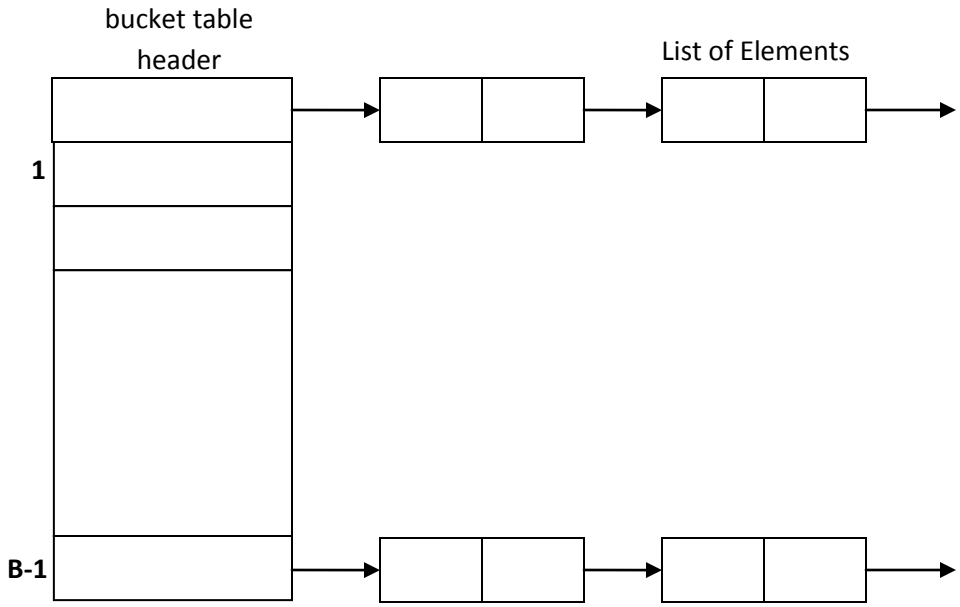
2. Close hashing or internal hashing

Closed or internal hashing, uses a fixed space for storage and thus limits the size of hash table.

1. Open Hashing Data Structure



The open hashing data organization

- The basic idea is that the records [elements] are partitioned into B classes, numbered 0,1,2 … B-l
- A Hashing function f(x) maps a record with key n to an integer value between 0 and B-l.
- Each bucket in the bucket table is the head of the linked list of records mapped to that bucket.

2. Close Hashing Data Structure

0	b
1	
2	
3	
4	c
5	d

- A closed hash table keeps the elements in the bucket itself.
- Only one element can be put in the bucket
- If we try to place an element in the bucket f(n) and find it already holds an element, then we say that a collision has occurred.
- In case of collision, the element should be rehashed to alternate empty location f1(x), f2(x), ... within the bucket table
- In closed hashing, collision handling is a very important issue.

Hashing Functions

Characteristics of a Good Hash Function

- A good hash function avoids collisions.
- A good hash function tends to spread keys evenly in the array.
- A good hash function is easy to compute.

Different hashing functions

1. Division-Method
2. Midsquare Methods
3. Folding Method
4. Digit Analysis
5. Length Dependent Method
6. Algebraic Coding
7. Multiplicative Hashing

1. Division-Method

- In this method we use modular arithmetic system to divide the key value by some integer divisor m (may be table size).
- It gives us the location value, where the element can be placed.
- We can write,
  L = (K mod m) + 1
  where L => location in table/file
  K => key value
  m => table size/number of slots in file
- Suppose, k = 23, m = 10 then
  L = (23 mod 10) + 1= 3 + 1=4, The key whose value is 23 is placed in 4th location.

2. Midsquare Methods

- In this case, we square the value of a key and take the number of digits required to form an address, from the middle position of squared value.
- Suppose a key value is 16, then its square is 256. Now if we want address of two digits, then you select the address as 56 (i.e. two digits starting from middle of 256).

