Algebră relațională (continuare)

Reuniune, intersecție, diferență

- $R_1 \cup R_2 = \{ t \mid t \in R_1 \vee t \in R_2 \}$
- $R_1 \cap R_2 = \{ t \mid t \in R_1 \wedge t \in R_2 \}$
- $R_1 - R_2 = \{ t \mid t \in R_1 \wedge t \notin R_2 \}$

Relațiile R_1 și R_2 trebuie să fie *compatibile*:

- același număr de atribute (aceeași *aritate*)

- atributele aflate pe aceeași poziție au domenii *compatibile* și *același nume*

Reuniune, intersecție, diferență în SQL

$R_1 \cup R_2$

```
SELECT DISTINCT *
FROM R1
UNION
SELECT DISTINCT *
FROM R2
```

$R_1 \cap R_2$

```
SELECT DISTINCT *
FROM R1
INTERSECT
SELECT DISTINCT *
FROM R2
```

$R_1 - R_2$

```
SELECT DISTINCT *
FROM R1
EXCEPT
SELECT DISTINCT *
FROM R2
```

Nu toți operatorii sunt esențiali

$$R_1 \cap R_2 = ((R_1 \cup R_2) — (R_1 — R_2)) — (R_2 — R_1)$$

Identifică tuplurile incluse în R_1 sau R_2

Elimină tuplurile ce aparțin doar R_1

Elimină tuplurile ce aparțin doar R_2

Produs cartezian

- Combinarea a doua relații

$$R_1(a_1, \ldots, a_n) \text{ și } R_2(b_1, \ldots, b_m)$$

$$R_1 \times R_2 = \{ t \mid t_1 \in R_1 \wedge t_2 \in R_2$$
$$\wedge\ t.a_1 = t_1.a_1 \ldots \wedge\ t.a_n = t_1.a_n$$
$$\wedge\ t.b_1 = t_2.b_1 \ldots \wedge\ t.b_m = t_2.b_m \}$$

```
SELECT DISTINCT *
FROM R1, R2
```

θ-Join

- Combinarea a doua relații R_1 și R_2 cu respectarea condiției *c*

$$R_1 \otimes_c R_2 = \sigma_c (R_1 \, X \, R_2)$$

$$\text{Students} \otimes_{\text{Students.sid=Enrolled.sid}} \text{Enrolled}$$

```
SELECT DISTINCT *
FROM Students,Enrolled
WHERE Students.sid =
Enrolled.sid
```

```
SELECT DISTINCT *
FROM Students
INNER JOIN Enrolled ON
Students.sid=Enrolled.sid
```

Equi-Join

- Combină două relații pe baza unei condiții compuse doar din egalități ale unor atribute aflate în prima și a doua relație și proiectează doar unul dintre atributele redundante (deoarece sunt egale)

$$R_1 \otimes_{E(c)} R_2$$

Courses

cid	*cname*
Alg1	Algorithms1
DB1	Databases1
DB2	Databases2

$\otimes_{E(\text{Courses.cid} = \text{Enrolled.cid})}$

Enrolled

sid	***cid***	*grade*
1234	Alg1	9
1235	Alg1	10
1234	DB1	10
1234	DB2	9
1236	DB1	7

=

cname	*sid*	*cid*	*grad*
Algorithms1	1234	Alg1	9
Algorithms1	1235	Alg1	10
Databases1	1234	DB1	10
Databases2	1234	DB2	9
Databases1	1236	DB1	7

Join Natural

- Combină două relații pe baza egalității atributelor ce au *același nume* și proiectează doar unul dintre atributele redundante

$$R_1 \otimes R_2$$

Courses

cid	*cname*
Alg1	Algorithms1
DB1	Databases1
DB2	Databases2

$\otimes$

Enrolled

sid	***cid***	*grade*
1234	Alg1	9
1235	Alg1	10
1234	DB1	10
1234	DB2	9
1236	DB1	7

=

cname	*sid*	*cid*	*grad*
Algorithms1	1234	Alg1	9
Algorithms1	1235	Alg1	10
Databases1	1234	DB1	10
Databases2	1234	DB2	9
Databases1	1236	DB1	7

Redenumirea

- Dacă atributele și relațiile au aceleași nume (de exemplu la *join*-ul unei relații cu ea însăși) este necesar să putem redenumi una din ele

$$\rho(R'\ (N_1 \to N'_1, N_2 \to N'_2\ ), R)$$

notație alternativă: $\rho_{R'\ (N'1,\ N'2\ )}(R)$,

- Noua relație R' are aceeași instanță ca R, iar structura sa conține atributul N'_i în locul atributului N_i

Redenumirea

$$\rho(\text{Courses2}\ (\text{cid} \rightarrow \text{code},\ \text{cname} \rightarrow \text{description}\ ),\ \text{Courses})$$

Courses

cid	*cname*	*credits*
Alg1	Algorithms1	7
DB1	Databases1	6
DB2	Databases2	6

$\rightarrow$

Courses2

code	*description*	*credits*
Alg1	Algorithms1	7
DB1	Databases1	6
DB2	Databases2	6

```
SELECT cid as code,
       cname as description,
       credits
FROM Courses Courses2
```

Operaţia de atribuire

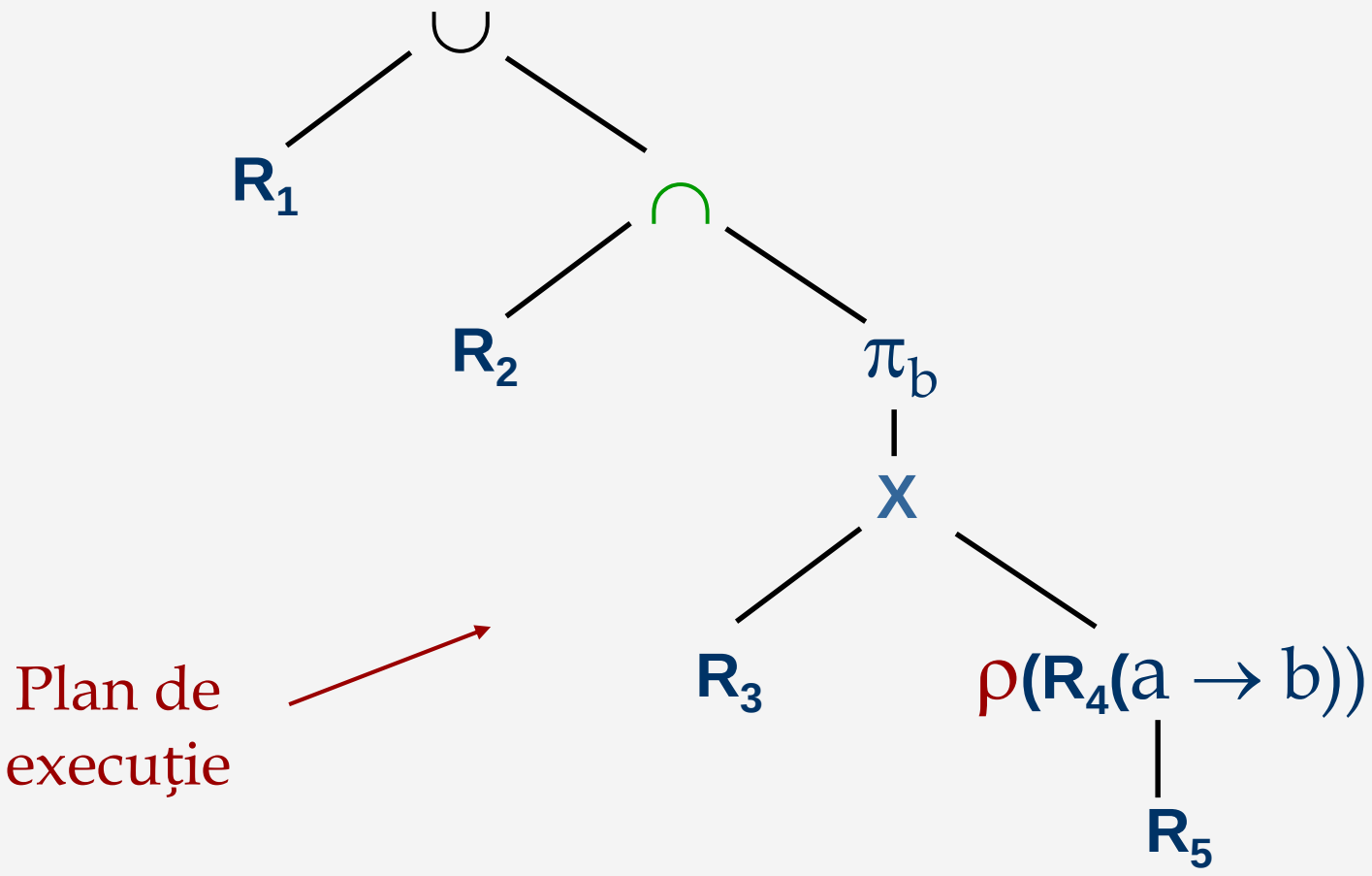
- Operatia de atribuire ($\leftarrow$) oferă un mod simplu de tratare a interogărilor complexe.
  - Atribuirile se fac intotdeauna într-o variabilă temporară

