

Disk Storage, Basic File Structure and Hashing

- Several types of data storage exists in most computer systems.
- These media are classified by the speed with which data can be accessed, cost per unit of data to buy them & their reliability.
- **Cache-** Fastest and most costly, is small, managed by the computer system hardware, not our concern to manage it in DBMS.
- **Main Memory-** Used for data that are available to be operated on, even if it is available in Mb or GB still its too small to store a database, its contents are lost in case of power failuare or system crash.
- **Flash Memory** – also known as *electrically erasable programmable read only memory* (EEPROM). Data survives in case of power failure, reading from flash is easy-writing is complex, supports only a limited number of erase cycles, used in computer systems which are embedded in other devices such as digital cameras.

- **Magnetic disk storage-** Direct access storage, entire database can be stored permanently, system moves data from disk to main memory and back, available in GBs, survives power failure
- **Optical Storage-** examples CDs (Compact Disk) & DVDs (Digital Video Disk) , available in record once or multiple write forms, data are stored optically and are read by a laser
- **Tape Storage-** Used primarily for backup and archival, Cheaper than disks, but access to data is slower, referred to as sequential access storage,

Basic File Structure

- A file is a collection of records where record consists of one or more fields.
- Primary objective of file organization- provide means for record retrieval and update.
- Factors involving in selecting a particular file organization are:
 - Economy of storage
 - Ease of retrieval
 - Convenience of updates
 - Reliability
 - Security
 - Integrity
 - Volume of transaction

Commonly used file organizations

- **Sequential File: -**

- Data records are stored in a specific sequence
- Records are physically ordered on the value of one of the fields
- Records could also be stored in the order of arrival

- **Relative File: -**

- Each record is placed at a fixed place in file
- Each record is associated with a integer key-value, which is mapped to a fixed slot in a file.

- **Direct File: -**

- Similar to relative file.
- popularly known as a hashed file.
- here the key value need not be an integer

- **Indexed Sequential File: -**

- an index is added to a sequential file to provide random access.
- An overflow area is maintained to allow insertions.

- **Index File: -**

- data records need not be sequenced
 - An index is maintained to improve access
- ✓ Every OS supports the concept of file.
 - ✓ File is a collection of records
 - ✓ Records of a file are mapped onto disk blocks

- When a database is stored in to files, there are two possibilities:

I) Use several files with an independent file for each table.

This approach uses fixed length records.

Files of fixed length records are easier to implement.

II) Use a single file for storing multiple tables.

The file will contain records of different record types, and hence variable length records. Files of variable length records are difficult to implement.

- **Fixed Length Records:**

- A file of FLR consists of records of equal size.
- Stored on secondary storage
- A secondary storage device is divided into blocks.
- A block is the unit of data transfer between disk and memory.
- Records are the logical unit of access of a file.
- Blocks are unit of input / output for secondary storage.
- When the block size B is larger than the record size R , then we can store $\lfloor B/R \rfloor$ (floor of B/R) records per block.
- Records of a file can be stored serially.
- A serial file is generated by appending records at the end.

- Deletion of a record from a serial file can be handled in a number of ways:
 - 1. all the records following the deleted record can be moved forward.
 - 2. last record can be brought in place of the deleted record.
 - 3. delete the record logically. (only marked as deleted, not physically)
- ✓ Logical deletion is more efficient.
- ✓ A new record might be inserted in the space occupied by the first deleted record.
- ✓ In physical deletion, more moving of records and additional block access.
- ✓ To quickly locate an empty space for an incoming record, a linked list of deleted records can be maintained.

Example

- Consider a file of student records. Each record is defined as:

```
struct student
{
    char name[30];
    int roll_no;
    int marks;
};
```

❑CASE-I:

Record no. 2,6,9 are deleted and all records moved to fill up the empty slots. (fig. 7.1.2)

❑CASE-II:

Record no. 2,6,9 are deleted and a chain of deleted records is created.

Address of the first deleted record is stored in the header.

Header is a special structure which is allocated a certain no. of bytes at the beginning of the file. (fig. 7.1.3)

0	Amit	105	80
1	Sohan	109	79
→ 2	Rita	101	76
3	Ramu	102	62
4	Shekhar	120	49
5	Lata	113	52
→ 6	Mangesh	115	56
7	Shyam	121	53
8	Nitu	116	43
→ 9	Rekha	130	78

Fig. 7.1.1 : A file with fixed-length records

0	Amit	105	80
1	Sohan	109	79
2	Ramu	102	62
3	Shekhar	120	49
4	Lata	113	52
5	Shyam	121	53
6	Nitu	116	43

Fig. 7.1.2 : Record 2,6 and 9 deleted and records moved

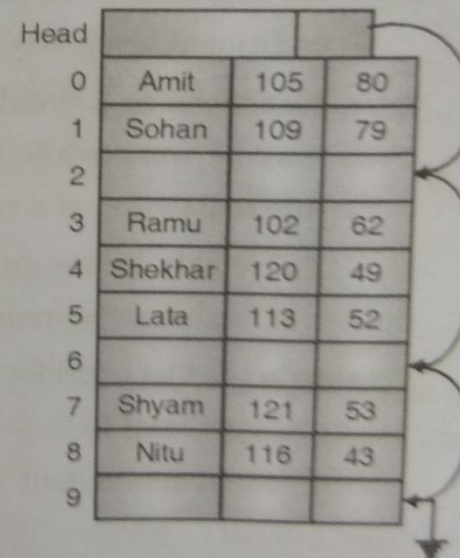


Fig. 7.1.3 : Records 2,6 and 9 deleted and records chained

- **Variable Length Records:** A file may have variable length records for several reasons:
 - Each table is stored in a separate file but one or more fields are of varying sizes.
 - Each table is stored in a separate file, but one or more fields may have multiple values (multi-value attributes)
 - Multiple tables are stored in one file.
- Techniques for implementing VLR-
 - The reserved space method
 - The pointer method
 - The combined method

- **The Reserved Space Method :**

- Variable length records can be represented using fixed length records.
- If there is an upper limit on the size of the record that is never exceeded then this upper limit can be used as the size of the record.
- Unused space (for records shorter than max size) can be filled with null values .

