Structura fizica a bazelor de date relationale (partea 3)

Ioana Ciuciu
ioana.ciuciu@ubbcluj.ro
http://www.cs.ubbcluj.ro/~oana/

Planificare

Sunt posibile mici modificari ale planificarii in timpul semestrului

Saptamana	Curs	Seminar	Laborator
S1	1. Concepte fundamentale ale bazelor de date. Modelare conceptuala	1. Modelul Entitate-Relatie. Modelul relational	1. Modelarea unei BD in modelul ER si implementarea ei in SQL Server
S2	2. Modelul relational de organizare a bazelor de date. Modelare conceptuala		
S3	3. Gestiunea bazelor de date relationale cu limbajul SQL (DDL)	2. Limbajul SQL – definirea si actualizarea datelor	2. Interogari SQL
S4	4. Gestiunea bazelor de date relationale cu limbajul SQL (DML)		
S5-6	5-6. Dependente functionale, forme normale	3. Limbajul SQL – regasirea datelor	3. Interogari SQL avansate
S7	7. Interogarea bazelor de date relationale cu operatori din algebra relationala	4. Proceduri stocate	4. Proceduri stocate. View. Trigger
S8	8. Structura fizica a bazelor de date relationale		
S9	9. FARA CURS (30 nov.)	5. View-uri. Functii definite de utilizator. Trigger	
S10-11	10-11. Indecsi. Arbori B. Fisiere cu acces direct	6. Formele normale ale unei relatii. Indecsi	
S12	12. Extensii ale modelului relational si baze de date NoSQL – curs invitat UBB, vineri 22 Dec. 2023		
S13	13. Evaluarea interogarilor in bazele de date relationale	7. Probleme	Examen practic
S14	14. Aplicatii		

Planul cursului

- Curs 11
  - Organizarea directa
  - Index pentru un atribut oarecare
  - Gestiunea indexurilor

Organizarea directă

C - **cheie** pentru o relaţie ***R***

Inregistrările relaţiei ***R*** se memorează într-un fişier (colecţie de date)

F={r_1, r_2, ..., r_n}, $K_i = \Pi_C(r_i)$), i=1, ..., n

Algoritm pentru determinarea răspunsului la interogarea: **C = K0**:

- căutare secvenţială
- consultarea unui index

Definim o funcţie:

h: {K_1, K_2, ..., K_n} → *A*

unde *A* = mulţimea adreselor de pe suport unde se poate memora ***F***

Semnificaţia: **h(K_i) = adresa unde se memorează înregistrarea r_i**, i=1, ..., n

Cerinţă:

h(K_i) ≠ h(K_j), pentru i ≠ j (deci **K_i ≠ K_j**)

(h să fie o funcţie injectivă) - greu de respectat pentru colecţii dinamice

Pentru eficienţă se permite existenţa unor **coliziuni**:

h(K_i) = h(K_j), i ≠ j

In acest caz înregistrările **r_i** şi **r_j** se numesc **sinonime,** iar valoarea **h(K_i)** este **adresa de început** a căutării (se foloseşte un algoritm de căutare în mulţimea de înregistrări sinonime).

Noţiuni:

1. **funcţie de dispersare**, sau **funcţie hashing**
2. **algoritm de dispersare**, sau **algoritm hashing**

Cerinţe pentru o **funcţie de dispersare**:

1. se calculează repede
2. minimizează numărul de coliziuni

Presupunem:

A = {0, 1, ..., m-1},

deci există **m** zone de memorie pentru a memora **n** înregistrări, **m>=n**

Pentru **h** este necesar ca: $0 \leq h(K_i) < m, i=1, ..., n$.

Tipuri de funcţii de dispersare:

- în expresia funcţiei să se folosească eventuale informaţii suplimentare privind distribuţia valorilor cheii
- **funcţii de dispersare bazate pe împărţire: restul împărţirii lui K la m**:

$$h(K) \equiv K \pmod m$$

Valorile vor fi în mulţimea A.

Rezultate bune (observate din teste) se obţin pentru **m = număr prim**.

