Chapter 14:Indexing

Dr. Abu Raihan Mostofa Kamal

November 22, 2022

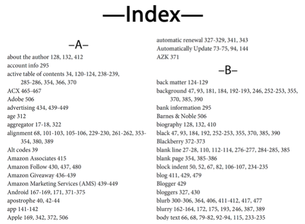
Table of Contents

1. Indexing: Badic Concept
2. Index Types: Primary and Secondary Index
3. Index Types: Dense and Sparse Index

Indexing: Basic Concepts

Indexing

An index for a file in a database system works in much the same way as the index in this textbook. If we want to learn about a particular topic (specified by a word or a phrase) in this textbook, we can search for the topic in the index at the back of the book, find the pages where it occurs, and then read the pages to find the information for which we are looking.



—Index—

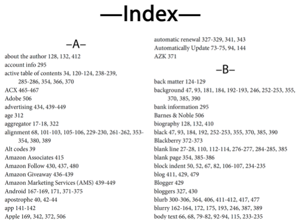
–A–

about the author 128, 132, 412
account info 295
active table of contents 34, 120-124, 238-239, 285-286, 354, 366, 370
ACX 465-467
Adobe 506
advertising 434, 439-449
age 312
aggregator 17-18, 322
alignment 68, 101-103, 105-106, 229-230, 261-262, 353-354, 380, 389
Alt codes 39
Amazon Associates 415
Amazon Follow 430, 437, 480
Amazon Giveaway 436-439
Amazon Marketing Services (AMS) 439-449
Android 167-169, 171, 371-375
apostrophe 40, 42-44
app 141-142
Apple 169, 342, 372, 506
automatic renewal 327-329, 341, 343
Automatically Update 73-75, 94, 144
AZK 371

–B–

back matter 124-129
background 47, 93, 181, 184, 192-193, 246, 252-253, 355, 370, 385, 390
bank information 295
Barnes & Noble 506
biography 128, 132, 410
black 47, 93, 184, 192, 252-253, 355, 370, 385, 390
Blackberry 372-373
blank line 27-28, 110, 112-114, 276-277, 284-285, 385
blank page 354, 385-386
block indent 50, 52, 67, 82, 106-107, 234-235
blog 411, 429, 479
Blogger 429
bloggers 327, 430
blurb 300-306, 364, 406, 411-412, 417, 477
blurry 162-164, 172, 175, 193, 246, 387, 389
body text 66, 68, 79-82, 92-94, 115, 233-235

Indexing: Basic Purpose



—Index—

–A–

about the author 128, 132, 412
account info 295
active table of contents 34, 120-124, 238-239, 285-286, 354, 366, 370
ACX 465-467
Adobe 506
advertising 434, 439-449
age 312
aggregator 17-18, 322
alignment 68, 101-103, 105-106, 229-230, 261-262, 353-354, 380, 389
Alt codes 39
Amazon Associates 415
Amazon Follow 430, 437, 480
Amazon Giveaway 436-439
Amazon Marketing Services (AMS) 439-449
Android 167-169, 171, 371-375
apostrophe 40, 42-44
app 141-142
Apple 169, 342, 372, 506
automatic renewal 327-329, 341, 343
Automatically Update 73-75, 94, 144
AZK 371

–B–

back matter 124-129
background 47, 93, 181, 184, 192-193, 246, 252-253, 355, 370, 385, 390
bank information 295
Barnes & Noble 506
biography 128, 132, 410
black 47, 93, 184, 192, 252-253, 355, 370, 385, 390
Blackberry 372-373
blank line 27-28, 110, 112-114, 276-277, 284-285, 385
blank page 354, 385-386
block indent 50, 52, 67, 82, 106-107, 234-235
blog 411, 429, 479
Blogger 429
bloggers 327, 430
blurb 300-306, 364, 406, 411-412, 417, 477
blurry 162-164, 172, 175, 193, 246, 387, 389
body text 66, 68, 79-82, 92-94, 115, 233-235

- **Fast and efficient** processing of queries in databases.
- These are sophisticated techniques that **reduce the overhead of reading the entire contents** of the database. (Typically records of database are enormous in number)

Indexing: Implementation Complexity

An Example

Suppose we need to search a students information based on its ID. Now one simple solution is to sort them based on the search key (i.e. ID).