Example : Let us consider a file of student records. Each record of a student is allowed to store marks of maximum of three subjects. Each record of a student is defined as :

```
Struct student
{
    char name [30] ;
    int rollno ;
    char sub1 [5] ;
    int mark1 ;
    char sub2 [5] ;
    int mark2 ;
    char sub3 [5] ;
    int mark3 ;
};
```

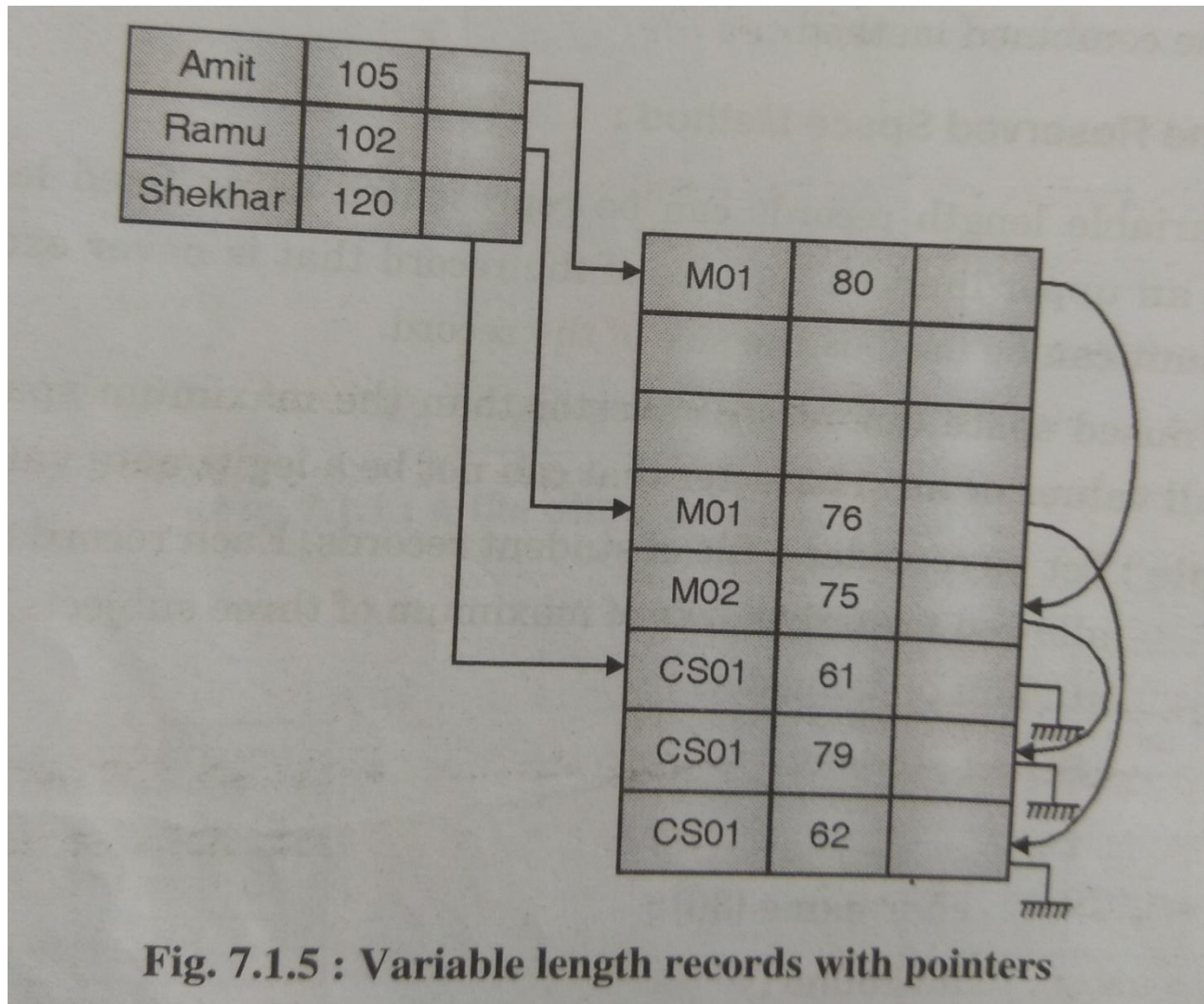
If we decide to use the reserved-space method to store student-record, we need to select a maximum record length. Fig. 7.1.4 shows a representation of sample database using reserved-space method.

Amit	105	M01	80	M02	75	CS01	79
Sohan	109	CS02	78	Null	Null	Null	Null
Ramu	102	M01	76	CS01	62	Null	Null
Shekhar	120	CS01	61	Null	Null	Null	Null

Fig. 7.1.4 : Representing variable length records using reserved-space method

- **The Pointer Method :**

- We could replace a repeating group by a pointer to the first group of the chain of the repeating groups.



- **The Combined Method :**

- The reserved space and pointer method can be combined together in various ways:

- We could use pointers for one repeating groups and reserved space for another repeating group
 - First few occurrences of a repeating group can be stored using reserved space and additional occurrences can be handled using the pointer method.

