

# Structura fizica a bazelor de date relationale (partea 3)

Ioana Ciuciu

[oana@cs.ubbcluj.ro](mailto:oana@cs.ubbcluj.ro)

<http://www.cs.ubbcluj.ro/~oana/>

# Planificare

Saptamana	Curs	Seminar	Laborator
S1	1. Concepte fundamentale ale bazelor de date	1. Modelul Entitate-Relatie. Modelul relational	1. Modelarea unei BD in modelul ER si implementarea ei in SQL Server
S2	2. Modelare conceptuala		
S3	3. Modelul relational de organizare a bazelor de date	2. Limbajul SQL – definirea si actualizarea datelor	2. Interogari SQL
S4	4. Gestiunea bazelor de date relationale cu limbajul SQL		
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	
S8	8. Structura fizica a bazelor de date relationale		
S9-10-11	9-10-11. Indeksi. Arbori B. Fisiere cu acces direct	5. View-uri. Functii definite de utilizator. Executie dinamica	4. Proceduri stocate. View. Trigger
S12	12. Evaluarea interogarilor in bazele de date relationale	6. Formele normale ale unei relatii. Indeksi	
S13	13. Extensii ale modelului relational si baze de date NoSQL	7. Probleme	Examen practic
S14	14. Aplicatii		

# Planul cursului

---

- ▶ Curs 10
  - ▶ Organizarea directa
  - ▶ Index pentru un atribut oarecare
  - ▶ Gestiunea indexurilor



## Organizarea directă

**C** - cheie pentru o relație **R**

Înregistrările relației **R** se memorează într-un fișier (colecție de date)

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

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

- căutare secvențială
- consultarea unui index

Definim o funcție:

$$h: \{K_1, K_2, \dots, K_n\} \rightarrow A$$

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

Semnificația:  **$h(K_i)$  = adresa unde se memorează înregistrarea  $r_i$ ,  $i=1, \dots, n$**

**Cerință:**

$$h(K_i) \neq h(K_j), \text{ pentru } i \neq j \text{ (deci } K_i \neq K_j)$$

( $h$  să fie o funcție injectivă) - greu de respectat pentru colecții dinamice

**Pentru eficiență se permite existența unor coliziuni:**

$$h(K_i) = h(K_j), i \neq j$$

În acest caz înregistrările  $r_i$  și  $r_j$  se numesc **sinonime**, iar valoarea  **$h(K_i)$**  este **adresa de început** a căutării (se folosește un algoritm de căutare în mulțimea de înregistrări sinonime).

## Noțiuni:

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

## Cerințe pentru o funcție de dispersare:

1. se calculează repede
2. minimizează numărul de coliziuni

Presupunem:

$$A = \{0, 1, \dots, m-1\},$$

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

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

## Tipuri de funcții de dispersare:

- în expresia funcției să se folosească eventuale informații suplimentare privind distribuția valorilor cheii
- **funcții de dispersare bazate pe împărțire: restul împărțirii lui K la m:**

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

Valorile vor fi în mulțimea A.

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

- **funcții de dispersare bazate pe înmulțire:**

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

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 fracționară a numărului  $\frac{c}{w} * K$

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

***Fie  $x$  un număr irațional. Dacă se reprezintă punctele  $\{x\}, \{2x\}, \dots, \{nx\}$  pe segmentul  $[0,1]$ , unde  $\{y\}$  este partea fracționară a numărului  $y$ , atunci cele  $(n+1)$  segmente de dreaptă care rezultă au cel mult trei lungimi diferite. În 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 obținute pentru  $x$  egal cu:

