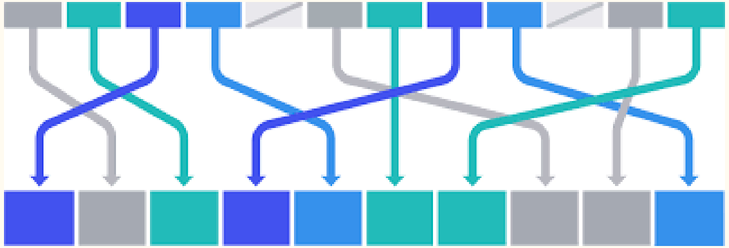
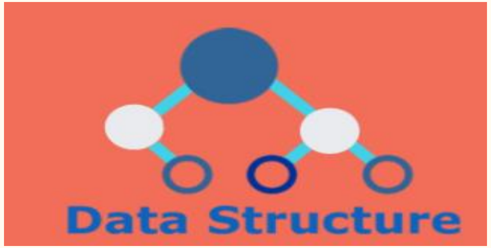
CHAPTER 4
HASHING AND FILE STRUCTURE





SUBJECT:DATA STRUCTURE
CODE:3130702

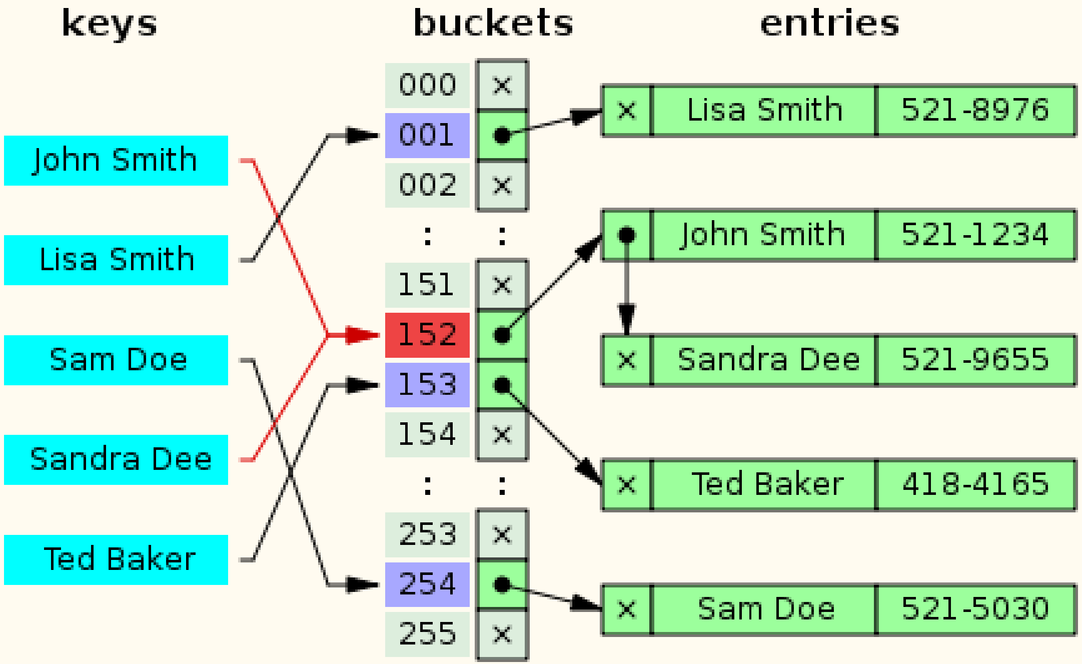
PREPARED BY:
ASST.PROF.PARAS NARKHEDE
(CSE DEPARTMENT,ACET)

HASHING

WHAT IS HASHING?