- ✓ Pointer based method tends to use less space, but performs more block access to find a record and hence complex to implement and maintain.

- **File Organization**

- The techniques used to represent and store records in the file is called the file organization.

- Techniques are:

1. Sequential file organization

2. Indexed file organization

- 2.1. Indexed Sequential or primary indexed

- 2.2. Multilevel indexed

3. Hashing

Sequential file

- Records are stored in the sequence of their primary key value.
- Records span over several blocks.
- Insertion of a new record requires shifting of records from the point of insertion to the end of file to create space for the incoming record.
- Deletion & updation operations are complex and requires shifting of records.

... Blocks.

Rollno	Name	Year	Marks
1000	Amit	1	60.48
1005	Pratap	2	53.68
1007	Mohan	1	70.43

Block 1

1010	Deepak	3	49.58
1012	Snehal	2	63.43
1015	Prakash	4	72.45

Block 2

⋮

1090	Suchitra	3	79.59
1092	Reehal	2	80.62
1095	Rachna	4	86.42

Block n

Fig. 7.3.1 : A sample sequential file with primary key as Rollno

Sequential file

- **Advantages:**

- 1) Reading of records in order of primary key is very efficient.
- 2) Finding of next record in order of primary key usually may not require additional block access. Next record may be found in the same block.
- 3) Searching on primary key is very fast.
- 4) Range based queries on primary key values will run very efficiently.

Sequential file

- **Disadvantages:**
 - 1) For non-primary key based search operation, it is not advantageous.
 - 2) Insert, Delete Update operations are expensive as it requires movement of records.

Indexed file organization (Ordered indices)

- Indexing is used to speed up retrieval of records.
- It is done with the help of a separate sequential file.
- This file is known as index file.
- Each record in index file consists of 2 fields-
 - Ordering Key
 - Block pointer, Pointing to the data block containing the said record.
- To find a specific record for the given ordering key value, the index is searched for the given key value.
- After getting the address of the block from index, the record can be easily retrieved.

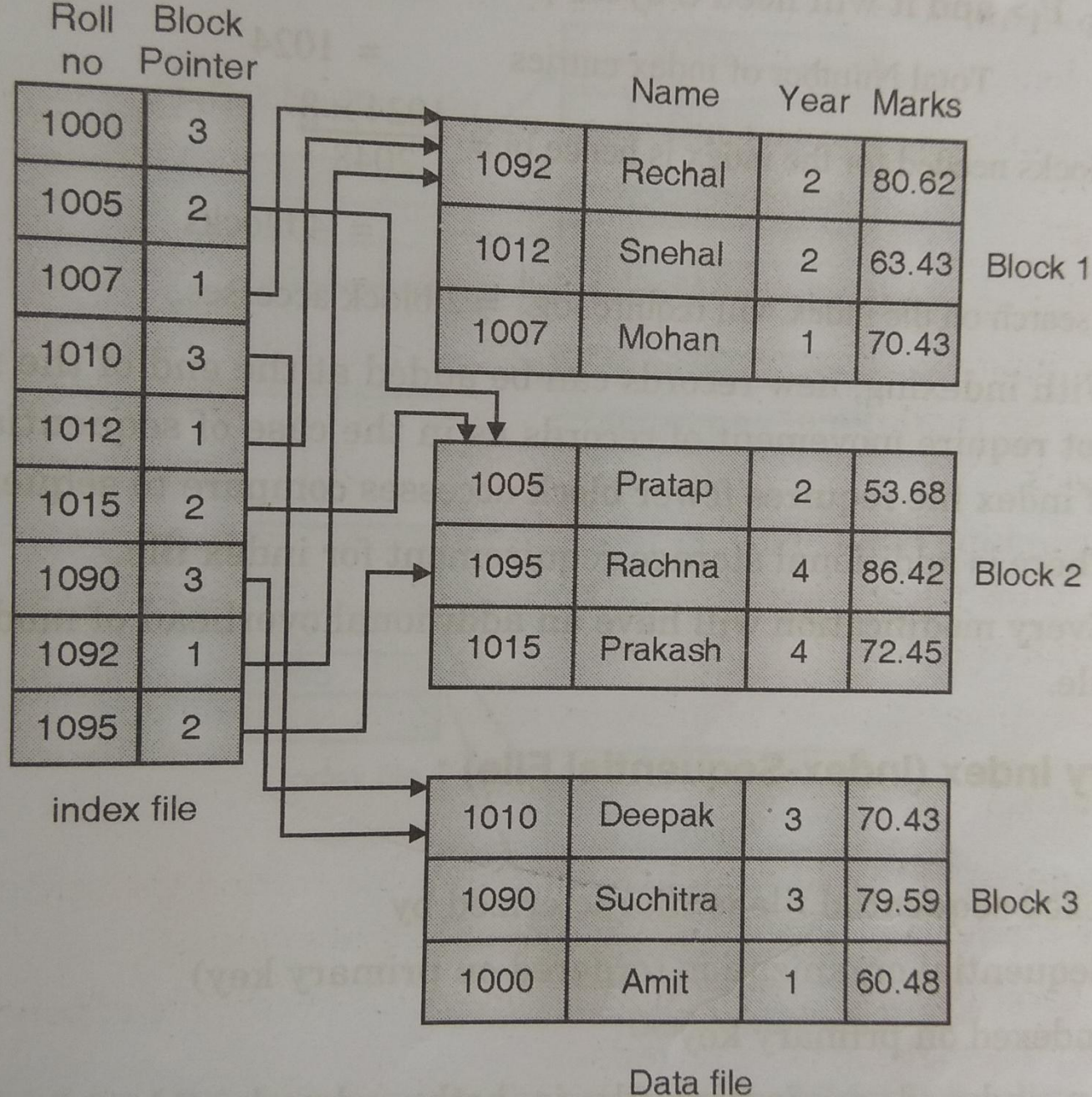


Fig. 7.3.2 : A simple index file

- **Advantages & Disadvantages:**
- Sequential file can be searched effectively on ordering key,

But when the search is based on non-key attribute, this approach is not advantageous.
- Multiple indexes can be maintained for each type of field to be used for searching. Thus indexing provides better searching.
- An index file requires less storage space than the main file.

