

Algebră Relațională

Extensii ale operatorilor algebrici relaționali

- Generalizarea proiecției
- Funcții de agregare
- Outer Join
- Modificarea bazei de date

Generalizarea proiecției

- Operatorul *proiecție* este extins prin permiterea utilizării funcțiilor aritmetice în lista de definire a proiecției.

$$\pi_{F_1, F_2, \dots, F_n}(R)$$

- R poate fi orice expresie din algebra relațională
- Fiecare dintre F_1, F_2, \dots, F_n sunt expresii aritmetice ce implică attribute din R și constante.

Funcții de agregare

- **Funcția de agregare** returneaza ca rezultat o valoare pe baza unei colecții de valori primite ca input

avg: valoarea medie

min: valoarea minimă

max: valoarea maximă

sum: suma

count: numărul înregistrărilor

- **Operator de agregare în algebra relațională**

$$G_{G_1, G_2, \dots, G_n}^{F_1(A_1), F_2(A_2), \dots, F_n(A_n)}(R)$$

- R poate fi orice expresie din algebra relațională
 - G_1, G_2, \dots, G_n e o listă de attribute pe baza cărora se grupează datele (poate fi goală)
 - Fiecare F_i este o funcție de agregare
 - Fiecare A_i este un nume de atribut

Agregare – exemplu

Relație R :

A	B	C
α	α	7
α	β	7
β	β	3
β	β	10

$G_{\text{sum}(C)}(R)$

sum_C
27

- Rezultatul agregării nu are un nume
 - se pot folosi operatorii de redenumire
 - se poate permite redenumirea ca parte a unui operator de agregare

Agregare & Grupare – exemplu

Relație *Completed*:




PersonName *G* count(PuzzleName) (Completed)

<i>PersonName</i>	<i>PuzzleName</i>
<i>Alex</i>	<i>Soma cube</i>
<i>Carla</i>	<i>Rubik cube 3x3</i>
<i>Bob</i>	<i>Cluebox Shrodinger Cat</i>
<i>Bob</i>	<i>Soma cube</i>
<i>Alex</i>	<i>Windmill cube</i>
<i>Bob</i>	<i>Mirror cube 3x3</i>
<i>Carla</i>	<i>Rubik cube 4x4</i>

<i>PersonName</i>	
<i>Alex</i>	<i>2</i>
<i>Bob</i>	<i>3</i>
<i>Carla</i>	<i>2</i>

Outer Join

■ Extensii ale operatorului join natural care împiedică pierderea informației:

- Left Outer Join 
- Right Outer Join 
- Full Outer Join 

■ Realizează joncțiunea și apoi adaugă la rezultat tuplurile dintr-una din relații (din stânga, dreapta sau ambele părți ale operatorului) care nu sunt conectate cu tupluri din celaltă relație.

■ Utilizează valoarea *null*:

- *null* semnifică faptul că valoarea e necunoscută sau nu există
- Toate comparațiile ce implică *null* sunt (*simply spus*) **false** prin definiție.

Modificarea bazei de date

■ Conținutul bazei de date poate fi modificat folosind următorii operatori:

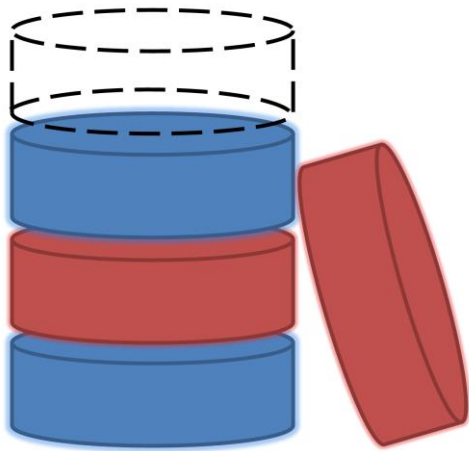
■ Ștergere $R \leftarrow R - E$

■ Inserare $R \leftarrow R \cup E$

■ Modificare $R \leftarrow \pi_{F_1, F_2, \dots, F_n}(R)$

■ Toți acești operatori sunt exprimați prin utilizarea operatorului de atribuire.

Indexarea bazelor de date relaționale



Regăsirea datelor

- Interogare folosind egalități:
 - *“Find student name whose Age = 20”*
- Interogare folosind intervale:
 - *“Find all students with Grade > 8.50”*
- Parcurgerea secvențială a fișierului este costisitoare
- Dacă datele sunt sortate în fișier:
 - Căutare binară pentru găsirea primei înregistrări (*cost ridicat*)
 - Parcurgerea fișierului pentru găsirea celorlalte înregistrări.

