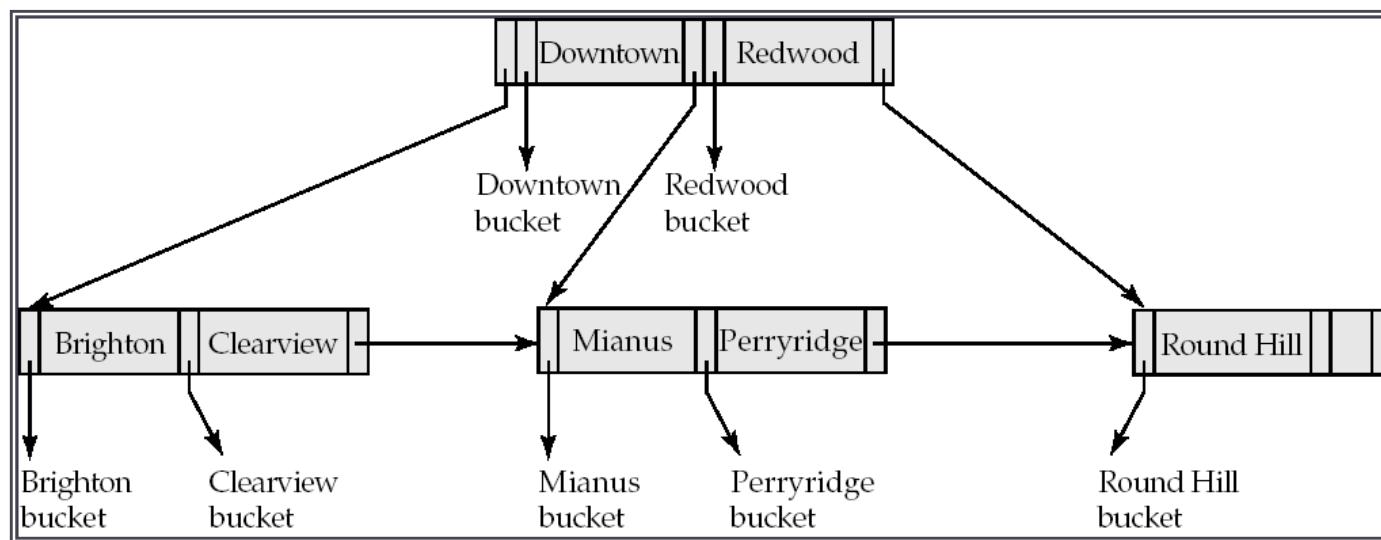
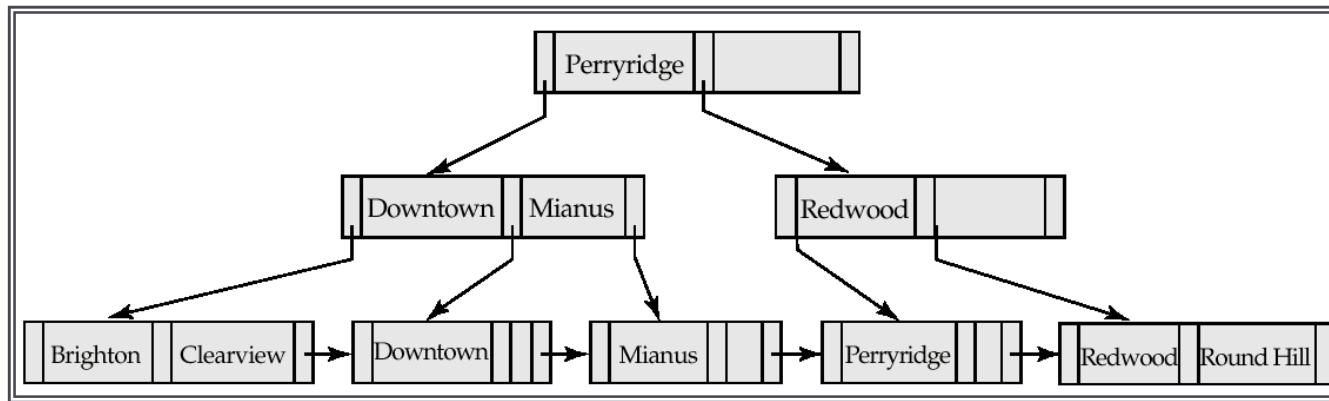
Indecsi B-arbore

- ▶ Asemănători B⁺-arborilor însă permit o singură apariție a valorilor cheilor de căutare
- ▶ Cheile de căutare în nodurile care nu sunt frunză nu mai apar nicăieri în arbore ceea ce necesită introducerea unui pointer adițional



- ▶ Pointerii B_i sunt pointeri către înregistrări sau bucketuri

Indecsi B-arboare Exemplu



Indecsi B-arbore

Observatii

▶ Avantaje

- ▶ Pot utiliza mai puține noduri decât B⁺-arborele corespunzător
- ▶ E posibil a se localiza valoarea căutată înainte de a ajunge la frunze

▶ Dezavantaje

- ▶ Nodurile care nu sunt frunze sunt mai mari ceea ce necesită reducerea numărului de valori stocate; înălțimea va fi mai mare
- ▶ Inserările și ștergerile sunt mai complicate
- ▶ Implementarea e mai dificilă
- ▶ Nu e posibil a fi scanat un tabel doar cu ajutorul frunzelor
- ▶ Avantajele nu cântăresc mai mult decât dezavantajele, B⁺-arborii fiind preferați de către SGBD-uri

Acces multi-cheie

- ▶ Pot fi utilizați mai mulți indecsi la o interogare

```
select account_number  
from account  
where branch_name = "Perryridge" and balance = 1000
```

- ▶ Strategii posibile pentru utilizarea indecsilor uni-atribut:
 - ▶ Utilizarea indexului cu cheia de căutare *branch_name*
 - ▶ Utilizarea indexului cu cheia de căutare *balance*
 - ▶ Utilizarea ambilor și efectuarea intersecției
- ▶ Dezavantaje:
 - ▶ Pot exista multe înregistrări ce satisfac numai una dintre condiții

Indecsi multi-cheie

- ▶ Cheile de căutare compuse sunt chei ce conțin mai mult de un atribut
- ▶ Ordinea lexicografică: $(a_1, a_2) < (b_1, b_2)$ dacă
 - ▶ $a_1 < b_1$ sau
 - ▶ $a_1 = b_1$ și $a_2 < b_2$

Ex. *(branch_name, balance)*

Pot fi rezolvate eficient condițiile de mai jos?

- a) **where branch_name = “Perryridge” and balance < 1000**
- b) **where branch_name < “Perryridge” and balance = 1000**

Hashing

- ▶ În organizarea de tip hash a fișierului/tabelului/relației înregistrările sunt grupate în bucketuri care pot fi localizate pe baza valorilor cheii de căutare
- ▶ Funcția hash $h:K \rightarrow B$ este o funcție de la mulțimea valorilor cheii de căutare la mulțimea adreselor tuturor bucketurilor
 - ▶ Localizează înregistrările pentru acces, inserare, ștergere
- ▶ Înregistrări cu valori diferite a cheii de căutare pot fi mapate la același bucket
 - ▶ Căutare secvențială în bucket

Organizarea de tip hash

Exemplu

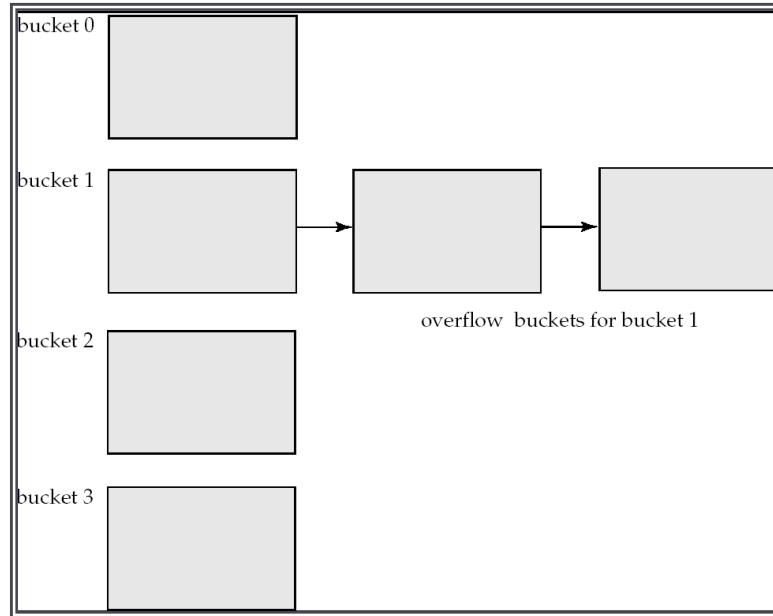
bucket 0		
bucket 1		
bucket 2		
bucket 3		
A-217	Brighton	750
A-305	Round Hill	350
bucket 4		
A-222	Redwood	700
bucket 5		
A-102	Perryridge	400
A-201	Perryridge	900
A-218	Perryridge	700
bucket 6		
bucket 7		
A-215	Mianus	700
bucket 8		
A-101	Downtown	500
A-110	Downtown	600
bucket 9		

Organizarea de tip hash
utilizând *branch_name*
drept cheie:

Reprezentarea binară a
caracterului i din alfabet
este considerat a fi
întregul i. Funcția hash:
suma reprezentărilor
binare modulo 10

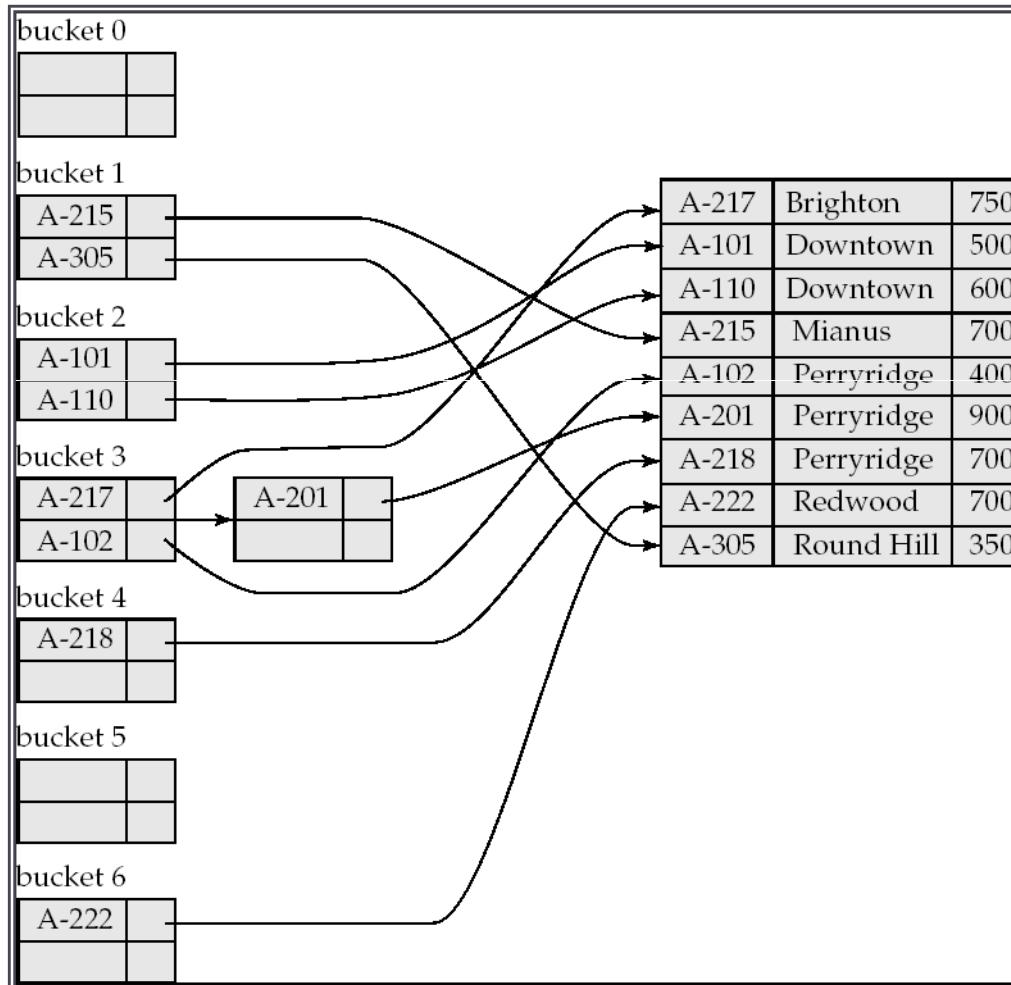
Funcții hash

- ▶ Cerințe
 - ▶ Uniformitate
 - ▶ Caracter aleatoriu
- ▶ Funcțiile hash tipice au la bază calcule pe reprezentarea binară internă a cheii de căutare
- ▶ Pot apărea situații de depășire a bucketului caz în care se utilizează bucketuri de exces



