

COURSE NAME: DBMS

COURSE CODE:23AD2102R

TOPIC:

INDEX STRUCTURES

What is Indexing ?

INDEXING is a data structure technique which allows you to quickly retrieve records from a database file.

Indexes are used to quickly locate data without having to search every record in multiple disk blocks

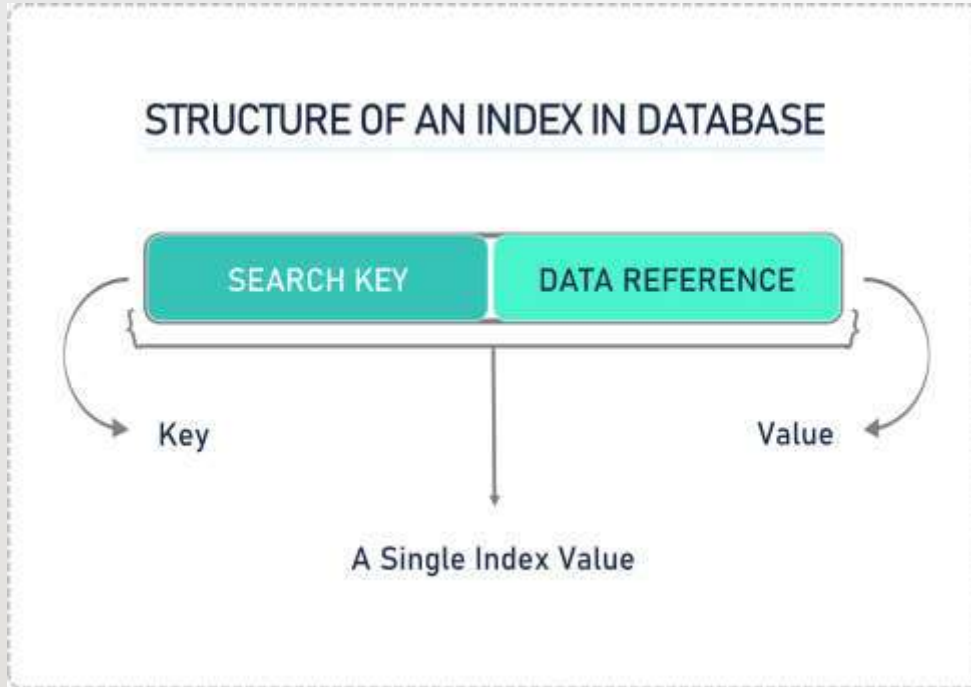
INDEX	
ABC, 164, 321 <i>n</i>	Anello, Douglas, 60
academic journals, 262, 280–82	animated cartoons, 21–24
Adobe eBook Reader, 148–53	antiretroviral drugs, 257–61
advertising, 36, 45–46, 127, 145–46, 167–68, 321 <i>n</i>	Apple Corporation, 203, 264, 302
Africa, medications for HIV patients in, 257–61	architecture, constraint effected through, 122, 123, 124, 318 <i>n</i>
Agee, Michael, 223–24, 225	archive.org, 112
agricultural patents, 313 <i>n</i>	<i>see also</i> Internet Archive
Aibo robotic dog, 153–55, 156, 157, 160	archives, digital, 108–15, 173, 222, 226–27
AIDS medications, 257–60	Aristotle, 150
air traffic, land ownership vs., 1–3	Armstrong, Edwin Howard, 3–6, 184, 196
Akerlof, George, 232	Arrow, Kenneth, 232
Alben, Alex, 100–104, 105, 198–99, 295, 317 <i>n</i>	art, underground, 186
alcohol prohibition, 200	artists:
<i>Alice's Adventures in Wonderland</i> (Carroll), 152–53	publicity rights on images of, 317 <i>n</i>
	recording industry payments to, 52, 58–59, 74, 195, 196–97, 199, 301, 329 <i>n</i> –30 <i>n</i>

Similar to Indexing in Textbooks

- A single-level index is an auxiliary file that makes it more efficient to search for a record in the data file.
- The index is usually specified on one field of the file (although it could be specified on several fields)
- One form of an index is a file of entries **<field value, pointer to record>**, which is ordered by field value
- The index is called an access path on the field.

$\langle K(1) = (\text{Aaron, Ed}), P(1) = \text{address of block 1} \rangle$
 $\langle K(2) = (\text{Adams, John}), P(2) = \text{address of block 2} \rangle$
 $\langle K(3) = (\text{Alexander, Ed}), P(3) = \text{address of block 3} \rangle$

What is index in Database?

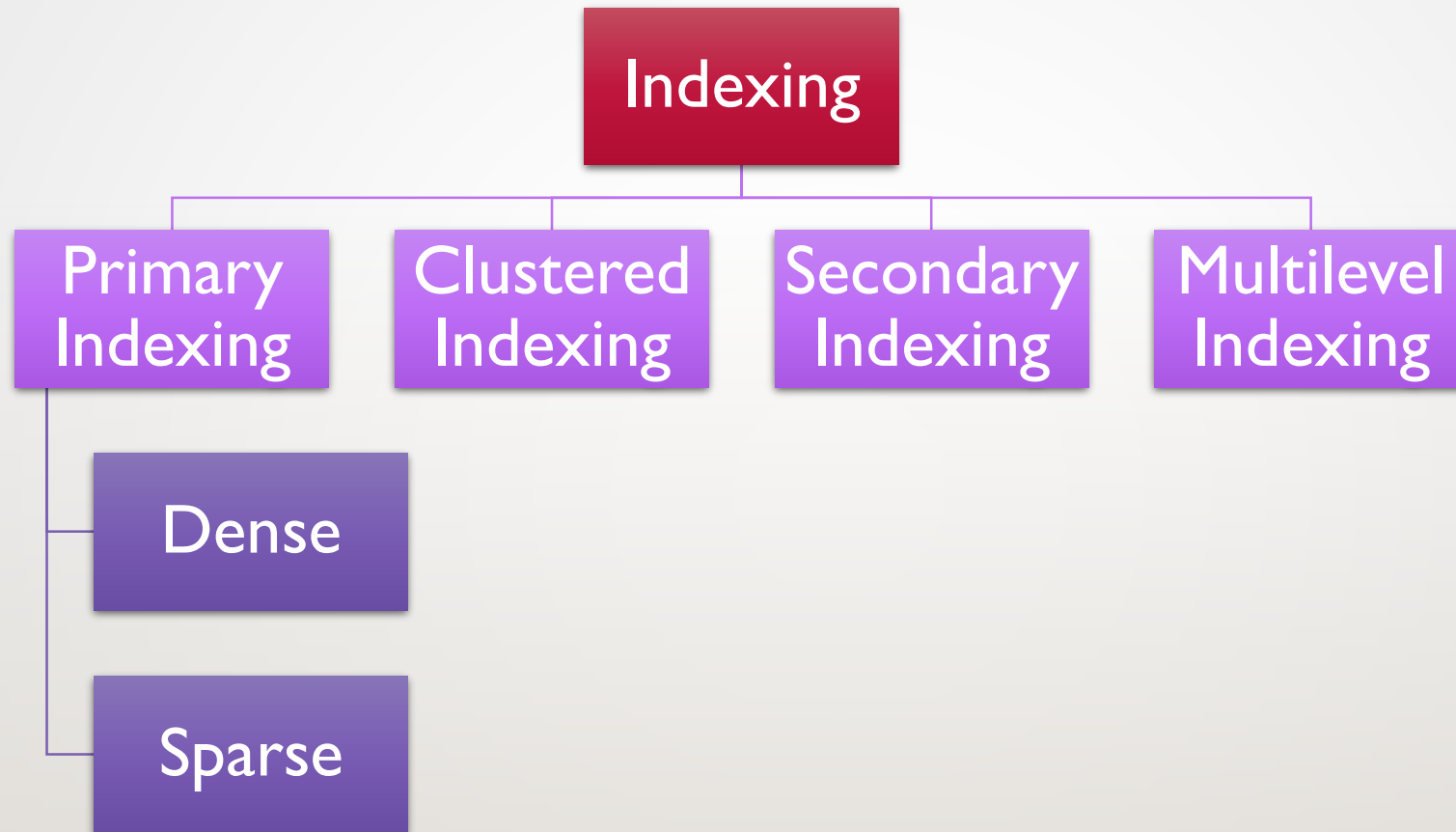


- An Index is a small table having only two columns. The first column comprises a copy of the primary or candidate key of a table. Its second column contains a set of pointers for holding the address of the disk block where that specific key value stored.