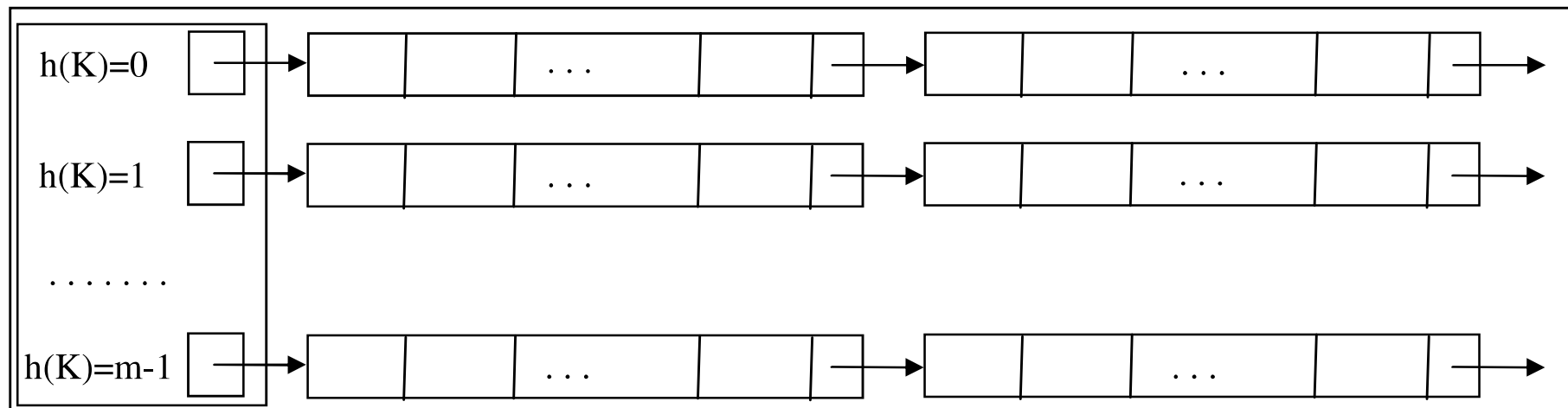
- $\theta = \frac{\sqrt{5}-1}{2} = 0.61803\dots$  (proporția de aur)
- $1 - \theta = 0.38196\dots$

## Rezolvarea coliziunilor

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

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

### Hashing static



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

Funcție hashing bună: puține liste vide, puține înregistrări sinonime.

## Hashing extensibil (dinamic)

Caracteristici:

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

Fie:

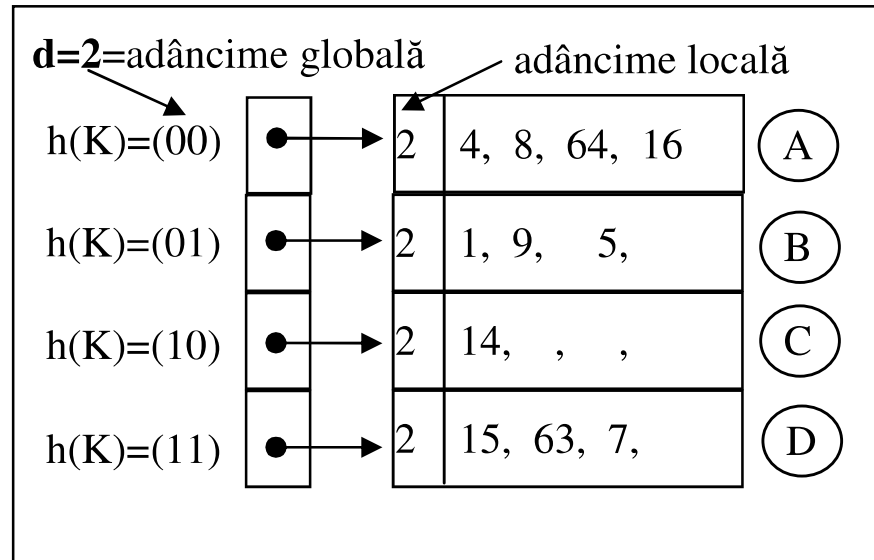
$$h(K) = (\dots a_2 a_1 a_0)_2$$

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

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

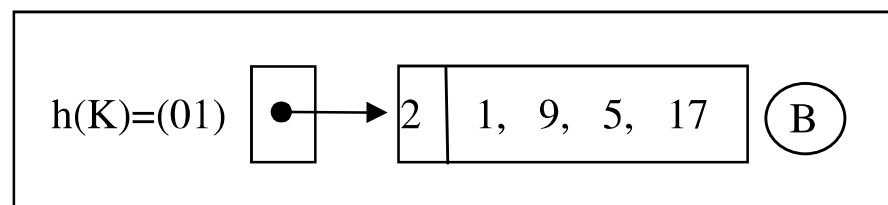
În exemplul următor vom presupune că  **$h(K)=K$** , iar **K** are valori întregi. Într-un bloc se pot memora patru înregistrări (în figură apare numai valoarea pentru cheie).