Indecși hash

- ▶ Organizează cheile de căutare cu pointerii asociați într-o structură de tip hash

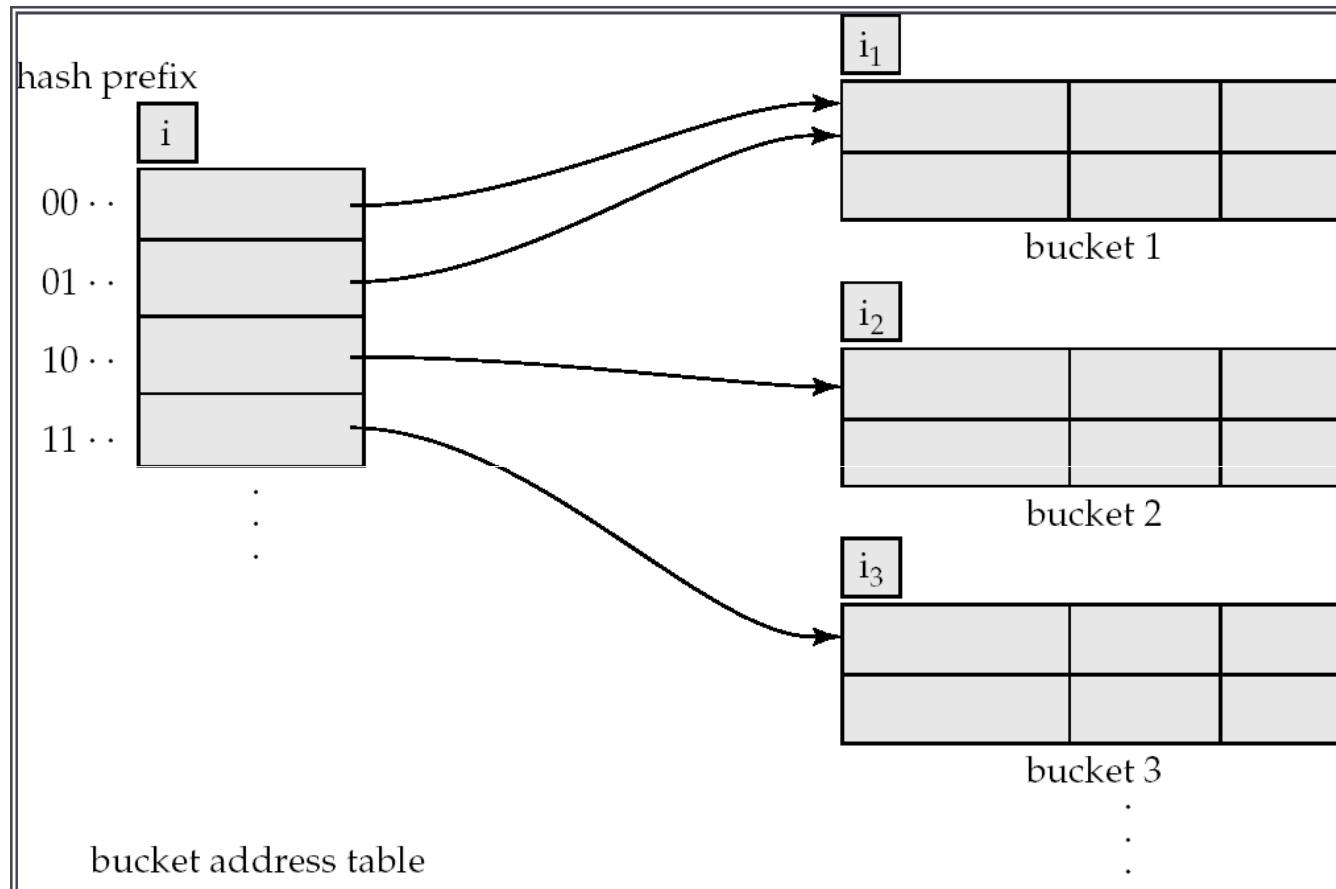


Hash dinamic

- ▶ Funcția h mapează valori a cheii de căutare la un set fix de adrese de bucketuri.
 - ▶ Dacă fișierul crește apar depășiri ale bucketurilor
 - ▶ Dacă fișierul se micșorează spațiu este alocat inutil
- ▶ Solutii:
 - ▶ Reorganizări periodice cu o nouă funcție hash (costisitoare, necesită întreruperea operațiunilor)
 - ▶ Numărul de bucketuri este modificat dinamic
- ▶ Hash extensibil: funcția hash e modificată dinamic
 - ▶ Generează valori într-o mulțime mare, tipic întregi pe 32 biți
 - ▶ La un anumit moment se utilizează doar un prefix al funcției hash (doar primii i biți) a cărui lungime scade sau crește după caz

Hash extensibil

Structura generală



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

Hash extensibil

Utilizare

- ▶ Fiecare bucket j stochează o valoare i_j
 - ▶ toate intrările care indică spre bucketul j vor avea aceeași valoare pe primii i_j biți
- ▶ Pentru a localiza bucketul ce conține cheia de căutare K_j :
 - ▶ Se calculează $h(K_j) = X$
 - ▶ Se utilizează primii i biți ai lui X și se urmează pointerul către bucketul potrivit
- ▶ Pentru a insera o înregistrare cu cheia de căutare K_j :
 - ▶ Se localizează bucketul j ca mai sus
 - ▶ Dacă este spațiu în bucket se inserează înregistrarea
 - ▶ Altfel bucketul este divizat și inserarea este reîncercată

Hash extensibil

Divizare bucket la inserare

Pentru a diviza bucketul j la inserarea unei valori K_j :

- ▶ Dacă $i > i_j$
 1. Se alocă un nou bucket z și $i_z = i_z = (i_j + 1)$
 2. Se actualizează a doua jumătate a tablei de adrese a bucketurilor pentru a indica spre z
 3. Se scot înregistrările din j și sunt reinserate în j sau z
 4. Se recalculează adresa bucketului pentru K_j și se inserează
- ▶ Dacă $i = i_j$
 1. Dacă se atinge o limită a lui i se utilizează bucketuri de exces
 2. Altfel
 1. Se incrementează i și se dublează dimensiunea tablei de adrese
 2. Se înlocuiește fiecare intrare în tabel cu două intrări care indică spre același bucket
 3. Se recalculează adresa bucketului pentru K_j și se inserează (acum $i > i_j$)

Hash extensibil

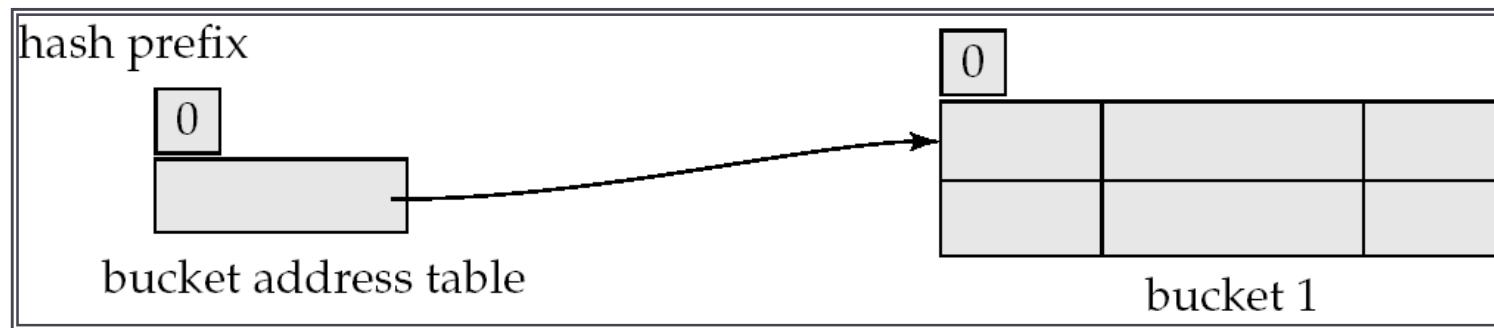
Ștergere

- ▶ Pentru a șterge o înregistrare
 - ▶ Se localizează bucketul și se șterge din el
 - ▶ Dacă bucketul devine gol acesta este șters cu modificările necesare în tabela de adrese
 - ▶ Pot fi contopite bucketuri care au aceeași valoare pentru i_j și același prefix $i_j - l$
 - ▶ Descreșterea dimensiunii tabelei de adrese este posibilă