3. Folding Method

- Most machines have a small number of primitive data types for which there are arithmetic instructions.
- Frequently key to be used will not fit easily in to one of these data types
- It is not possible to discard the portion of the key that does not fit into such an arithmetic data type

- The solution is to combine the various parts of the key in such a way that all parts of the key affect for final result such an operation is termed folding of the key.
- That is the key is actually partitioned into number of parts, each part having the same length as that of the required address.
- Add the value of each parts, ignoring the final carry to get the required address.
- This is done in two ways :
    - **Fold-shifting:** Here actual values of each parts of key are added.
        - Suppose, the key is : 12345678, and the required address is of two digits,
        - Then break the key into: 12, 34, 56, 78.
        - Add these, we get 12 + 34 + 56 + 78 : 180, ignore first 1 we get 80 as location
    - **Fold-boundary:** Here the reversed values of outer parts of key are added.
        - Suppose, the key is : 12345678, and the required address is of two digits,
        - Then break the key into: 21, 34, 56, 87.
        - Add these, we get 21 + 34 + 56 + 87 : 198, ignore first 1 we get 98 as location

4. Digit Analysis

- This hashing function is a distribution-dependent.
- Here we make a statistical analysis of digits of the key, and select those digits (of fixed position) which occur quite frequently.
- Then reverse or shifts the digits to get the address.
- For example, if the key is : 9861234. If the statistical analysis has revealed the fact that the third and fifth position digits occur quite frequently, then we choose the digits in these positions from the key. So we get, 62. Reversing it we get 26 as the address.

5. Length Dependent Method

- In this type of hashing function we use the length of the key along with some portion of the key j to produce the address, directly.
- In the indirect method, the length of the key along with some portion of the key is used to obtain intermediate value.

6. Algebraic Coding

- Here a n bit key value is represented as a polynomial.
- The divisor polynomial is then constructed based on the address range required.
- The modular division of key-polynomial by divisor polynomial, to get the address-polynomial.
- Let f(x) = polynomial of n bit key = $a_1 + a_2x + \ldots\ldots + a_nx^{n-1}$
- d(x) = divisor polynomial = $x^1 + d_1 + d_2x + \ldots + d_1x^{1-1}$
- then the required address polynomial will be f(x) mod d(x)

7. Multiplicative Hashing

- This method is based on obtaining an address of a key, based on the multiplication value.

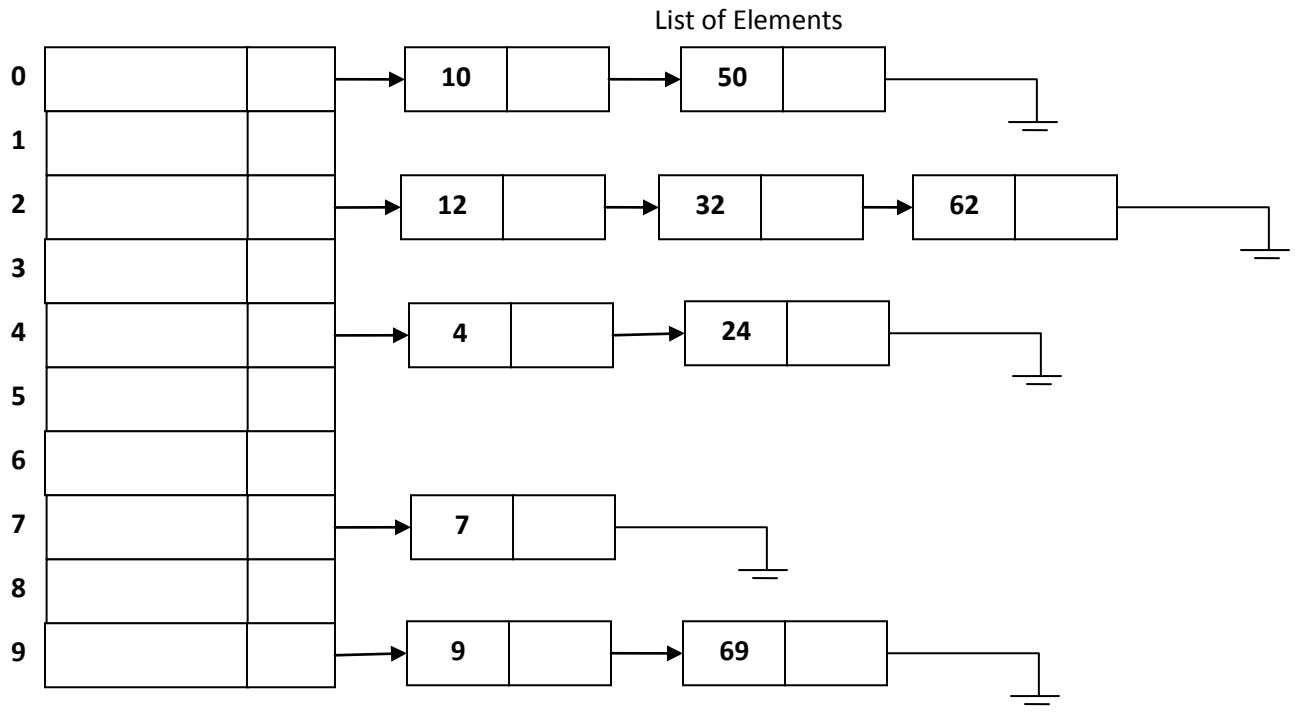
- If k is the non-negative key, and a constant c, (0 < c < 1), compute kc mod 1, which is a fractional part of kc.
- Multiply this fractional part by m and take a floor value to get the address
- $\lfloor m\ (kc\ mod\ 1) \rfloor$
- 0 < h (k) < m