- **Sequential search** requires, on the average **O(n) comparisons** to **locate an element**, so many comparisons are not desirable for a large database of elements.
- **Binary search** requires much fewer comparisons on the average **O (log n)** but there is an additional requirement that the **data should be sorted**. Even with best sorting algorithm, sorting of elements require O(n log n) comparisons.
- There is **another** widely used **technique** for **storing of data** called **hashing**. It does away with the requirement of keeping data sorted (as in binary search) and its best case timing complexity is of constant order O(1). In its worst case, hashing algorithm starts behaving like linear search.
- **Best case** timing **behavior** of searching using hashing = **O(1)**
- **Worst case** timing Behavior of searching using hashing = **O(n)**

THE HASH TABLE



keys

John Smith

Lisa Smith

Sam Doe

Sandra Dee

Ted Baker

buckets

000 ×

001 ●

002 ×

: :

151 ×

152 ●

153 ●

154 ×

: :

253 ×

254 ●

255 ×

entries

× Lisa Smith 521-8976

● John Smith 521-1234

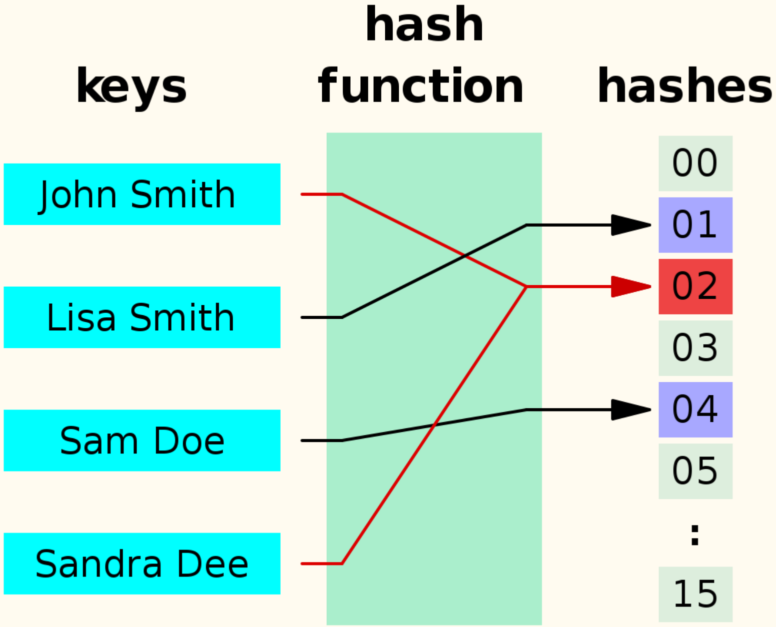
× Sandra Dee 521-9655

× Ted Baker 418-4165

× Sam Doe 521-5030

HASHING FUNCTION

Hash function is a function which is applied on a key by which it produces an integer, which can be used as an address of hash table. Hence one can use the same hash function for accessing the data from the hash table. In this the integer returned by the hash function is called hash key.



AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

TYPES OF HASHING FUNCTION

Different hashing functions

- Division-Method
- Mid square Methods
- Folding Method
- Digit Analysis
- Length Dependent Method
- Algebraic Coding
- Multiplicative Hashing

DIVISION METHOD

In this the hash function is dependent upon the remainder of a division. For example:-if the record 52,68,99,84 is to be placed in a hash table and let us take the table size is 10.

Then:

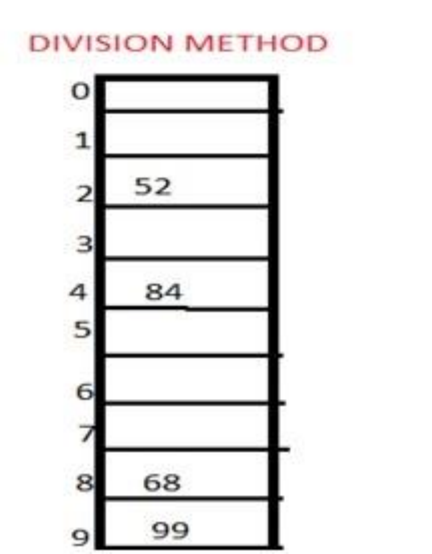
h(key)=record% table size.