- A binary search on sequential file with $r=1024$ records of fixed length with record size $R=128$ bytes stored on disk with block size $B = 2048$ bytes.
- No. of blocks 'b' required to store the file =
 $(1024 \times 128) / 2048 = 64$
- No. of block accesses for searching a record = $\log_2 64 \cong 6$.
- Suppose we want to construct an index on a key field that is $V=4$ bytes long and the block pointer is $P=4$ bytes long.
- A record of an index file is of the form $\langle V_i, P_i \rangle$ & it will need 8 bytes per entry.

- Total no. of index entries = 1024
- No. of blocks needed for the index is hence $= (1024 \times 8) / 2048 = 4$ blocks
- Binary search on the index will require $\log_2 4 = 2$ blocks access.
- With indexing new records can be added at the end of the main file. It will not require movement of records as in case of sequential.
- Updation of index requires fewer block access.
- There is additional storage requirement for index.
- Every modification need modification of index also.

Index-sequential file organization (Primary index)

- An index-sequential file is both ordered and indexed.
- Records are organized in the sequence of a key field known as primary key.
- An index to the file is added to support random access.
- Each record in the index file consists of 2 fields- a key field & the block pointer.
- The number of records in the index file is same as no. of blocks in the main data file.

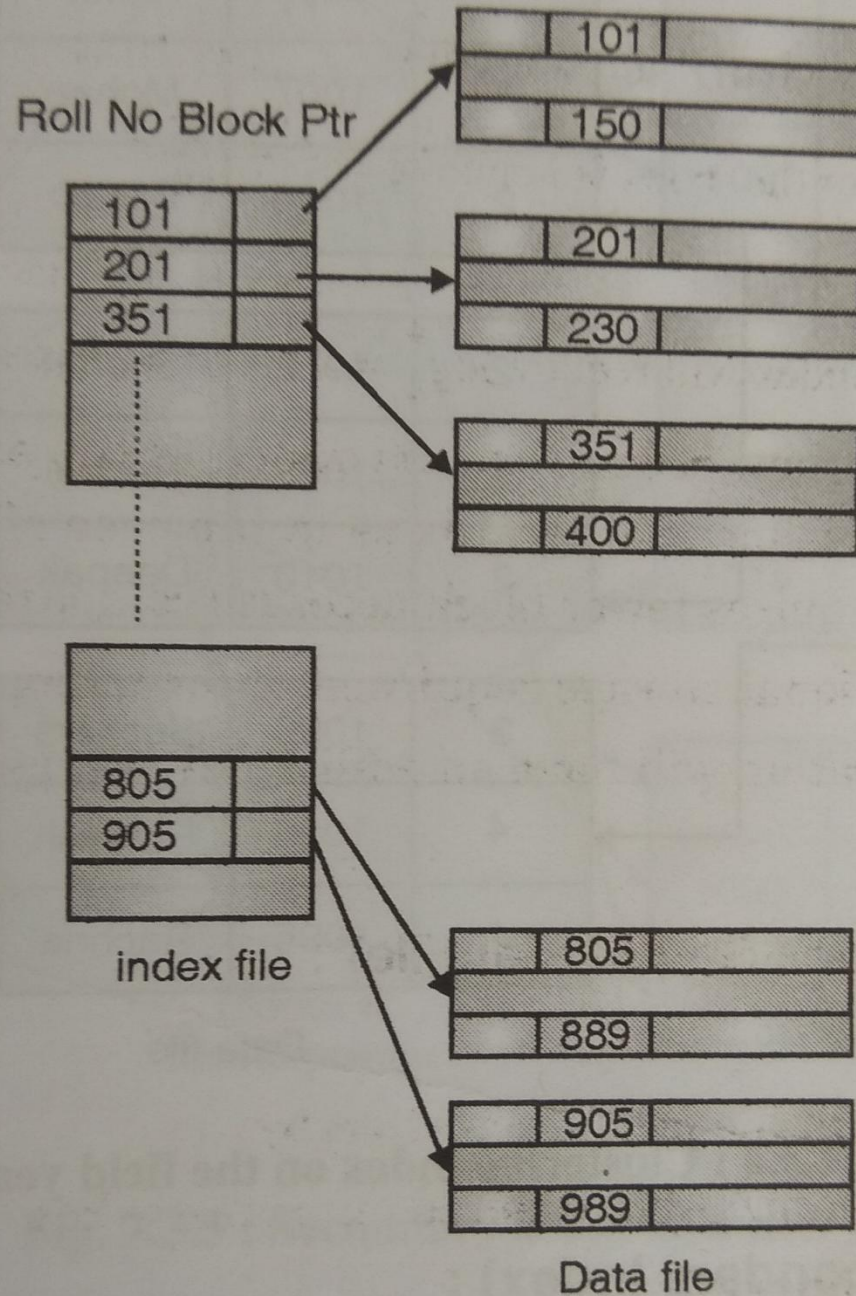


Fig. 7.3.3 : Primary index on the primary key field roll number

Multilevel index on primary key

- The basic idea behind multilevel index is to keep a very small portion of index in the memory.
 - Binary search can be used on the index file.
 - If the index file is too large with b blocks , binary search on index will need $\log_2 b$ block accesses.
- This may not be acceptable in many cases.
- Number of block accesses can be reduced by having multilevel index.
- Outer index is used to locate a block containing inner index.
- A block containing inner index give the address of the block containing the said record.
- A record can be retrieved in 3 block accesses.