Collision Resolution Strategies (Synonym Resolution)

- Collision resolution is the main problem in hashing.
- If the element to be inserted is mapped to the same location, where an element is already inserted then we have a collision and it must be resolved.
- There are several strategies for collision resolution. The most commonly used are :
    1. **Separate chaining** - used with open hashing
    2. **Open addressing** - used with closed hashing

1. Separate chaining

- In this strategy, a separate list of all elements mapped to the same value is maintained.
- Separate chaining is based on collision avoidance.
- If memory space is tight, separate chaining should be avoided.
- Additional memory space for links is wasted in storing address of linked elements.
- Hashing function should ensure even distribution of elements among buckets; otherwise the timing behavior of most operations on hash table will deteriorate.

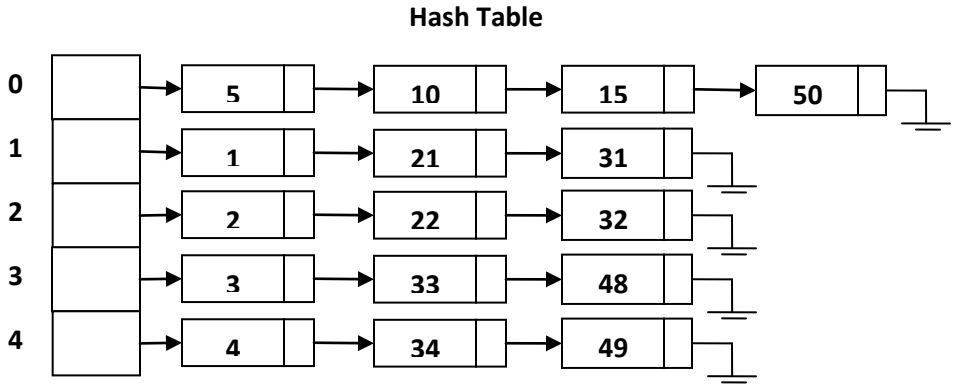


A Separate Chaining Hash Table

Example : The integers given below are to be inserted in a hash table with 5 locations using chaining to resolve collisions. Construct hash table and use simplest hash function. 1, 2, 3, 4, 5, 10, 21, 22, 33, 34, 15, 32, 31, 48, 49, 50

An element can be mapped to a location in the hash table using the mapping function **key % 10**.