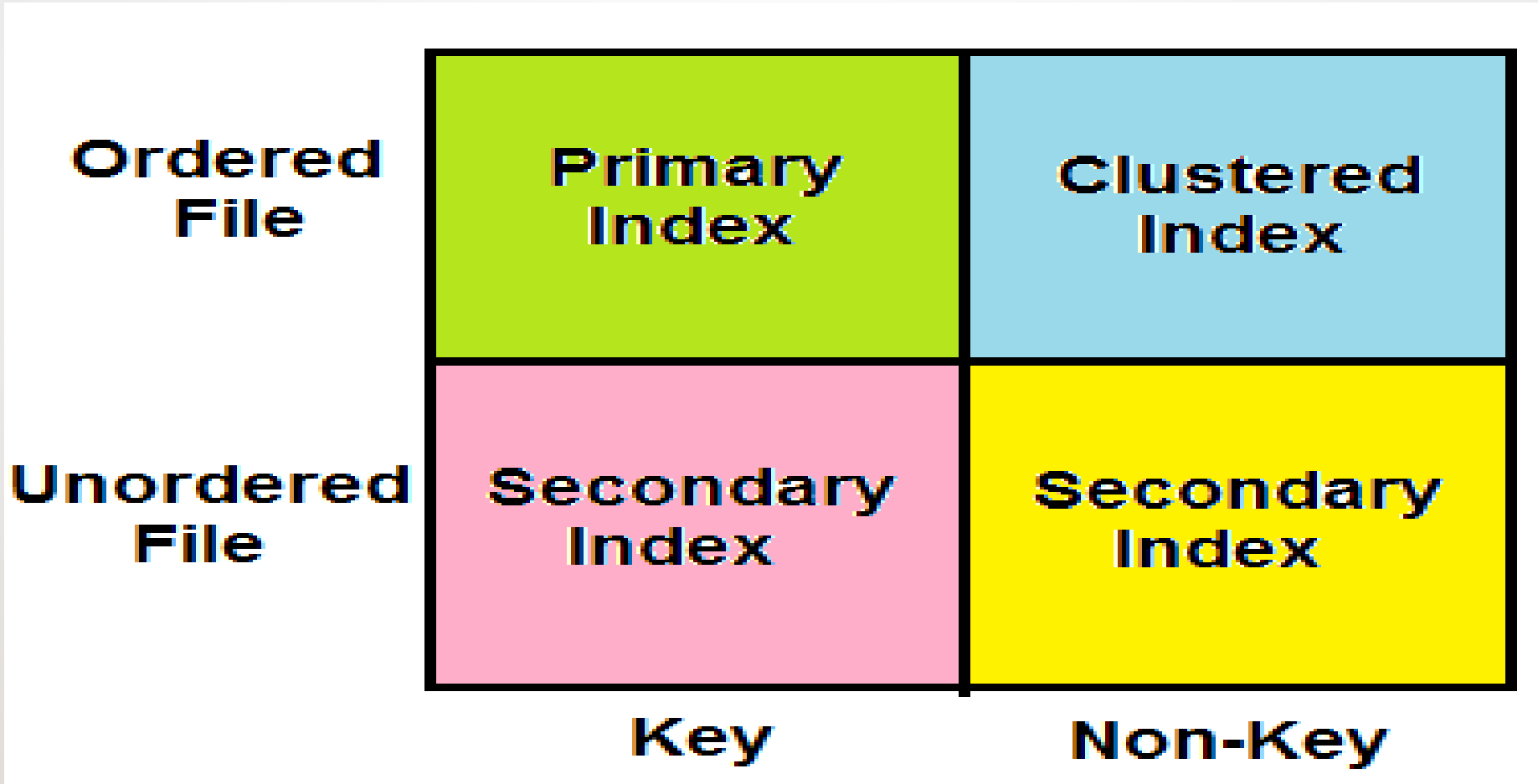
An index -

- Takes a search key as input
- Efficiently returns a collection of matching records.

Types of Indexing

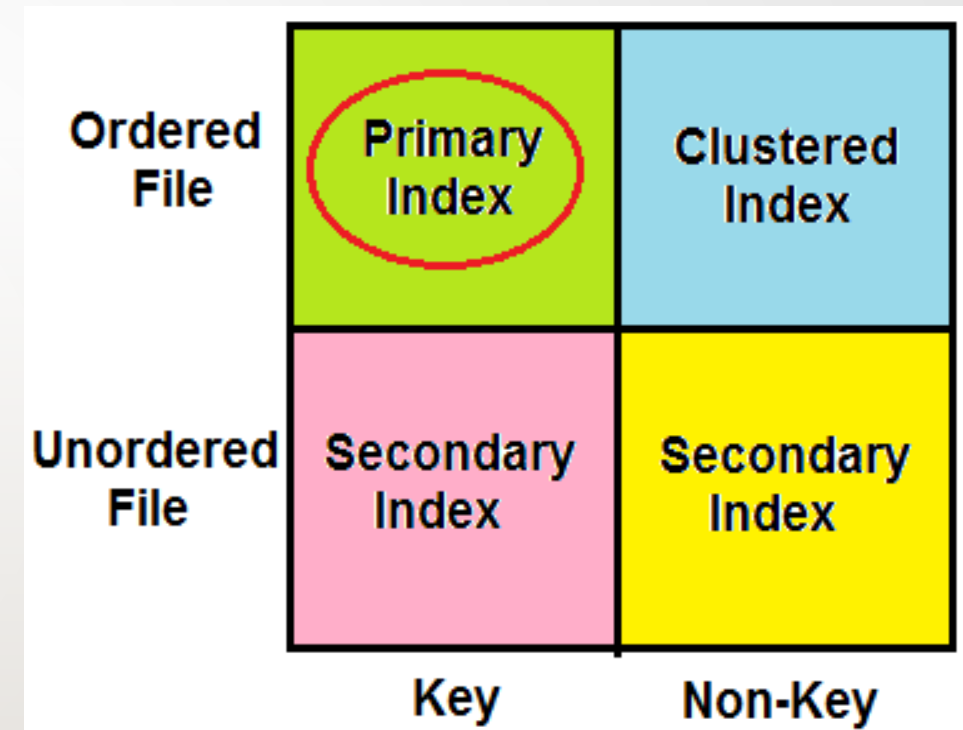


Types of Indexing



Primary Indexing

- Primary Index is an ordered file which is of fixed length size with two fields.
- The first field is the same as a primary key and second field is a pointer that points to that specific data block.
- The primary Indexing is further divided into two types.
 - Dense Index
 - Sparse Index



Dense Index



A record is created for each search key valued in the database.