$$Temp \leftarrow \pi_x(R_1 \; X \; R_2)$$

- Rezultatul expresiei din dreapta $\leftarrow$ este atribuit variabilei din stânga operatorului $\leftarrow$.
- Variabilele pot fi utilizate apoi în alte expresii
  - result $\leftarrow$ Temp – R_3

Expresii complexe

$$R_1 \cup (R_2 \cap \pi_b (R_3 \; X \; \rho(R_4(a \rightarrow b), R_5)))$$



Determinați numele tuturor studenților cu note la cursul ‘BD1’

Solutie 1: π_{name} (($\sigma_{cid='BD1'}$(Enrolled)) $\otimes$ Students)

Solutie 2: ρ ($Temp_1$, $\sigma_{cid='BD1'}$(Enrolled))

ρ ($Temp_2$, $Temp_1 \otimes$ Students)

π_{name} ($Temp_2$)

Solutie 3: π_{name} ($\sigma_{cid='BD1'}$(Enrolled $\otimes$ Students))

Determinați numele tuturor studenților cu note la cursuri cu 5 credite

- Informația cu privire la credite se găsește în relatia *Courses*, și prin urmare se adaugă un join natural:

$$\pi_{name}\,(\,(\sigma_{credits=5}(Courses)) \otimes Enrolled \otimes Students\,)$$

- O soluție mai eficientă:

$$\pi_{name}\,(\,\pi_{sid}(\pi_{cid}(\sigma_{credits=5}(Courses)) \otimes Enrolled) \otimes Students\,)$$

Modulul de optimizare a interogărilor e capabil să transforme prima solutie în a doua!

Determinați numele tuturor studenților cu note la cursuri cu 4 sau 5 credite

- Se identifică toate cursurile cu 4 sau 5 credite, apoi se determină studenții cu note la unul dintre aceste cursuri:

$$\rho\ (\mathit{TempCourses}, (\sigma_{\text{credits=4} \vee \text{credits=5}}(\text{Courses})))$$

$$\pi_{\text{name}}\ (\mathit{TempCourses} \otimes \text{Enrolled} \otimes \text{Students}\ )$$

- *TempCourses* se poate defini și utilizând reuniunea!

- Ce se întâmplă dacă înlocuim $\vee$ cu $\wedge$ în interogare?

Determinați numele tuturor studenților cu note la cursuri cu 4 si 5 credite

- Abordarea anterioară nu funcționează! Trebuie identificați în paralel studenții cu note la cursuri de 4 credite și studenții cu note la cursuri de 5 credite, apoi se intersectează cele două mulțimi (*sid* este cheie pentru *Students*):

$$\rho\ (\mathit{Temp4},\ \pi_{\text{sid}}(\sigma_{\text{credits=4}}\ (\text{Courses}) \otimes \text{Enrolled}))$$

$$\rho\ (\mathit{Temp5},\ \pi_{\text{sid}}(\sigma_{\text{credits=5}}\ (\text{Courses}) \otimes \text{Enrolled}))$$

$$\pi_{\text{name}}\ ((\mathit{Temp4} \cap \mathit{Temp5}) \otimes \text{Students})$$

Extensii ale operatorilor algebrici relaționali

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

Generalizarea proiecției

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

$$\pi_{F1,\, F2,\ldots,\, Fn}\,(R)$$

- *R* poate fi orice expresie din algebra relațională
- Fiecare dintre F_1, F_2, …, F_n sunt expresii aritmetice ce implică atribute 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

$$\mathcal{G}_{G_1, G_2, \ldots, G_n \quad F_1(A_1), F_2(A_2), \ldots, F_n(A_n)}(R)$$

- *R* poate fi orice expresie din algebra relațională
  - G_1, G_2 …, 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

Aggregarea – Exemple

Relatie R:

A	B	C
α	α	7
α	β	7
β	β	3
β	β	10

$\mathcal{g}_{\text{sum(C)}}(R)$

sum(C)
27

- Rezultatul agregarii nu are un nume
  - se pot folosi operatorii de redenumire
  - se poate permite redenumirea ca parte a unei operatii de agregare

Outer Join

- Extensii ale operatorului join natural care impiedică pierderea informației:
  - Left Outer Join 
  - Right Outer Join 
  - Full Outer Join 
- Realizează joncțiunea și apoi adaugă la rezultat tuplurile dintr-una din relatii (din stânga, dreapta sau ambele părți ale operatorului) care nu sunt conectate cu tupluri din celaltă relație.
- Utilizeaza valoarea *null*:
  - *null* semnifică faptul că valoarea e necunoscută sau nu există
  - Toate comparatiile ce implică *null* sunt (*simplu spus*) **false** prin definiție.