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

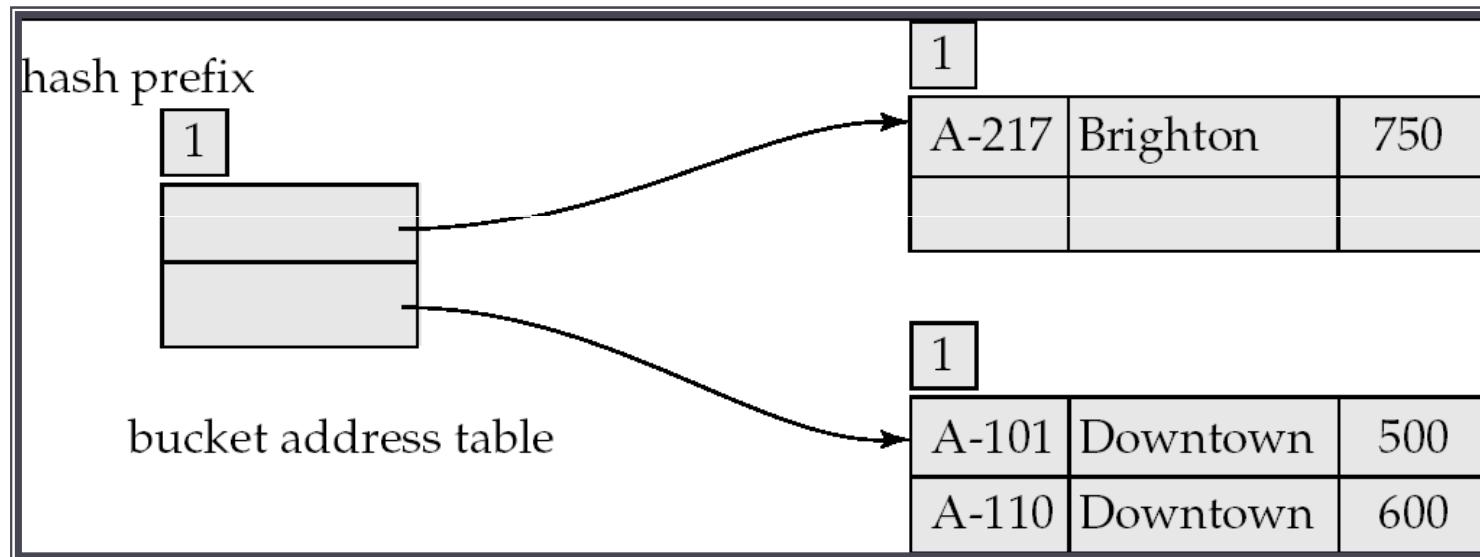


structura hash inițială, dimensiune bucket = 2

Hash extensibil

Exemplu

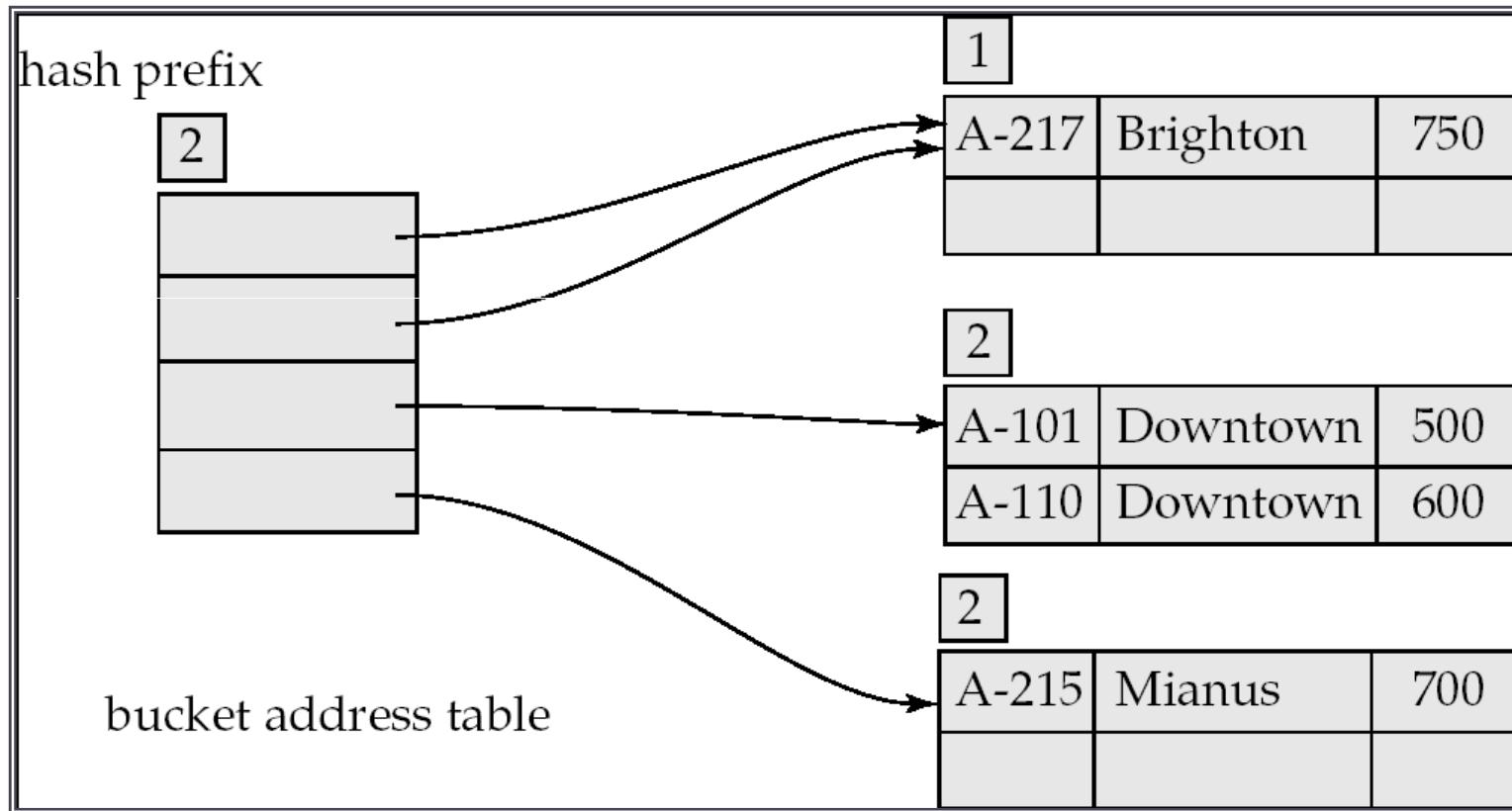
- ▶ Structura după inserarea unei înregistrări Brighton și a două înregistrări Downtown



Hash extensibil

Exemplu

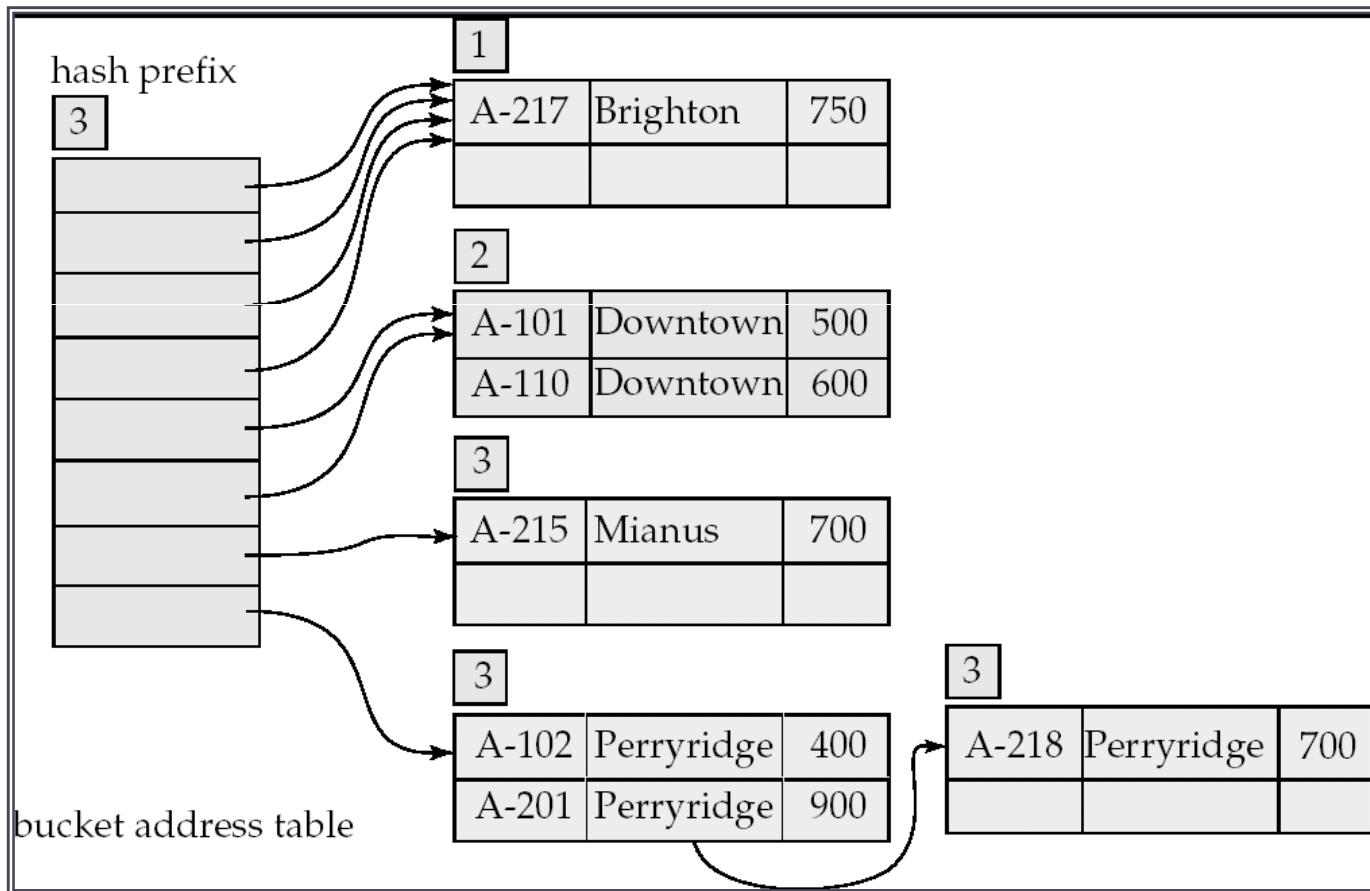
- ▶ După inserarea înregistrării Mianus



Hash extensibil

Exemplu

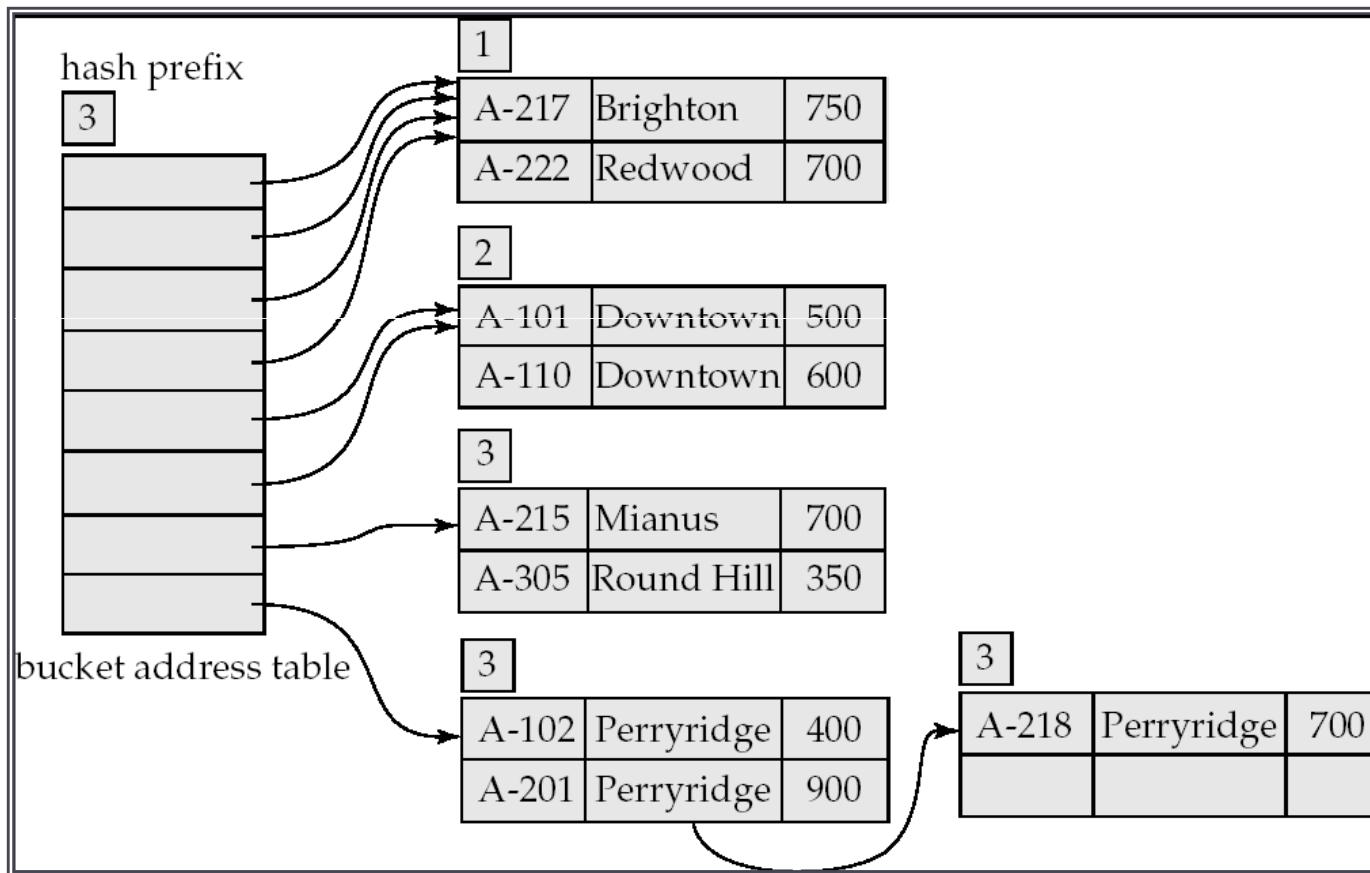
- ▶ După inserarea a trei înregistrări Perryridge



Hash extensibil

Exemplu

- ▶ După inserarea înregistrărilor Redwood și Round Hill



Hashing extensibil

Observații

- ▶ **Beneficii**
 - ▶ Performanța nu se degradează cu creșterea fișierului
 - ▶ Minimizează consumul de memorie
- ▶ **Dezavantaje**
 - ▶ Tabela de adrese a bucketurilor poate deveni foarte mare
 - ▶ Soluție: utilizarea unui B^+ -arbore pentru a localiza înregistrarea dorită în tabela de adrese
 - ▶ Modificarea dimensiunii tabelei de adrese este costisitoare
- ▶ **În funcție de tipul interogării:**
 - ▶ Hashingul e indicat când se specifică o valoare a cheii de căutare
 - ▶ Dacă se lucrează cu intervale de valori e mai rapid indexul ordonat
- ▶ **În practică:**
 - ▶ Postgres suportă indecșii hash
 - ▶ Oracle suportă organizarea statică de tip hash, nu și indecși hash
 - ▶ SQLServer suportă numai B^+ -arbori

Indecsi bitmap

- ▶ Proiectați pentru a trata eficient interogările cu mai multe chei de căutare
- ▶ Aplicabili pentru atribute care iau un set redus de valori distincte
- ▶ Uplele relației sunt considerate a fi numerotate
- ▶ Structura:
 - ▶ Un sir de biți pentru fiecare valoare a atributului
 - ▶ Sirul are lungimea numărului de înregistrări
 - ▶ Valoarea 1 semnifică egalitate cu valoarea căreia îi este asociat bitmapul

record number	name	gender	address	income_level	Bitmaps for gender	Bitmaps for income_level
0	John	m	Perryridge	L1	m 1 0 0 1 0	L1 1 0 1 0 0
1	Diana	f	Brooklyn	L2	f 0 1 1 0 1	L2 0 1 0 0 0
2	Mary	f	Jonestown	L1		L3 0 0 0 0 1
3	Peter	m	Brooklyn	L4		L4 0 0 0 1 0
4	Kathy	f	Perryridge	L3		L5 0 0 0 0 0