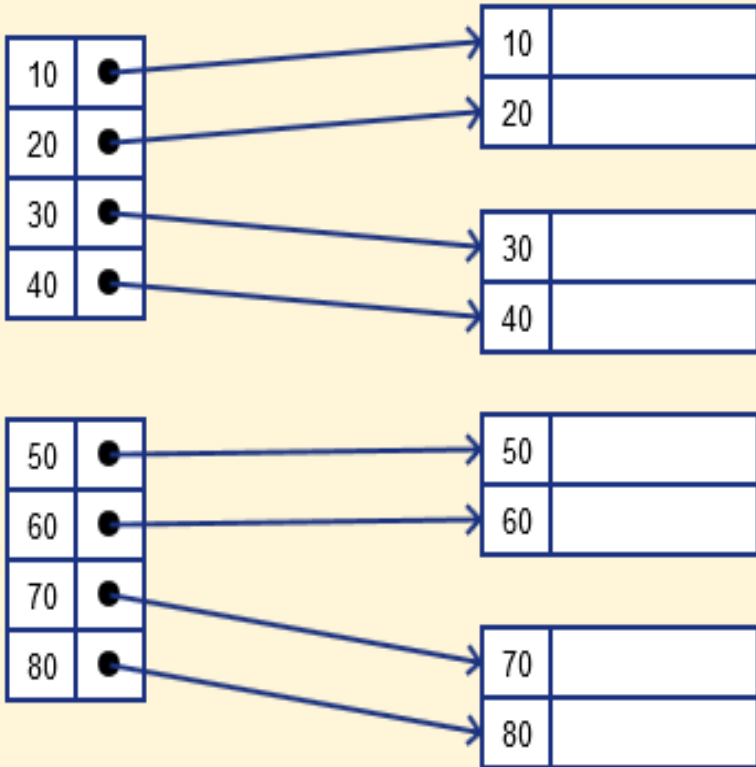


Searching is faster

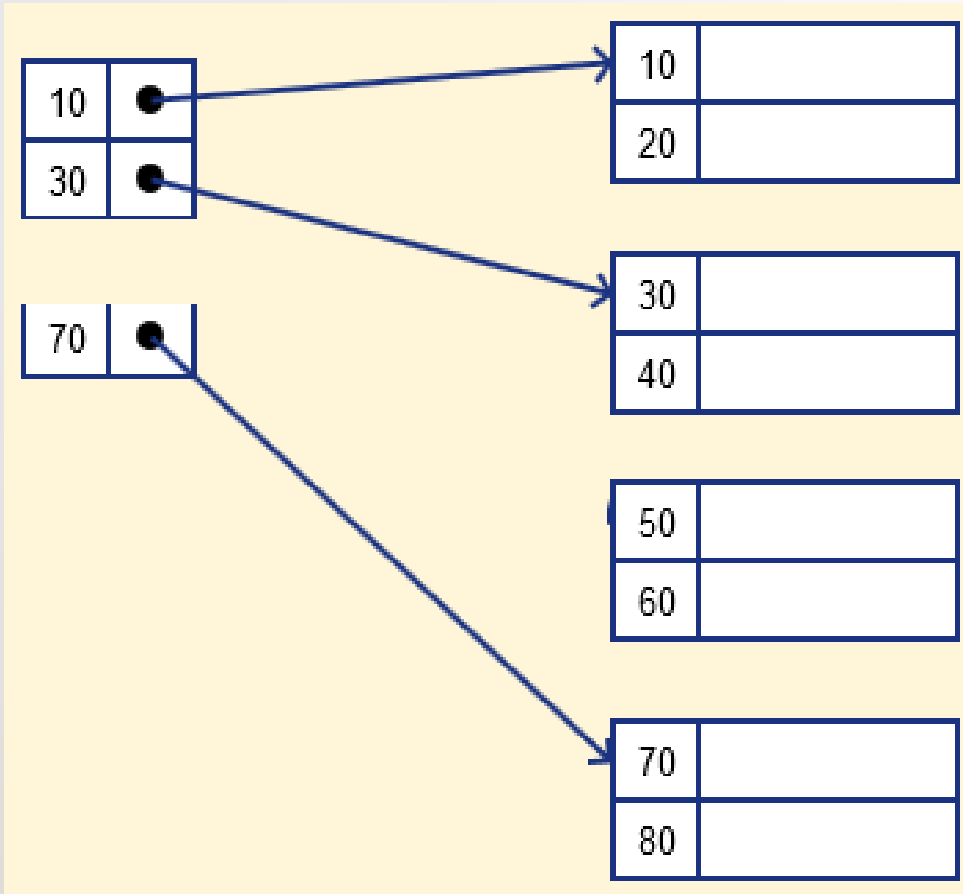


Requires more space to store index records

No. of records in IT = No. of records in HD



Sparse Index



- Sparse index contains only the anchor records
- To locate a record, we find the index record with the largest search key value \leq search key value we are looking for.
- We start at that record pointed to by the index record, and proceed along with the pointers in the file (sequentially) until we find the desired record.

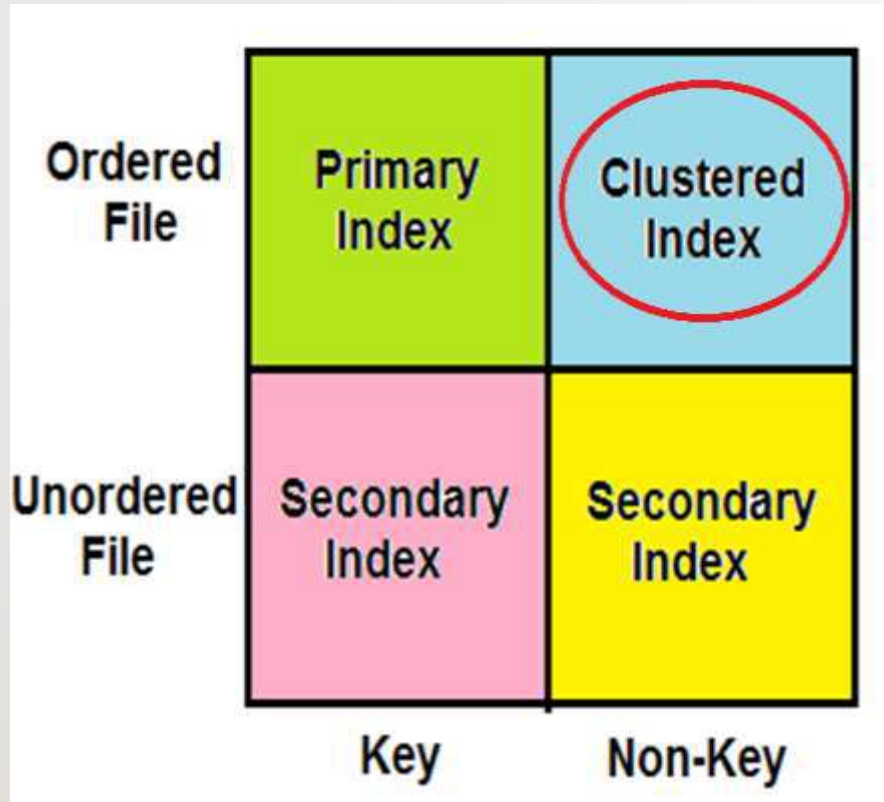
No. of records in IT = No. of blocks in HD

Time Complexity = $\log_2 N + 1$

Indexes as Access Paths

- The index file usually occupies considerably less disk blocks than the data file because its entries are much smaller
- A binary search on the index yields a pointer to the file record
- Indexes can also be characterized as dense or sparse
 - A **dense index** has an index entry for every search key value (and hence every record) in the data file.
 - A **sparse (or nondense) index**, on the other hand, has index entries for only some of the search values

Clustered Index



- Clustering index is defined on an ordered data file. The data file is ordered on a non-key field.
- In some cases, the index is created on non-primary key columns which may not be unique for each record.
- In such cases, in order to identify the records faster, we will group two or more columns together to get the unique values and create index out of them. This method is known as the clustering index.
- Basically, records with similar characteristics are grouped together and indexes are created for these groups.

Indexes as Access Paths

Contains block pointer which points to the next block data with the same clustering field value.