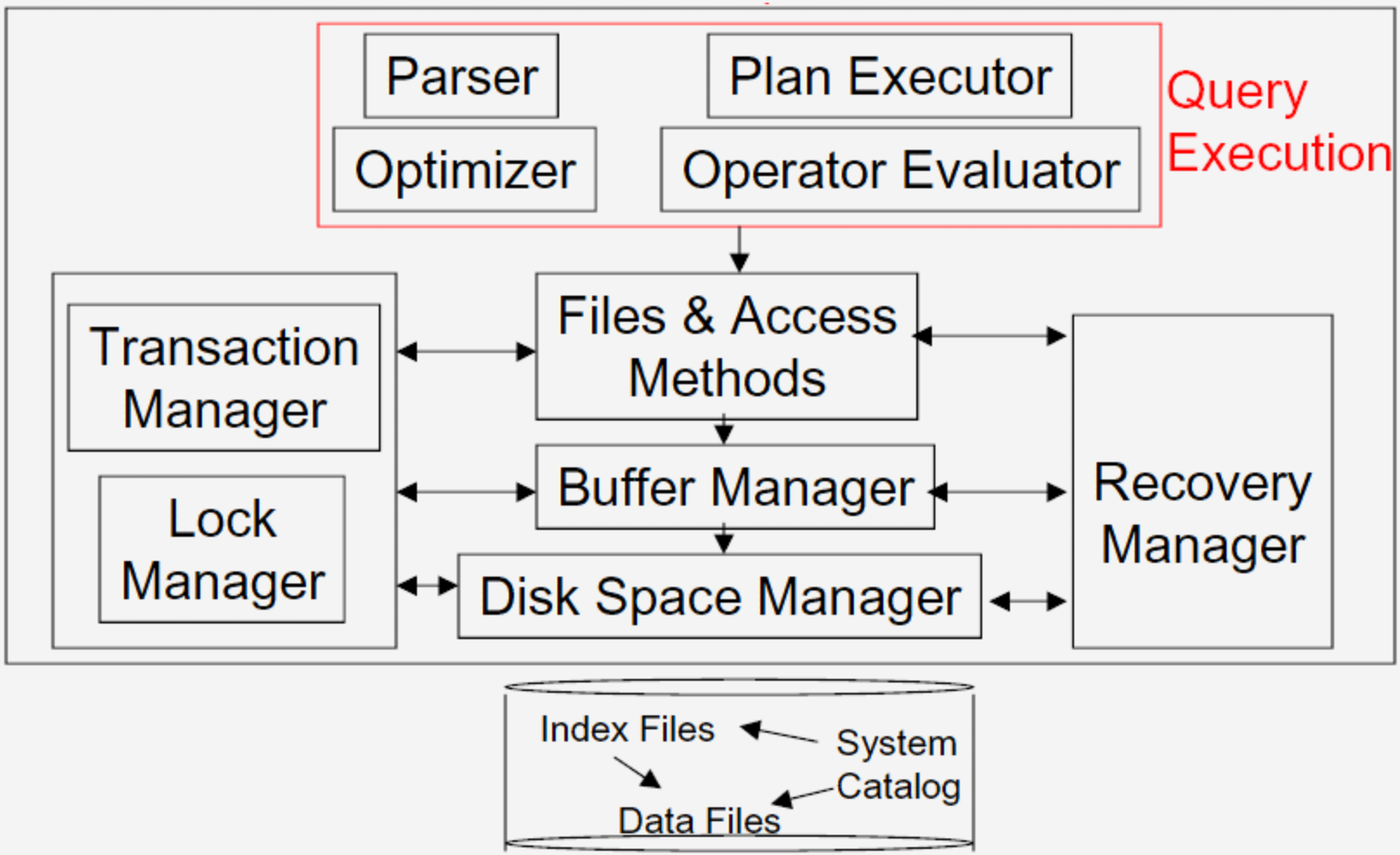
Modificarea bazei de date

- Conținutul bazei de date poate fi modificat folosind următorii operatori:
  - Ștergere $R \leftarrow R - E$
  - Inserare $R \leftarrow R \cup E$
  - Modificare $R \leftarrow \pi_{F1, F2, \ldots, Fn}(R)$

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

Structura fizică
a
bazelor de date

Structura unui SGBD



Structura fizică a fişierelor BD

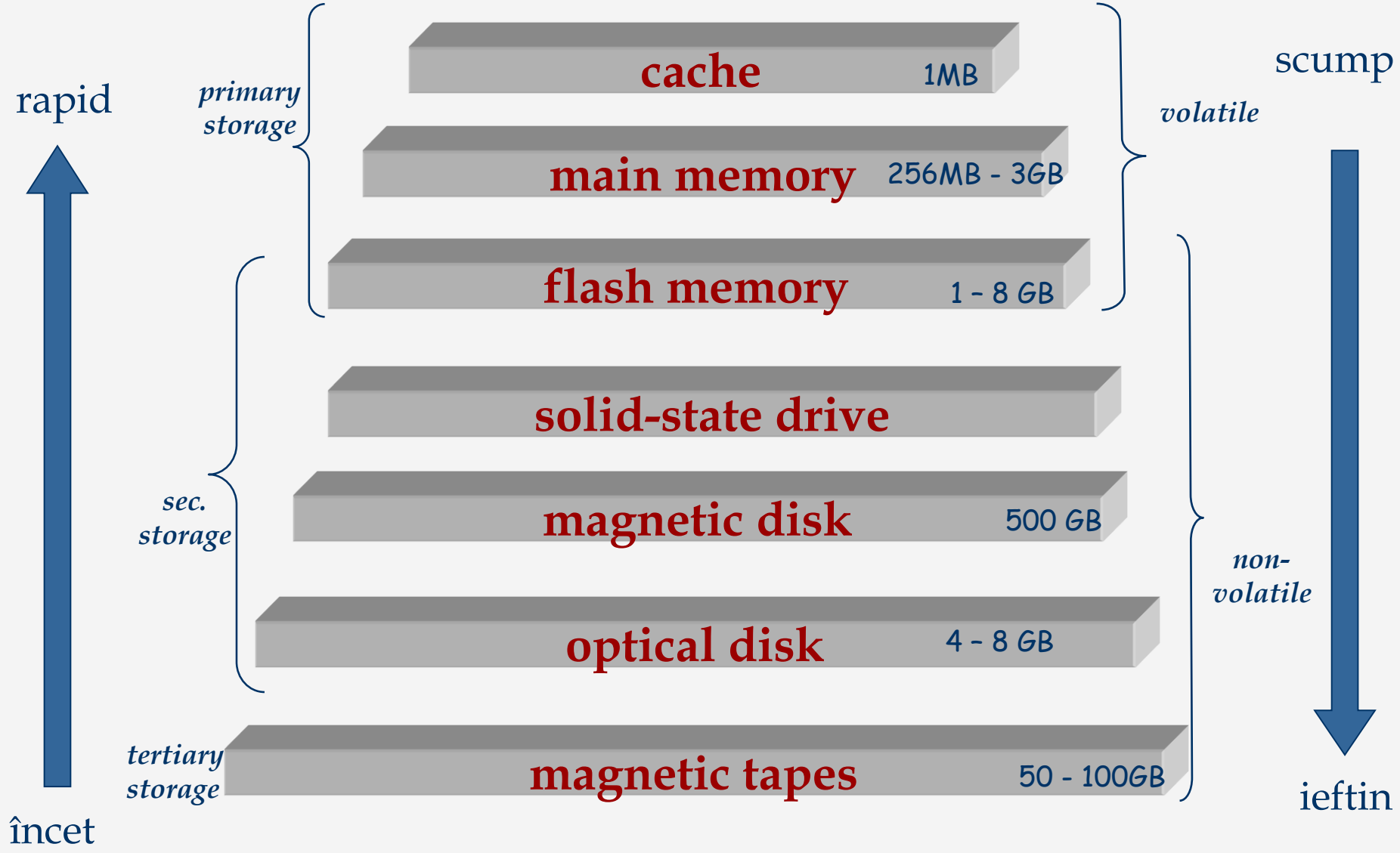
- SGBD-urile stochează informația pe disc magnetic
- Acest lucru are implicații majore în proiectarea unui SGBD!
  - READ: transfer date de pe disc în memoria internă
  - WRITE: transfer date din memoria internă pe disc

Ambele operații sunt costisitoare, comparativ cu operațiile *in-memory*, deci trebuie planificate corespunzător!

De ce nu stocăm totul în memoria internă?

- Răspuns (tipic):
  - Costă prea mult
  - Memoria internă este volatilă (datele trebuie să fie persistente)
- Procedură tipică("ierarhie de stocare")
  - RAM – pentru datele utiliz. curent (*primary storage*)
  - *Hard-disks* – pentru baza de date (*secondary storage*)
  - Bandă – pentru arhivarea versiunilor anterioare ale datelor (*tertiary storage*)

Ierarhia mediilor de stocare



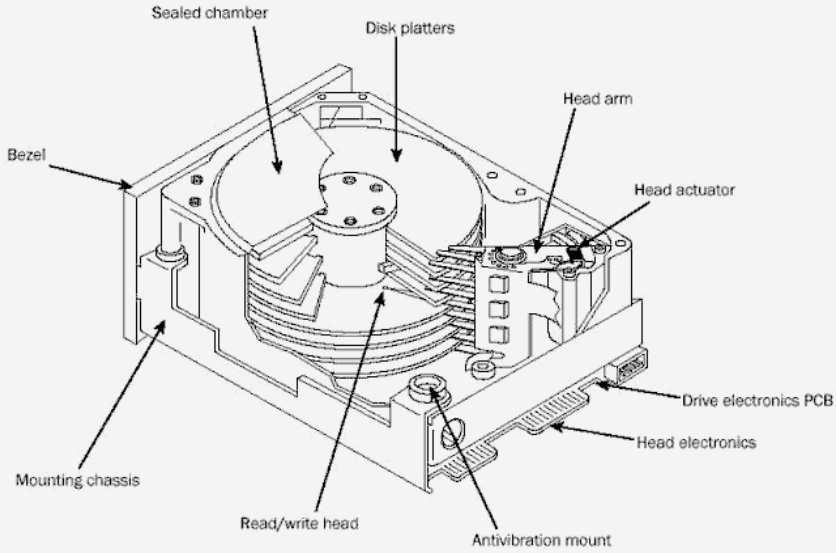
Legea lui Moore

- Gordon Moore: "*Integrated circuits are improving in many ways, following an exponential curve that doubles every 18 months*"
  - Viteza procesoarelor
  - Numărul de biți de pe un chip
  - Numărul de octeți (*bytes*) pe un hard disk
- Parametrii ce NU urmează legea lui Moore:
  - Viteza de accesare a datelor în memoria internă
  - Viteza de rotație a discului