Indecsi bitmap

Observatii

- ▶ Interogările sunt rezolvate utilizând operatori pe biți:
 - ▶ Intersecția – AND
 - ▶ Reuniunea – OR
 - ▶ Complementarierea – NOT
- ▶ **where gender ='m' and income_level ='L'**
 $(10010 \text{ AND } 10100) = 10000$
 - ▶ Nu e necesar accesul fișierului
 - ▶ Utili când interogarea necesită numărare
- ▶ Implementare eficientă:
 - ▶ La ștergere se preferă utilizarea unui bitmap de existență
 - ▶ Bitmapurile sunt împachetate în cuvinte (tipul word) de 32 sau 64 biți (operatorul and pe un cuvânt – o singură instrucțiune CPU)
 - ▶ Bitmapuri pot fi utilizate pe nivelul frunză în B^+ -arbori pentru valori ale cheii de căutare ce corespund unui număr mare de înregistrări

Definirea indecșilor în standardul SQL

- ▶ Creare:

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

E.g.: **create index b-index on branch(branch_name)**

- ▶ Ștergere:

drop index <index-name>

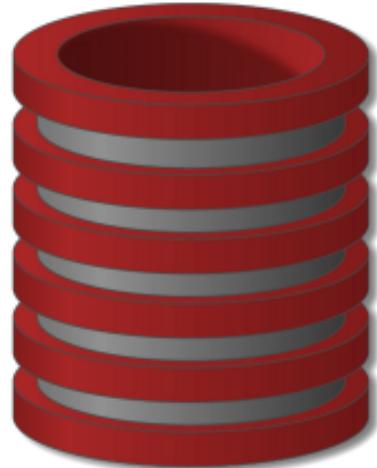
- ▶ Majoritatea SGBD-urilor permit specificarea tipului de index

Indexarea în Oracle

- ▶ Oracle suportă B⁺-arbori implicit la crearea indexului cu comanda SQL
- ▶ Un nou atribut nenul row-id este adăugat tuturor indecșilor pentru a garanta că toate valorile cheii de căutare sunt unice
- ▶ Indecșii sunt suportați pe:
 - ▶ Atribute și liste de atribute
 - ▶ Rezultatul unei funcții peste atribute
- ▶ Indecși bitmap sunt suportați cu declararea
`create bitmap index <index-name> on <relation-name> (<attribute-list>)`
- ▶ Indecși hash nu sunt suportați dar există suport pentru organizarea hash statică

Bibliografie

- ▶ Capitolul 11 în *Avi Silberschatz Henry F. Korth S. Sudarshan. “Database System Concepts”*. McGraw-Hill Science/Engineering/Math; 6 edition (January 27, 2010)



BAZE DE DATE

Procesarea interogărilor

@FII (2011-2012)

prezentat de Mihaela Elena Breabă

Tematică curs

- ▶ **Proiectarea bazelor de date relaționale**
 - ▶ Normalizare și denormalizare
 - ▶ Modelul entitate-asociere, diagrame UML
 - ▶ Constrângeri și declanșatoare
 - ▶ View-uri
 - ▶ Indecși
- ▶ **Procesarea interogărilor**
- ▶ **Managementul tranzacțiilor**
- ▶ **OLAP, Baze de date distribuite, NoSQL, Data Mining**

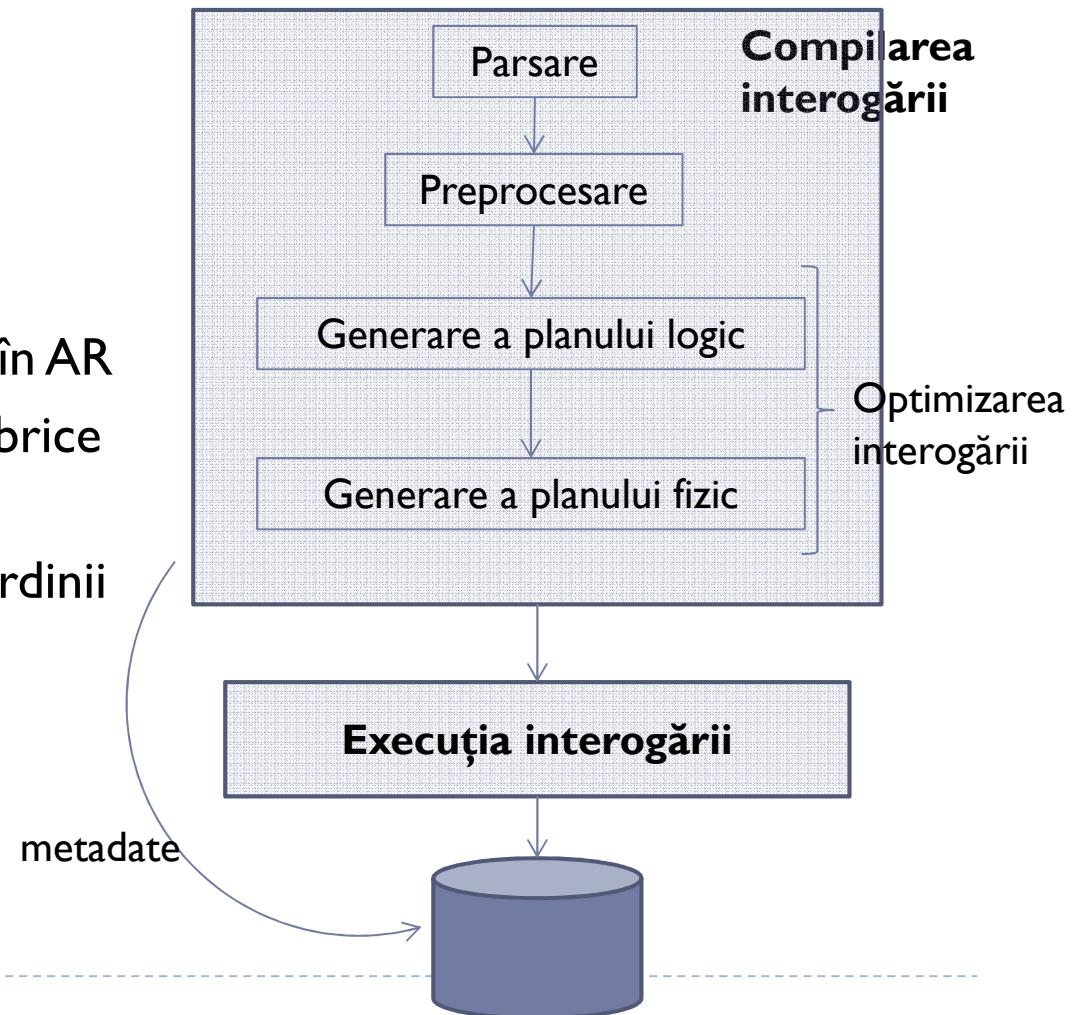
Cuprins

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

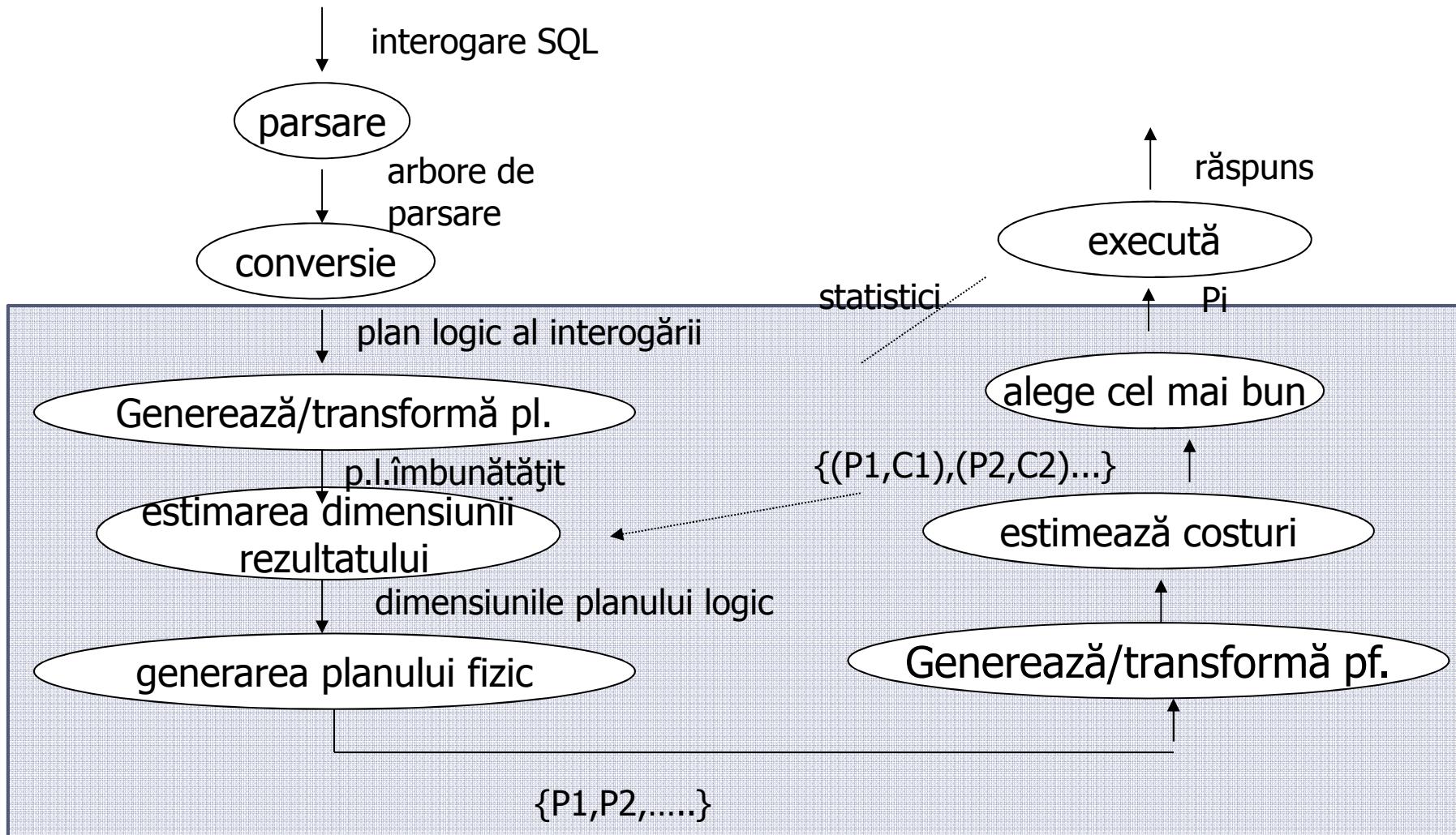
Etapele procesării interogărilor

▶ Compilarea interogării

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

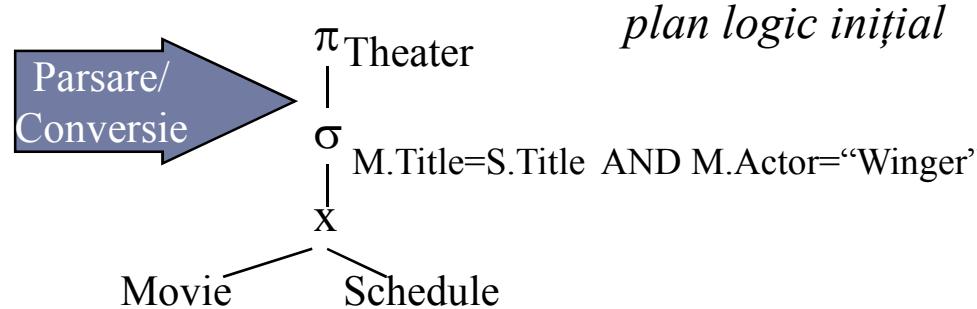


Optimizarea interogărilor

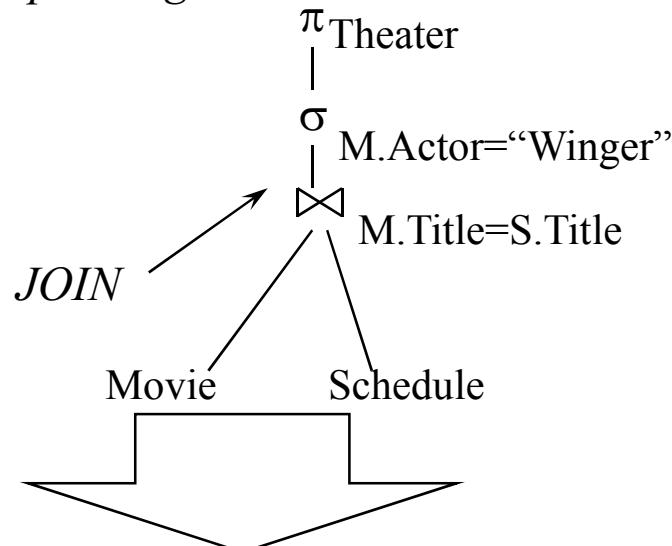


Drumul unei interogări

```
SELECT Theater  
FROM Movie M, Schedule S  
WHERE  
    M.Title = S.Title  
    AND M.Actor="Winger"
```