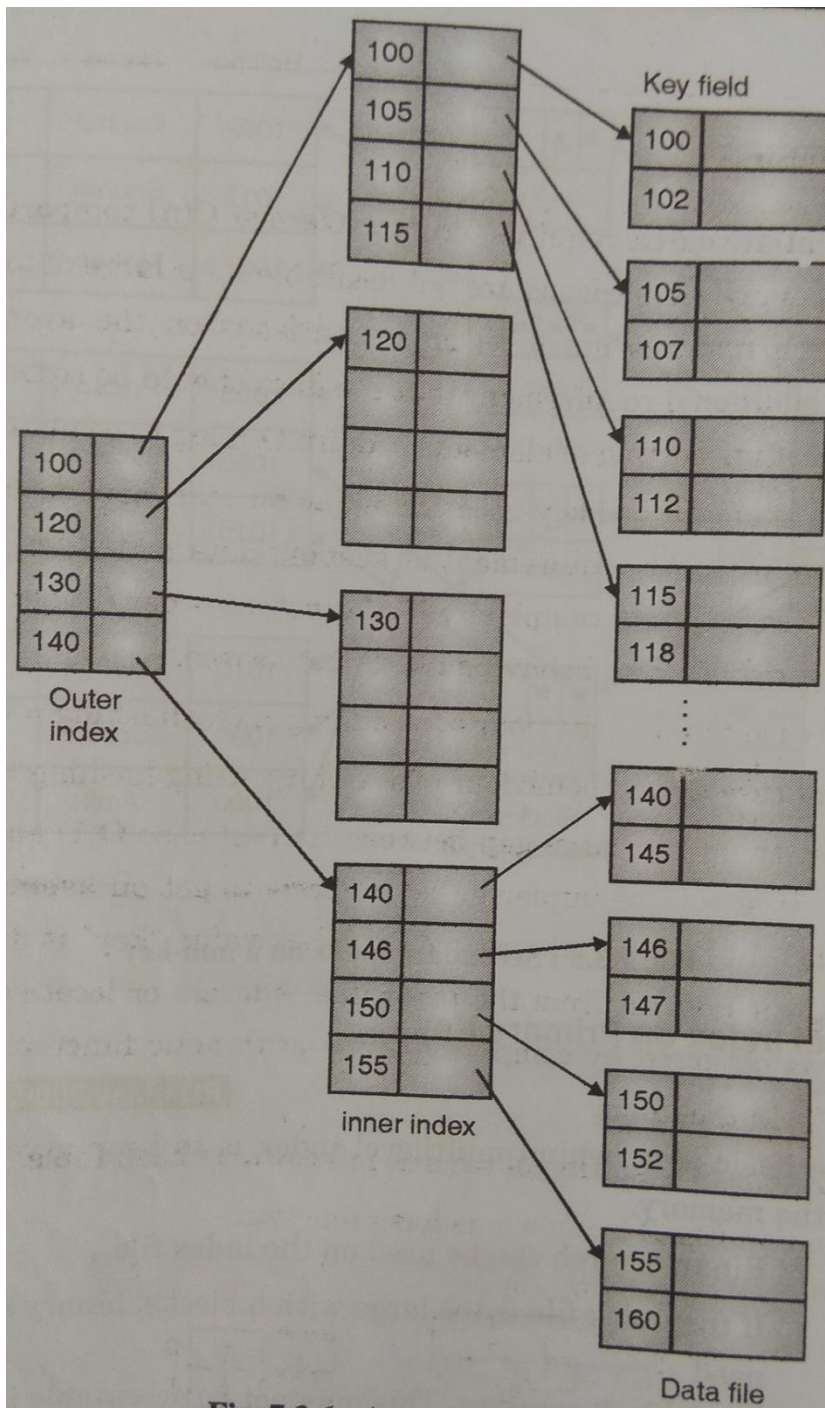


Fig. 7.3.6 : A two level index

Hashing

- Sequential Search requires , on the average $O(n)$ comparisons to locate an element.
- So many comparisons are not desirable for a large databases.
- Binary search requires fewer comparisons but database needs to be sorted.
- There is another widely used technique for storing data called 'hashing' .
- Hashing is a technique used in database management systems (DBMS) to efficiently retrieve data stored in a database.
- It does not require keeping data sorted.
- Its best case timing complexity is of constant order ($O(1)$).

- In hashing, a mathematical function is applied to the data (known as the "hash key") to generate a fixed-length "hash value." The hash value is then used to index the data in the database, allowing for fast and efficient retrieval.
- Hashing is commonly used in DBMS for indexing large datasets, as it allows for fast access to specific data elements based on the hash value, rather than having to search the entire dataset.

Terms

- Hash Key: A hash key, also known as a key or an input, is a piece of data that is used as an input to a hash function. The hash function takes the hash key as input and generates a unique hash value as output. The hash value is then used as an index to store or retrieve data in a database.
- A hash key can be any type of data, such as a number, string, or record. In order for the hash function to work effectively, the hash key must be able to produce unique hash values for each unique input, and it must be deterministic, meaning that for a given input, it should always generate the same output.
- For example, if a database contains records of employees, the hash key might be the employee ID or the combination of the first and last name of an employee.