Hash Table Location	Mapped element
0	5, 10, 15, 50
1	1, 21, 31
2	2, 22, 32
3	3, 33, 48
4	4, 34, 49



2. Open Addressing

- Separate chaining requires additional memory space for pointers. Open addressing hashing is an alternate method of handling collision.
- In open addressing, if a collision occurs, alternate cells are tried until an empty cell is found.
    a. Linear probing
    b. Quadratic probing
    c. Double hashing.

a) Linear Probing

- In linear probing, whenever there is a collision, cells are searched sequentially (with wraparound) for an empty cell.
- Fig. shows the result of inserting keys {5,18,55,78,35,15} using the hash function **(f(key)= key%10)** and linear probing strategy.

	Empty Table	After 5	After 18	After 55	After 78	After 35	After 15
0							15
1							
2							
3							
4							
5		5	5	5	5	5	5
6				55	55	55	55
7						35	35
8			18	18	18	18	18
9					78	78	78

- Linear probing is easy to implement but it suffers from "**primary clustering**"

- When many keys are mapped to the same location (clustering), linear probing will not distribute these keys evenly in the hash table. These keys will be stored in neighborhood of the location where they are mapped. This will lead to clustering of keys around the point of collision

b) Quadratic probing

- One way of reducing "primary clustering" is to use quadratic probing to resolve collision.
- Suppose the "key" is mapped to the location j and the cell j is already occupied. In quadratic probing, the location j, (j+1), (j+4), (j+9), ... are examined to find the first empty cell where the key is to be inserted.
- This table reduces primary clustering.
- It does not ensure that all cells in the table will be examined to find an empty cell. Thus, it may be possible that key will not be inserted even if there is an empty cell in the table.

c) Double Hashing

- This method requires two hashing functions f1 (key) and f2 (key).
- Problem of clustering can easily be handled through double hashing.
- Function f1 (key) is known as primary hash function.
- In case the address obtained by f1 (key) is already occupied by a key, the function f2 (key) is evaluated.
- The second function f2 (key) is used to compute the increment to be added to the address obtained by the first hash function f1 (key) in case of collision.
- The search for an empty location is made successively at the addresses f1 (key) + f2(key), f1 (key) + 2f2 (key), f1 (key) + 3f2(key),...

What is File?

- A file is a collection of records where a record consists of one or more fields. Each contains the same sequence of fields.
- Each field is normally of fixed length.
- A sample file with four records is shown below:

Name	Roll No.	Year	Marks
AMIT	1000	1	82
KALPESH	1005	2	54
JITENDRA	1009	1	75
RAVI	1010	1	79

- There are four records
- There are four fields (Name, Roll No., Year, Marks)
- Records can be uniquely identified on the field 'Roll No.' Therefore, Roll No. is the key field.
- A database is a collection of files.
- Commonly, used file organizations are :
  1. Sequential files
  2. Relative files
  3. Direct files
  4. Indexed Sequential files
  5. Index files
- Primitive Operations on a File :
  1. Creation
  2. Reading
  3. Insertion
  4. Deletion
  5. Updation
  6. Searching

Sequential Files

It is the most common type of file. In this type of file:

- A fixed format is used for record.
- All records are of the same length.
- Position of each field in record and length of field is fixed.
- Records are physically ordered on the value of one of the fields - called the ordering field.

Block 1

Name	Roll No.	Year	Marks
AMIT	1000	1	82
KALPESH	1005	2	54
JITENDRA	1009	1	75
RAVI	1010	1	79

Block 2

NILESH	1011	2	89

Some blocks of an ordered (sequential) file of students records with Roll no. as the ordering field

Advantages of sequential file over unordered files :

- Reading of records in order of the ordering key is extremely efficient.
- Finding the next record in order of the ordering key usually, does not require additional block access. Next record may be found in the same block.
- Searching operation on ordering key is must faster. Binary search can be utilized. A binary search will require $\mathbf{\log_2 b}$ block accesses where b is the total number of blocks in the file.

Disadvantages of sequential file :

- Sequential file does not give any advantage when the search operation is to be carried out on non-ordering field.
- Inserting a record is an expensive operation. Insertion of a new record requires finding of place of insertion and then all records ahead of it must be moved to create space for the record to be inserted. This could be very expensive for large files.
- Deleting a record is an expensive operation. Deletion too requires movement of records.
- Modification of field value of ordering key could be time consuming. Modifying the ordering field means the record can change its position. This requires deletion of the old record followed by insertion of the modified record.