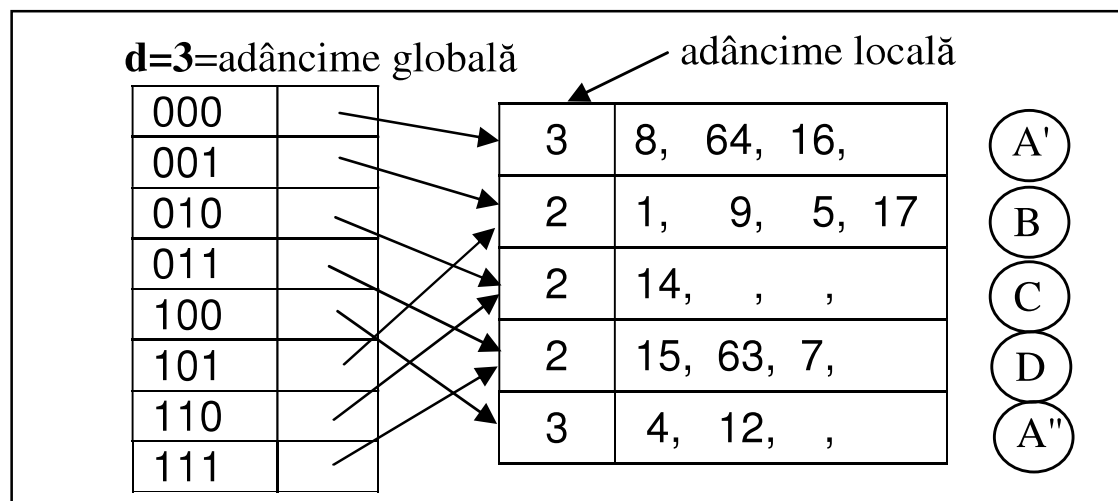
La adăugarea unei noi înregistrări se determină valoarea funcției **h** pentru cheie și rezultă o adresă de bloc. Apar două situații:

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



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

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



Pentru blocul A, care se divide, se iau trei poziții 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 citește și consultă blocul de la adresa calculată

La **ștergerea** unei înregistrări:

- se determină blocul care conține înregistrarea, iar această înregistrare se elimină din bloc
- dacă blocul devine vid, atunci se unesc două blocuri și adâncimea locală pentru noul bloc se micșorează. Dacă adâncimea locală  $<$  adâncimea globală pentru fiecare bloc, atunci se micșorează adâncimea globală și se înjumătățește indexul.

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

2. **Liste interpătrunse.** Zona de memorie pentru fișier 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 înlănțuită ce începe la această adresă (folosind zona **LEG**). Dacă valoarea căutată nu e în listă, ea se adaugă într-o zonă liberă și se înserează în lista înlănțuită parcursă.

**Exemplu** pentru:  $h(K) \equiv K \pmod{13}$  și colecția următoare de înregistrări:

<b>K</b>	16	23	36	25	19	32	29	49	22
<b>h(K)</b>	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). Spațiul liber se caută începând cu o anumită adresă (de ex. inițial adresa maximă, după folosirea valorii aceasta se micșorează cu 1).

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

**Căutarea** unei valori folosește metoda descrisă la adăugare.

**Stergerea** unei înregistrări se face prin căutarea ei în lista înlănțuită ce începe la adresa furnizată de funcția de hashing. Dacă se găsește, atunci poziția care se eliberează este posibil să fie ocupată de o altă înregistrare din lista înlănțuită ce începe la această adresă. Exemplu la ștergerea valorii 36 urmată imediat de căutarea valorii 25.