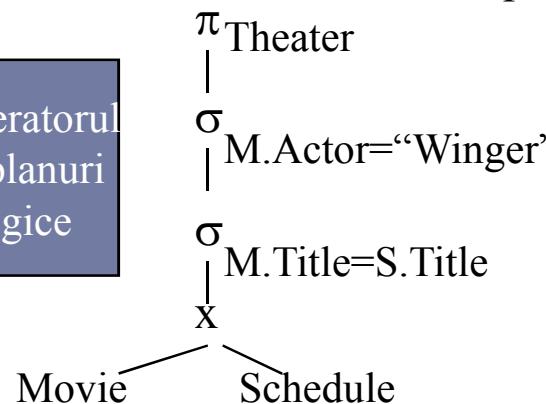


alt plan logic

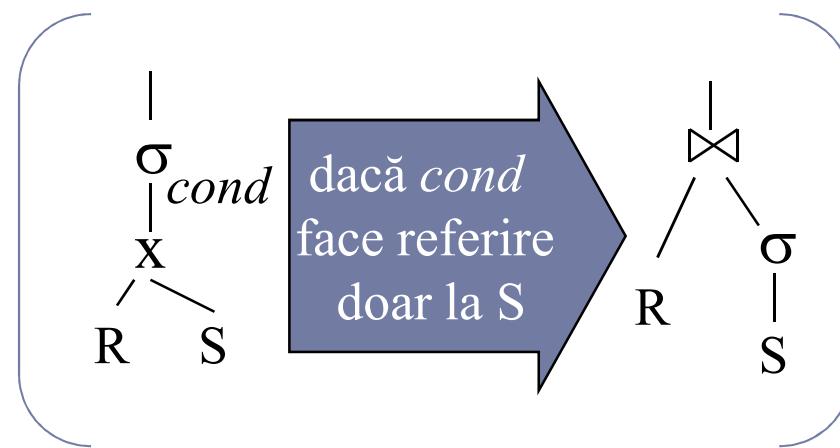
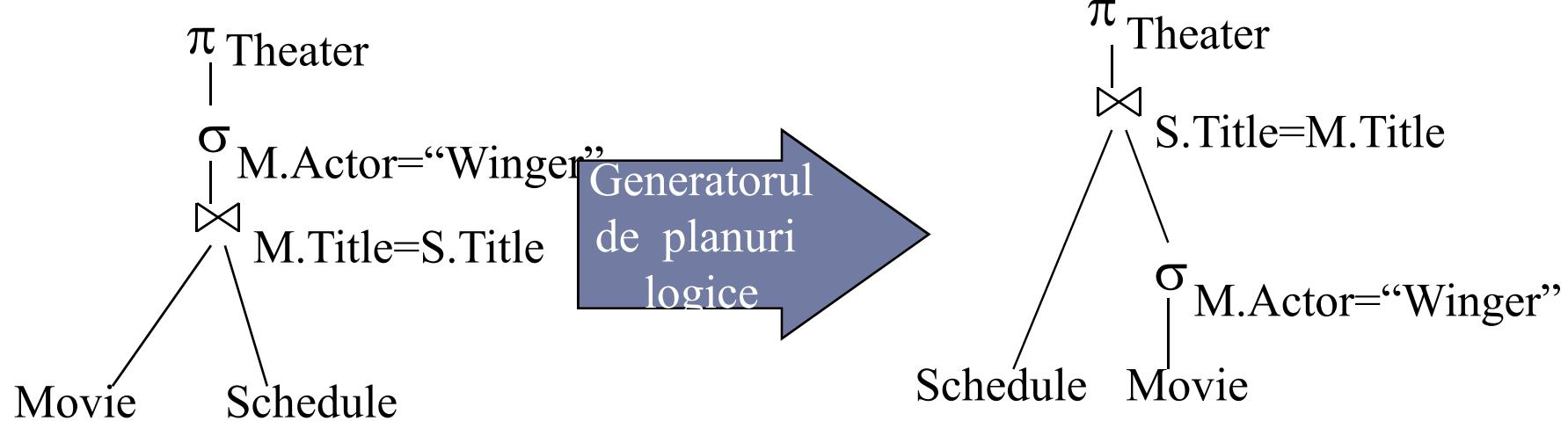


Generatorul de planuri logice
aplică rescrieri algebrice

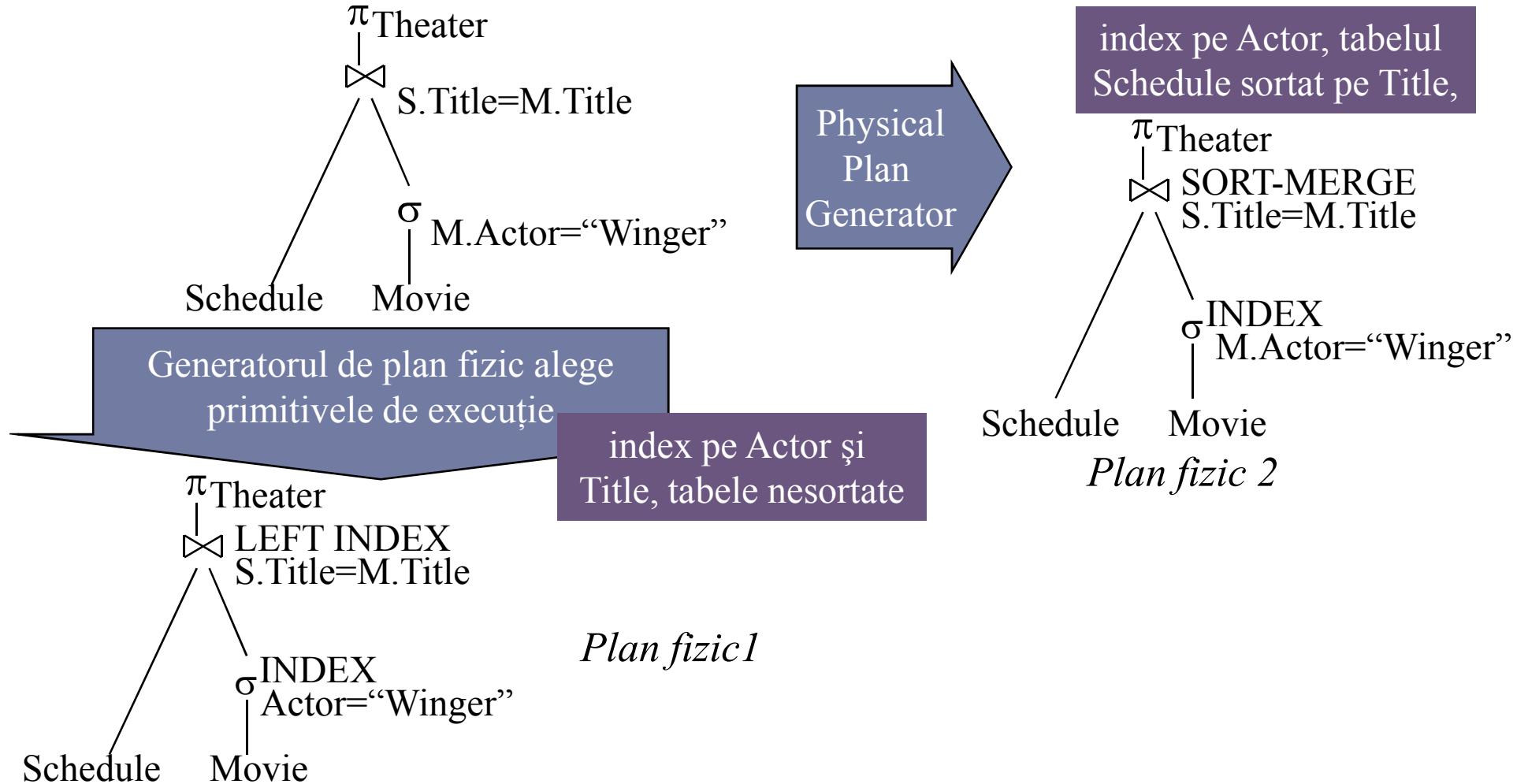
alt plan logic



Drumul unei interogări



Drumul unei interogări



Analiza sintactică

- ▶ Gramatică independentă de context

`<query> ::= <SFW> | (<query>)`

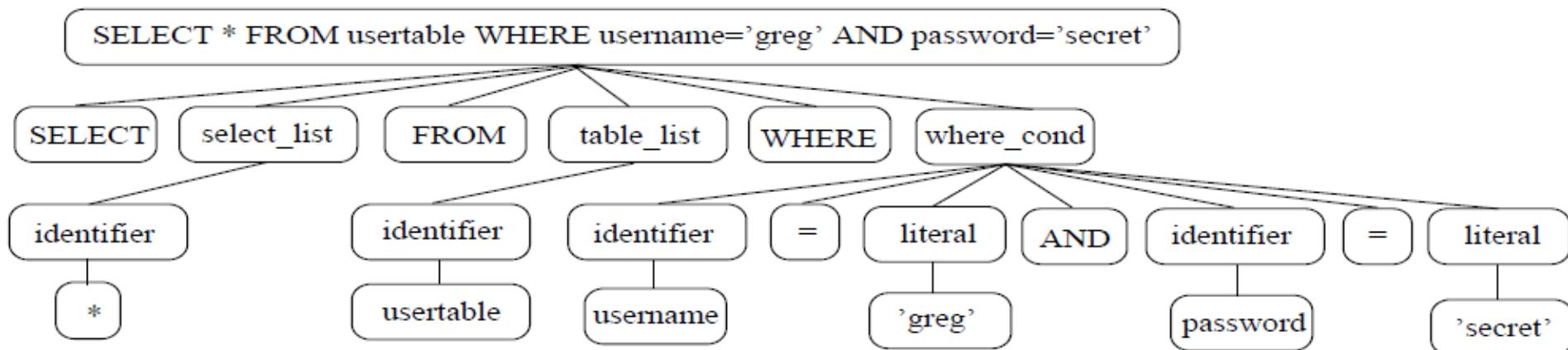
`<SFW> ::= SELECT <select_list> FROM <table_list> WHERE <where_cond>`

`<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>

Analiza semantică

Preprocesare

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

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

Operatori în algebra relațională (revizitat)

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



Operatorul de selecție

- ▶ Realația 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
α	1
α	1
β	1
β	2

A	C
α	1
β	1
β	2

Operatorul reuniune

▶ Relațiile r și s

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

▶ $r \cup s$:

A	B
α	1
α	2
β	1
β	3

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

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
γ	10	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

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 attributele redenumite în A_1, A_2, \dots, A_n .