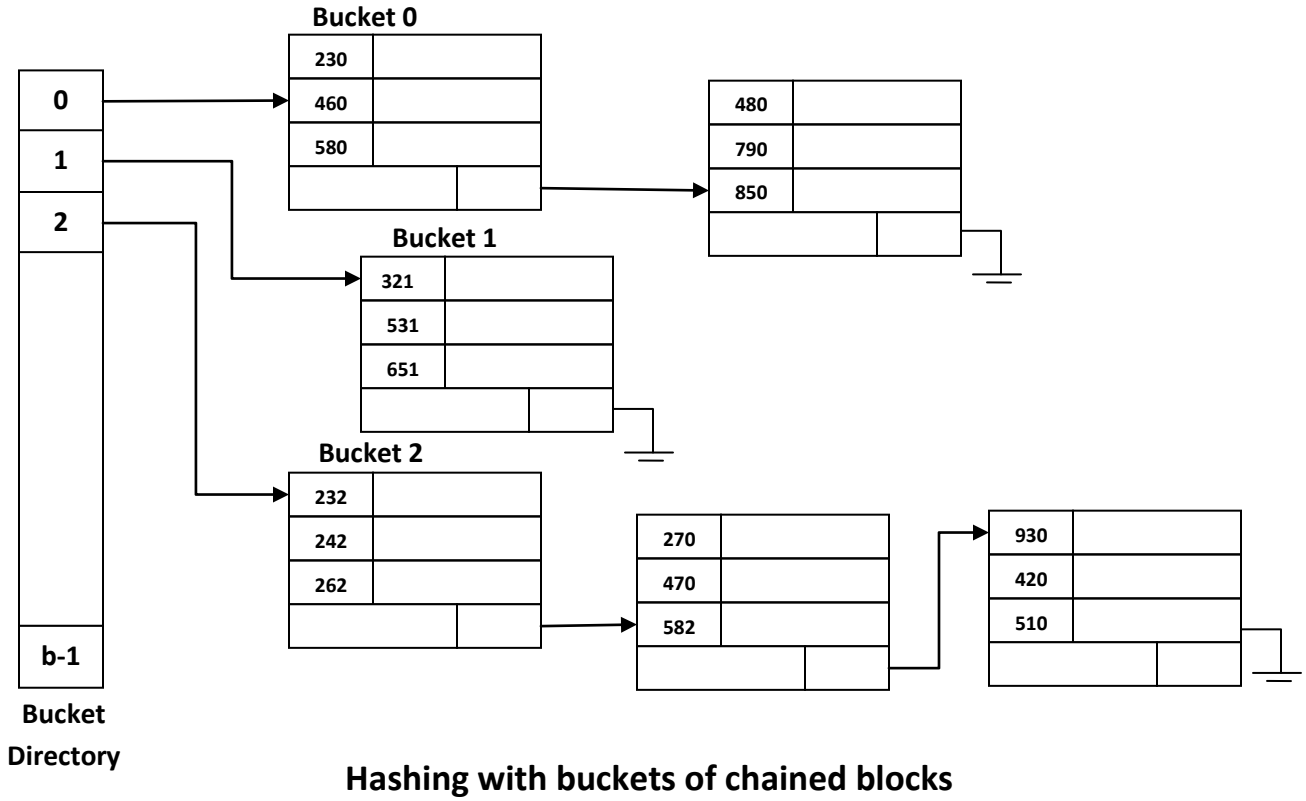
Hashing (Direct file organization):

- It is a common technique used for fast accessing of records on secondary storage.
- Records of a file are divided among buckets.
- A bucket is either one disk block or cluster of contiguous blocks.
- A hashing function maps a key into a bucket number. The buckets are numbered 0, 1,2...b-1.
- A hash function f maps each key value into one of the integers 0 through b - 1.
- If x is a key, f(x) is the number of bucket that contains the record with key x.