Indecși

- Indecșii sunt fișiere/structuri de date speciale utilizate pentru accelerarea execuției interogărilor.
- Un index se crează pe baza unei *chei de căutare*
 - *Cheia de căutare* e un atribut/set de attribute folosit(e) pentru a căuta înregistrările dintr-o tabelă/fișier.
 - *Cheia de căutare* este diferită de cheia primară / cheia candidat / supercheia
- Un index conține o colecție de înregistrări speciale și permite regăsirea eficientă a “*tuturor înregistrărilor tablei indexate pentru care cheia de căutare are valoarea k*”.

Caracteristicile indecșilor

- *Propagarea modificărilor*
 - Adăugarea/ștergerea de înregistrări în tabela indexată implică actualizarea tuturor indecșilor definiți pentru acea tabelă.
 - Orice modificare a valorilor câmpurilor ce aparțin cheii de căutare presupune actualizarea indecșilor bazați pe acea cheie de căutare

Caracteristicile indecșilor

■ *Dimensiunea indecșilor*

- Deoarece utilizarea unui index presupune citirea acestuia în memoria internă → dimensiune indexului trebuie să fie redusă.
- Ce se întâmplă dacă indexul e (totuși) prea mare?
 - Utilizare structură de indexare parțială
 - Se indexează fișierul index
(stratificat sau pe o structură arborescentă)

- Teme importante:

- Care este structura unei înregistrări de index?
(conținutul indexului)
- Cum sunt organizate aceste înregistrări?
(tehnica de indexare)

Variante de structurare a conținutului unui index

- (1) înregistrările originale ce includ cheia de căutare
- (2) $\langle k, rid \rangle$, rid – referință spre înreg. cu valoarea k a cheii de căutare
- (3) $\langle k, \text{listă de } rid\text{-uri} \rangle$

- Alegerea variantei de stocare a intrărilor (înregistrărilor) din index e ortogonală cu tehnica de indexare.
 - Exemple de tehnici de indexare: arbori B+, acces direct
 - Indexul mai conține și informații auxiliare de direcționare a căutărilor spre înregistrările căutate

Variante de structurare a conținutului unui index

■ Varianta (1):

- Indexul și înregistrările originale sunt stocate împreună
- Pentru o colecție de înregistrări se poate crea cel mult un index structurat conform variantei (1) (în caz contrar avem înregistrări duplicate → redundanță → potențiale inconsistențe)
- Dacă înregistrările sunt voluminoase, numărul de blocuri în care se stochează înregistrările este mare → dimensiunea informației auxiliare din index este la rândul său mare.

Variante de structurare a conținutului unui index

■ Variantele (2) și (3):

■ Intrările din index referă înregistrările originale

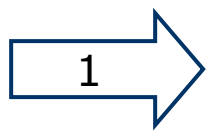
- Intrările din index sunt în general mai mici decât înregistrările (deci sunt variante mai eficiente decât varianta (1), mai ales dacă cheia de căutare este mică).
- Informația auxiliară din index folosită pentru a direcționa căutarea, este la rândul său mai redusă decât în cazul variantei (1).

■ Varianta (3) este mult mai compactă decât varianta (2), dar implică dimensiune variabilă a spațiului de stocare a intrărilor, chiar dacă cheia de căutare este de dimensiune fixă.

Crearea unui index

	<i>Name</i>	<i>Age</i>	<i>Grade</i>
rid_1	John	22	8.50
rid_2	Jack	21	9.00
		...	
rid_n	Peter	22	10.00

cheie de
căutare

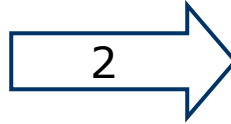


$rid_1 : 22$

$rid_2 : 21$

...

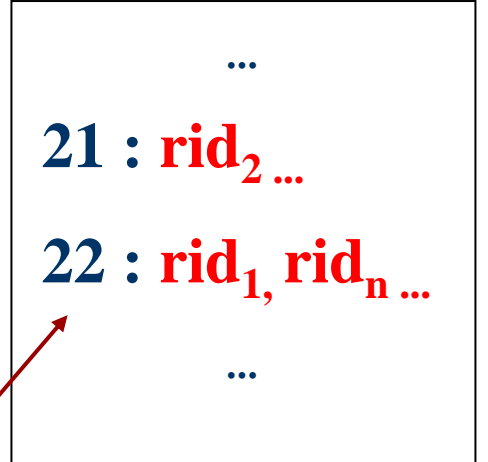
$rid_n : 22$



$22 : rid_1, rid_n \dots$

$21 : rid_2 \dots$

...