2=52%10

8=68%10

9=99%10

4=84%10



DIVISION METHOD

0	
1	
2	52
3	
4	84
5	
6	
7	
8	68
9	99

MID SQUARE METHOD

In this method firstly key is squared and then mid part of the result is taken as the index. For example: consider that if we want to place a record of 3101 and the size of table is 1000. So 3101*3101=9616201 i.e. **h (3101) = 162 (middle 3 digit)**

DIGIT FOLDING METHOD

In this method the key is divided into separate parts and by using some simple operations these parts are combined to produce a hash key. For example: consider a record of 12465512 then it will be divided into parts i.e. 124, 655, 12. After dividing the parts combine these parts by adding it.

$$H(key)=124+655+12$$

$$=791$$

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,

The key is : **9861234**

If the statistical analysis has revealed the fact that the **third** and **fifth** position digits occur quite frequently,

We **choose** the **digits** in **these positions** from the key

So we get, **62**. **Reversing** it we get **26 as the address**

LENGTH DEPENDENT METHOD

In this type of hashing function **we use the length of the key** along **with some portion of the key** 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**.

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**2**x + ……. + a**n**xn**-**1

d(x) = divisor polynomial = d**1** + d**2**x + …. + d**n**xn**-**1

Required **address** polynomial will be **f(x) mod d(x)**

MULTI CAPTIVE 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**

CHARACTERISTIC OF GOOD HASHING FUNCTION

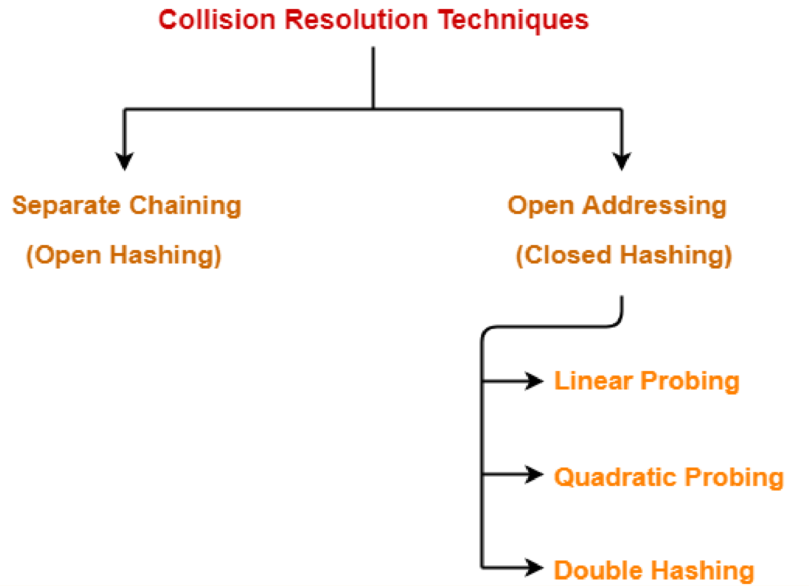
1. The hash function should generate different hash values for the similar string.
2. The hash function is easy to understand and simple to compute.
3. The hash function should produce the keys which will get distributed, uniformly over an array.
4. A number of collisions should be less while placing the data in the hash table.
5. The hash function is a perfect hash function when it uses all the input data

COLLISION

It is a situation in which the hash function returns the same hash key for more than one record, it is called as collision. Sometimes when we are going to resolve the collision it may lead to a overflow condition and this overflow and collision condition makes the poor hash function.

COLLISION RESOLUTION TECHNIQUE

If there is a problem of collision occurs then it can be handled by apply some technique. These techniques are called as collision resolution techniques. There are generally four techniques which are described below.