But it may not work in reality, because:

1. The index would itself be **very large** (hard to fit in main momory for millions of records)
2. Even though keeping the index sorted reduces the search time, finding a student can still be rather **time-consuming**.
3. Updating a sorted list whenever students are **added or removed** from the database can be **very expensive**.

Hence, more **sophisticated indexing techniques** are used in database systems.

Indexing: Broad Classification

There are two basic kinds of indices:

1. **Ordered indices**. Based on a sorted ordering of the values.
2. **Hash indices**. Based on a uniform distribution of values across a range of **buckets**. The bucket to which a value is assigned is determined by a function, called a **hash function**.

Note: Concentration will be on ordered indexing.

Indexing Technique: Which one to use?

No one technique is the best. Rather, each technique is best suited to particular database applications. Each technique must be evaluated on the basis of these factors:

1. **Access types:** The types of access that are supported efficiently. Access types can include finding records with a **specified attribute value** and finding records whose attribute values fall in **a specified range**.
2. **Access time:** The time it takes to find a particular data item, or set of items, using the technique in question.
3. **Insertion and Deletion time:** The time it takes to insert a new data item or to remove a data. This value includes the time it takes to update the index structure.
4. **Space overhead:** The additional space occupied by an index structure. Provided that the amount of additional space is moderate, it is usually worthwhile to sacrifice the space to achieve improved performance.

Search Key

Search Criteria

An attribute or set of attributes used to look up records in a file is called a search key. Note that this definition of key **differs** from that used in primary key, candidate key, and superkey. Multiple search keys are common in use.

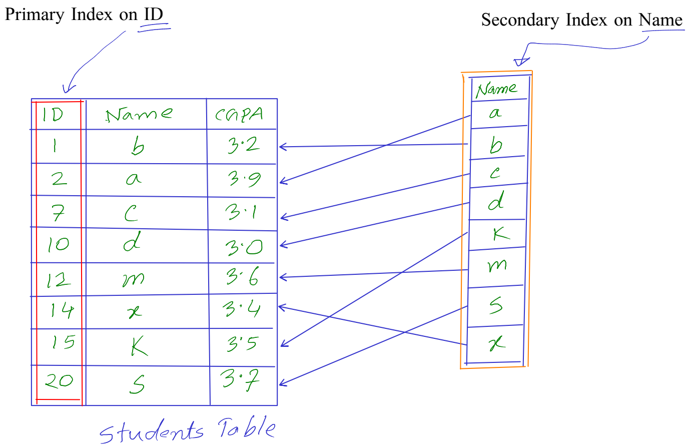
Ordered Index: Types

Ordered index can be of 2 types:

1. **Clustering Index:** Cluster index is a type of index which sorts the data rows in the table on their key values. In the Database, there is only one clustered index per table. A clustered index defines the order in which data is stored in the table which can be sorted in only one way. So, there can be an only a single clustered index for every table. In an RDBMS, usually, the primary key allows you to create a clustered index based on that specific column.This also called **Primary Index**.
2. **Nonclustering Indices:** A Non-clustered index stores the data at one location and indices at another location. The index contains pointers to the location of that data. A single table can have many non-clustered indexes as an index in the non-clustered index is stored in different places. Called **Secondary Index**.