- **funcţii de dispersare bazate pe înmulţire**:

$$h(K) = \left\lfloor m * \left\{ \frac{c}{w} * K \right\} \right\rfloor,$$

unde:

- w este valoarea maximă pentru zona de memorie unde se memorează valorile cheii
- c este o constantă primă cu w, deci $\frac{c}{w}$ este un număr subunitar (poate fi considerat ca "întregul" **c** reprezentat în zona de memorie cu virgula la stânga numărului)

  $\left\{ \frac{c}{w} * K \right\}$ este partea fracţionară a numărului $\frac{c}{w} * K$

Funcţiile de acest tip folosesc un **rezultat important din teoria numerelor**:

Fie x un număr iraţional. Dacă se reprezintă punctele {x}, {2x}, ..., {nx} pe segmentul [0,1], unde {y} este partea fracţionară a numărului y, atunci cele (n+1) segmente de dreaptă care rezultă au cel mult trei lungimi diferite. In plus, următorul punct {(n+1)x} va fi în unul din cele mai mari segmente existente.

Rezultate bune (produc cele mai uniforme segmente pe [0,1]) au fost obţinute pentru **x** egal cu:

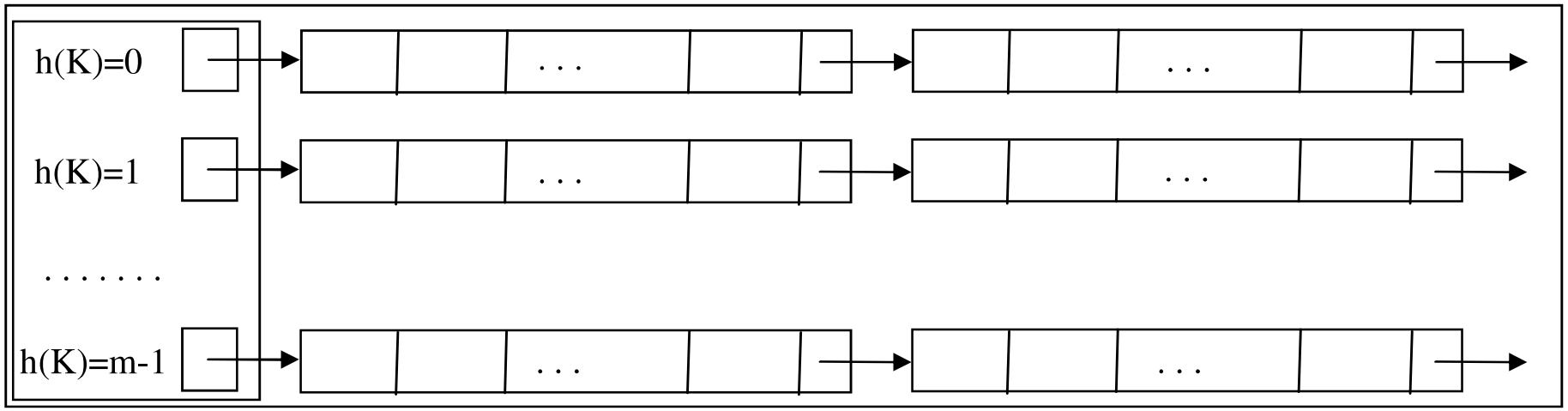
- $\theta = \frac{\sqrt{5}-1}{2} = 0.61803...$ (proporţia de aur)
- $1 - \theta = 0.38196...$

Rezolvarea coliziunilor

La fiecare metodă folosită pentru memorarea înregistrărilor, inclusiv la **organizarea directă**, sunt necesari algoritmi pentru: **adăugarea, căutarea şi ştergerea unei înregistrări**.

1. **Liste independente**: datele sinonime se păstrează într-o listă înlănţuită. Pentru optimizare se poate construi o lista de blocuri (într-un bloc se memorează mai multe înregistrări transferate împreună în memoria internă). In cadrul listei, înregistrările sinonime se **pot memora** în ordinea descrescătoare a frecvenţelor de căutare.

Hashing static



Observaţie. Dacă înregistrările sinonime nu se pot memora într-un singur bloc, atunci sunt necesare citiri multiple de blocuri.

Funcţie hashing bună: puţine liste vide, puţine înregistrări sinonime.

Hashing extensibil (dinamic)

Caracteristici:

1. Se construieşte un index cu adresele spre blocurile din zona de memorie.
2. Numărul de înregistrări sinonime nu poate depăşi dimensiunea unui bloc.