intrare

fișier index
(inversat)

Clasificarea indecșilor

- Indecși *primari* vs *secundari*:
 - Un index *primar* se formează pe baza unei chei de căutare ce include cheia primară.
 - Un index e *unic* atunci când cheia de căutare conține o cheie candidat
 - Nu vor fi intrări duplicate
 - În general, indecșii *secundari* conțin duplicate

Clasificarea indecșilor

- Indecși *clustered* (*grupat*) vs. *un-clustered* (*negrupat*):
 - Un index este *clustered* (grupat) dacă ordinea înregistrărilor este aceeași (sau foarte apropiată) cu ordinea intrărilor din index.
 - Varianta (1) implică grupare; de asemenea în practică, gruparea implică utilizarea primei variante de structurare a indecșilor.
 - O tabelă poate fi indexată grupat (*clustered*) pentru cel mult o cheie de căutare.
 - Costul regăsirii datelor prin intermediul unui index e influențat *decisiv* de grupare!

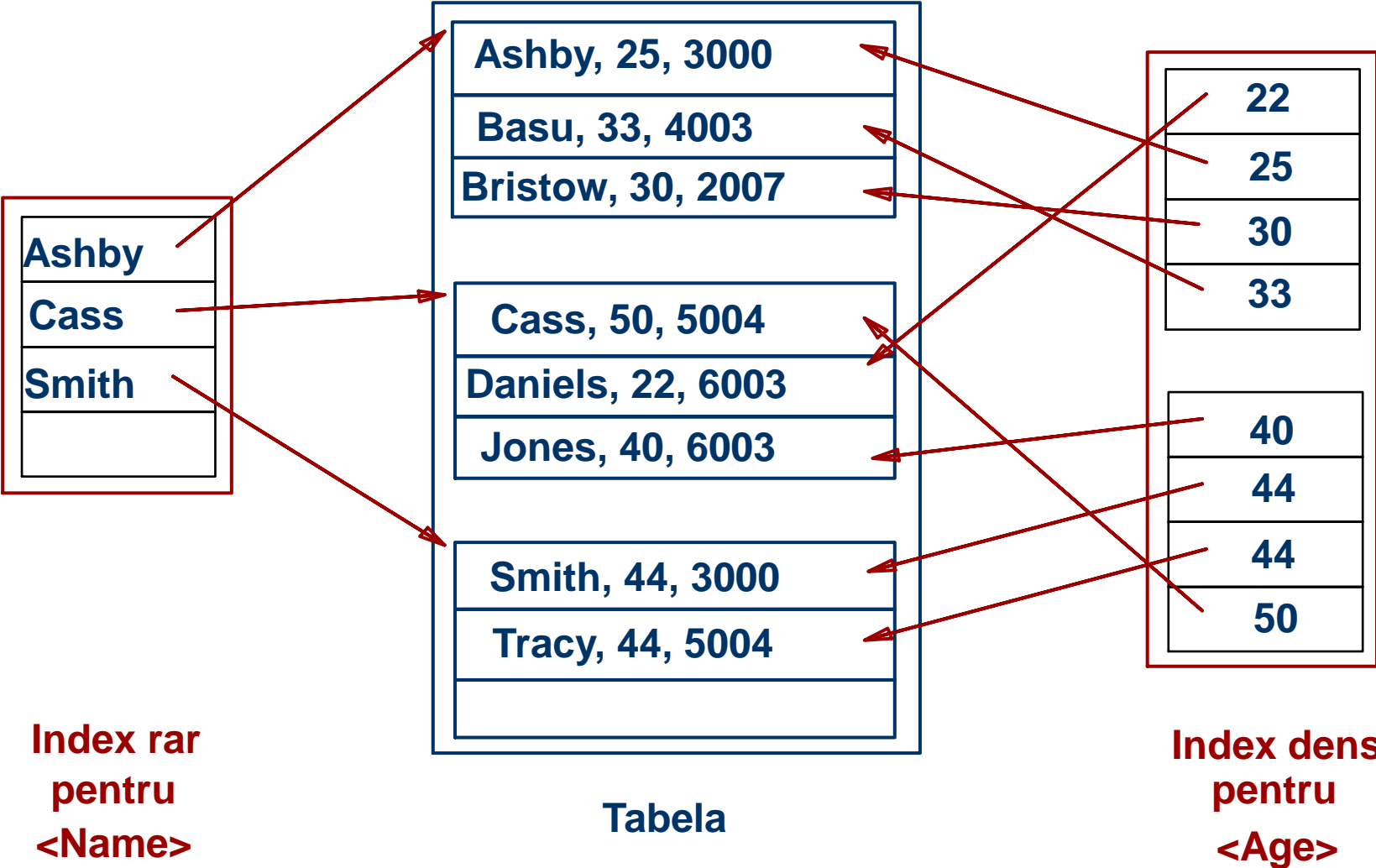
Index Clustered vs. Un-clustered

- Pentru a construi indecși grupați:
 - Se ordonează înregistrările în cadrul fișierului (*heap file*)
 - Se rezervă spațiu în fiecare pagină de memorie, pentru a se absorbi viitoarele inserări
 - dacă spațiul suplimentar e consumat, se vor forma liste înlănțuite de pagini suplimentare
 - E necesară o reorganizare periodică pentru a se asigura o performanță ridicată
- Întreținerea indecșilor grupați e foarte costisitoare

Clasificarea indecșilor

- Indecși *denși* vs. *rari*:
- Un index este *dens* dacă are cel puțin o intrare pentru fiecare valoare a cheii de căutare (ce se regăsește în înregistrări)
 - Mai multe intrări pot avea aceeași valoare pentru cheia de căutare în cazul utilizării variantei (2) de stocare
 - Varianta (1) presupune implicit ca indexul e dens.
- Un index e *rar* dacă memorează o intrare pentru fiecare pagină de înregistrări
 - Toți indecșii rari sunt grupați (*clustered*)!
 - Indecșii rari sunt mai compacti.