- The blocks making up each bucket could either be contiguous blocks or they can be chained together in a linked list.
- Translation of bucket number to disk block address is done with the help of bucket directory. It gives the address of the first block of the chained blocks in a linked list.
- Hashing is quite efficient in retrieving a record on hashed key. The average number of block accesses for retrieving a record.

$$= 1 \text{ (bucket directory)} + \frac{\boldsymbol{No\ of\ records}}{\boldsymbol{No\ of\ buckets\ x\ No\ of\ records\ per\ block}}$$

- Thus the operation is b times faster (b = number of buckets) than unordered file.
- To insert a record with key value x, the new record can added to the last block in the chain for bucket f(x). If the record does not fit into the existing block, record is stored in a new block and this new block is added at the end of the chain for bucket f(x).
- A well designed hashed structure requires two block accesses for most operations

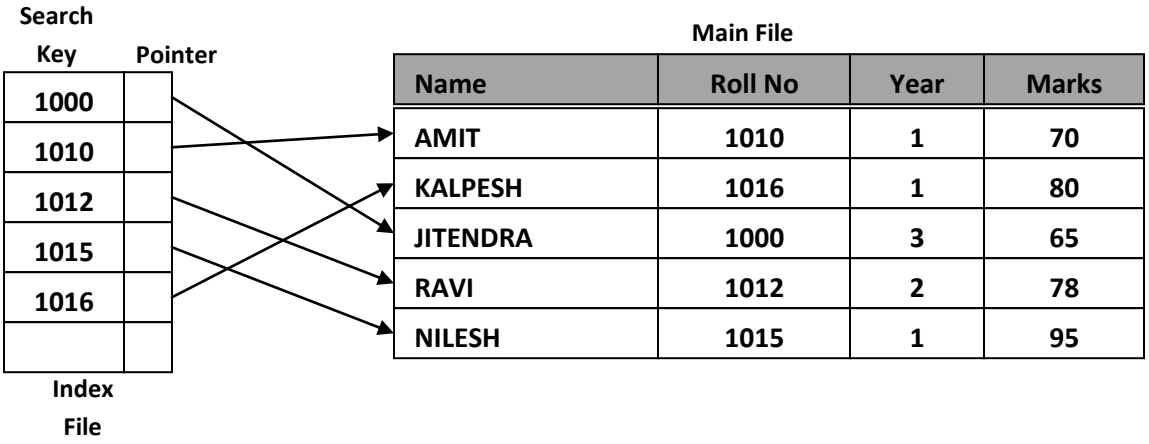


Hashing with buckets of chained blocks

Indexing

- Indexing is used to speed up retrieval of records.

- It is done with the help of a separate sequential file. Each record of in the index file consists of two fields, a key field and a pointer into the main file.
- To find a specific record for the given key value, index is searched for the given key value.
- Binary search can used to search in index file. After getting the address of record from index file, the record in main file can easily be retrieved.



- Index file is ordered on the ordering key Roll No. each record of index file points to the corresponding record. Main file is not sorted.

Advantages of indexing over sequential file:

- Sequential file can be searched effectively on ordering key. When it is necessary to search for a record on the basis of some other attribute than the ordering key field, sequential file representation is inadequate.
- Multiple indexes can be maintained for each type of field used for searching. Thus, indexing provides much better flexibility.
- An index file usually requires less storage space than the main file. A binary search on sequential file will require accessing of more blocks. This can be explained with the help of the following example. Consider the example of a sequential file with r = 1024 records of fixed length with record size R = 128 bytes stored on disk with block size B = 2048 bytes.
- Number of blocks required to store the file = $\frac{1024\ x\ 128}{2048} = 64$
- Number of block accesses for searching a record = $\log_2 64 = 6$
- 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 <V;, Pj> and it will need 8 bytes per entry.
- Total Number of index entries = 1024
- Number of blocks b' required to store the file = $\frac{1024\ x\ 8}{2048} = 4$
- Number of block accesses for searching a record = $\log_2 4 = 2$