(Definitions have been adopted from www.guru99.com)

Primary and Secondary Index: Concepts



Primary Index on ID

Secondary Index on Name

ID	Name	CGPA
1	b	3·2
2	a	3·9
7	c	3·1
10	d	3·0
12	m	3·6
14	x	3·4
15	K	3·5
20	S	3·7

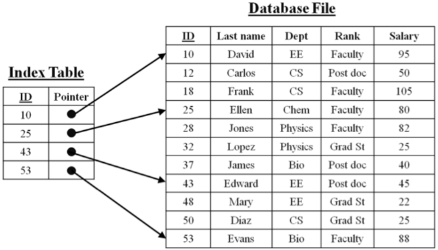
Students Table

Name
a
b
c
d
K
m
S
x

Dense and Sparse Index

Index Entry

An index entry, or index record, consists of a **search-key value** and **pointers** to one or more records with that value as their search-key value. The pointer to a record consists of the identifier of a disk block and an offset within the disk block to identify the record within the block.



Index Table

ID	Pointer
10	●
25	●
43	●
53	●

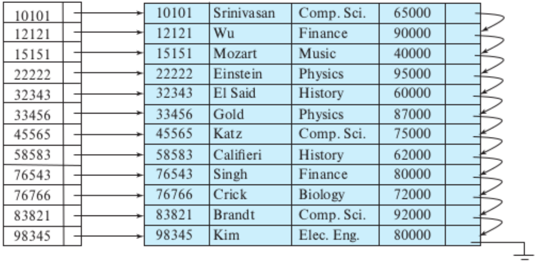
Database File

ID	Last name	Dept	Rank	Salary
10	David	EE	Faculty	95
12	Carlos	CS	Post doc	50
18	Frank	CS	Faculty	105
25	Ellen	Chem	Faculty	80
28	Jones	Physics	Faculty	82
32	Lopez	Physics	Grad St	25
37	James	Bio	Post doc	40
43	Edward	EE	Post doc	45
48	Mary	EE	Grad St	22
50	Diaz	CS	Grad St	25
53	Evans	Bio	Faculty	88

Dense Index

Dense Index: Definition

In a dense index, **an index entry** appears **for every search-key value** in the file. In a dense clustering index, the index record contains the search-key value and a pointer to the **first data record** with that search-key value.



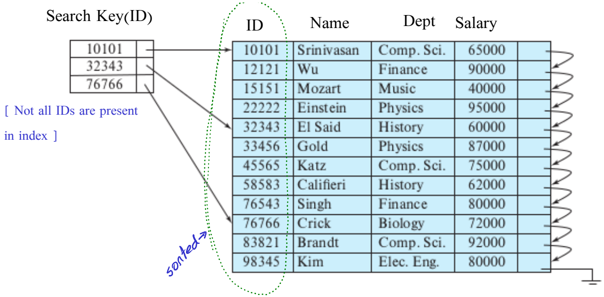
10101
12121
15151
22222
32343
33456
45565
58583
76543
76766
83821
98345

10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Sparse Index

Sparse Index: Definition

In a sparse index, an index entry appears for **only some of the search- key values**. Sparse indices can be used only if the relation is stored in **sorted order** of the search key; that is, if the index is a clustering index.



Search Key(ID)

10101
32343
76766

[Not all IDs are present in index]

sorted→

ID	Name	Dept	Salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Dense and Sparse Index: Pros and Cons

Dense Index:

- It is generally **faster** to locate a record if we have a dense index rather than a sparse index

Sparse Index:

- Sparse indices have advantages over dense indices in that they **require less space** and they impose **less maintenance overhead** for insertions and deletions.

Access Time and Space Overhead: A trade-off

- There is a trade-off that the system designer must make between access time and space overhead.
- Trade-off depends on the specific application
- The **dominant** cost in processing a database request is the time that it takes to **bring a block from disk into main memory**.
- Once we have brought in the block, the time to scan the entire block is negligible.

Multilevel Indices

Why?

Large indices are stored as sequential files on disk because of its size (can not be directly loaded into Main Memory).

For instance, Suppose we build a dense index on a relation with 1,000,000 tuples. Index entries are smaller than data records, so let us assume that 100 index entries fit on a 4-kilobyte block. Thus, our index occupies 10,000 blocks. If the relation instead had 100,000,000 tuples, the index would instead occupy 1,000,000 blocks, or 4 gigabytes of space.(Main Memory has other functions as well)

Multilevel Indices (Cont.)