Terms

- Hash Function: A hash function is a mathematical function that takes an input (known as a hash key) and produces an output (known as a hash value) that is a fixed length.
- Hash functions are used to index data stored in a database.
- The hash function maps the input data (the hash key) to a unique hash value, which is then used as an index to store or retrieve data in the database.

Properties of hash function

- **Deterministic:** For a given input, the hash function should always generate the same output.
- **Unique:** The hash function should map each unique input to a unique hash value.
- **Efficient:** The hash function should be fast and efficient, so that it can be applied to large datasets in a timely manner.
- **Fixed Length:** The output of the hash function should have a fixed length, regardless of the size or complexity of the input.

Example

- Search Key (24,52,91,67,48,83)
- Hash Function (kmod10, kmodn, mid square, folding method etc.)
- Hash Function is Kmod10
- So, $24 \bmod 10 = 4$
- $52 \bmod 10 = 2$
- If 62 is the next value, there is already a value at 2nd

Location, this situation is called collision.

Hash Table

0	
1	91
2	52
3	83
4	24
5	
6	
7	67
8	48
9	

Open hashing or external hashing

- Open hashing, also known as separate chaining, is a technique used in hash tables to resolve collisions, which occur when two or more elements are mapped to the same index in the hash table.
- In open hashing, each index in the hash table is associated with a linked list, and elements that are mapped to the same index are stored in the same linked list.
- When an element is to be inserted into the hash table, the hash function is applied to the element's key to generate an index. If the index is empty, the element is inserted directly. If the index is already occupied, the element is inserted into the linked list associated with that index.

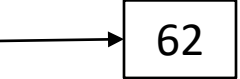
Example

- Search Key (24,52,91,67,48,83)
- Hash Function (kmod10, kmodn, mid square, folding method etc.)
- Hash Function is Kmod10
- So, $24 \bmod 10 = 4$
- $52 \bmod 10 = 2$
- If 62 is the next value, there is already a value at 2nd

Location, this situation is called collision.

Hash Table

0	
1	91
2	52
3	83
4	24
5	
6	
7	67
8	48
9	



- When an element is to be inserted into the hash table, the hash function is applied to the element's key to generate an index. If the index is empty, the element is inserted directly. If the index is already occupied, the element is inserted into the linked list associated with that index.
- When searching for an element in the hash table, the hash function is applied to the search key to generate the index. The linked list associated with the index is then searched for the desired element. If the element is found, it is returned. If the element is not found, the search is concluded and the result is reported as not found.
- Open hashing provides an efficient way to handle collisions in hash tables and is often used in database management systems. The main advantage of open hashing is that it allows for a dynamic and flexible hash table, as the linked lists can grow or shrink as needed to accommodate changes in the number of elements stored in the hash table.

Closed hashing

- Closed hashing requires that the hash table has a fixed size and does not use linked lists, so it can be more memory-efficient than open hashing.
- However, its efficiency can be affected by the quality of the hash function and the collision resolution strategy used.
- In particular, if the hash function produces a large number of collisions, or if the collision resolution strategy is not effective in finding unoccupied slots, performance can degrade significantly.

Balanced Tree (B-Tree)

- ❑ Multilevel indexing

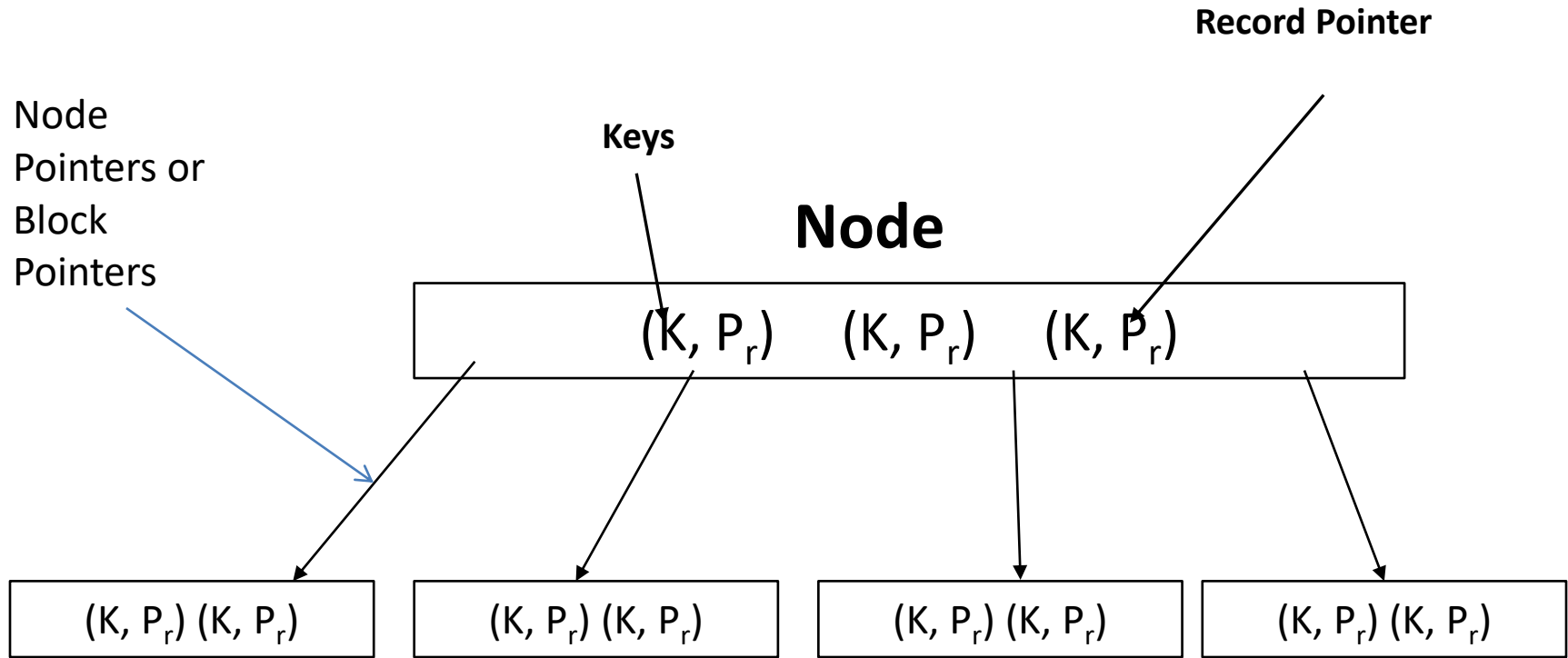
- ❑ Terminologies:

- ❖ Block/Node Pointer

- ❖ Record Pointer

- ❖ Order of Tree

- ❖ Keys



If 'n' = number of keys, then 'n+1' will be the number of children or node pointers. (e.g. n=3, pointers=4 in above example)

Order of tree is the maximum number of children a node can have. (e.g. 4 in above example)

Properties of B - Tree

1) Root:

- Children between '2' and 'P'.
- Keys between '1' and 'P-1'

Let order of tree $P=4$

Node	Min Key	Max Key
1)Root Node	1	3
2) Internal Node	1	3
3) Leaf Node	1	3

2) Internal Nodes:

- Children between $\text{ceiling}(P/2)$ and P
- Keys between $\text{ceiling}(P/2) - 1$ and $P-1$.

3) Leaf Node: All should be at the same level

- Keys between $\text{ceiling}(P/2) - 1$ and $P-1$.

Finding Order of B-Tree

- Q) Given $K=12$ Bytes, Block Size= 280 Bytes, P_r =Pointer to the record= 8 Bytes, P_b =pointer to the block = 10 Bytes.
- Answer → let order be 'n'

$$\text{So, } n * P_b + (n-1) * (K+P_r) \leq 280 \text{ Bytes}$$

Solving above equation,

$$n*(10) + (n-1) * (12+8) \leq 280$$

$$10n + (n-1) * (20) \leq 280$$

$$10n + (20n - 20) \leq 280$$

$$30n - 20 \leq 280$$

$$30n \leq 280 + 20$$

$$30n \leq 300$$

$$n \leq 10$$

Insertion in B- Tree

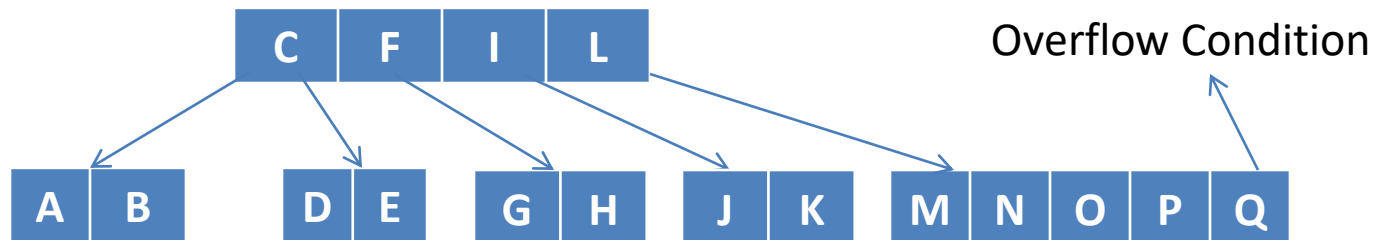
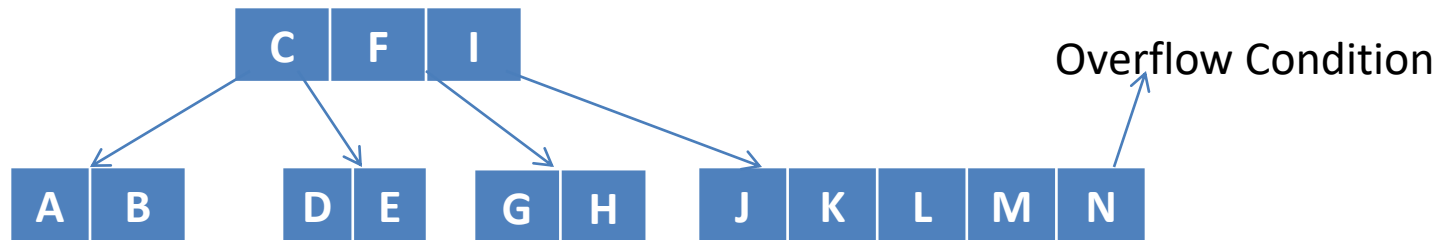
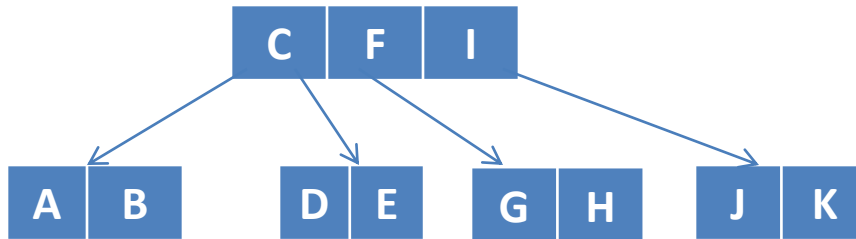
- Key \rightarrow A,B,C,D,E,,G,H,I,J,K,L,M,N,O,P,Q
- $P=5$, Therefore min no. of keys = $\text{ceiling}(P/2)-1=2$
- Max Keys = $P-1 = 4$.