Componerea 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

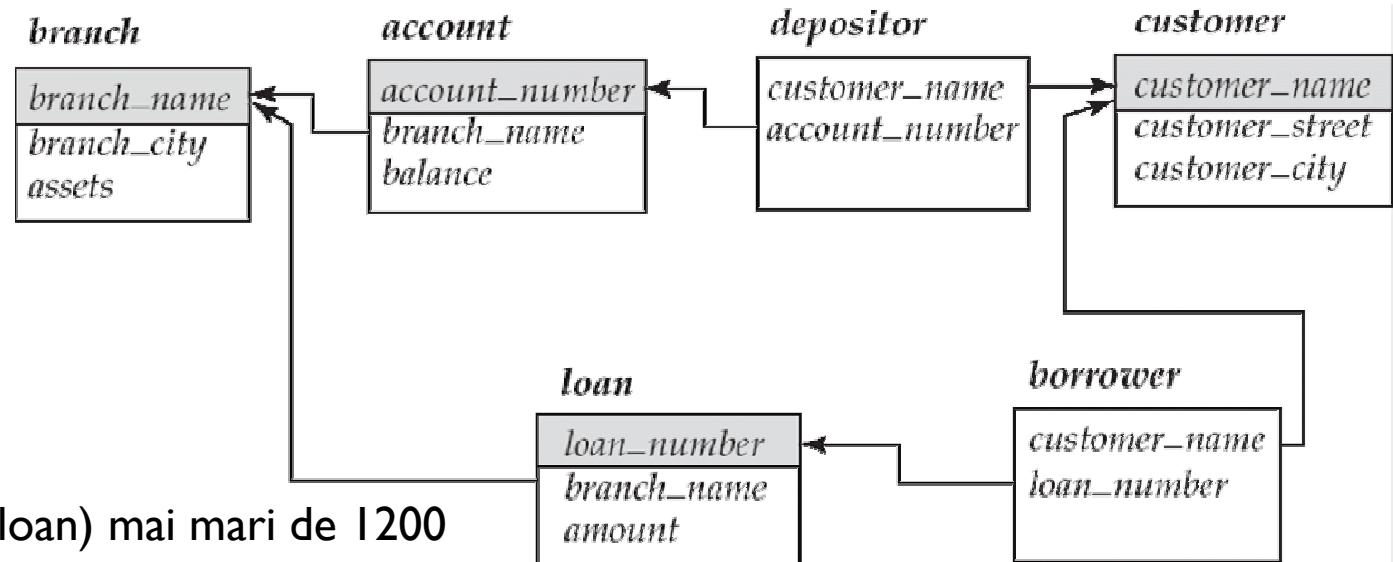
2. $\sigma_{A=C}(r \times s)$

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

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

Exprimarea interogărilor în algebra relatională



- ▶ 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 cont sau ambele la bancă
 $\Pi_{customer_name} (borrower) \cup \Pi_{customer_name} (depositor)$

Interogări

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

$$\begin{aligned} & \prod_{customer_name} (\sigma_{branch_name="Perryridge"} \\ & (\sigma_{borrower.loan_number = loan.loan_number}(borrower \times loan))) \end{aligned}$$

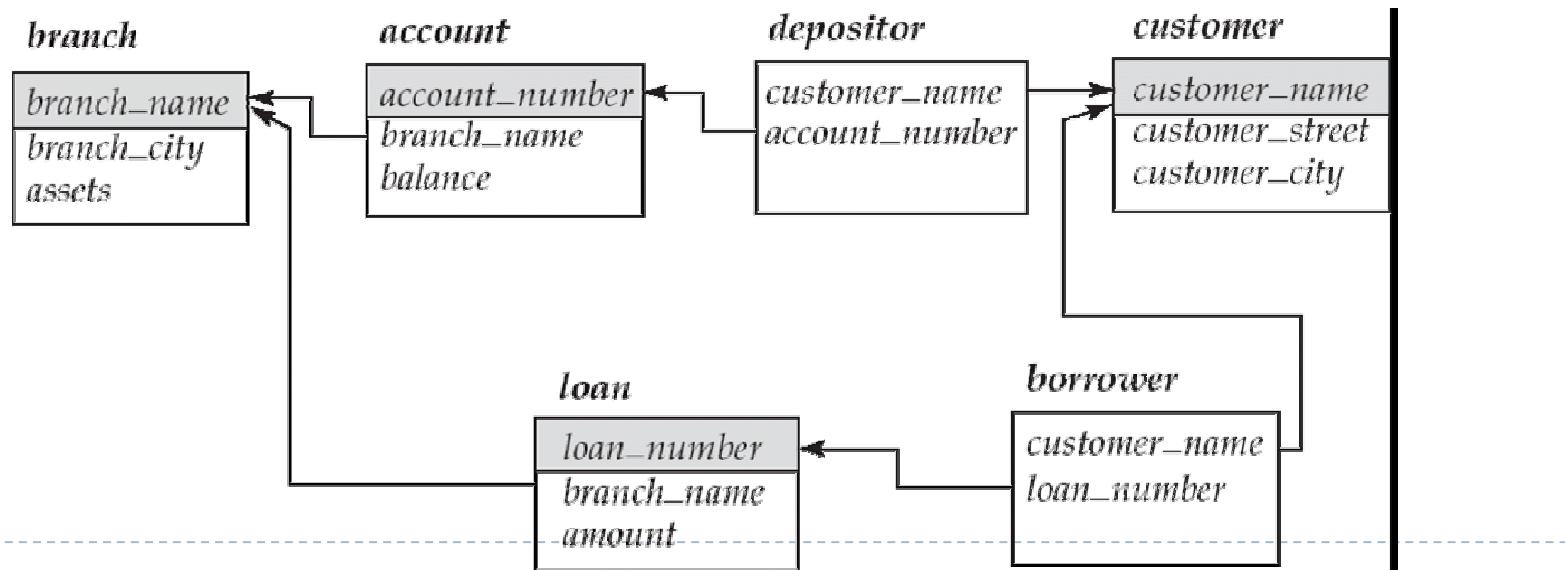
- ▶ Numele tuturor clientilor care au un împrumut la filiala Perryridge dar nu au un cont la nici o filială a băncii

$$\begin{aligned} & \prod_{customer_name} (\sigma_{branch_name = "Perryridge"} \\ & (\sigma_{borrower.loan_number = loan.loan_number}(borrower \times loan)) - \\ & \prod_{customer_name}(depositor) \end{aligned}$$

Interogări

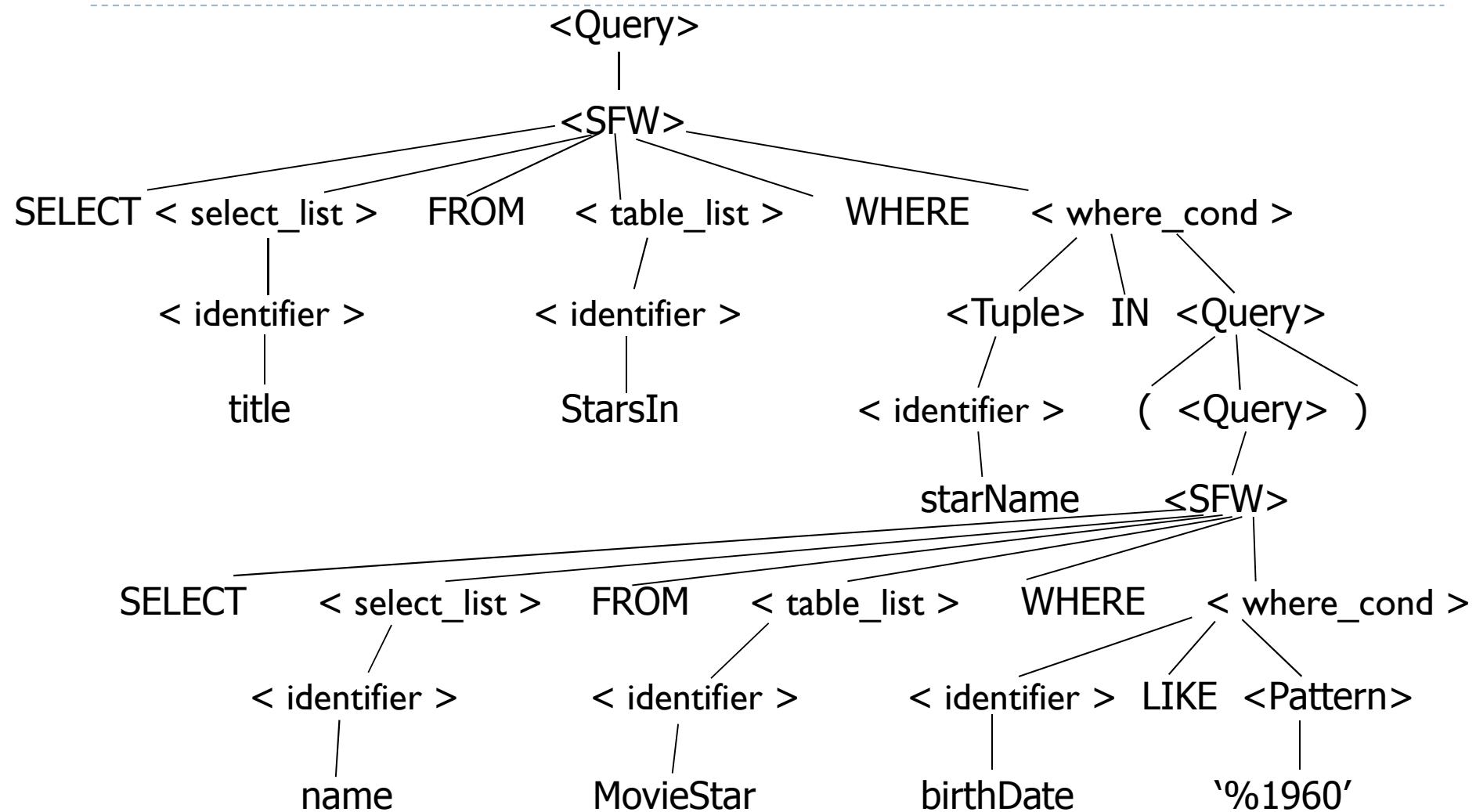
- ▶ Numele tuturor clientilor care au un împrumut la filiala Perryridge

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

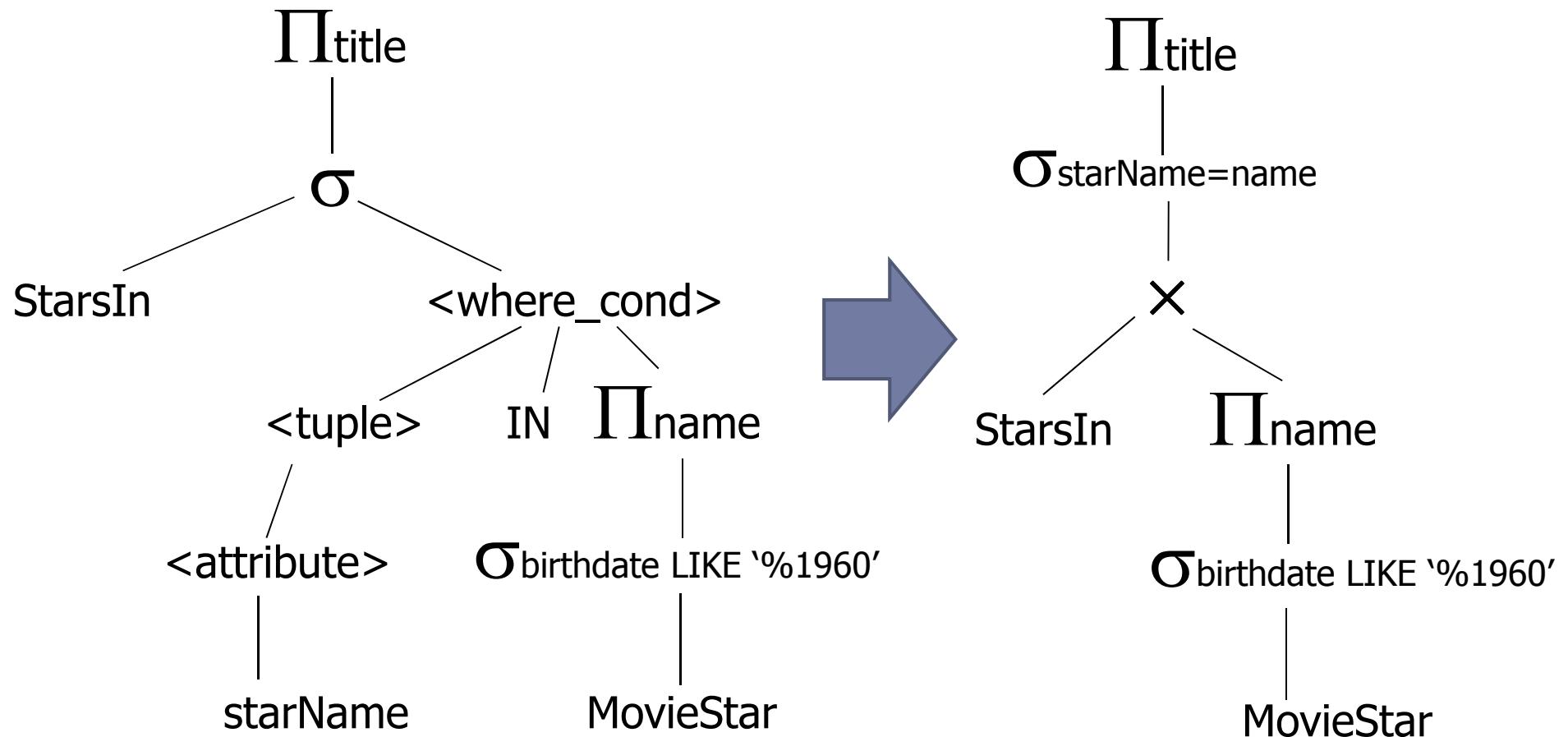


Conversia unei interogări la compilare

Arbore de parsare



Conversia unei interogări la compilare Arbore în algebra relațională



Operatori adiționali

- ▶ Intersecția pe multimi
 - ▶ Joinul natural
 - ▶ Agregarea
 - ▶ Joinul extern
 - ▶ Teta-joinul
-
- ▶ Toți cu excepția agregării pot fi exprimați utilizând operatori de bază

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	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ϵ

s

- ▶ $r \bowtie s$

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

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

Exemplu

- ▶ Cea mai mare balanță din tabela account

account

<i>account_number</i>
<i>branch_name</i>
<i>balance</i>

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

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

Funcții de agregare și operatori

- ▶ Funcții de agregare:

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

- ▶ Operatorul de agregare în algebra relațională

$$G_1, G_2, \dots, G_n \mathcal{G}_{F_1(A_1), F_2(A_2), \dots, F_n(A_n)}(E)$$

- ▶ 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

Exemplu

- ▶ relația r

A	B	C
α	α	7
α	β	7
β	β	3
β	β	10

- ▶ $g_{\text{sum}(c)}(r)$

sum(c)
27

- ▶ Care operații de agregare nu pot fi exprimate pe baza celorlalți operatori relationali?