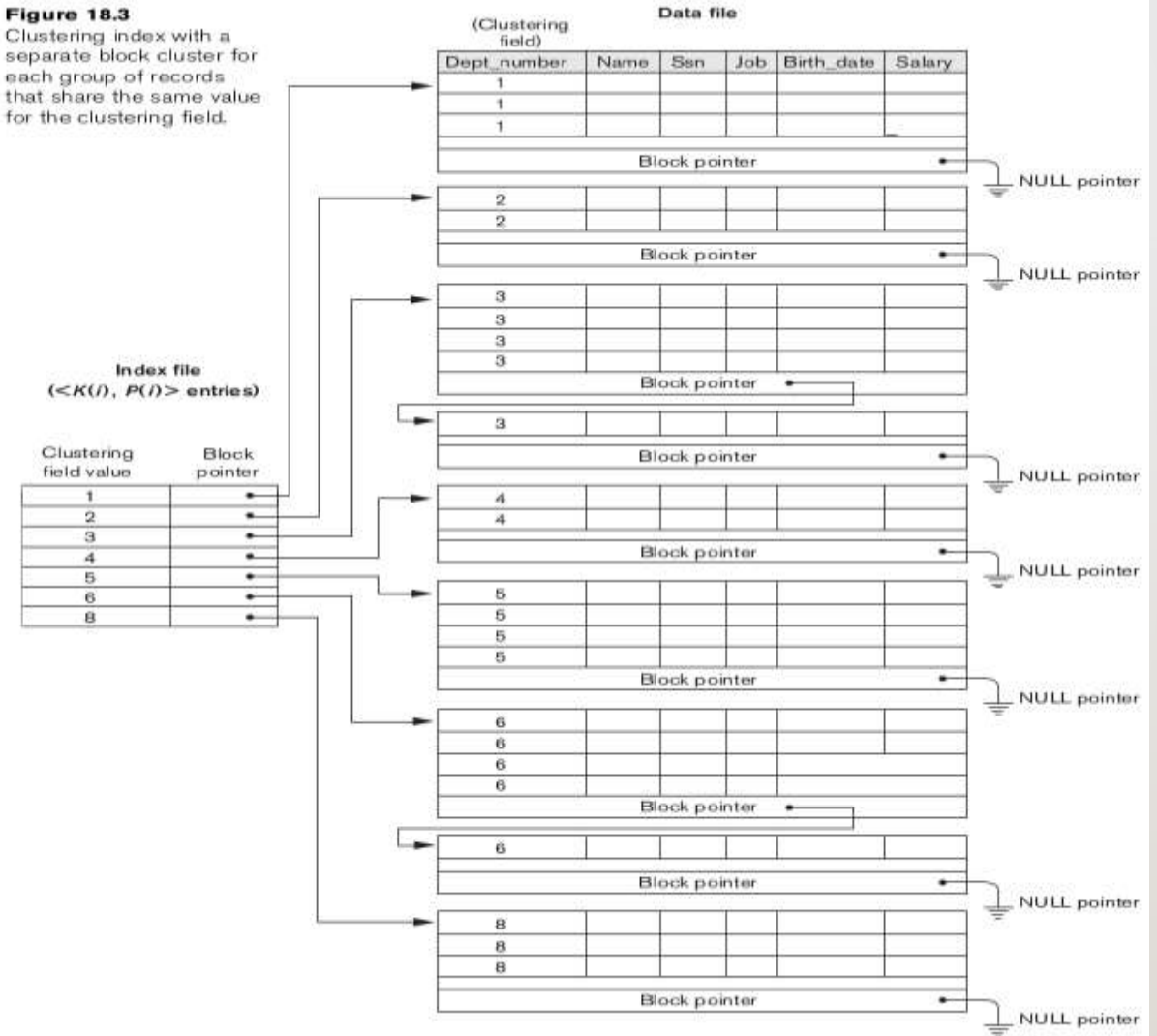
Searching criteria is little bit increased.

Uses Sparse index

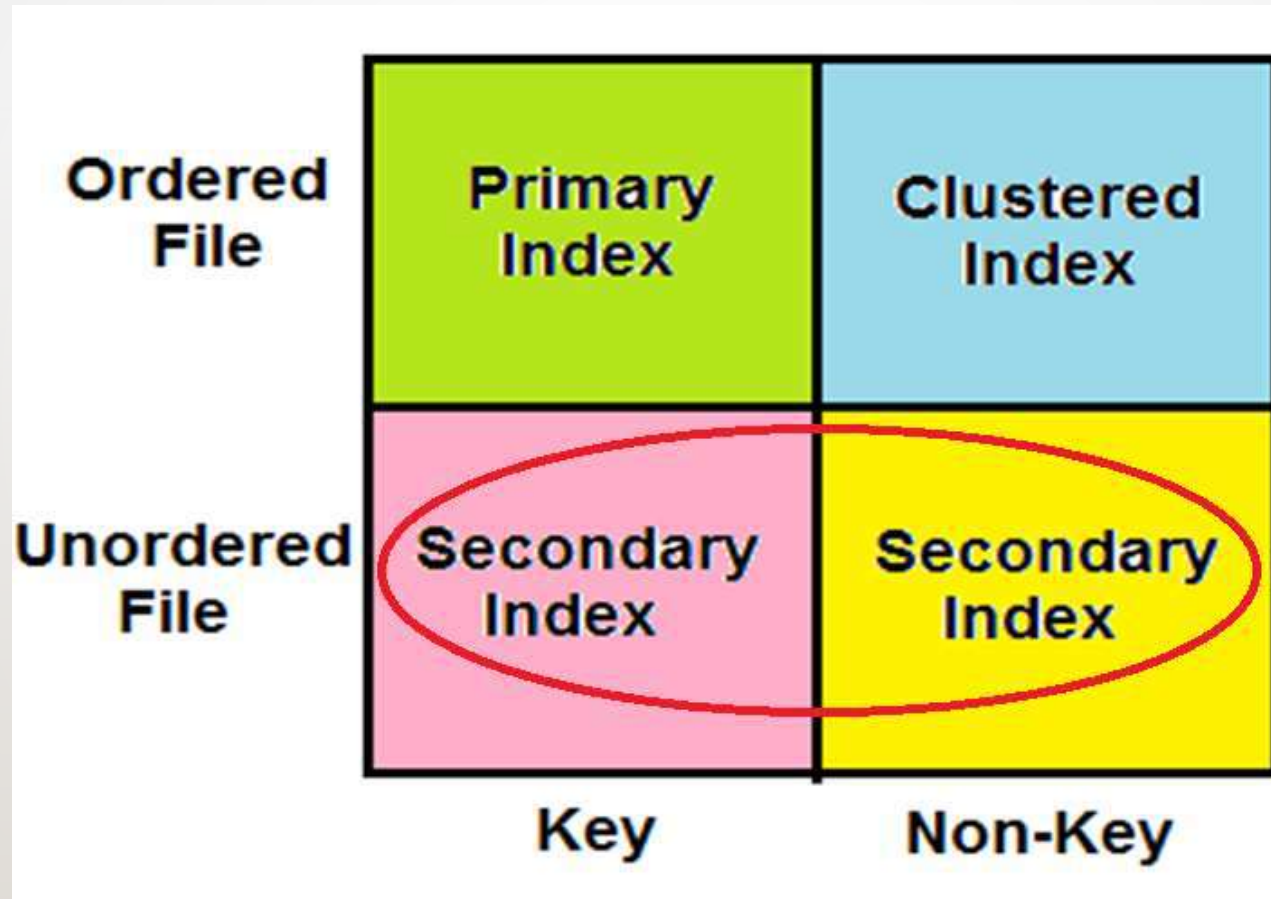
Time Complexity = $\log_2 N + 1 + 1..$

Figure 18.3

Clustering index with a separate block cluster for each group of records that share the same value for the clustering field.

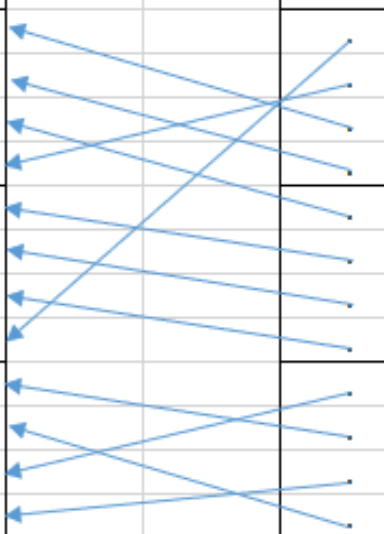


Secondary Indexing



Unordered File With Secondary Key

Eid	Ename	Pno			Ptr	Key (Pno)
1	A	40				23
2	B	51				33
3	A	62				40
4	C	33				51
5	A	71				62
6	D	82				71
7	E	91				82
8	F	23				91
9	K	100				98
10	L	150				100
11	H	98				120
12	C	120				150
HD					IT	



Secondary Index Example

- File is ordered on Eid(Primary Key)
- Search to be done using Pno
- So, Index table will maintain Pno as a key and in ordered.