Clasificarea indecșilor

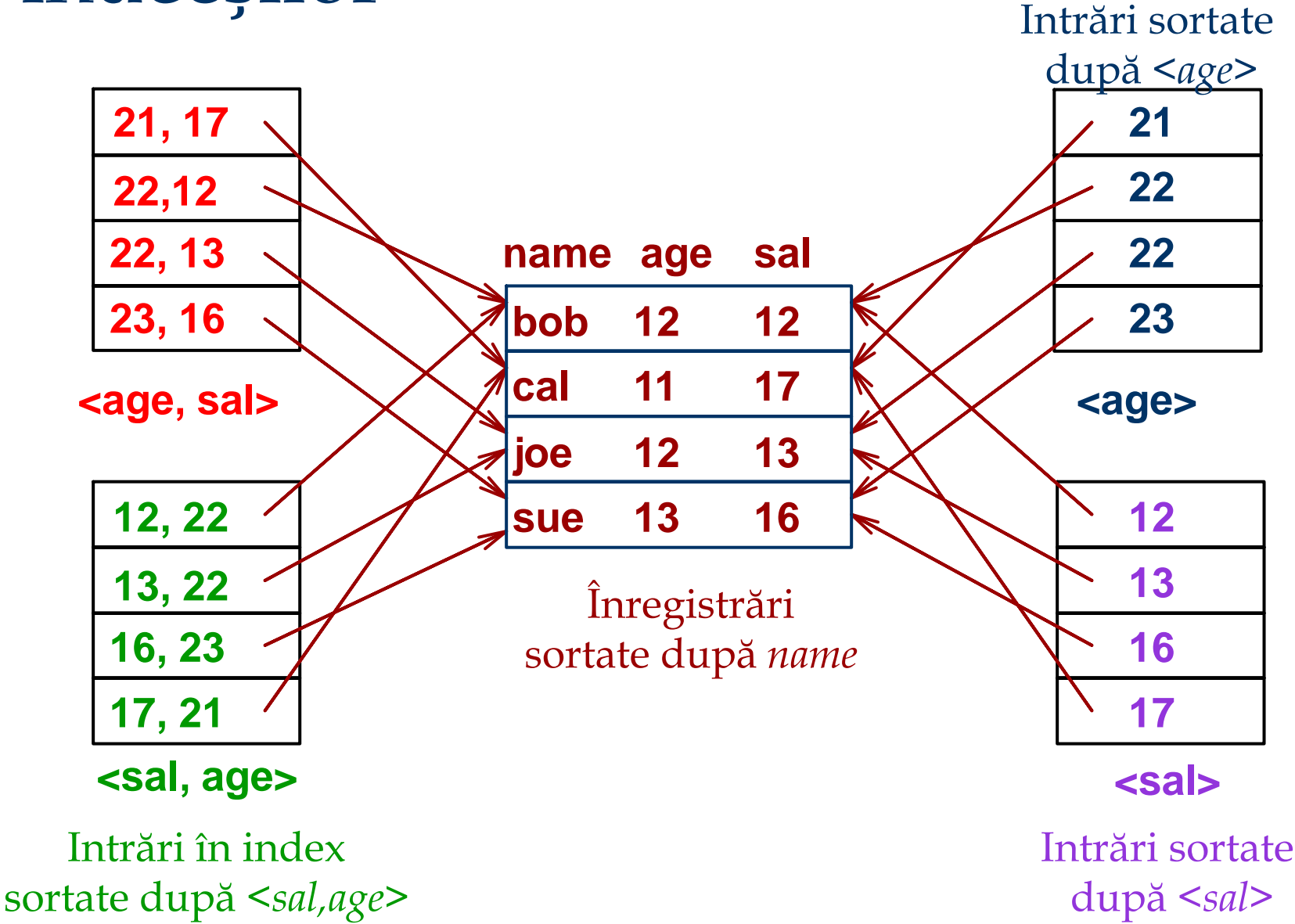