### 3. **Fără înlănțuire** (adresare deschisă).

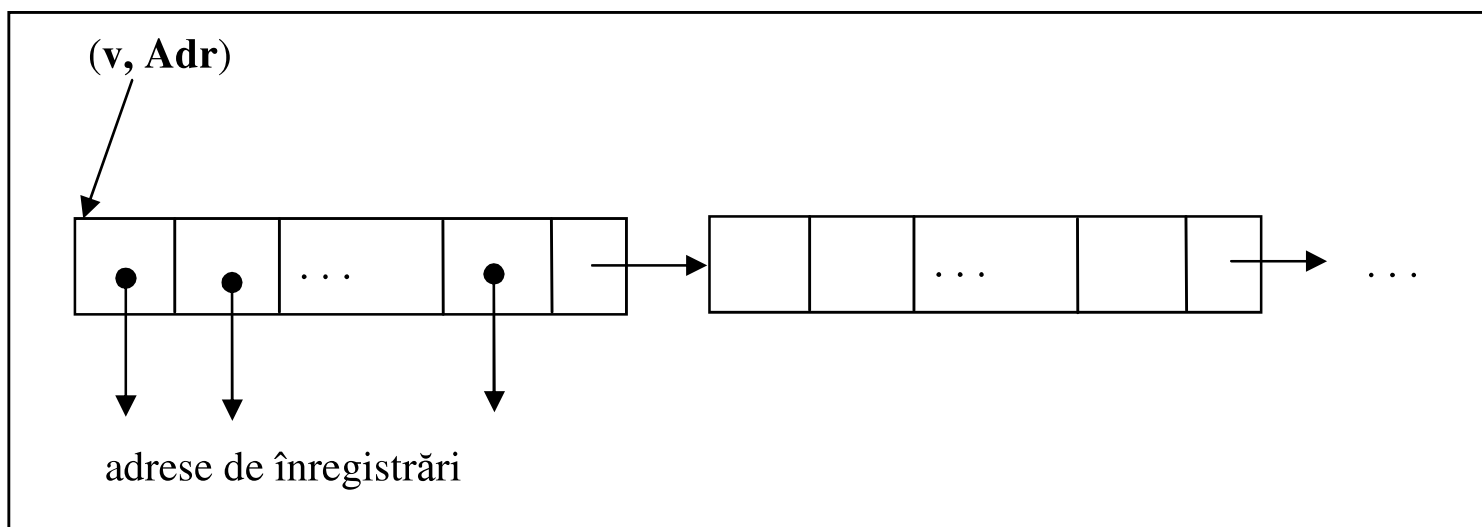
Zona dorită (la căutare, adăugare) se obține parcurgând zonele de la adresele:  $h(K)$ ,  $h(K)+1$ , ...,  $h(K)-1$ . Exemplu pentru colecția cu valorile pentru cheie din mulțimea {5, 21, 24, 22, 23, 34, 35} și  $h(K) \equiv K \pmod{13}$ .

# 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 aceeași valoare **v**, care se pot păstra în blocuri separate, deci la **(v, Adr)** zona **Adr** este **adresa unei înregistrări** din fișier 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 înlanțui.



## b. Index memorat ca un B<sup>+</sup>-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 mulțime de adrese.
  - se poate utiliza o listă de blocuri cu astfel de adrese (soluția 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<sup>+</sup>-arbore de ordin 3 la care se adaugă înregistrări la care valoarea unui atribut (pentru care se construiește acest index) sunt {5, 10, 20, 100, 25, 3, 200, 1, 3, 15, 13, 20, 20, 20}.

## 2. Construirea unei structuri de fișier multilistă sau fișier inversat.

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

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

### Exemplu.

**b.** Mulțimea  $M_i$  se memorează în una din variantele:

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

### Exemplu.

**Situații** în care organizarea inversată este avantajoasă.

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

(anstudiu=2 and media  $\geq$  9) **or** (sectia="I" and media>8 and bursa  $\neq$  'M')

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



# 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 construiește indexul)
- cu o expresie ce folosesc valorile coloanelor

După **implementarea fizică:**

- B-arbore (sau B<sup>+</sup>-arbore)
- bitmap

După **tipul de valori:**

- valori unice
- valori care se pot repeta

## ***2. Modul de construire***

- implicit - la definirea unei restricții de cheie unică
- prin instrucțiune (depinde de SGBD):

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

## **Exemple:**

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

## **3. Utilizare**

- la operații de modificare a colecției de date (adăugare, ștergere, modificare de înregistrări) pentru verificarea restricțiilor
- la căutarea unei valori a cheii sau a tributului folosit la crearea indexului
- la filtrarea tabelor
- la evaluarea operațiilor de join
- la sortări

## **4. Costul gestiunii unui index**

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

**Referințe:**

- [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 și 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