$\Rightarrow$ Latența devine progresiv mai mare

- Timpul de transfer între nivelele ierarhiei mediilor de stocare este tot mai mare în comparație cu timpul de calcul

Discuri magnetice

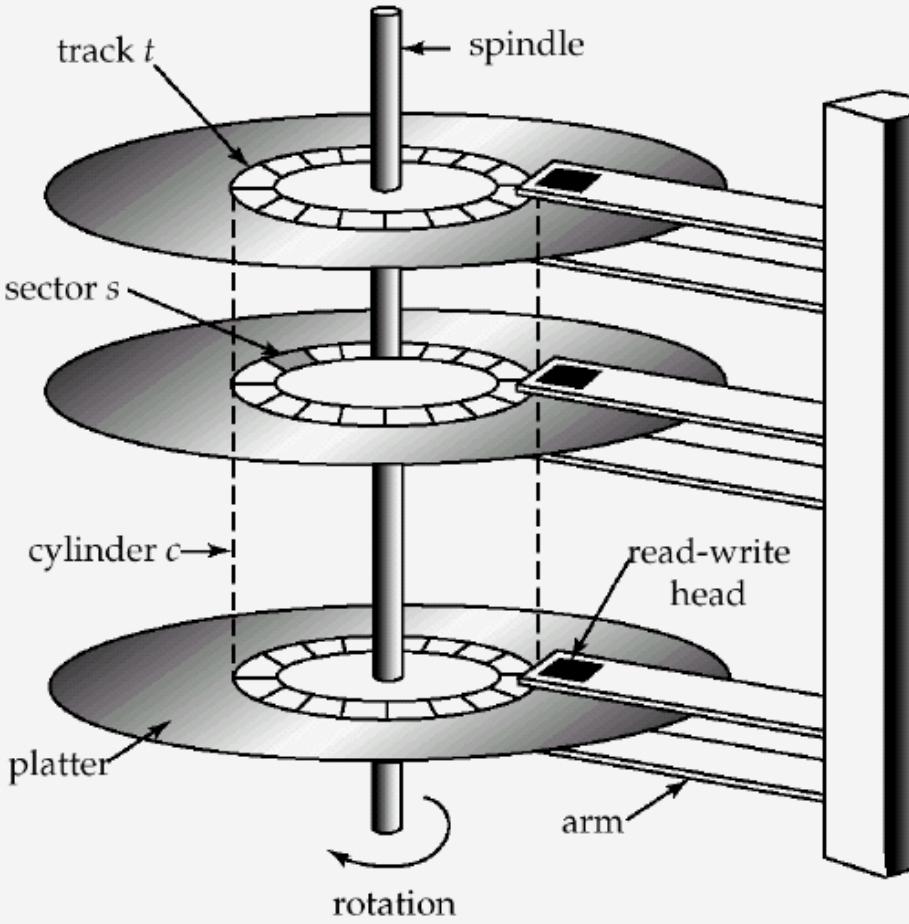


- Utilizat ca mediu de stocare secundar
- Avantaj major asupra benzilor: *acces direct*
- Datele sunt stocate şi citite în unități numite blocuri sau pagini.
- Spre deosebire de memoria internă, timpul de transfer blocurilor/paginilor variază în funcție de poziția acestora pe disc.

! Poziția relativă a paginilor pe disc
are un impact major asupra perfomanței unui SGBD!

Componentele unui disc dur (*hard disk*)

- Rotația platanelor (90rps)
- Ansamblu de brațe ce se deplasează pentru poziționarea capului magnetic pe pista dorită. Pistele aflate la aceeași distanță de centrul platanelor formează un cilindru (imaginar!).
- Un singur cap citește/ scrie la un moment dat.
- Un bloc e un multiplu de sectoare (care e fix).



Accesarea unei pagini (bloc)

- Timp de acces (citire/scriere) a unui bloc:

  seek time (mutare braț pentru poziționarea capului de citire/scriere pe pistă)

  rotational delay (timp poziționare bloc sub cap)

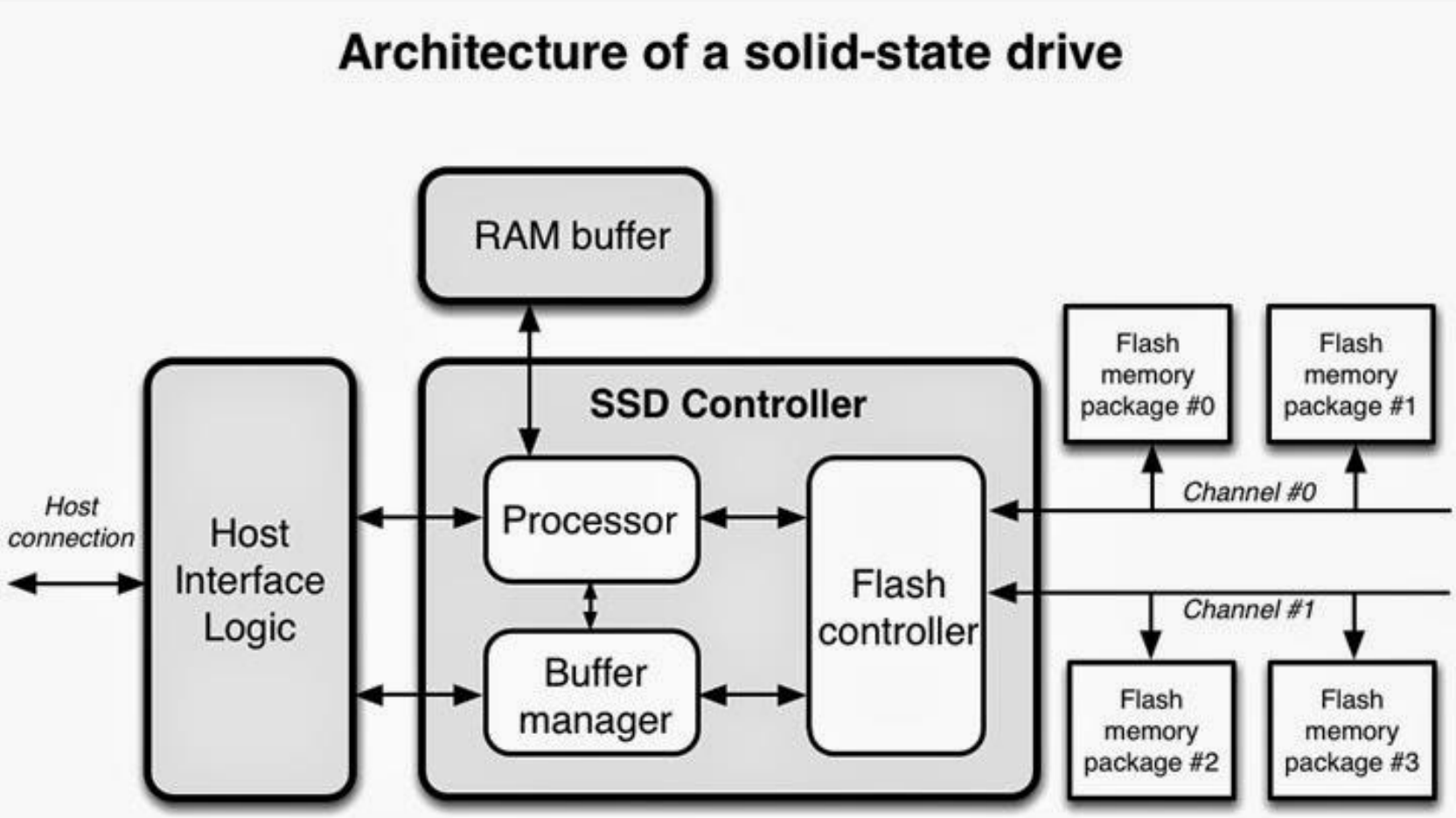
  transfer time (transfer date de pe/pe disc)
- *Seek time* și *rotational delay* domină.
  - *Seek time* variază între 1 și 20msec
  - *Rotational delay* variază între 0 și 10msec
  - *Transfer rate* e de aproximativ 1msec pe 4KB (pagină)
- Reducerea costului I/O: reducere *seek/rotational delays*!
- Soluții *hardware* sau *software*?