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

- ▶ $loan \text{ } \overline{\bowtie} \text{ } borrower$ (*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

Join extern

➤ Join extern dreapta

loan \bowtie^{\square} *borrower*

<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>	<i>customer_name</i>
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	<i>null</i>	<i>null</i>	Hayes

➤ Join extern plin

loan \bowtie^{\square} *borrower*

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

Exemple interogări

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

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

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

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

- ▶ Clientii care au conturi de la măcar cele două filiale Downtown și Uptown

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

Echivalență expresiilor

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

Reguli de echivalență

- I. 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. $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$

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

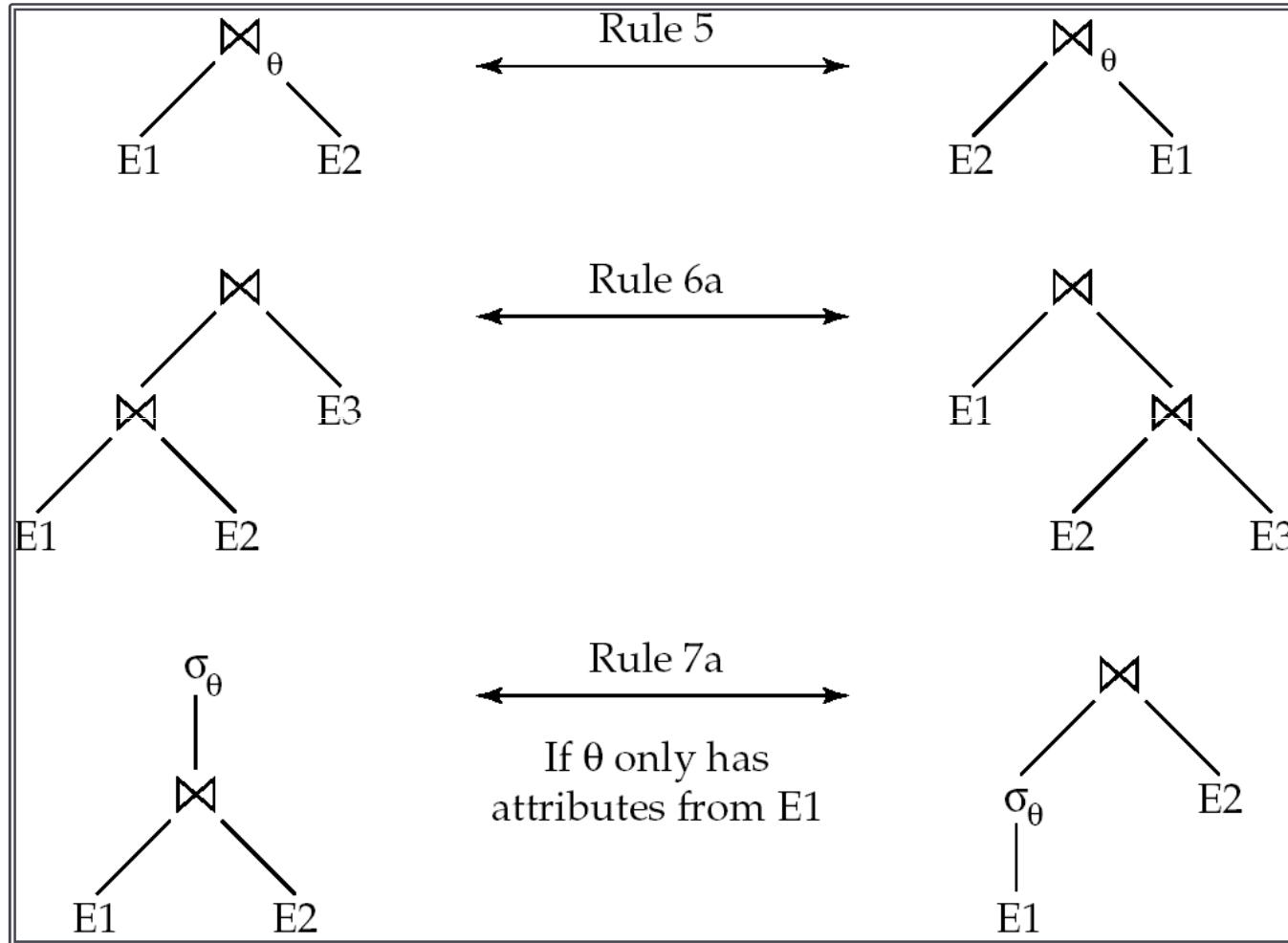
$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

- 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 attribute doar din E_2 și E_3

Reguli de echivalență



Reguli de echivalență

7. Distribuția selecției asupra operatorului de teta-join

a) când θ_0 implică attribute 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 .

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

Reguli de echivalență

8. Distribuția proiecției asupra teta-joinului

a) dacă θ implică numai atrbute 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 atrbute din E_1 și respectiv E_2

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

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

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

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

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

Optimizări

Împingerea selecțiilor

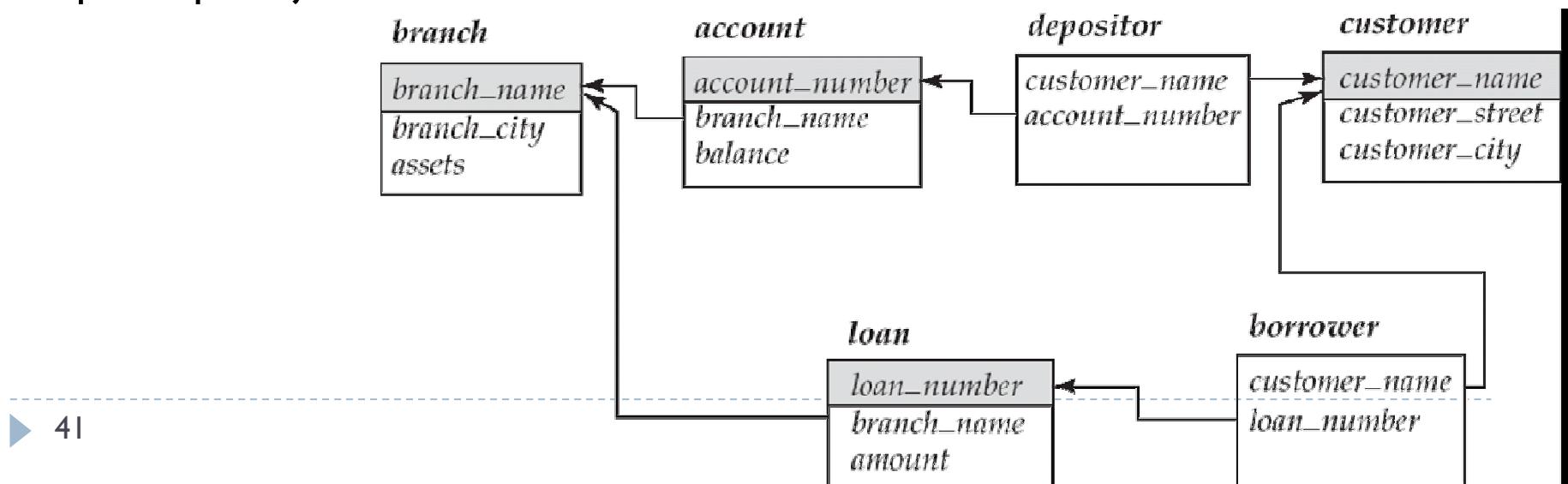
- ▶ Numele clienților care au un cont la o filială din Brooklyn

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

- ▶ 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



Optimizări

Împingerea selecțiilor

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

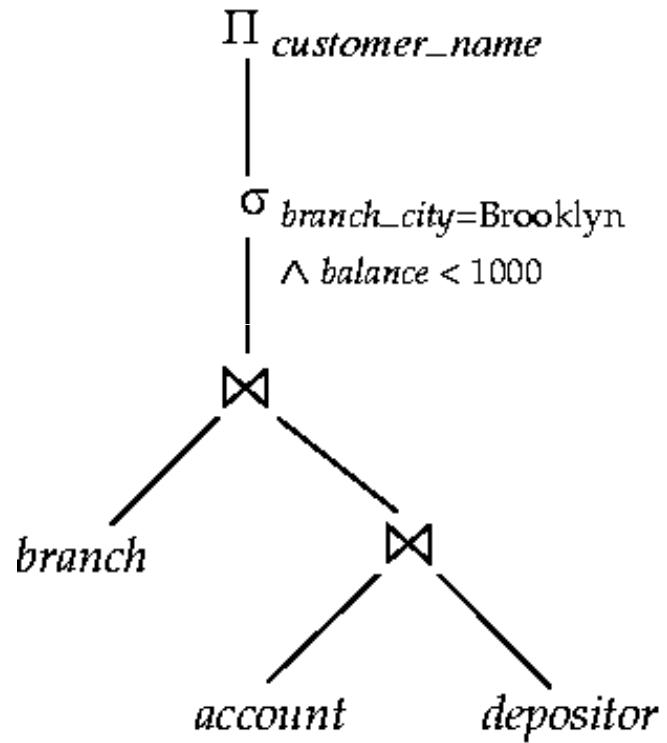
$$\Pi_{customer_name}(\sigma_{branch_city = "Brooklyn" \wedge balance > 1000} \\ (branch \bowtie (account \bowtie depositor)))$$

- ▶ Regula 6a (asociativitatea la join)

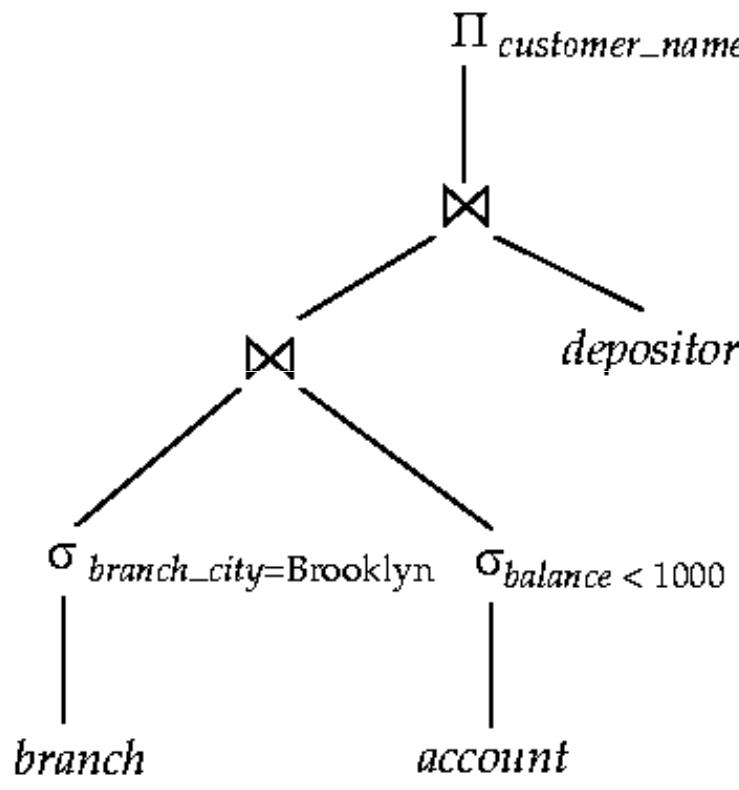
$$\Pi_{customer_name}((\sigma_{branch_city = "Brooklyn" \wedge balance > 1000} \\ (branch \bowtie account)) \bowtie depositor)$$

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

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



(a) Initial expression tree



(b) Tree after multiple transformations

Optimizări

Împingerea proiecțiilor

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

- ▶ Eliminarea atributelor care nu sunt necesare din rezultatele intermediare

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

- ▶ Realizarea devreme a proiecției reduce dimensiunea relațiilor din join

Optimizări

Ordonarea joinurilor

- ▶ Pentru orice relații r_1, r_2 , and 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

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

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.
 - ▶ $n = 7 \rightarrow 665280$, $n = 10 \rightarrow 176$ billion!
- ▶ Pentru a reduce numărul de ordonări supuse evaluării se utilizează programarea dinamică

Estimarea costurilor

- ▶ n_r : numărul de uple în relația r .
- ▶ b_r : numărul de blocuri conținând uple din r .
- ▶ l_r : dimensiunea unui uplu din r .
- ▶ f_r : factorul de bloc al lui r — nr. de uple din r ce intră într-un bloc
- ▶ $V(A, r)$: numărul de valori distincte care apar în r pentru atributul A ; e echivalent cu dimensiunea proiecției $\prod_A(r)$ (pe seturi).
- ▶ Dacă uplele lui r sunt stocate împreună într-un fișier, atunci:

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

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 $n_r / 2$.