- If an index is **small** enough to be **kept entirely in main memory**, the search time to find an entry is low.
- However, for larger size index, the search for an entry in the index then requires **several disk-block reads**.
- Binary search can be used on the index file to locate an entry, but the search still has a large cost. If the index would occupy **b** blocks, binary search requires as many as $\lceil log_2 b \rceil$ blocks to be read. Even this is quite high for a very large database.
- To deal with this problem, we treat the index just as we would treat any other sequential file, and we construct a **sparse outer index** on the original index, which we now call the inner index.

Multilevel Indices (Cont.): A simple Analogy

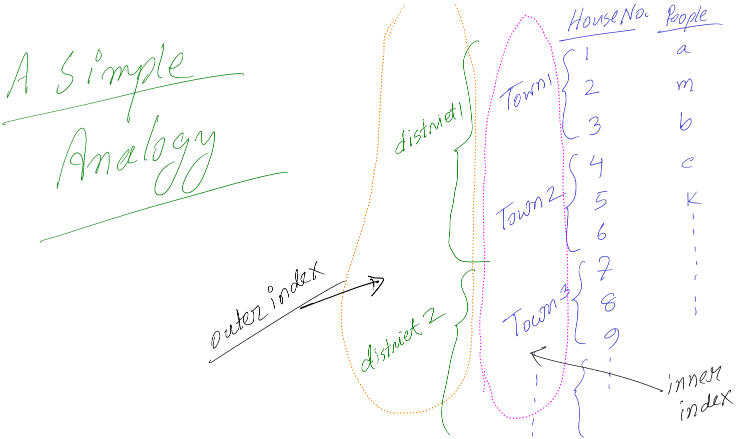


Figure: Multi-level index: Analogy

Multilevel Indices (Cont.):

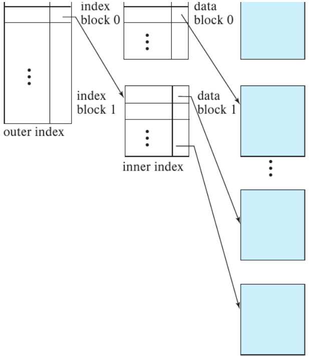


Figure: Multi-level index: Basic Idea

B+ Tree

To cover this topic we need to recap few concepts:

- Disk Structure
- How data is stored on disk
- Concept of indexing
- Multi-level Indexing (already done)
- m-way search trees
- B Trees
- B+ Trees

Bitmap Indices

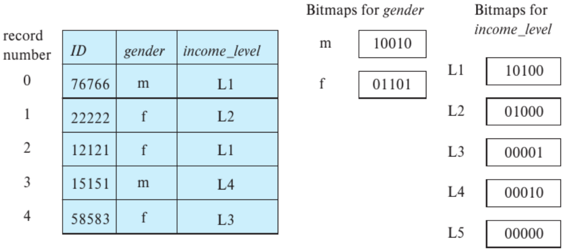
Defintion

A **bitmap** is simply an array of bits. In its simplest form, a bitmap index on the attribute A of relation r consists of one bitmap for each value that A can take. Each bitmap has **as many bits as the number of records** in the relation.

Where to use?

It has an efficient implementation when total number of distinct values of the selected attribute is limited. Example: Gender, Blood Group, or any customized class. Class must be non-overlapping.

Bitmap Indices: General Idea



record number	*ID*	*gender*	*income_level*
0	76766	m	L1
1	22222	f	L2
2	12121	f	L1
3	15151	m	L4
4	58583	f	L3

Bitmaps for *gender*

m	10010
f	01101

Bitmaps for *income_level*

L1	10100
L2	01000
L3	00001
L4	00010
L5	00000

Now consider the following SQL:

select ID
from citizens
where gender='m' and income=L1;

Now we can readily get the result (i.e. index of record) by and-ing the vectors. You will special benefit when you need count(*) function. No need to look for the records even, only bit vectors can compute the result.