Aranjarea paginilor/blocurilor pe disc

- Conceptul de *next block*:
  - blocuri pe aceeași pistă, urmate de
  - blocuri pe același cilindru, urmate de
  - blocuri pe cilindri adiacenți
- Blocurile dintr-un fișier trebuie dispuse secvențial pe disc (`*next*'), pentru a minimiza *seek delay* și *rotational delay*.
- În cazul unei scanări secvențiale, citirea de pagini în avans (*pre-fetching*) este esențială!

Solid State Drive

- SSD-urile conțin mai multe componente NAND flash (16/32 GB)



Solid State Drive - Avantaje

- Latență foarte mică
  - *seek time* este zero
- Viteze mari de citire și scriere
- Mai robuste fizic
  - Rezistente la șocuri
  - Zero părți mobile
    - Silențioase
    - Consumă puțin
- Excelează la citiri/scrieri de dimensiuni reduse
- “Imun” la fragmentarea datelor

Solid State Drive – Dezavantaje

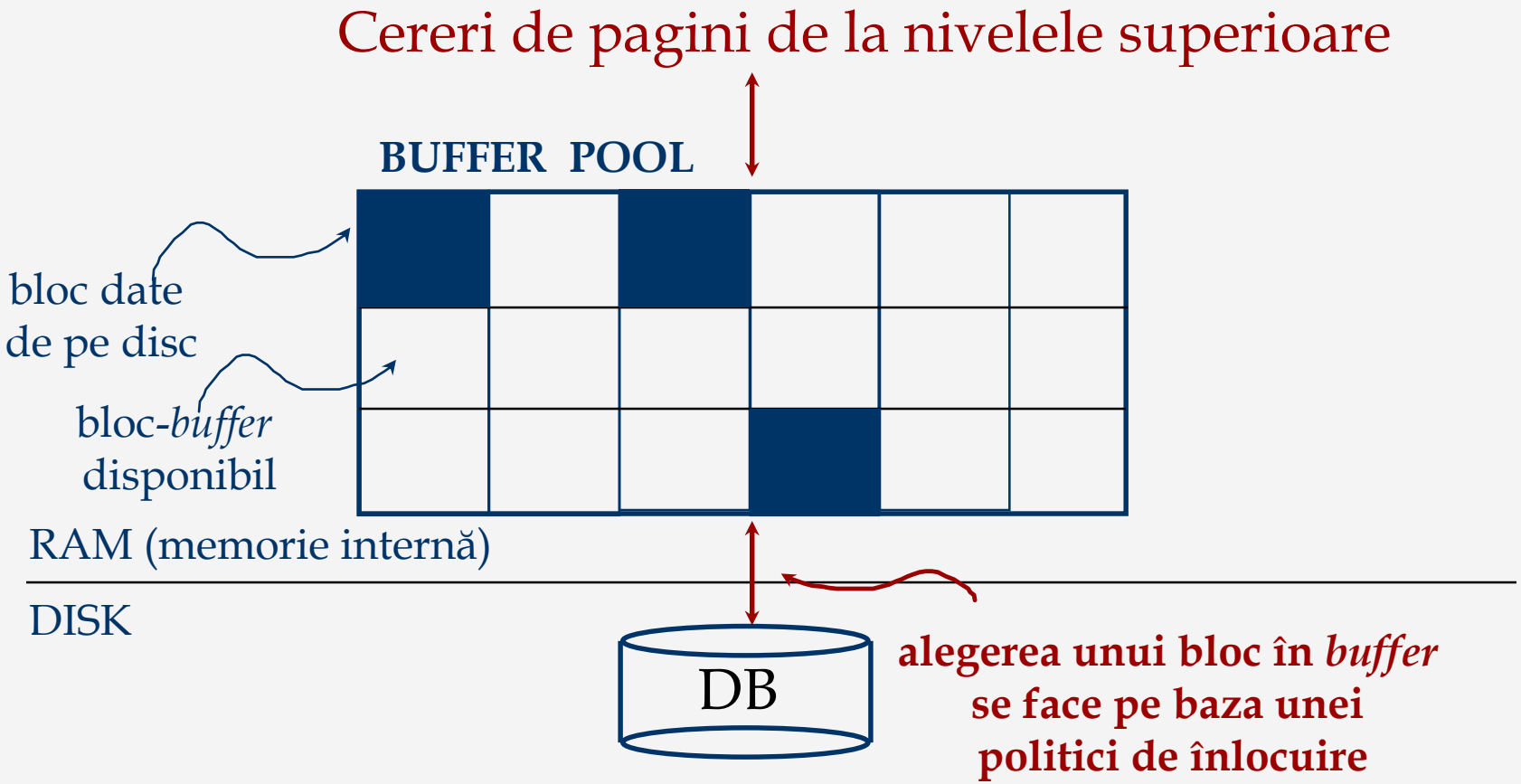
- Cost per GB mult mai mare comparativ cu discurile magnetice
- Dimensiuni
  - HDD 3.5'' cu 4TB sunt relativ comune
  - SSD 3.5'' cu 2TB sunt disponibile (rare și scumpe)
- Cicluri de citire/scriere limitate
  - 1 - 2 millioane cicluri de scriere $\Rightarrow$ uzura MLC (multi-level cell)
  - Sub 5 milioane cicluri de scriere $\Rightarrow$ uzura SLC

Gestionare *buffer* (zona de lucru) de către SGBD

- ***Buffer*** – partiție a memoriei interne utilizată pentru stocarea de cópii ale blocurilor de date.
- ***Buffer manager*** – modul SGBD responsabil cu alocarea spațiului de *buffer* în memoria internă.

- ***Buffer manager***-ul este apelat când este necesară accesarea unui bloc de pe disc
  - SGBD-ul operează asupra datelor din memoria internă

Gestionare *buffer* de către SGBD



- Se actualizează o tabelă cu perechi <nr_bloc_buffer, id_bloc_date>

La cererea unui bloc de date…

- Dacă blocul nu se regăsește în *buffer*:
  - Se alege un bloc disponibil pt. înlocuire
  - Dacă blocul conține modificări acesta este transferat pe disc
  - Se citește blocul dorit în locul vechiului bloc
- Blocul e *fixat* și se returnează adresa sa

! Dacă cererile sunt predictibile (ex. scanări secvențiale) pot fi citite în avans mai multe blocuri la un moment dat

Gestionare *buffer* de către SGBD

- Programul care a cerut blocul de date trebuie să îl elibereze și să indice dacă blocul a fost modificat:
  - folosește un dirty bit.
- Același bloc de date din *buffer* poate fi folosit de mai multe programe:
  - folosește un pin count. Un bloc-*buffer* e un candidat pentru a fi înlocuit ddacă pin count = 0.
- Modulele *Concurrency Control* & *Crash Recovery* pot implica acțiuni I/O adiționale la înlocuirea unui bloc-*buffer*.

Politici de înlocuire a blocurilor în *buffer*

- ***Least Recently Used*** (LRU): utilizează șablonul de utilizare a blocurilor ca predictor al utilizării viitoare. Interogările au șabloane de acces bine definite (ex, scanările secvențiale), iar un SGBD poate utiliza informațiile din interogare pentru a prezice accesările ulteriore ale blocurilor.
- ***Toss-immediate***: eliberează spațiul ocupat de un bloc atunci când a fost procesat ultima înregistrare stocată în blocul respectiv
- ***Most recently used*** (MRU): după procesarea ultimei înregistrări dintr-un bloc, blocul este eliberat (*pin count* e decrementat) și devine blocul utilizat cel mai recent.