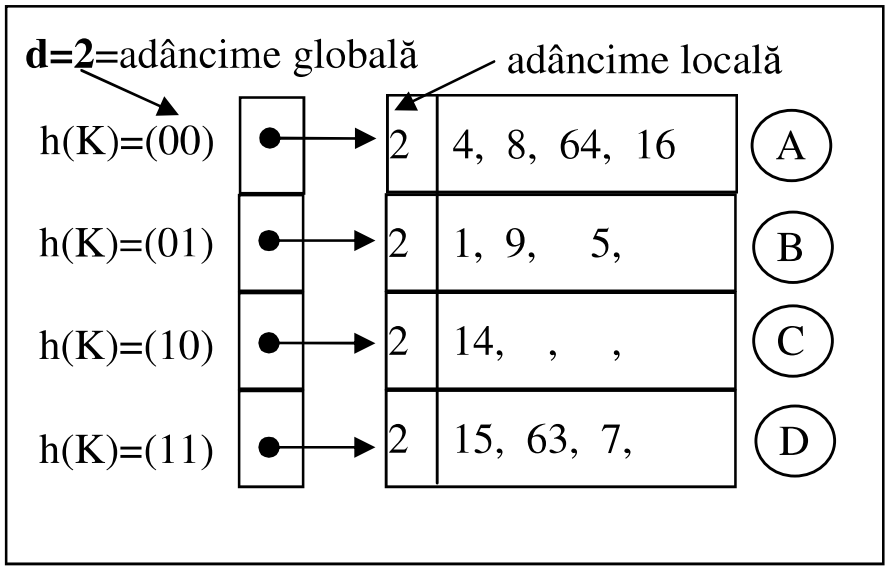
Fie:

$$h(K) = (....a_2 a_1 a_0)_2$$

reprezentarea adresei (pentru înregistrarea care are o valoare K pentru cheie) **în baza 2.**

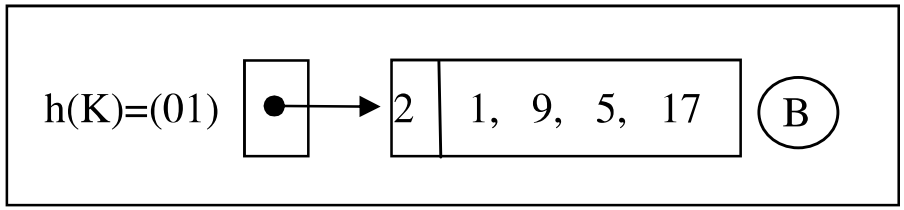
Din **h(K)** se pot lua numai ultimele poziţii. Fie **d** nr. de poziţii, care dau o **adresă în index.** Această valoare este **adâncimea globală** a indexului. Fiecare bloc va avea o **adâncime locală**, precizată în antetul blocului (iniţial este adâncimea globală).

In exemplul următor vom presupune că **h(K)=K**, iar **K** are valori intregi. Intr-un bloc se pot memora patru înregistrări (în figură apare numai valoarea pentru cheie).



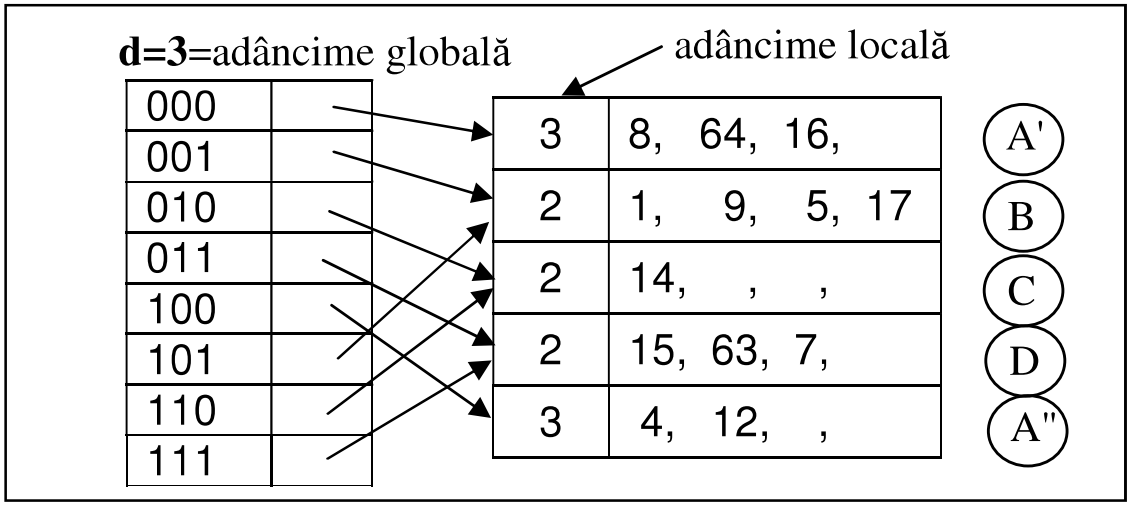
La adăugarea unei noi înregistrări se determină valoarea funcţiei **h** pentru cheie şi rezultă o adresă de bloc. Apar două situaţii:

a. In bloc **există spaţiu disponibil**, deci noua valoare se poate adăuga, de ex**. 17** în blocul B:



b. In bloc nu există spaţiu **disponibil**, de ex**.** se adaugă **12** (în blocul A).

b1. **Adâncimea globală = adâncimea locală a blocului**. In acest caz blocul se divide în două blocuri (se adaugă un bloc nou). Redistribuirea datelor între aceste blocuri se face luând încă **o poziţie binară pentru adâncimea globală** (pe exemplu ⇒ d=3) şi **indexul se dublează.**



Pentru blocul A, care se divide, se iau trei poziţii binare în adresă, iar **d global** este egal cu **d local** pentru aceste două blocuri.

b2. **Adâncimea globală > adâncimea locală a blocului**: blocul curent se divide, adâncimea globală nu se schimbă, în index se modifică o singură adresă (pentru blocul nou apărut).

Dacă, de ex., se adaugă **21**, atunci blocul B se divide, dar nu se dublează indexul.

Căutarea unei înregistrări precizată prin valoarea K0 a cheii:

- se evaluează h(K0)
- se citeşte şi consultă blocul de la adresa calculată

La **ştergerea** unei înregistrări:

- se determină blocul care conţine înregistrarea, iar această înregistrare se elimină din bloc
- dacă blocul devine vid, atunci se unesc două blocuri şi adâncimea locală pentru noul bloc se micşorează**.** Dacă adâncimea locală **<** adâncimea globală pentru fiecare bloc, atunci se micşorează adâncimea globală şi se înjumătăţeşte indexul.

Utilizarea unei funcţii pentru crearea unui **index hash:** în blocurile din index se memorează adresele la înregistrări.

2. **Liste interpătrunse.** Zona de memorie pentru fişier are structura:

K	**INF**	**LEG**
r		

La **adăugare**, dacă zona de la adresa h(K) (rezultată prin calcul) este ocupată deja de altă înregistrare, atunci se parcurge lista înlănţuită ce începe la această adresă (folosind zona **LEG**). Dacă valoarea căutată nu e în listă, ea se adaugă într-o zonă liberă şi se înserează în lista înlănţuită parcursă.

Exemplu pentru: **h(K) ≡ K (mod 13)** şi colecţia următoare de înregistrări:

K	16	23	36	25	19	32	29	49	22
h(K)	3	10	10	12	6	6	3	10	9

Adăugarea înregistrărilor, în ordinea în care sunt precizate în tabel, este dată în tabelul următor (se precizează numai valoarea pentru cheie). Spaţiul liber se caută începând cu o anumită adresă (de ex.iniţial adresa maximă, după folosirea valorii aceasta se micşorează cu 1).

	0	1	2	3	4	5	6	7	8	9	10	11	12
K				16		22	19	49	29	32	23	25	36
LEG				8			9			5	12	7	11

Căutarea unei valori foloseşte metoda descrisă la adăugare.

Stergerea unei înregistrări se face prin căutarea ei în lista înlănţuită ce începe la adresa furnizată de funcţia de hashing. Dacă se găseşte, atunci poziţia care se eliberează este posibil să fie ocupată de o altă înregistrare din lista înlănţuită ce începe la această adresă. Exemplu la ştergerea valorii 36 urmată imediat de căutarea valorii 25.

3. **Fără înlănţuire** (adresare deschisă).