Collision Resolution Techniques

Separate Chaining

(Open Hashing)

Open Addressing

(Closed Hashing)

Linear Probing

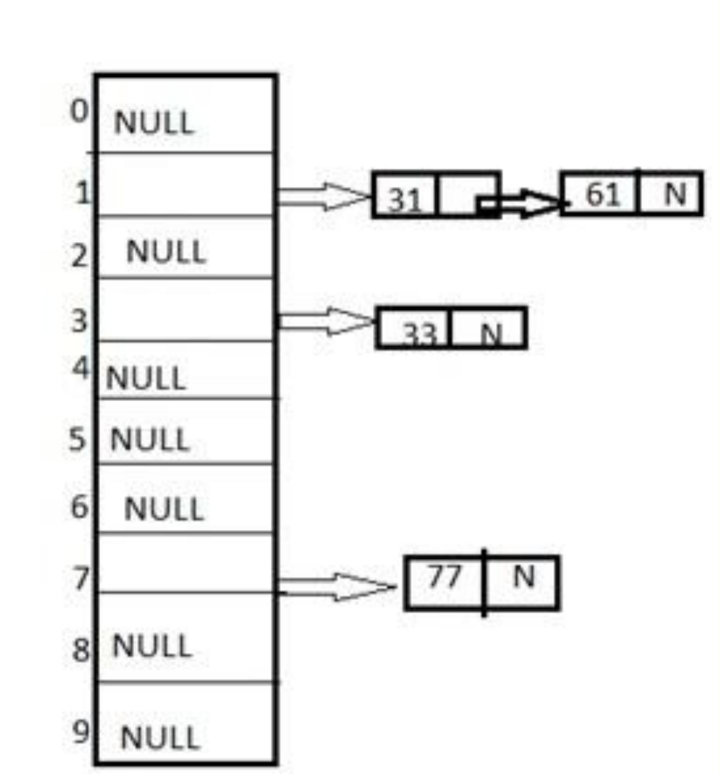
Quadratic Probing

Double Hashing

CHAINING

It is a method in which additional field with data i.e. chain is introduced. A chain is maintained at the home bucket. In this when a collision occurs then a linked list is maintained for colliding data.

Example: Let us consider a hash table of size 10 and we apply a hash function of H(key)=key % size of table. Let us take the keys to be inserted are 31,33,77,61. In the above diagram we can see at same bucket 1 there are two records which are maintained by linked list or we can say by chaining method.



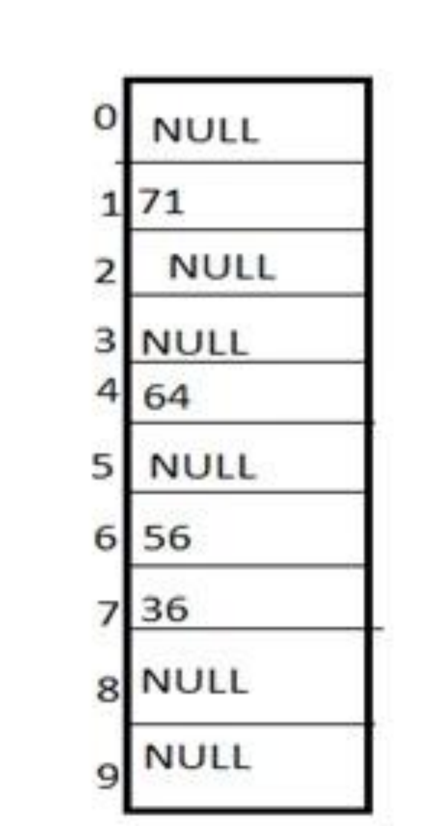
AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

LINEAR PROBING

It is very easy and simple method to resolve or to handle the collision. In this collision can be solved by placing the second record linearly down, whenever the empty place is found. In this method there is a problem of clustering which means at some place block of a data is formed in a hash table.