Politici de înlocuire a blocurilor în *buffer*

- *Buffer Manager* poate utiliza **informații statistice** cu privire la probabilitatea ca o anumită cerere să refere un anumit bloc sau chiar o anumită relație
- Politicile de înlocuite pot avea un impact determinant în ceea ce privește numărul de I/Os – dependent de șablonul de acces.
- *Sequential flooding*: problemă generată de LRU + scanări secvențiale repetate.
  - Nr blocuri-buffer < Nr blocuri în tabelă → fiecare cerere de pagină determină un I/O. MRU e preferabil într-o astfel de situație.

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 atribute 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 tabelei indexate pentru care cheia de căutare are valoarea* ***k***".

Proprietățile 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

Proprietățile indecșilor

- *Dimensiunea indecșilor*
  - *D*eoarece utilizarea unui index presupune citirea acestuia în memoria internă → dimensiune indexului trebui 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) <**k**, *rid*>, rid – referință spre înregistrarea cu valoarea **k** pentru cheia de căutare

(3) <**k**, listă de *rid*-uri>

- 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

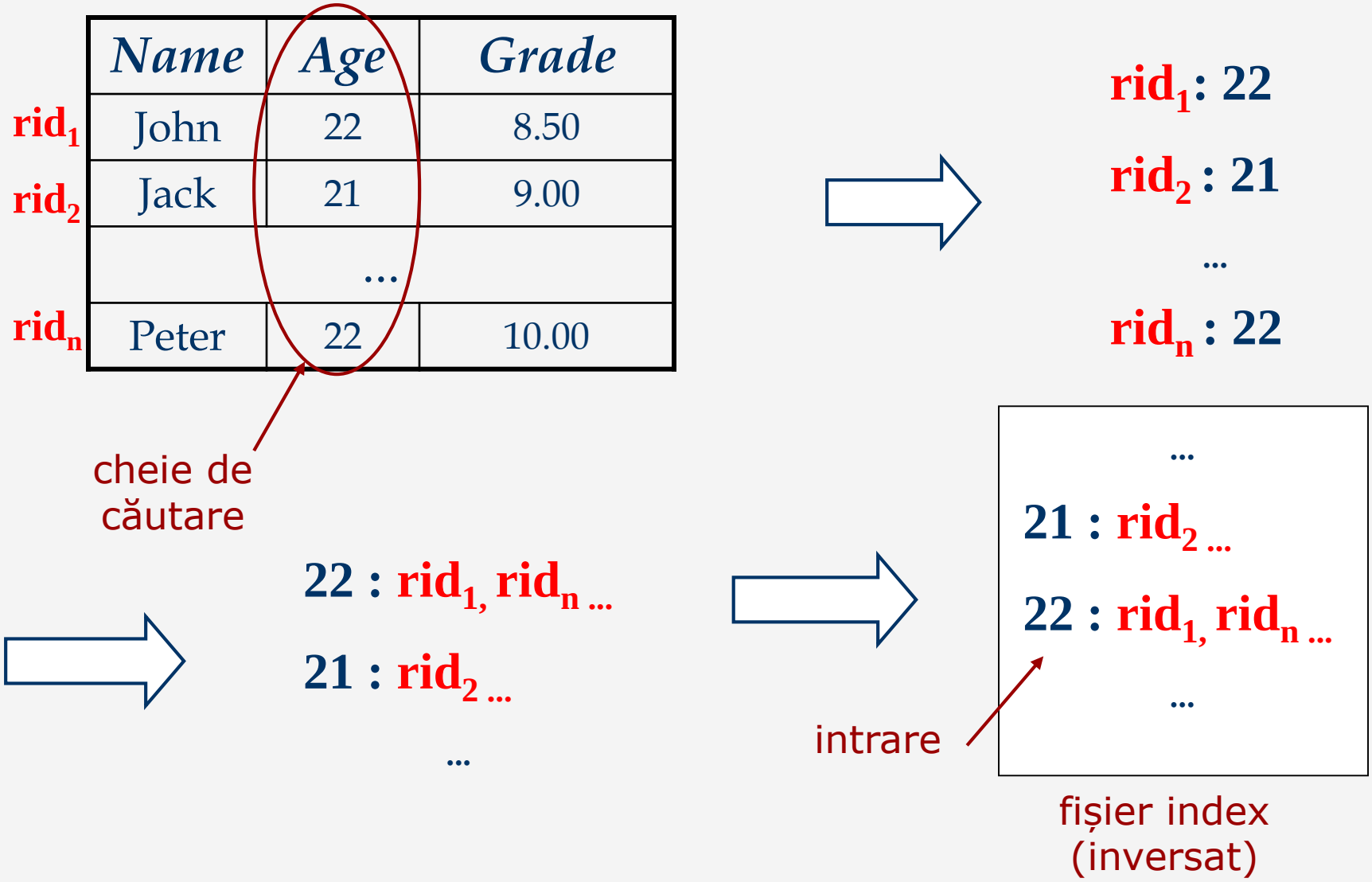
- Varianta (1):
  - Indexul și înregistrările originale sunt stocate împreună
  - Pentru o colecție de înregistrări se poate crea cel mult un index structurat conform variantei (1) (în caz contrar → înregistrări duplicate → redundanță → potențiale inconsistențe)
  - Dacă înregistrările sunt voluminoase, 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



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* vs. *un-clustered*:
  - 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*) cel mult o pentru o cheie de căutare.
  - Costul regăsirii datelor prin intermediul unui index e influențat *decisiv* de grupare!