$$\text{Time Complexity} = \log_2 N + 1$$

Secondary Indexing

Unordered File with Non-key

Eid	Ename	Pno		A Block		Ptr	Key(Ename)
1	A	40	←	A	←	.	A
2	B	51	←	A	←	.	B
3	A	62	←	A	←	.	C
4	C	33	←		←	.	D
5	A	71	←	B Block	←	.	E
6	D	82	←	B	←	.	F
7	E	91	←		←	.	H
8	F	23	←		←	.	K
9	K	100	←		←	.	L
10	L	150	←	C Block	←		
11	H	98	←	C	←		
12	C	120	←	C	←		
HD				Blocks of Record Pointers			IT
Secondary index by indirection							

Secondary Index Example

- Search done by Ename(Non-key)
- Index file contains Ename as key and is ordered.
- Maintains intermediate index layer which contains block of record pointers.
- Pointer in IT points to a particular block and the record pointers in that block will point to the record in HD.

$$\text{Time Complexity} = \log_2 N + 1 + 1$$

Types of Single-Level Indexes

Primary Index	Clustering Index	Secondary Index
ordered file	ordered file	ordered file a secondary means of accessing a file
Data file is ordered on a key field (distinct value for each record)	Data file is ordered on a non-key field (no distinct value for each record)	Data file is ordered may be on candidate key has a unique value or a non-key with duplicate values
file content <key field, pointer>	file content <key field, pointer>	file content <key field, pointer>
one index entry for each disk block. key field value is the first record in the block, which is called the block anchor	one index entry for each distinct value of the field; the index entry points to the first data block that contains records with that field value	The index is an ordered file with two fields: 1field value. 2it is either a block pointer or a record pointer.
nondense (sparse) index	nondense (sparse) index	If key, dense. If non key, dense or sparse index

Multi-Level Indexes

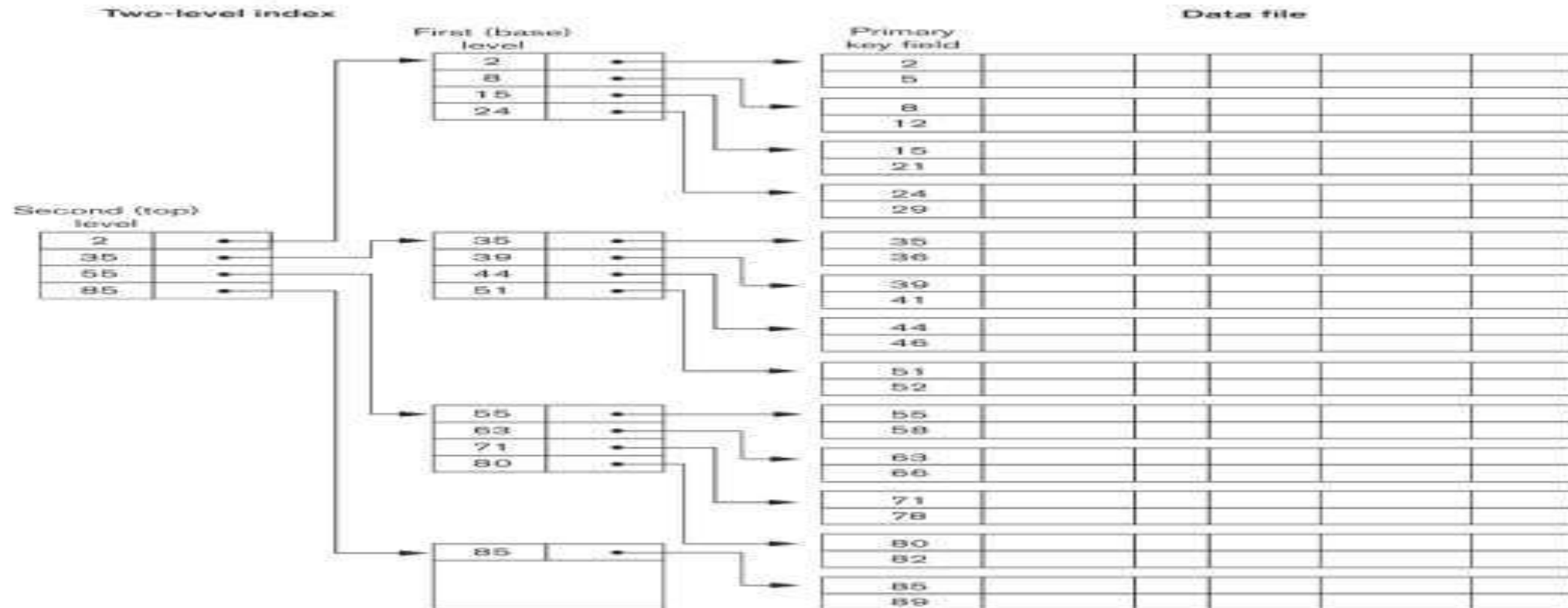
A Two-level Primary Index
Dynamic Multilevel Indexes Using B-trees and
B+-trees

Multi-Level Indexes

- Because a single-level index is an ordered file, we can create a primary index *to the index itself*;
- In this case, the original index file is called the *first-level index* and the index to the index is called the *second-level index*.
- We can repeat the process, creating a third, fourth, ..., top level until all entries of the *top level* fit in one disk block
- A multi-level index can be created for any type of first-level index (primary, secondary, clustering) as long as the first-level index consists of *more than one* disk block

A Two-Level Primary Index

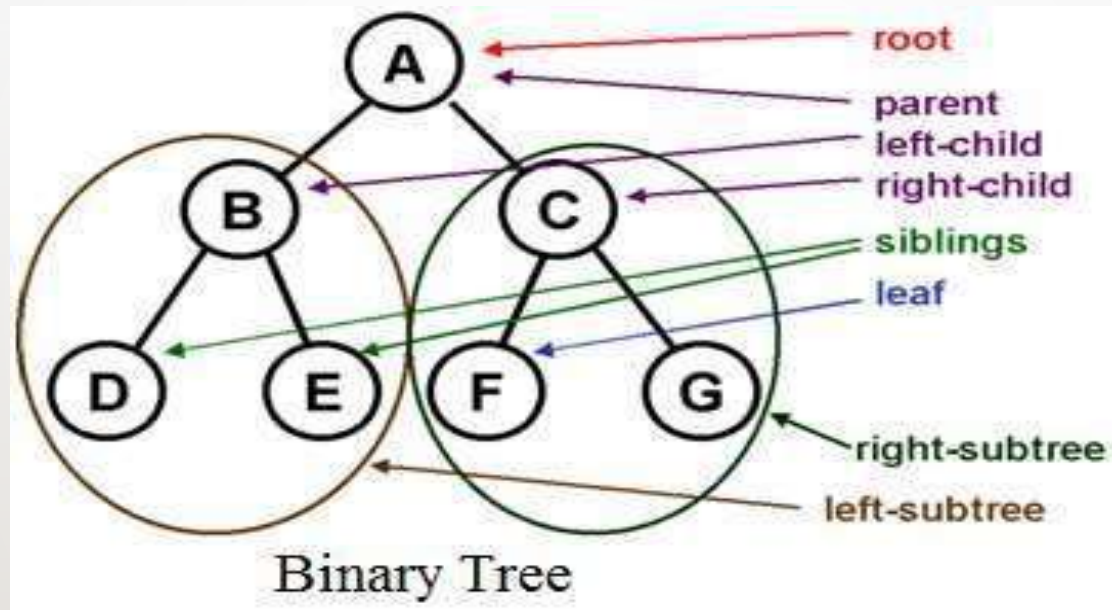
Figure 18.6
A two-level primary index resembling ISAM (Indexed Sequential Access Method) organization.



- Such a multi-level index is a form of *search tree*
- However, insertion and deletion of new index entries is a severe problem because every level of the index is an *ordered file*.