Clasificarea indecșilor

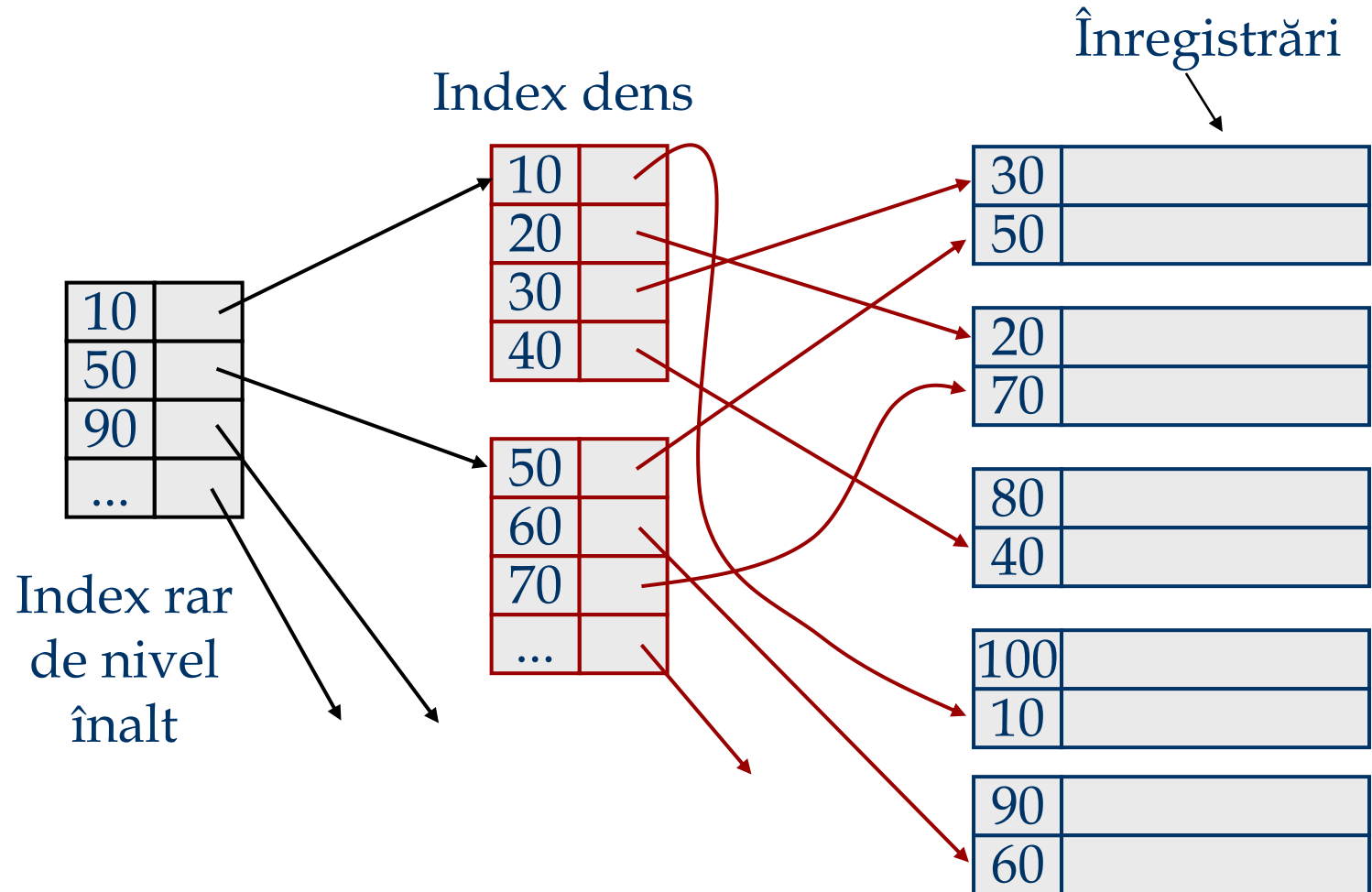
Chei de căutare compuse:

- Utile atunci când căutarea se realizează după o combinație de câmpuri
 - Interogări cu egalitate (valoarea fiecărui câmp e egală cu o valoare constantă):
age=20 și sal =25
 - Interogări cu interval (valoarea unui câmp nu e o constantă):
age=20 și sal > 30
- Intrările din index sunt sortate după cheia de căutare pentru a putea fi utile interogărilor cu interval.
 - Ordine lexicografică, sau
 - Ordine spațială.

Clasificarea indecșilor



Indecși multinivel (indexare multiplă)



Exemplu

- Înregistrările tablei Students sunt stocate într-un fișier sortate după câmpul *Age*. Fiecare pagină poate stoca un număr maxim de 3 înregistrări

<i>ID</i>	<i>Name</i>	<i>Age</i>	<i>Grade</i>	<i>Course</i>
53831	Melanie	11	7.8	Music
53832	George	12	8.0	Music
53666	John	18	9.4	ComputerS
53688	Sam	19	9.2	Music
53650	Sam	19	9.8	ComputerS

- Determinați intrările în fiecare dintre următorii indecși (folosiți $\langle page_id, slot\ no \rangle$ pentru identificarea unui tuplu).

Exemplu

<i>ID</i>	<i>Name</i>	<i>Age</i>	<i>Grade</i>	<i>Course</i>
53831	Melanie	11	7.8	Music
53832	George	12	8.0	Music
53666	John	18	9.4	ComputerS
53688	Sam	19	9.2	Music
53650	Sam	19	9.8	ComputerS

1. *Age* – Dens, varianta (1)

- întreaga tabelă

2. *Age* – Dens, varianta (2)

$(11, \langle 1, 1 \rangle) (12, \langle 1, 2 \rangle) (18, \langle 1, 3 \rangle) (19, \langle 2, 1 \rangle) (19, \langle 2, 2 \rangle)$

3. *Age* – Dens, varianta (3)

$(11, \langle 1, 1 \rangle) (12, \langle 1, 2 \rangle) (18, \langle 1, 3 \rangle) (19, \langle 2, 1 \rangle, \langle 2, 2 \rangle)$

4. *Age* – Rar, varianta (1)

- un astfel de index nu poate fi construit (definiție)

Exemplu

<i>ID</i>	<i>Name</i>	<i>Age</i>	<i>Grade</i>	<i>Course</i>
53831	Melanie	11	7.8	Music
53832	George	12	8.0	Music
53666	John	18	9.4	ComputerS
53688	Sam	19	9.2	Music
53650	Sam	19	9.8	ComputerS

5. *Age* – rar, varianta (2)

(11,<1,1>) (19,<2,1>) - ordinea intrărilor e semnificativă

6. *Age* – rar, varianta (3)

(11,<1,1>) (19,<2,1>,<2,2>) - ordinea intrărilor e semnificativă

7. *Grade* – Dens, varianta (1)

7.8, 8.0, 9.2, 9.4, 9.8

8. *Grade* – Dense, varianta (2)

(7.8,<1,1>)(8.0,<1,2>)(9.2,<2,1>)(9.4,<1,3>)(9.8,<2,2>)

Exemplu

<i>ID</i>	<i>Name</i>	<i>Age</i>	<i>Grade</i>	<i>Course</i>
53831	Melanie	11	7.8	Music
53832	George	12	8.0	Music
53666	John	18	9.4	ComputerS
53688	Sam	19	9.2	Music
53650	Sam	19	9.8	ComputerS

9. *Grade* – Dense, varianta (3)

$(7.8, \langle 1, 1 \rangle)(8.0, \langle 1, 2 \rangle)(9.2, \langle 2, 1 \rangle)(9.4, \langle 1, 3 \rangle)(9.8, \langle 2, 2 \rangle)$

10. *Grade* – Rar, varianta (1)

- un astfel de index nu poate fi construit (definiție)

11. *Grade* – Rar, varianta (2)

- nu e posibil (valorile cheii de căutare nu sunt ordonate)