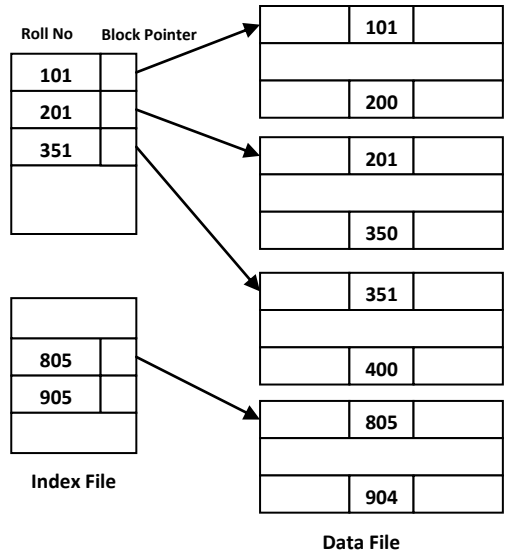
- With indexing, new records can be added at the end of the main file. It will not require movement of records as in the case of sequential file. Updation of index file requires fewer block accesses compare to sequential file

Types of Indexes:

1. Primary indexes
2. Clustering indexes
3. Secondary indexes

Primary Indexes (Indexed Sequential File):

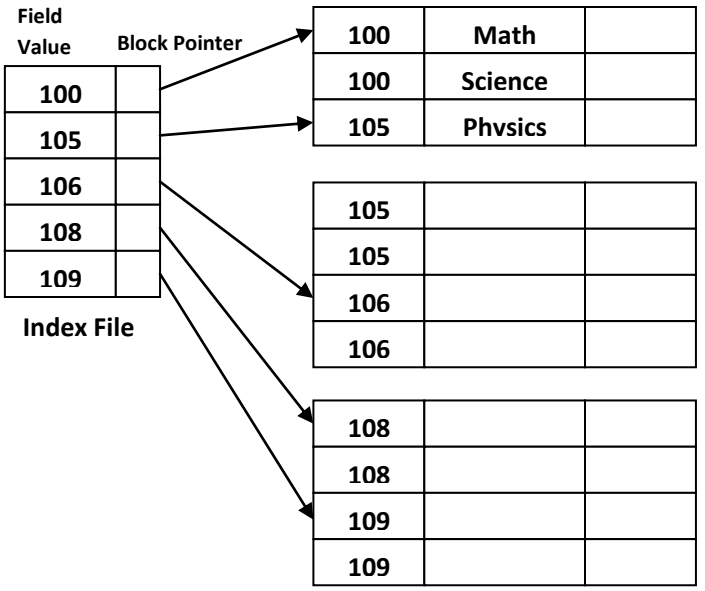
- An indexed sequential file is characterized by
    - Sequential organization (ordered on primary key)
    - Indexed on primary key
- An indexed sequential file is both ordered and indexed.
- Records are organized in sequence based on a key field, known as primary key.
- An index to the file is added to support random access. Each record in the index file consists of two fields: a key field, which is the same as the key field in the main file.
- Number of records in the index file is equal to the number of blocks in the main file (data file) and not equal to the number of records in the main file (data file).
- To create a primary index on the ordered file shown in the Fig. we use the rollno field as primary key. Each entry in the index file has rollno value and a block pointer. The first three index entries are as follows.
    - <101, address of block 1>
    - <201, address of block 2>
    - <351, address of block 3>
- Total number of entries in index is same as the number of disk blocks in the ordered data file.
- A binary search on the index file requires very few block accesses