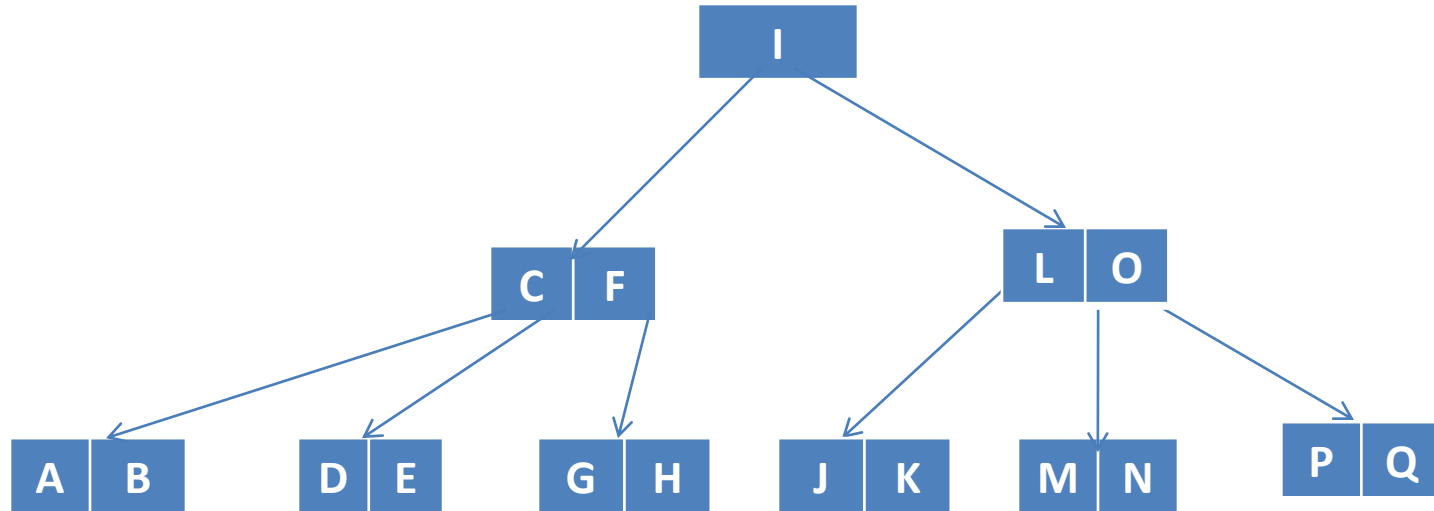
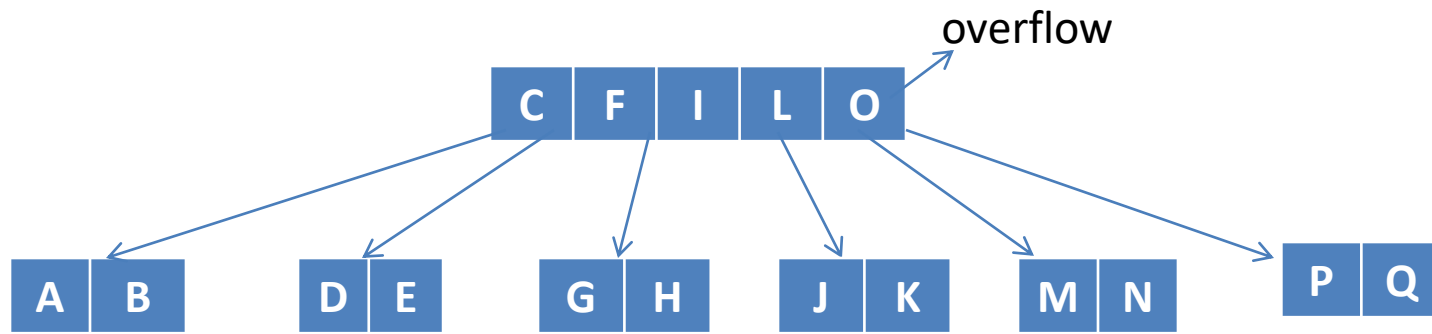


- Here $n=5$, therefore $\text{ceiling}(n/2) = 5/2=3$, 3rd element is 'C'



Insertion in B- Tree





Balanced Tree (B-Tree)

- A B-tree is a data structure used in database management systems to efficiently store and search large amounts of data.
- It is a balanced tree, which means that all the leaves are at the same level, and it has a variable number of keys and children per node.
- In a B-tree, each node can contain a number of keys and corresponding pointers to other nodes.

- The keys in each node are sorted in ascending order, and the pointers point to nodes that contain keys within certain ranges.
- The root of the B-tree is the node that contains the smallest key, and the leaves are the nodes that do not have any children.
- B-trees are used to store data in a way that allows for efficient search, insertion, and deletion operations.
- They are particularly useful for indexing data in a database, as they provide fast access to records based on their key values.

- B-trees can also be used to store and manage disk-based data, as they can minimize the number of disk I/O operations required to access and modify data.
- B-trees are commonly used in database systems to index large tables and provide fast search performance.
- They can be used for a variety of operations, including range queries, exact matches, and partial matches.
- The design of B-trees allows them to handle large amounts of data efficiently, and their performance can be optimized by adjusting the parameters that determine the maximum size of each node in the tree.