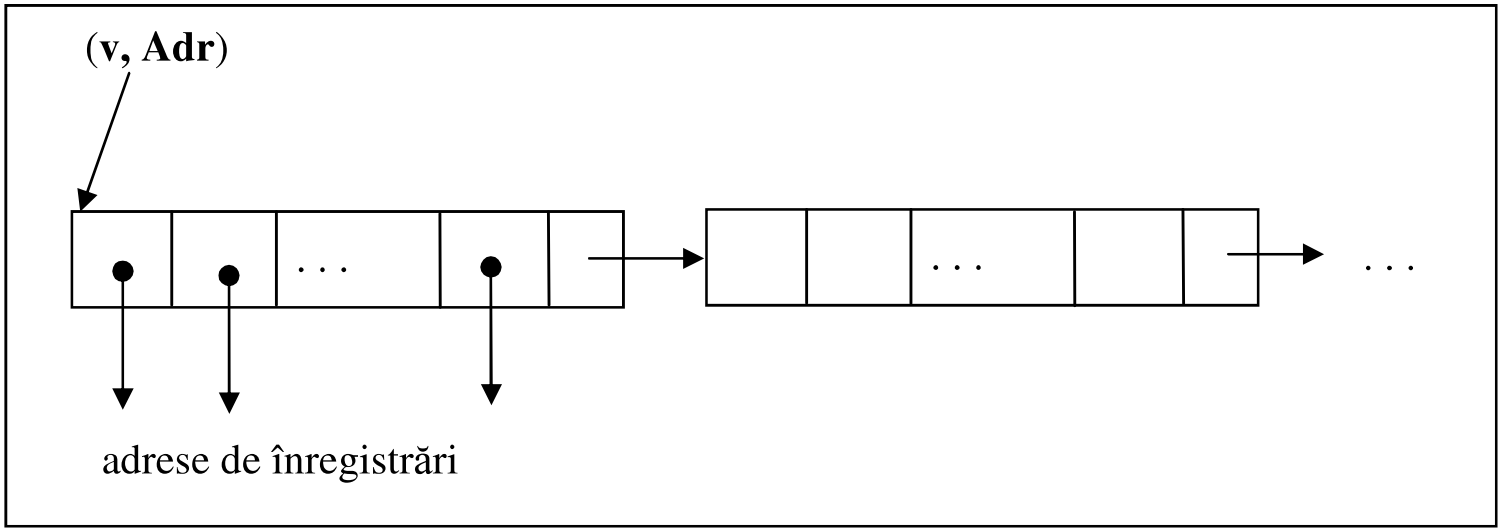
Zona dorită (la căutare, adăugare) se obţine parcurgând zonele de la adresele: h(K), h(K)+1, ..., h(K)-1. Exemplu pentru colecţia cu valorile pentru cheie din mulţimea {5, 21, 24, 22, 23, 34, 35} şi h(K) $\equiv$ K (mod 13).

Index pentru un atribut oarecare

1. **Transformarea modului de construire a unui index pentru o cheie**

a. **Index memorat ca un B-arbore**

- Fiecare valoare **v** pentru **cheie** apare în B-arbore sub forma **(v, Adr)**, unde **Adr** este adresa (unică) a înregistrării cu valoarea **v** pentru cheie
- Pentru un **atribut oarecare pot exista mai multe înregistrări cu aceeaşi valoare v**, care se pot păstra în blocuri separate, deci la **(v, Adr)** zona **Adr** este **adresa unei înregistrări** din fişier sau **adresa unui bloc din B-arbore** unde se memorează aceste înregistrări. Dacă înregistrările nu se pot memora într-un singur bloc, atunci blocurile necesare se pot înlănţui.



b. Index memorat ca un B^+-arbore

- pentru o **cheie** fiecare valoare **v** apare într-un nod terminal sub forma **(v, Adr)**. Nodurile interne se folosesc numai pentru "dirijarea" căutării (unele valori din nodurile terminale se regăsesc în nodurile interioare).
- pentru un **atribut** trebuie memorată o mulţime de adrese.
  - se poate utiliza o listă de blocuri cu astfel de adrese (soluţia amintită la B-arbore)
  - valoarea **v** se repetă pentru fiecare înregistrare. Pot apare blocuri care continuă ultima valoare din blocul anterior

Exemplu pentru un B^+-arbore de ordin 3 la care se adaugă înregistrări la care valoarea unui atribut (pentru care se construieşte acest index) sunt {5, 10, 20, 100, 25, 3, 200, 1, 3, 15, 13, 20, 20, 20}.