Example: Let us consider a hash table of size 10 and hash function is defined as H(key)=key % table size. Consider that following keys are to be inserted that are 56,64,36,71.

In this diagram we can see that 56 and 36 need to be placed at same bucket but by linear probing technique the records linearly placed downward if place is empty i.e. it can be seen 36 is placed at index 7.

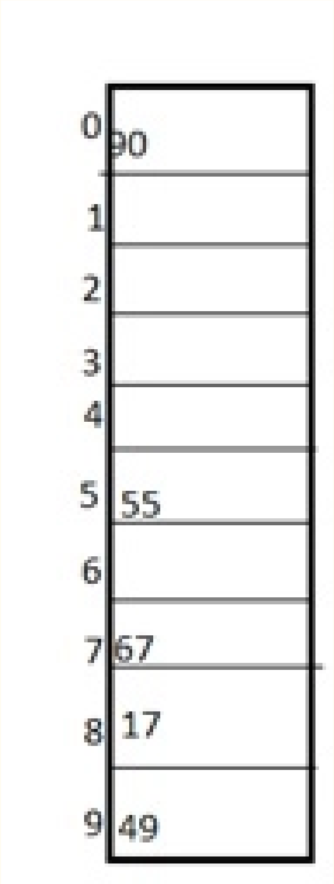


0	NULL
1	71
2	NULL
3	NULL
4	64
5	NULL
6	56
7	36
8	NULL
9	NULL

QUADRATIC PROBING

This is a method in which solving of clustering problem is done. In this method the hash function is defined by the H(key)=(H(key)+x*x)%table size. Let us consider we have to insert following elements that are:-67, 90,55,17,49.

In this we can see if we insert 67, 90, and 55 it can be inserted easily but at case of 17 hash function is used in such a manner that :-(17+0*0)%10=17 (when x=0 it provide the index value 7 only) by making the increment in value of x. let x =1 so (17+1*1)%10=8.in this case bucket 8 is empty hence we will place 17 at index 8.



0	90
1	
2	
3	
4	
5	55
6	
7	67
8	17
9	49

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

DOUBLE HASHING

It is a technique in which two hash function are used when there is an occurrence of collision. In this method 1 hash function is simple as same as division method. But for the second hash function there are two important rules which are

1. It must never evaluate to zero.
2. Must sure about the buckets, that they are probed.

The hash functions for this technique are:

H1(key)=key % table size

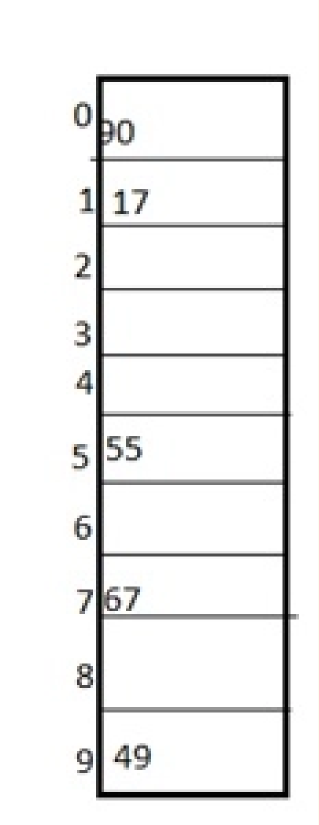
H2(key)=P-(key mod P)

Where, **p** is a prime number which should be taken smaller than the size of a hash table.

Example: Let us consider we have to insert 67, 90,55,17,49.

In this we can see 67, 90 and 55 can be inserted in a hash table by using first hash function but in case of 17 again the bucket is full and in this case we have to use the second hash function which is H2(key)=P-(key mode P) here p is a prime number which should be taken smaller than the hash table so value of p will be the 7.

i.e. H2(17)=7-(17%7)=7-3=4 that means we have to take 4 jumps for placing the 17. Therefore 17 will be placed at index 1.