Multi-Level Indexes

Tree structure



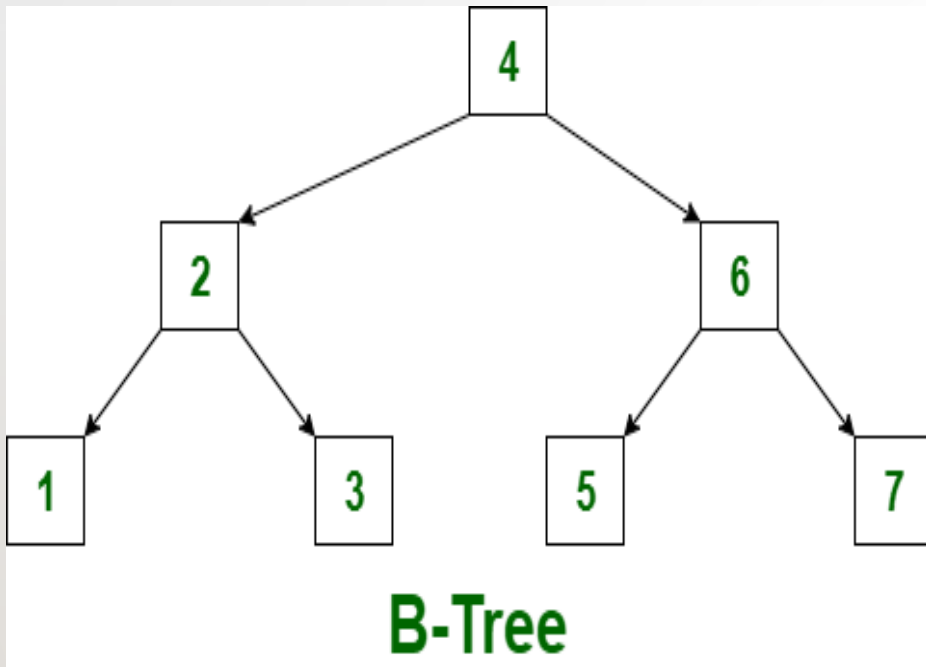
Dynamic Multilevel Indexes Using B-Trees and B+- Trees

- Most multi-level indexes use B-tree or B+-tree data structures because of the insertion and deletion problem
- This leaves space in each tree node (disk block) to allow for new index entries
- These data structures are variations of search trees that allow efficient insertion and deletion of new search values.
- In B-Tree and B+-Tree data structures, each node corresponds to a disk block
- Each node is kept between half-full and completely full

Dynamic Multilevel Indexes Using B-Trees and B+- Trees

- An insertion into a node that is not full is quite efficient
- If a node is full the insertion causes a split into two nodes
- Splitting may propagate to other tree levels
- A deletion is quite efficient if a node does not become less than half full
- If a deletion causes a node to become less than half full, it must be merged with neighboring nodes

Dynamic Multilevel Indexes Using B-Trees and B+- Trees

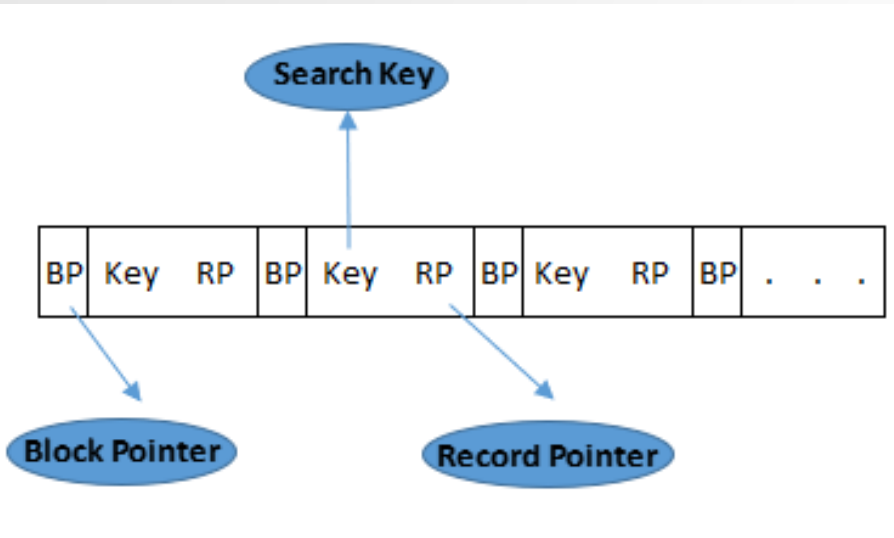


- Balanced Tree
- In multilevel indexing, inserting and deleting a record is difficult, as the corresponding entries in index tables also need to be changed.
- B-Trees makes these tasks simple.
- Elements are in sorted order

B-Tree

A B tree of order m contains the following properties:

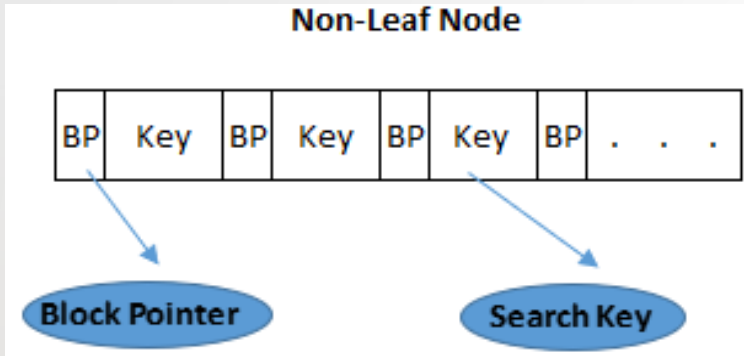
- ⑩ Every node in a B-Tree contains at most $m-1$ keys and m children.
- ⑩ Every node in a B-Tree except the root node and the leaf node contain at least $m/2$ children.
- ⑩ The root nodes must have at least 2 nodes.
- ⑩ All leaf nodes must be at the same level.
- ⑩ It is not necessary that, all the nodes contain the same number of children but, each node must have $m/2$ number of nodes.



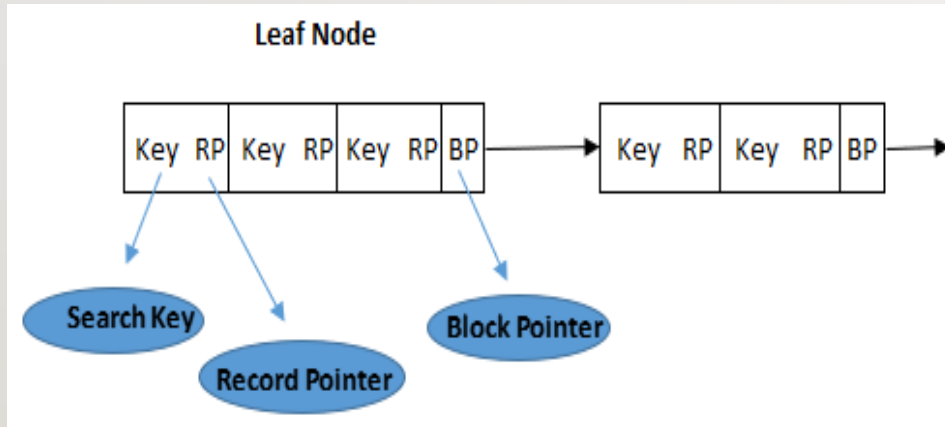
B-Tree

B+ Tree

Non-Leaf Node



Leaf Node



- B+ Tree is an extension of B Tree which allows efficient insertion, deletion and search operations.
- In B Tree, records (data) can only be stored on the leaf nodes while internal nodes can only store the key values.
- The leaf nodes of a B+ tree are linked together in the form of a singly linked lists to make the search queries more efficient.

Difference between B-tree and B+-tree

	B Tree	B+ Tree
1	Search keys can not be repeatedly stored.	Redundant search keys can be present.
2	Data can be stored in leaf nodes as well as internal nodes	Data can only be stored on the leaf nodes.
3	Searching for some data is a slower process since data can be found on internal nodes as well as on the leaf nodes.	Searching is comparatively faster as data can only be found on the leaf nodes.
4	Deletion of internal nodes are so complicated and time consuming.	Deletion will never be a complexed process since element will always be deleted from the leaf nodes.
5	Leaf nodes can not be linked together.	Leaf nodes are linked together to make the search operations more efficient.