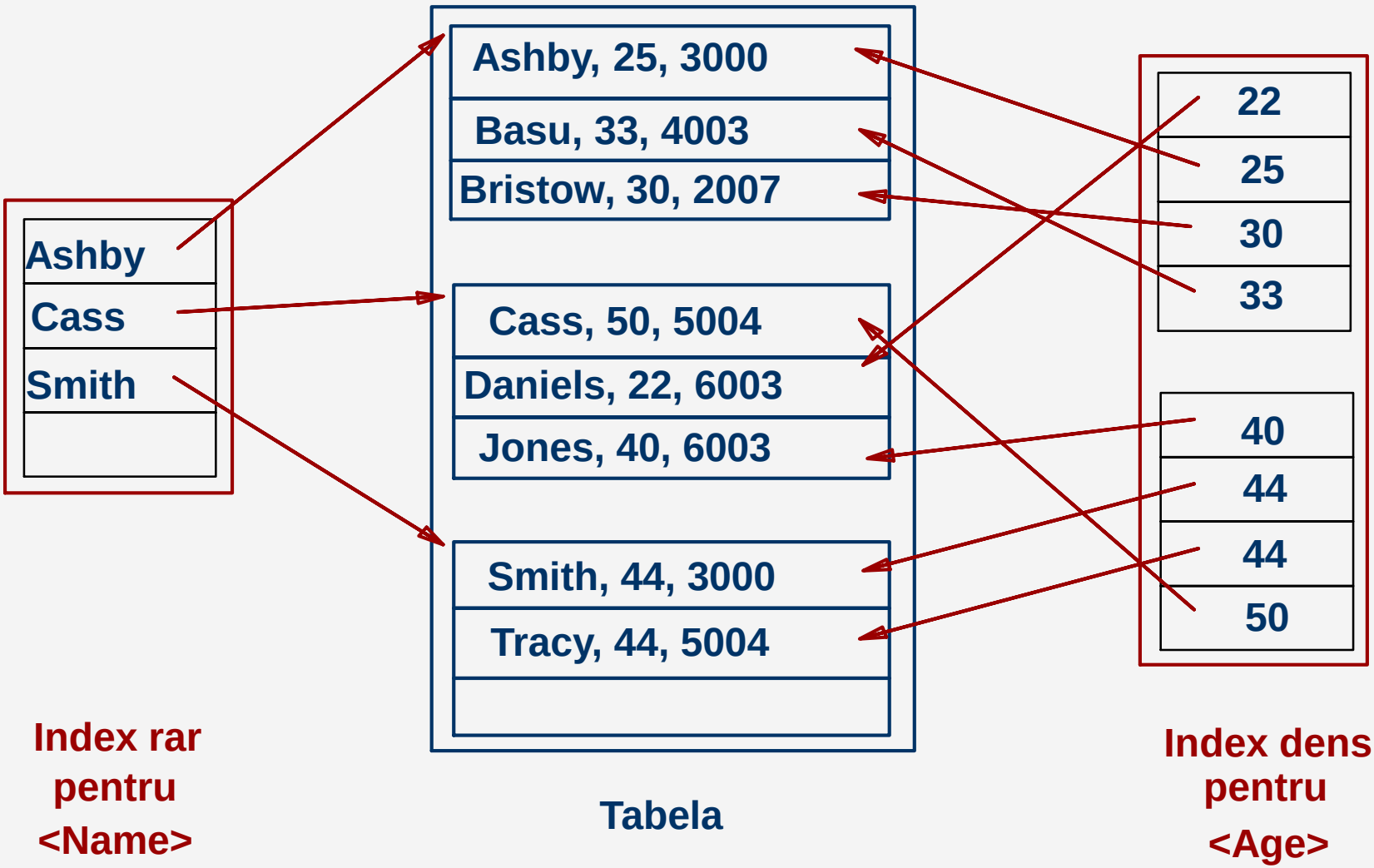
Index Clustered vs. Un-clustered

- 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 compacți.

Clasificarea indecșilor



Clasificarea indecșilor

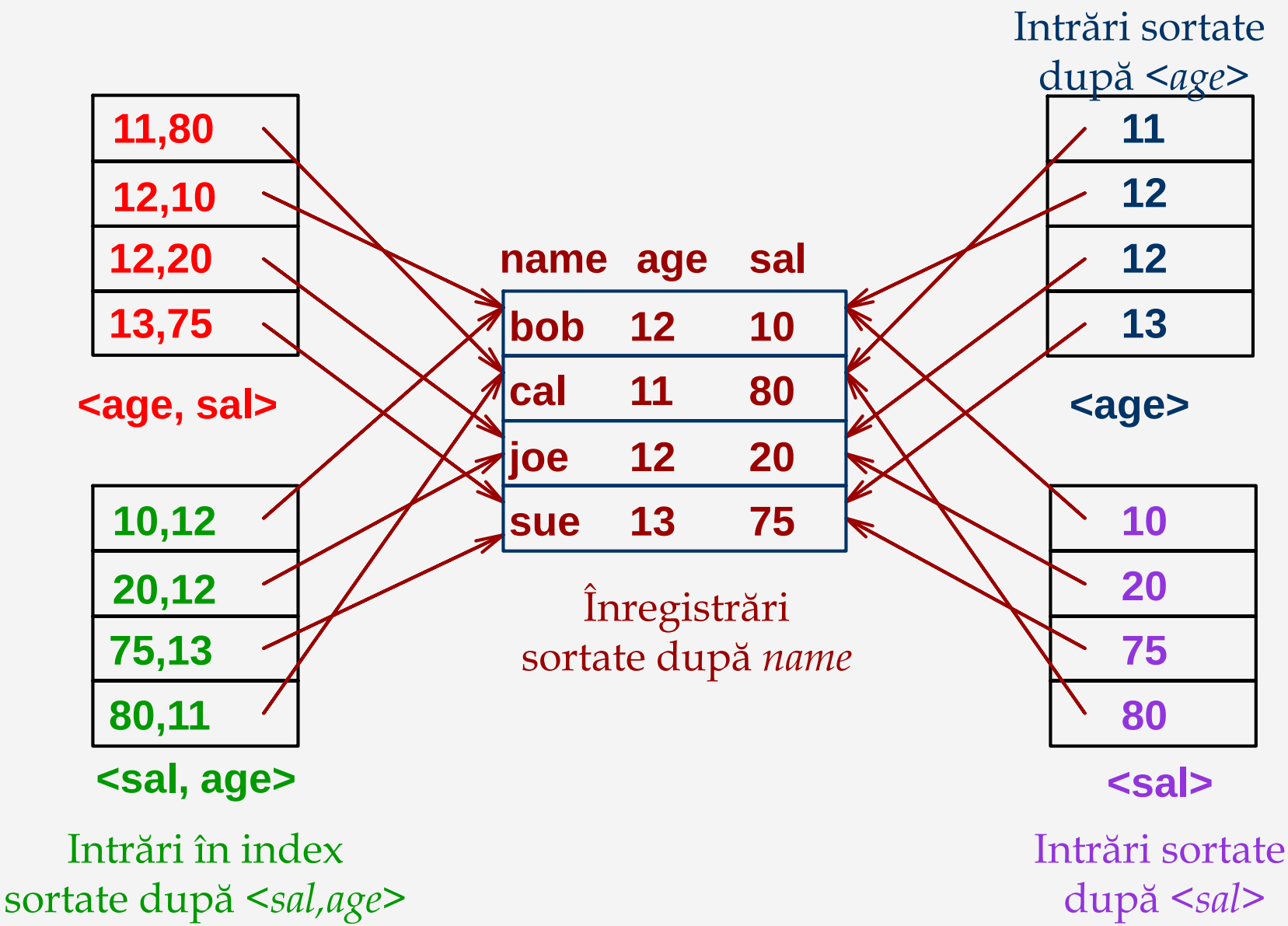
- Chei de căutare compuse:
- Utile atunci când căutarea se realizează după o combinație de câmpuri
  - Interogări cu egalitate: valoarea fiecărui câmp e egală cu o valoare constantă:

    age=20 și *sal =75*
  - Interogări cu interval: valoare unui câmp nu e o constantă:

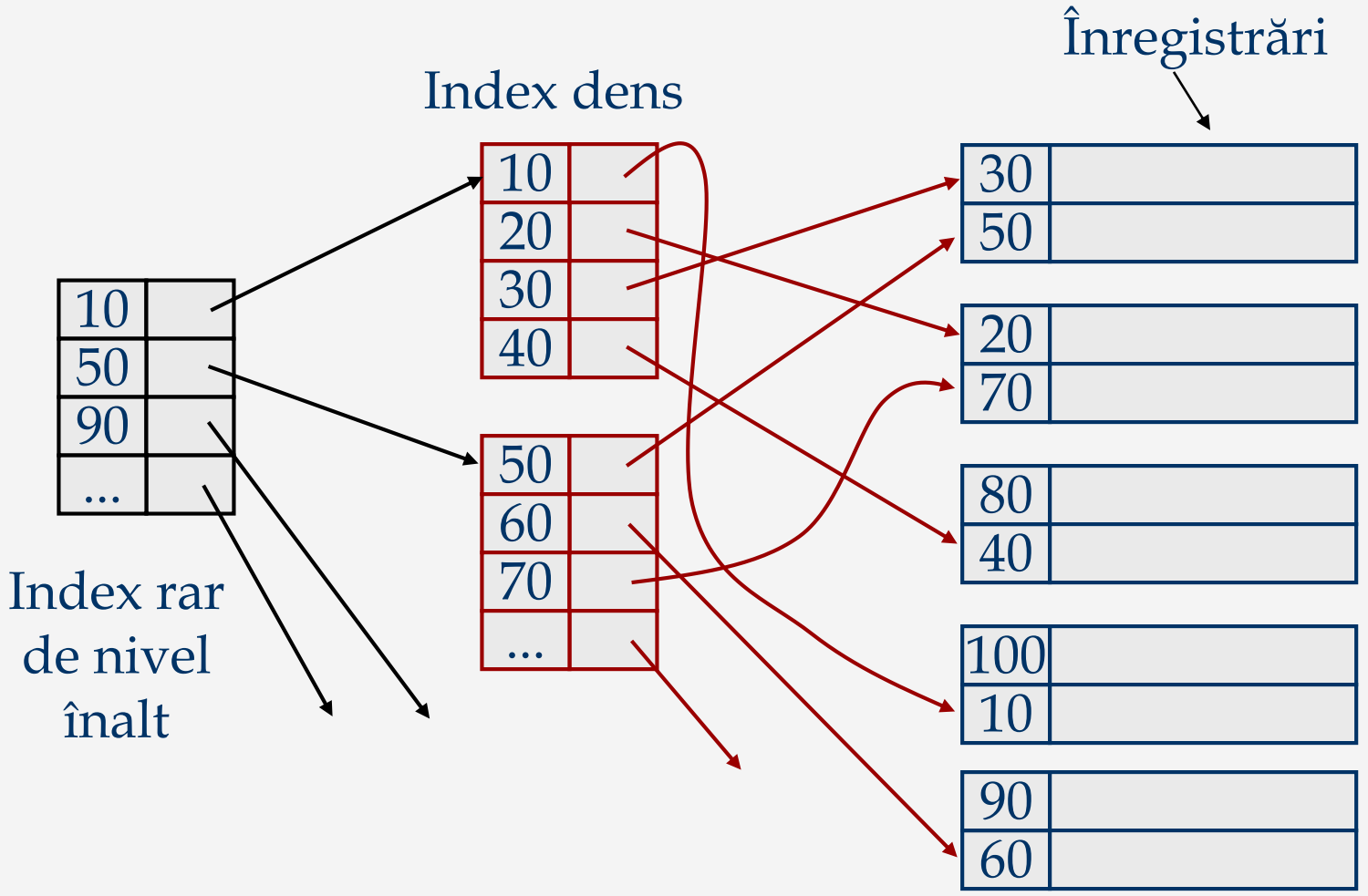
    age=20 și *sal > 10*

- 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 Secundari



Exemplu

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

ID	*Name*	*Age*	*Grade*	*Course*
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 <*page_id, slot no*> pentru identificarea unui tuplu).

Exemple

ID	*Name*	*Age*	*Grade*	*Course*
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,<1,1>) (12,<1,2>)(18,<1,3>)(19,<2,1>)(19,<2,2>)

3. *Age* – Dens, varianta (3)
   (11,<1,1>) (12,<1,2>)(18,<1,3>)(19,<2,1>,<2,2>)

4. *Age* – Rar, varianta (1)
   - un astfel de index nu poate fi construit (definiție)

Exemplu

ID	*Name*	*Age*	*Grade*	*Course*
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

ID	*Name*	*Age*	*Grade*	*Course*
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,<1,1>)(8.0,<1,2>)(9.2,<2,1>)(9.4,<1,3>)(9.8,<2,2>)

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)

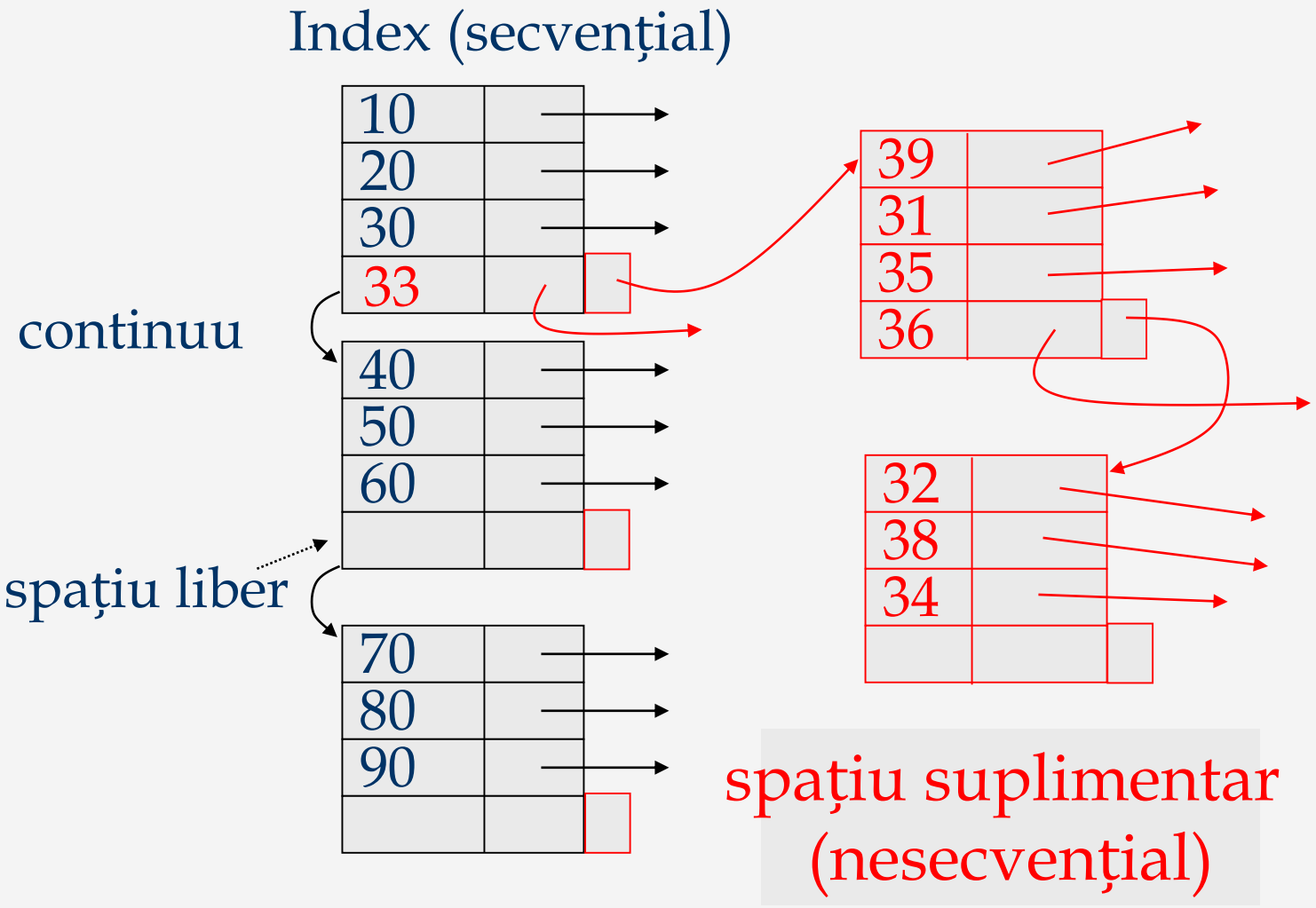
12. *Grade* – Rar, varianta (3)

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

Indecși convenționali

- Avantaje:
  - Simpli
  - Indexul e un fișier secvențial → potrivit pentru parcurgeri
- Dezavantaje:
  - Inserările sunt costisitoare, și/sau
  - Se pierd secvențialitatea & echilibrul

Exemplu


Index (secvențial)
10
20
30
33
continuu
40
50
60
spațiu liber
70
80
90
39
31
35
36
32
38
34
spațiu suplimentar
(nesecvențial)

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 înțtelegem 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 numrul 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 <*age*,*sal*> 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 <*age*,*sal*> sau <*sal*,*age*> e foarte bun.
- În cazul: *age*=30 AND 3000<*sal*<5000:
  - Index grupat <*age*,*sal*> e mai bun decât index pe <*sal*,*age*>
- 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?

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

- 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!

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

- Egalitate cu duplicate:
  - Index grupat pe *E.hobby*

```
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
```

Planuri *Index-Only*

- Un plan *index-only* e posibil dacă cheia de căutare e <dno,age> sau <age,dno>
  - Care variantă e mai bună?
  - Rămâne aceeași varianta pentru a doua interogare?

```
SELECT  E.dno,  COUNT (*)
FROM  Emp E
WHERE  E.age=30
GROUP BY E.dno
```

```
SELECT  E.dno,  COUNT (*)
FROM  Emp E
WHERE  E.age>30
GROUP BY E.dno
```