0	90
1	17
2	
3	
4	
5	55
6	
7	67
8	
9	49

FILE STRUCTURE

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

CONCEPTS OF FIELD

A field is an area in a fixed or known location in a unit of data such as a record, message header, or computer instruction that has a purpose and usually a fixed size. In some contexts, a field can be subdivided into smaller fields. Here are some examples:

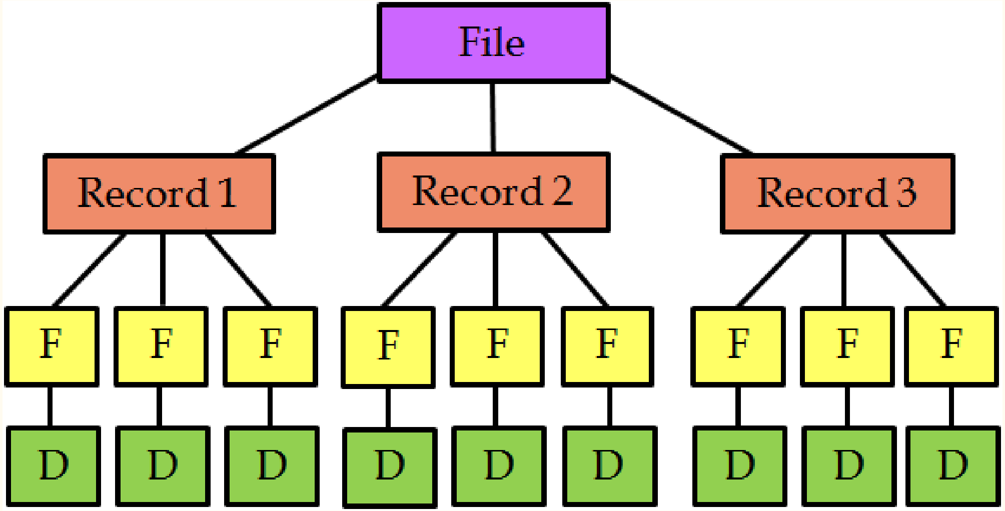
1)In a database table, a field is a data structure for a single piece of data. Fields are organized into records, which contain all the information within the table relevant to a specific entity. For example, in a table called *customer contact information*, *telephone number* would likely be a field in a row that would also contain other fields such as *street address* and *city*. The records make up the table rows and the fields make up the columns.

2) In a form that you fill out on a Web site, each box that asks you for information is a text entry field.

3) In the header of a variable-length transmission unit, a two-byte subfield in the header (which is really a field itself) could identify the length in bytes of the message

RECORDS AND FILES

File is a collection of records related to each other. The file size is limited by the size of memory and storage medium.



There are two important features of file:

1. File Activity

2. File Volatility

File activity specifies percent of actual records which proceed in a single run.

File volatility addresses the properties of record changes. It helps to increase the efficiency of disk design than tape.

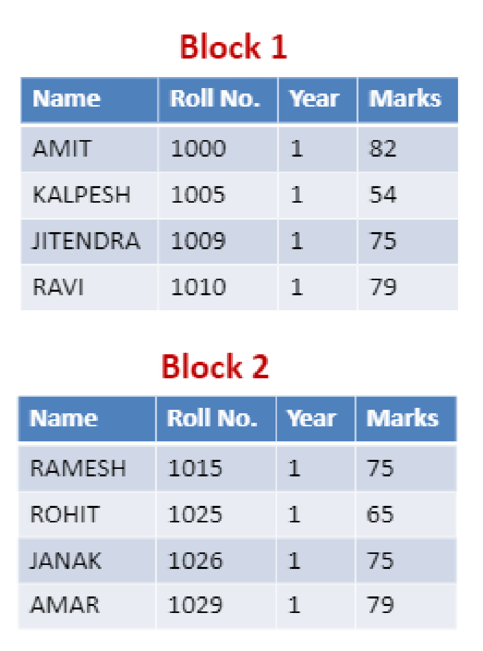
File Organization

File organization ensures that records are available for processing. It is used to determine an efficient file organization for each base relation.