Difference between B-tree and B+-tree

- In a B-tree, pointers to data records exist at all levels of the tree
 - In a B+-tree, all pointers to data records exist at the leaf-level nodes
 - A B+-tree can have less levels (or higher capacity of search values) than the corresponding B-tree
-
- Similarities between B-tree and B+-tree
 - All leaf nodes at the same level
 - Nodes contents not less than the half

Example

- Using a B-Tree Index of order $P = 3$. insert the following values in the order 8, 5, 1, 7, 3, 12, 9, 6.

Employee id	name	salary	department
8	Saleh	10000	1
5	Ahmed	20000	1
1	Jasem	30000	2
7	Nader	15000	2
3	Saleh	13000	3
12	Waleed	8000	3
9	Salim	11000	4
6	Raed	8000	5

B-tree Structures

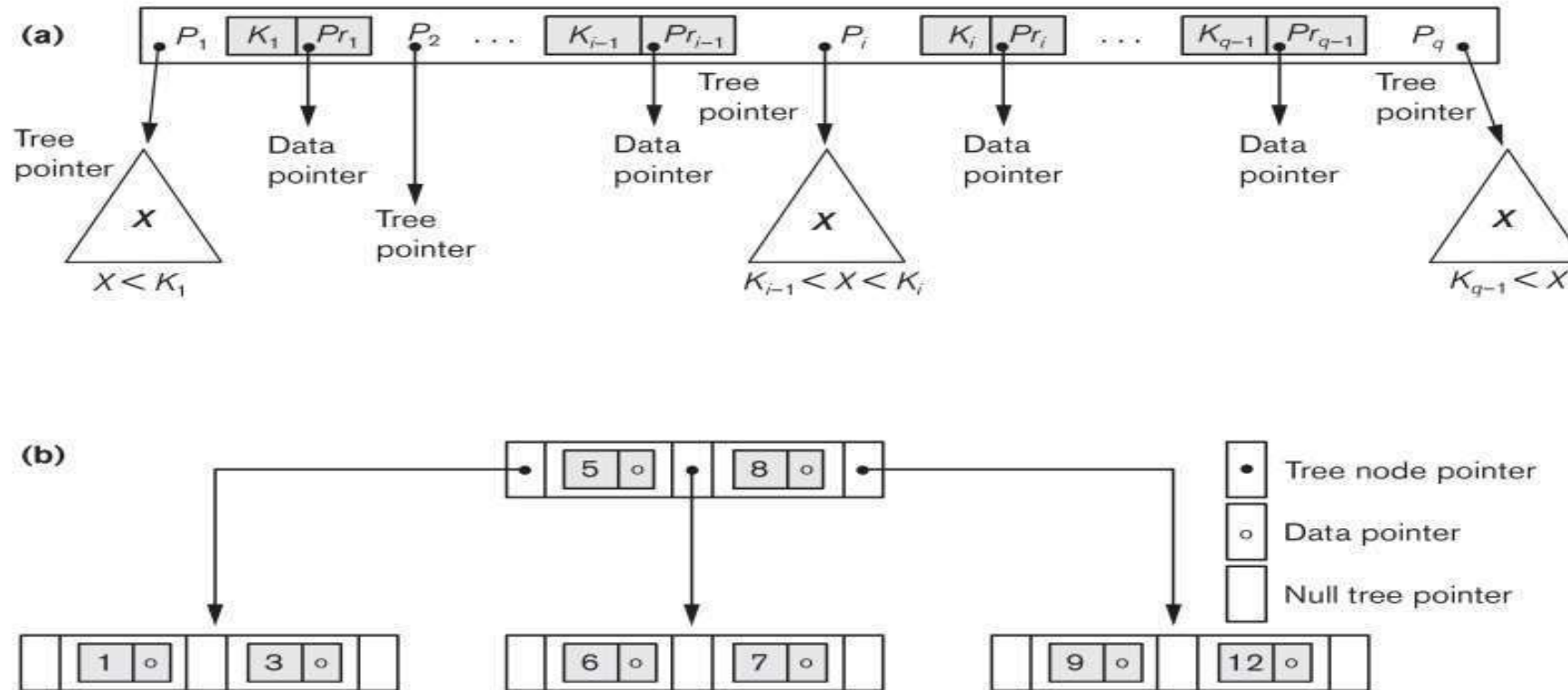


Figure 18.10

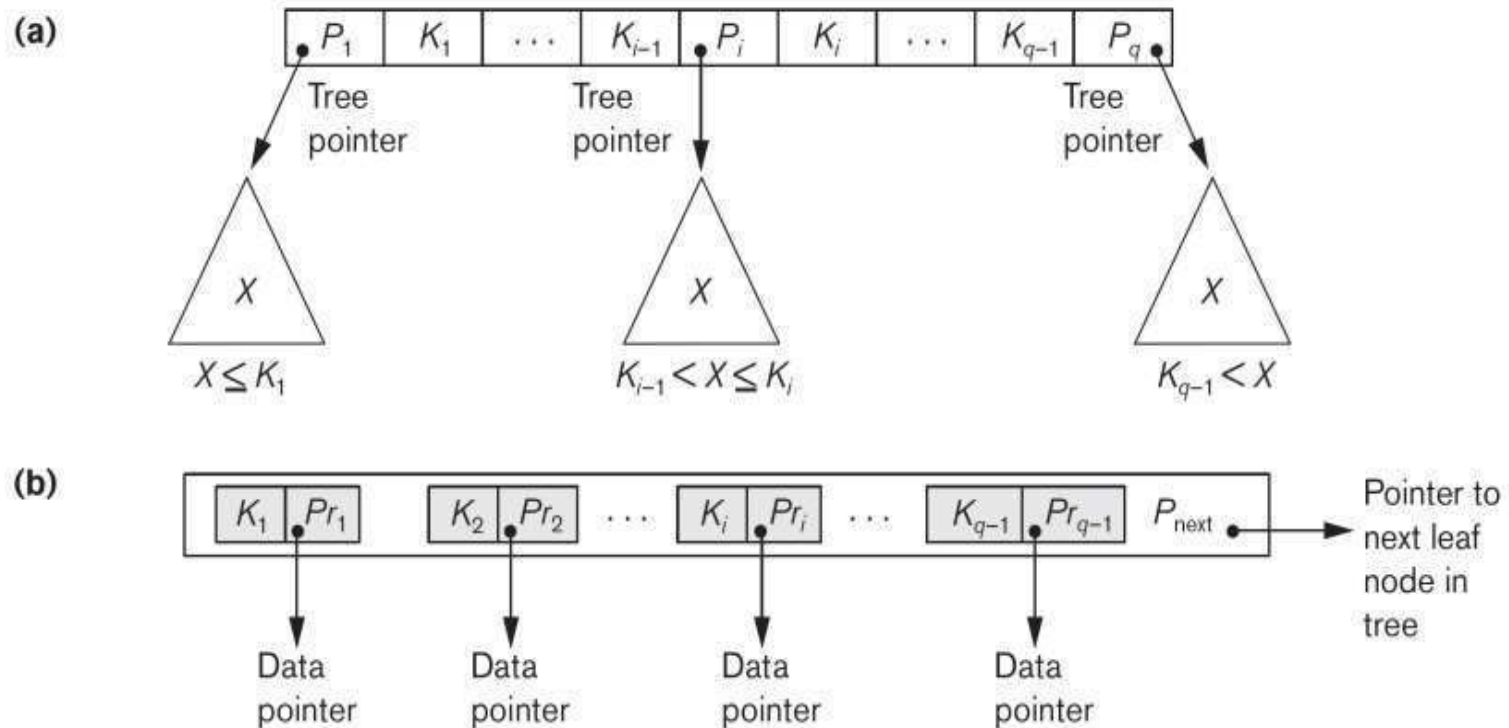
B-tree structures. (a) A node in a B-tree with $q - 1$ search values. (b) A B-tree of order $p = 3$. The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6.

The Nodes of a B⁺-tree

Figure 18.11

The nodes of a B⁺-tree. (a) Internal node of a B⁺-tree with $q - 1$ search values.

(b) Leaf node of a B⁺-tree with $q - 1$ search values and $q - 1$ data pointers.