12. *Grade* – Rar, varianta (3)

- nu e posibil (valorile cheii de căutare nu sunt ordonate)

Indecși convenționali

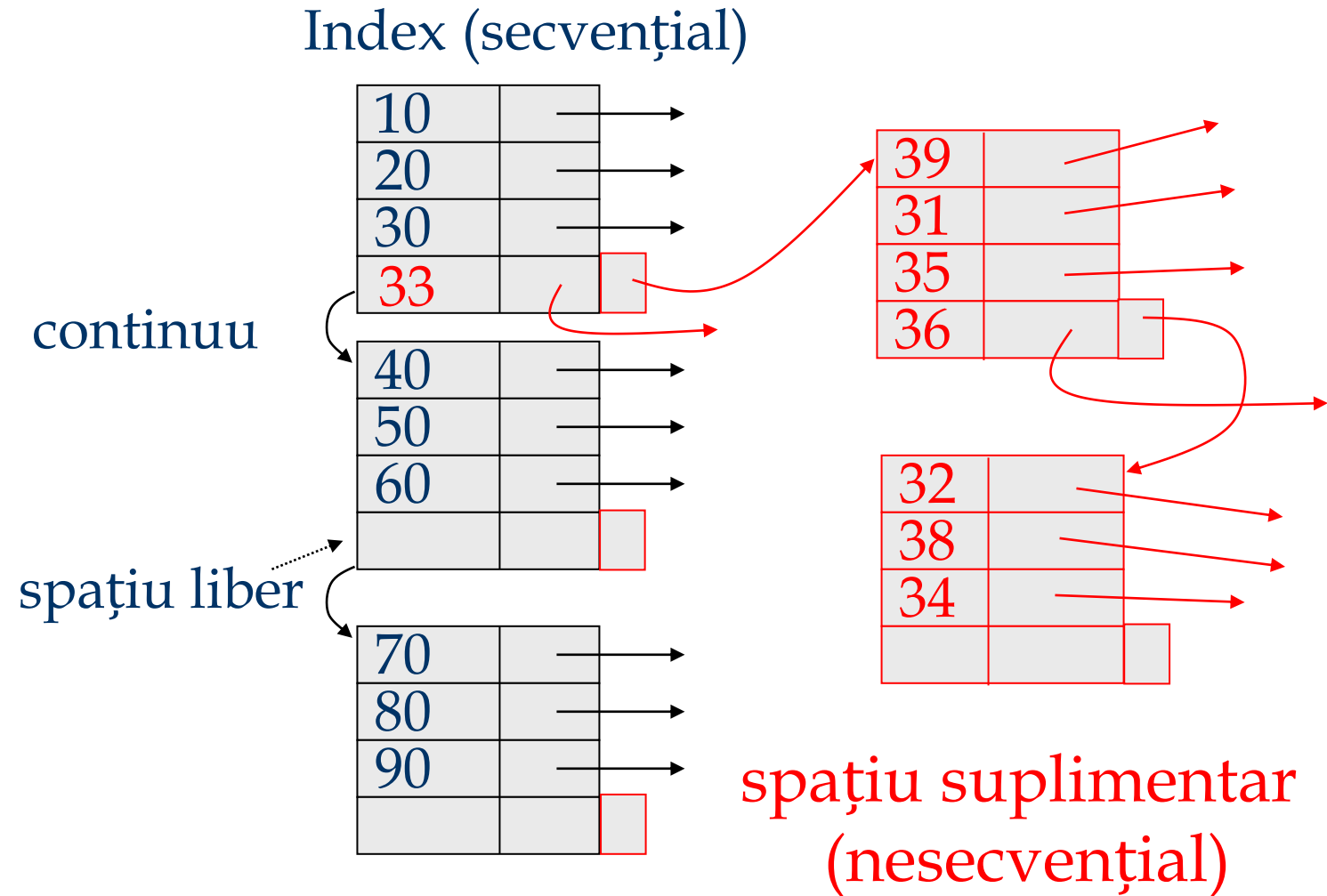
- Avantaje:

- Simpli
- Indexul e un fișier secvențial → potrivit pentru parcurgeri

- Dezavantaje:

- Inserările și/sau ștergerile sunt costisitoare
- Se pierde secvențialitatea & echilibrul

Exemplu



Înțelegere *workload*

- Pentru fiecare interogare utilizată:
 - Care sunt relațiile/tabelele accesate?
 - Care sunt atributele/câmpurile returnate?
 - Care dintre atribute sunt implicate în condiții de selecție/join? Cât de selective sunt aceste condiții?
- Pentru fiecare actualizare:
 - Care dintre atribute sunt implicate în condiții de selecție/join? Cât de selective sunt aceste condiții?
 - Ce tip de actualizare este (INSERT/DELETE/UPDATE), și care sunt atributele afectate?