2. Construirea unei structuri de fişier multilistă sau fişier inversat.

Pentru **(A, v_i)**, unde **v_i** este o valoare pentru **atributul A**, există o mulţime **M_i** de înregistrări posibile. După modul de memorare a mulţimii **M_i** se obţin cele două tipuri de structuri.

a. Mulţimea **M_i** se memorează ca o **listă înlănţuită**, adresa de început a listei se păstrează în index, iar legăturile se memorează în zona fişierului şi este asociată la valoarea atributului. Fiecare înregistrare va apare într-o singură listă construită pentru A (are o singură valoare pentru A).

Exemplu.

b. Mulţimea **M_i** se memorează în una din variantele:

- **listă de adrese** (ca indexul unei cărţi, unde la anumite cuvinte cheie se precizează paginile unde apar aceste cuvinte)
- **complementara** listei de adrese
- un **vector "caracteristic" cu valori binare**, câte o poziţie pentru fiecare înregistrare. Acest mod de memorare este folosit de **indecşi bitmap**.

Exemplu.

Situaţii în care organizarea inversată este avantajoasă.

Determinarea răspunsului la o interogare precizată în **forma normală conjuctivă**:

`(anstudiu=2 and media ≥ 9)` **`or`** `(sectia="I" and media>8 and bursa ≠ 'M')`

c. Combinaţie între cele două variante: mulţimea **M_i** se divide în mai multe subliste. Cu varianta **b** se precizează adresele de început, iar elementele dintr-o sublistă se păstrează înlănţuit.

Gestiunea indexurilor

1. Tipuri de indexuri:

După **expresia de index**

- cu o singură coloană
- cu două sau mai multe coloane (valorile din index sunt concatenări de valori ale atributelor pentru care se construieşte indexul)
- cu o expresie ce folosesc valorile coloanelor

După **implementarea fizică**:

- B-arbore (sau B^+-arbore)
- bitmap

După **tipul de valori**:

- valori unice
- valori care se pot repeta

2. Modul de construire

- implicit - la definirea unei restricţii de cheie unică
- prin instrucţiune (depinde de SGBD):

$$\text{CREATE INDEX}\left[\left\{\begin{matrix}\text{UNIQUE}\\ \text{BITMAP}\end{matrix}\right\}\right]\text{nume\_index ON nume\_tabel}\left(\left\{\begin{matrix}\text{coloana [, coloana]...}\\ \text{expresie}\end{matrix}\right\}\right)\ \text{[optiuni]}$$

Exemple:

```
CREATE UNIQUE INDEX i1 ON studenti(cnp);
CREATE UNIQUE INDEX i2 ON studenti(sectia,nrmatricol);
CREATE INDEX i3 ON studenti(upper(nume));
CREATE BITMAP INDEX i4 ON studenti(sectia);
```

3. Utilizare

- la operaţii de modificare a colecţiei de date (adăugare, ştergere, modificare de înregistrări) pentru verificarea restricţiilor
- la căutarea unei valori a cheii sau a tributului folosit la crearea indexului
- la filtrarea tabelelor
- la evaluarea operaţiilor de join
- la sortări

4. Costul gestiunii unui index

- spaţiu suplimentar de memorie
- timp suplimentar folosit pentru gestiunea indexului la modificarea sursei de date

Referinţe:

[Si10] SILBERSCHATZ A., KORTZ H., SUDARSHAN S., *Database System Concepts*, McGraw-Hill, 2010, cap. 10, 11.

[Ra07] RAMAKRISHNAN, R., *Database Management Systems*, McGraw-Hill, 2007, cap. 7, 8, 9, 10.

[Kn76] KNUTH, D.E., *Tratat de programare a calculatoarelor. Sortare şi căutare*. Ed.Tehnica, Bucuresti 1976.

[Ga08] GARCIA-MOLINA, H., ULLMAN, J., WIDOM, J., *Database Systems: The Complete Book,* Pearson Prentice Hall, 2008, cap.12, 13, 14