ACTIVITIES/ CASE STUDIES/ IMPORTANT FACTS RELATED TO THE SESSION

Given that the order of an internal node in a B^+ tree, index is the maximum number of children it can have. Imagine a child pointer takes 6 bytes, the search field value takes 14 bytes, and the block size is 512 bytes. Then, what is the order of the internal node?

Example

Using a B+-tree index of order $p = 3$, $p_{\text{leaf}} = 2$.

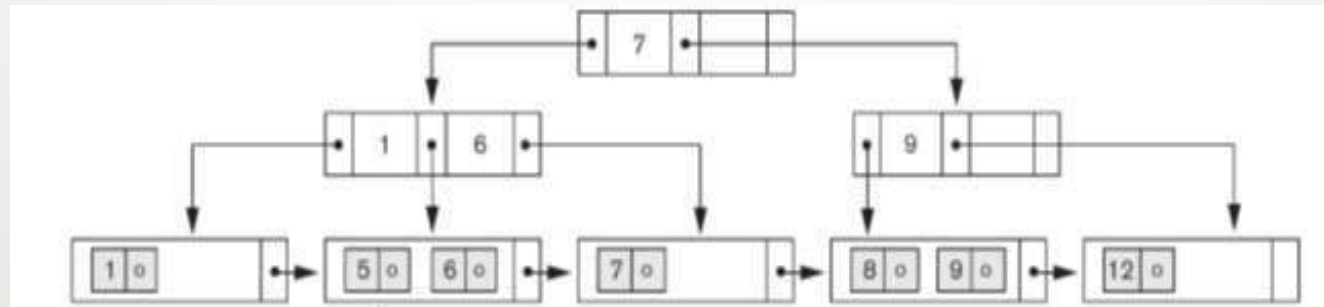
Insert the following values in the order 8, 5, 1, 7, 3, 12, 9, 6.

Employee id	name	salary	department
8	Saleh	10000	1
5	Ahmed	20000	1
1	Jasem	30000	2
7	Nader	15000	2
3	Saleh	13000	3
12	Waleed	8000	3
9	Salim	11000	4
6	Raed	8000	5



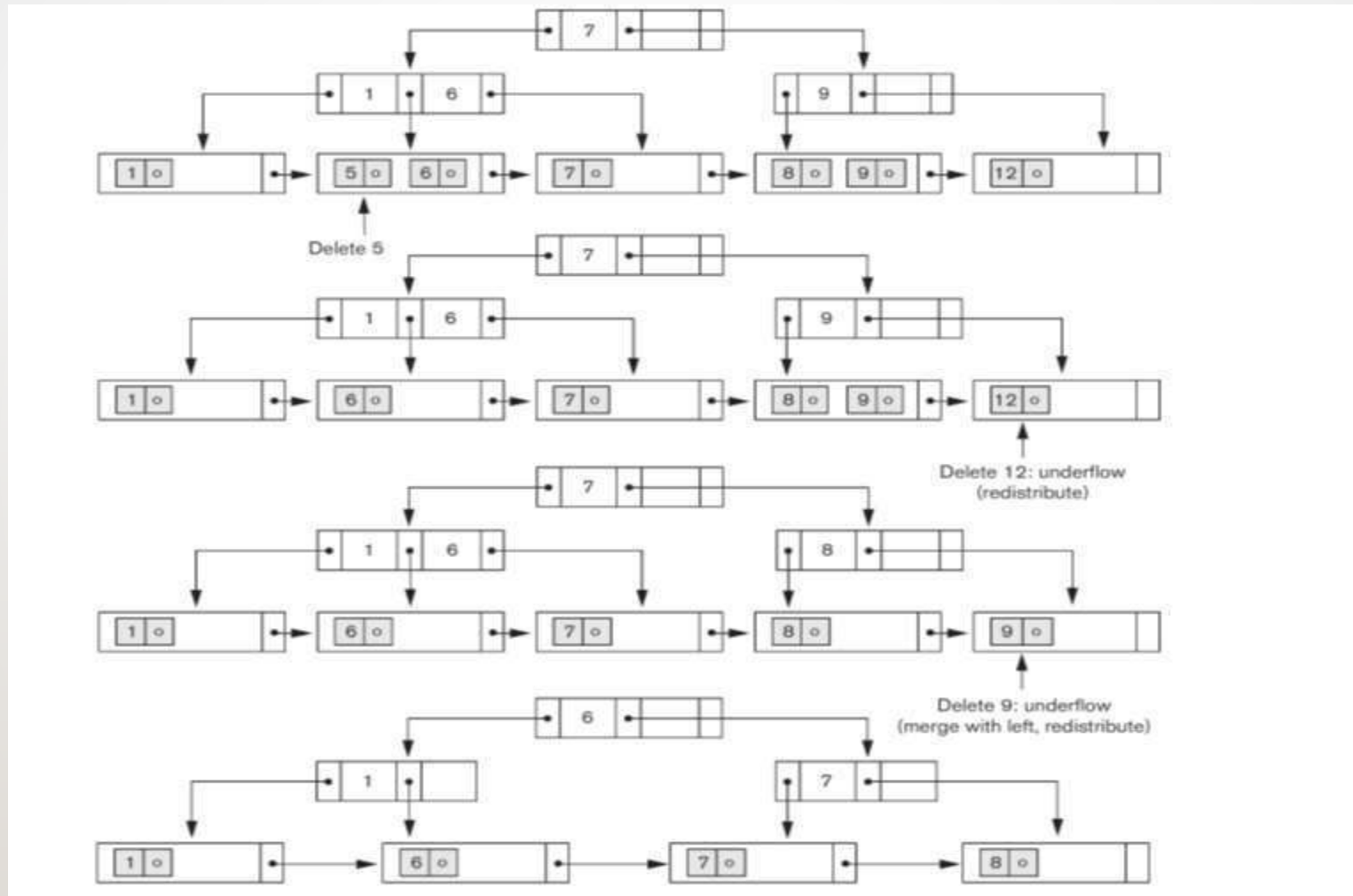
Example

Given a B+-Tree Index of order $P = 3$, $P_{\text{leaf}} = 2$.



Delete the nodes, 5, 12, 9

Example



- Indexing is a technique for improving database performance by reducing the number of disk accesses necessary when a query is run.
- An index is a form of data structure.
- It's used to swiftly identify and access data and information present in a database table.
- Indexing is used to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed

SELF-ASSESSMENT QUESTIONS

1. The fields on which clustering index is defined are of type_____

- (a) Key and non-ordering
- (b) Non-key and ordering
- (c) Key and ordering
- (d) Non-key and non-ordering

2. If a block can hold either 3 records or 10 key pointers and a database contains "n" records, then how many blocks do we need to hold the data file and the dense index?

- (a) $\lceil 3n/30 \rceil$
- (b) $N/10$
- (c) $N/30$
- (d) $N/3$

1. **What is an index in a database and what is its purpose?**
2. **What are the common types of index structures used in database management systems? Describe the advantages and disadvantages of each.**
3. **How does a B-tree index work and what are its properties? How is it different from other index structures?**
4. **Explain the concept of clustering and non-clustering indexes. What are the differences between these two types of indexes and when would you use each one?**

REFERENCES FOR FURTHER LEARNING OF THE SESSION

Reference Books:

1. "Database Management Systems" by Raghu Ramakrishnan and Johannes Gehrke - This book covers the basics of database management systems, including the concept of index structures.
2. "Database Systems: Design, Implementation, and Management" by Carlos Coronel, Steven Morris, and Peter Rob - This book provides a comprehensive introduction to database systems, including index structures and their importance in optimizing database performance.
3. "Database Indexing: A Practical Guide for Developers" by Will Iverson - This book focuses specifically on the concept of indexing in database management systems, providing practical advice and examples for developers.

Sites and Web links:

1. <https://docs.microsoft.com/en-us/sql/relational-databases/indexes/indexes?view=sql-server-ver15>
2. <https://dev.mysql.com/doc/refman/8.0/en/mysql-indexes.html>
3. <https://www.postgresql.org/docs/current/indexes.html>

THANK YOU



Team – DBMS