Primary Index on ordering key field roll number

Clustering Indexes

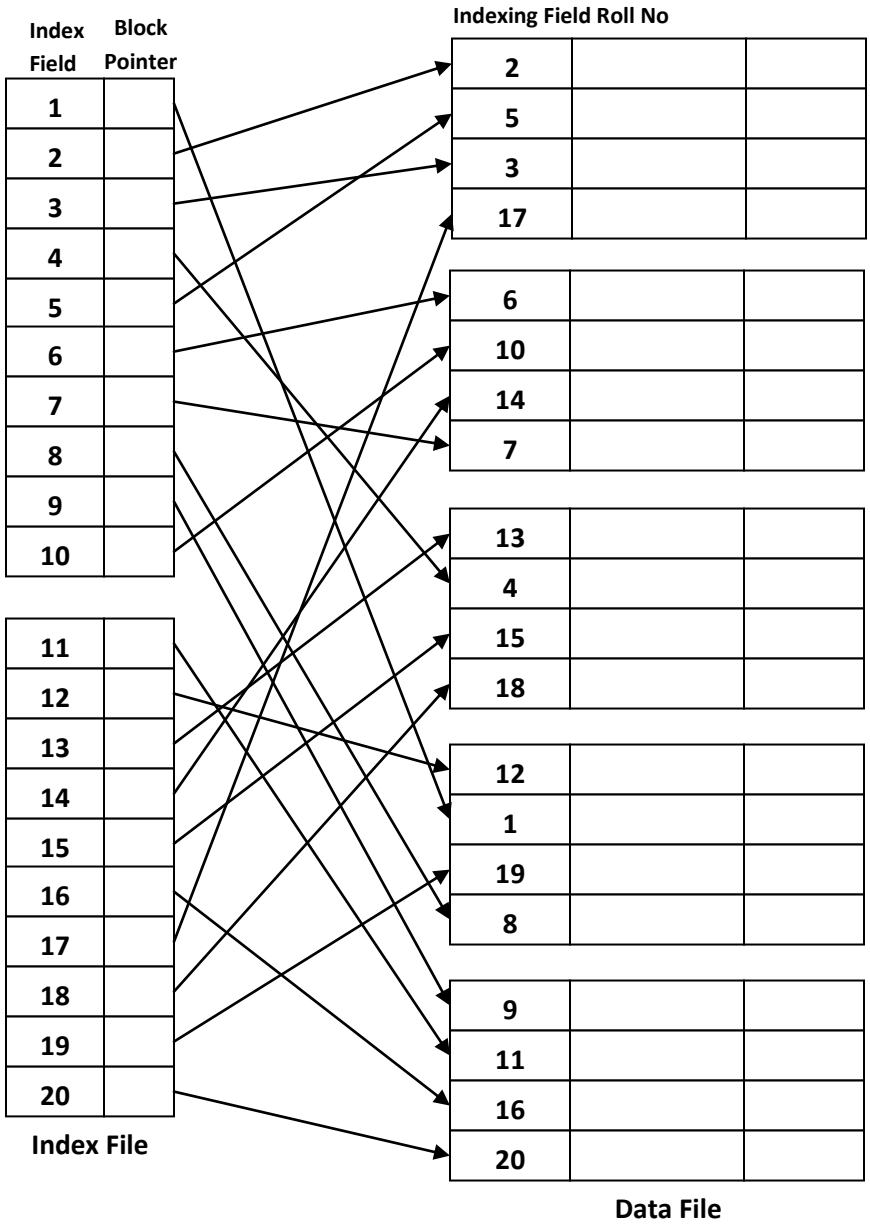
- If records of a file are ordered on a non-key field, we can create a different type of index known as clustering index.
- A non-key field does not have distinct value for each record.
- A Clustering index is also an ordered file with two fields.



Example of clustering index on roll no

Secondary indexes (Simple Index File)

- While the hashed, sequential and indexed sequential files are suitable for operations based on ordering key or the hashed key. Above file organizations are not suitable for operations involving a search on a field other than ordering or hashed key.
- If searching is required on various keys, secondary indexes on these fields must be maintained. A secondary index is an ordered file with two fields.
    - Some non-ordering field of the data file.
    - A block pointer
- There could be several secondary indexes for the same file.
- One could use binary search on index file as entries of the index file are ordered on secondary key field. Records of the data files are not ordered on secondary key field.
- A secondary index requires more storage space and longer search time than does a primary index.
- A secondary index file has an entry for every record whereas primary index file has an entry for every block in data file.
- There is a single primary index file but the number of secondary indexes could be quite a few.



A secondary index on a non-ordering key field