Estimarea dimensiunii selecțiilor complexe

- ▶ Selectivitatea unei condiții θ_i este probabilitatea ca un uplu în relația r să satisfacă θ_i
 - ▶ dacă numărul de uple 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
 - ▶ Negația
- $$\sigma_{\theta_1 \vee \theta_2 \vee \dots \vee \theta_n}(r): \quad n_r * \left(1 - \left(1 - \frac{s_1}{n_r} \right) * \left(1 - \frac{s_2}{n_r} \right) * \dots * \left(1 - \frac{s_n}{n_r} \right) \right)$$

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

Estimarea dimensiunii joinului

- ▶ pentru produsul cartezian $r \times s$: $n_r.n_s$ uple, fiecare uplu 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: $\frac{n_r * n_s}{V(A,s)}$ sau $\frac{n_r * n_s}{V(A,r)}$
 - ▶ minimul este considerat de acuratețe mai mare
 - ▶ dacă sunt disponibile histograme se calculează formulele anterioare pe fiecare celulă pentru cele două relații

Estimarea dimensiunii pentru alte operații

- ▶ Proiecția $\Pi_A(r) : V(A, r)$
- ▶ Agregarea: ${}_A\mathcal{G}_F(r) : V(A, r)$
- ▶ Operații pe multimi
 - ▶ $r \cup s : n_r + n_s$
 - ▶ $r \cap s : \min(n_r, n_s)$
 - ▶ $r - s : n_r$
- ▶ 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

Optimizarea planului fizic

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:
$$b * t_T + S * t_S$$

Algoritmi pentru selectie

- ▶ Căutare liniară
 - ▶ 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 uplu ce satisface condiția: $\lceil \log_2(b_r) \rceil * (t_T + t_S)$; dacă există mai multe uple 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 uple returnate: $(h_i + n) * (t_T + t_S)$
 - ▶ index primar, comparație: $h_i * (t_T + t_S) + t_S + t_T * b$

Algoritmi pentru selecții complexe

- ▶ **Conjuncție:** $\sigma_{\theta_1 \wedge \theta_2 \wedge \dots \wedge \theta_n}(r)$
 - ▶ utilizarea unui index pentru θ_1 , și verificarea celorlalte condiții, pe măsură ce upile sunt aduse în memorie
 - ▶ utilizarea unui index multi-cheie
 - ▶ intersecția identificatorilor (pointerilor la înregistrări) returnați de indecesii asociati condițiilor urmată de citirea înregistrărilor
- ▶ **Disjuncție:** $\sigma_{\theta_1 \vee \theta_2 \vee \dots \vee \theta_n}(r)$
 - ▶ reuniunea identificatorilor

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

Join cu bucle imbricate

- ▶ Pentru teta-join: $r \bowtie_{\theta} s$
for each uplu t_r **in** r **do begin**
 for each uplu t_s **in** s **do begin**
 if (t_r, t_s) satisface θ
 adaugă $t_r \bullet t_s$ la rezultat
 end
end
- ▶ relația interioară – s
- ▶ relația exterioară – r
- ▶ Costul estimat: $(n_r * b_s + b_r) * t_T + (n_r + b_r) * t_s$

Join indexat cu bucle imbricate

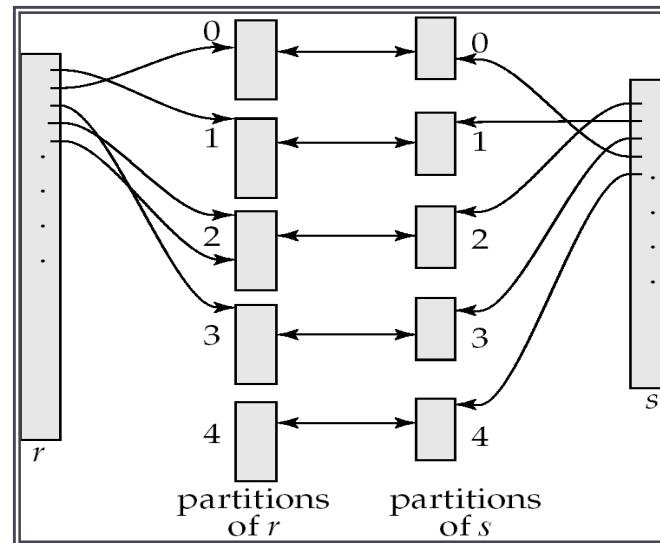
- ▶ Căutările în index pot înlocui scanarea fișierelor dacă:
 - ▶ e un echi-join sau join natural
 - ▶ există un index pe atributul de join al relației interioare
- ▶ pentru fiecare uplu t_r în relația exterioară r se utilizează indexul pentru localizarea uplelor din s care satisfac condiția de join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

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 și joinuri naturale
- ▶ **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
 - ▶ se sortează rezultatul după adresele uplelor relației nesortate
 - ▶ se scanează relația nesortată în ordinea adreselor fizice și se realizează fuziunea cu rezultatul anterior pentru a înlocui adresele cu uplele asociate

Join hash

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



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 reuniunii joinurilor individuale (aplicabil numai versiunii set a reuniunii)
 $(r \bowtie_{\theta_1} s) \cup (r \bowtie_{\theta_2} s) \cup \dots \cup (r \bowtie_{\theta_n} s)$

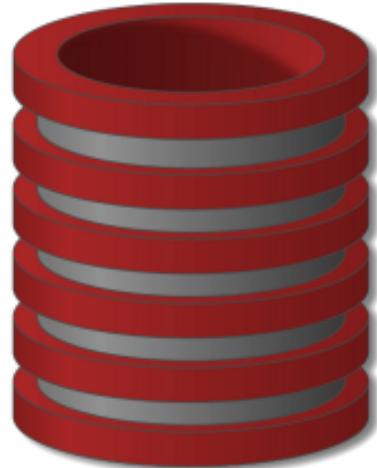
Evaluare expresiilor

▶ 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: uple 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 uple
 - ▶ varianta la producător: operatorul scrie în buffer uple iar părintele scoate din buffer (la umplerea bufferului există timpi de așteptare)

Bibliografie

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



BAZE DE DATE

Tranzactii

@FII (2011-2012)

prezentat de Mihaela Elena Breabăñ

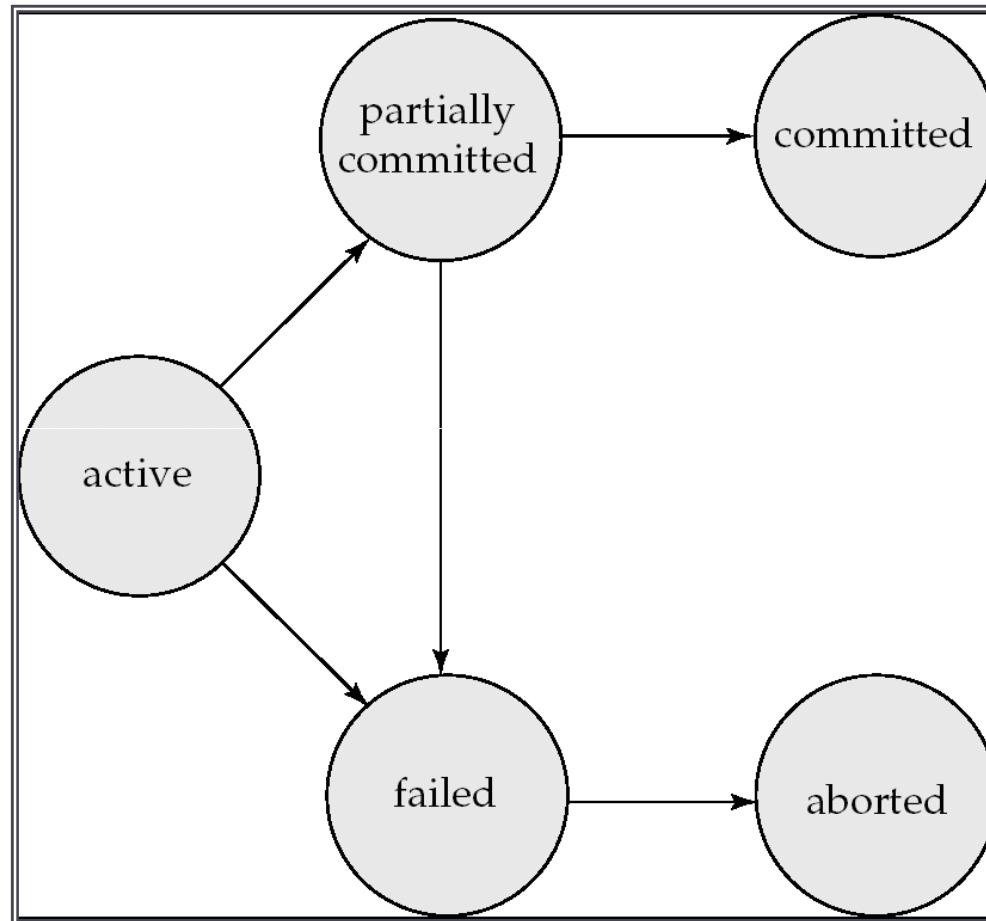
Cuprins

- ▶ Proprietățile tranzacțiilor
- ▶ Stările unei tranzacții
- ▶ Execuții concurente
- ▶ Definirea tranzacțiilor în SQL

Concept și proprietăți

- ▶ Tranzacția e o secvență de operații executate asupra bazei de date, cu următoarele proprietăți:
 - ▶ Atomicitate – operațiile vor fi executate în totalitate, altfel niciuna nu va fi executată
 - ▶ Consistență – o tranzacție aduce o bază de date dintr-o stare consistentă într-o alta stare tot consistentă
 - ▶ Izolare – execuția unei tranzacții nu este influențată de execuția alteia
 - ▶ Durabilitate – efectul execuției tranzacției trebuie să fie de durată (modificarea este una persistentă)
- ▶ Ex: transferul unei sume între 2 conturi A și B
 1. **read(A)**
 2. $A := A - 50$
 3. **write(A)**
 4. **read(B)**
 5. $B := B + 50$
 6. **write(B)**

Stările unei tranzacții



Execuție serială

T_1	T_2
<code>read(A)</code> $A := A - 50$ <code>write (A)</code> <code>read(B)</code> $B := B + 50$ <code>write(B)</code>	<code>read(A)</code> $temp := A * 0.1$ $A := A - temp$ <code>write(A)</code> <code>read(B)</code> $B := B + temp$ <code>write(B)</code>

T_1	T_2
	<code>read(A)</code> $temp := A * 0.1$ $A := A - temp$ <code>write(A)</code> <code>read(B)</code> $B := B + temp$ <code>write(B)</code>

Execuție concurentă

T ₁	T ₂
read(A) $A := A - 50$ write(A)	read(A) $temp := A * 0.1$ $A := A - temp$ write(A)
read(B) $B := B + 50$ write(B)	read(B) $B := B + temp$ write(B)

T ₁	T ₂
read(A) $A := A - 50$ write(A)	read(A) $temp := A * 0.1$ $A := A - temp$ write(A) read(B)

Definirea unei tranzacții în SQL

- ▶ O tranzacție începe implicit
- ▶ Se încheie la una din comenzi:
 - ▶ COMMIT (modificările devin permanente și începe o nouă tranzacție)
 - ▶ ROLLBACK (tranzacția este anulată și se revine la starea anterioară începerii ei)
- ▶ În Oracle:
 - ▶ Comanda COMMIT este emisă implicit înainte și după o comanda DDL și la deconectarea de la baza de date
 - ▶ O ieșire forțată emite comanda ROLLBACK

Bibliografie

- ▶ Capitolele 15 și 16 în *Avi Silberschatz Henry F. Korth S. Sudarshan. “Database System Concepts”.* McGraw-Hill Science/Engineering/Math; 4th edition