Alegerea indecșilor

- Ce index trebuie creat?
- Pentru fiecare index, care e varianta de structurare utilizată?
- **Abordare:** Se analizează cele mai importante interogări. Pentru fiecare se consideră cel mai optim plan de execuție folosind indecșii existenți. Vom crea un nou index doar dacă acesta va genera un plan mai bun.
 - Evident, acest lucru implică să înțelegem cum evaluează un SGBD interogările și cum sunt create **planurile de execuție!**
- Înainte de a crea un nou index, trebuie să evaluăm impactul actualizării acestuia!
 - Indecșii accelerează execuția interogărilor, dar încetinesc actualizările. De asemenea, necesită spațiu suplimentar de stocare.

Alegerea indecșilor

- Atributele din clauza WHERE sunt chei de căutare “candidat”
 - Egalitățile sugerează o structură cu acces direct
 - Intervalele sugerează utilizarea unei structuri arborescente
 - Gruparea e foarte utilă în aceste situații; dar și în cazul egalităților atunci când numărul de duplicate este mare.
- Atunci când clauza WHERE are mai multe condiții se pot lua în considerare chei de căutare compuse
 - Ordine atributelor e importantă pentru a evalua condițiile cu intervale
 - Pentru interogările importante acești indecși pot determina strategii de execuție **index-only**
- Vor fi aleși indecșii de care beneficiază cele mai multe interogări posibile. Deoarece se poate defini un singur index grupat pe o tabelă, acesta va fi ales astfel încât să beneficieze de acest index o interogare foarte importantă

Chei de căutare compuse

- Pentru returnarea înregistrărilor din *Employees* cu $age=30$ AND $sal=4000$, un index pe $\langle age, sal \rangle$ e mai potrivit decât un index pe age sau unul pe sal .
- Dacă avem: $20 < age < 30$ AND $3000 < sal < 5000$:
 - Un index arborescent grupat pe $\langle age, sal \rangle$ sau $\langle sal, age \rangle$ e foarte bun.
- În cazul: $age=30$ AND $3000 < sal < 5000$:
 - Index grupat $\langle age, sal \rangle$ e mai bun decât index pe $\langle sal, age \rangle$
- Indecșii compuși sunt mai voluminoși și sunt actualizați mai des

Exemple de indecși grupați

- Index arborescent pe *E.age*
 - Cât de selectivă e condiția?
 - Indexul e grupat?
- Utilizare GROUP BY
 - Dacă mai multe înregistrări au *E.age > 10*, un index pe *E.age* urmat de ordonarea înregistrărilor e costisitor
 - Index grupat pe *E.dno* e mai potrivit!
- Egalitate cu duplicate:
 - Index grupat pe *E.hobby*

```
SELECT E.dno  
FROM Employees E  
WHERE E.age>40
```

```
SELECT E.dno, COUNT (*)  
FROM Employees E  
WHERE E.age>10  
GROUP BY E.dno
```

```
SELECT E.dno  
FROM Employees E  
WHERE E.hobby='Stamps'
```

Planuri *Index-Only*

- Anumite interogări pot fi executate fără a accesa tabelele originale atunci când este disponibil un index potrivit.

<E.dno>

```
SELECT D.mgr  
FROM Dept D, Emp E  
WHERE D.dno=E.dno
```

<E.dno,E.eid> Tree index!

```
SELECT D.mgr, E.eid  
FROM Dept D, Emp E  
WHERE D.dno=E.dno
```

<E.dno>

```
SELECT E.dno, COUNT(*)  
FROM Emp E  
GROUP BY E.dno
```

<E.dno,E.sal> Tree index!

```
SELECT E.dno, MIN(E.sal)  
FROM Emp E  
GROUP BY E.dno
```

*<E.age,E.sal> or <E.sal, E.age>
Tree index!*

```
SELECT AVG(E.sal)  
FROM Emp E  
WHERE E.age=25 AND  
E.sal BETWEEN 3000 AND 5000
```

Indexarea azi

- Indecși unidimensionali
 - B+ Trees, Hash Files
- Indecși multidimensionali
 - pentru baze de date spațiale (GIS)
 - R - Arbori
- Indexare multidimensională
 - Interogări pe intervale
- Indexare avansată
 - Indexarea memoriei interne