For example, if we want to retrieve employee records in alphabetical order of name. Sorting the file by employee name is a good file organization. However, if we want to retrieve all employees whose marks are in a certain range, a file is ordered by employee name would not be a good file organization.

SEQUENTIAL FILE ORGANIZATION

- Storing and sorting in contiguous block within files on tape or disk is called as **sequential access file organization**.
- In sequential access file organization, all records are stored in a sequential order. The records are arranged in the ascending or descending order of a key field.
- Sequential file search starts from the beginning of the file and the records can be added at the end of the file.
- In sequential file, it is not possible to add a record in the middle of the file without rewriting the file.



Block 1

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

Block 2

Name	Roll No.	Year	Marks
RAMESH	1015	1	75
ROHIT	1025	1	65
JANAK	1026	1	75
AMAR	1029	1	79

Advantages of sequential file

- It is simple to program and easy to design.
- Sequential file is best use if storage space.

Disadvantages of sequential file

- Sequential file is time consuming process.
- It has high data redundancy.
- Random searching is not possible.

INDEXED FILE ORGANIZATION

- Indexed sequential access file combines both sequential file and direct access file organization.
- In indexed sequential access file, records are stored randomly on a direct access device such as magnetic disk by a primary key.
- This file have multiple keys. These keys can be alphanumeric in which the records are ordered is called primary key.
- The data can be access either sequentially or randomly using the index. The index is stored in a file and read into memory when the file is opened.

Advantages of Indexed sequential access file organization

- In indexed sequential access file, sequential file and random file access is possible.
- It accesses the records very fast if the index table is properly organized.
- The records can be inserted in the middle of the file.
- It provides quick access for sequential and direct processing.
- It reduces the degree of the sequential search.

Disadvantages of Indexed sequential access file organization

- Indexed sequential access file requires unique keys and periodic reorganization.
- Indexed sequential access file takes longer time to search the index for the data access or retrieval.
- It requires more storage space.
- It is expensive because it requires special software.
- It is less efficient in the use of storage space as compared to other file organizations.

RELATIVE/RANDOM FILE ORGANIZATION

A relative file is a file in which each record is identified by its ordinal position in the file (record 1, record 2 and so on). This means that records can be accessed randomly as well as sequentially:

- For sequential access, simply execute a READ or WRITE statement to access the next record in the file.
- For random access, define a data item as the relative key. Then specify in the data item the ordinal number of the record that you need to READ or WRITE.

Because records can be accessed randomly, access to relative files is fast.

Although you can declare variable length records for a relative file, this can be wasteful of disk space because the system assumes the maximum record length for all WRITE statements to the file and pads the unused character positions. This is done so that the COBOL file handling routines can quickly calculate the physical location of any record, given the record's record number in the file.

As relative files always contain fixed length records, no space is saved by specifying data compression. In fact, if data compression is specified for a relative file, it is ignored by the File Handler.

Each record in a relative file is followed by a two-byte record marker which indicates the current status of the record. The status of a record can be:

x"0D0A" - record present

x"0D00" - record deleted or never written

When you delete a record from a relative file, the record's contents are not removed immediately. The record's record marker is updated to show that it has been deleted, but the contents of the deleted record remain physically in the file until a new record is written. If you need to remove the data from the file for security reasons, follow the procedure below:

1. Use REWRITE to overwrite the record, for example with space characters.
2. Delete the record.

To define a relative file, specify ORGANIZATION IS RELATIVE in the SELECT clause for the file in your COBOL program.

To access records randomly, you must also:

- Specify ACCESS MODE IS RANDOM or ACCESS MODE IS DYNAMIC in the SELECT clause for the file
- Define a relative key in the Working-Storage Section of your program

INDEXING STRUCTURE FOR INDEX FILES

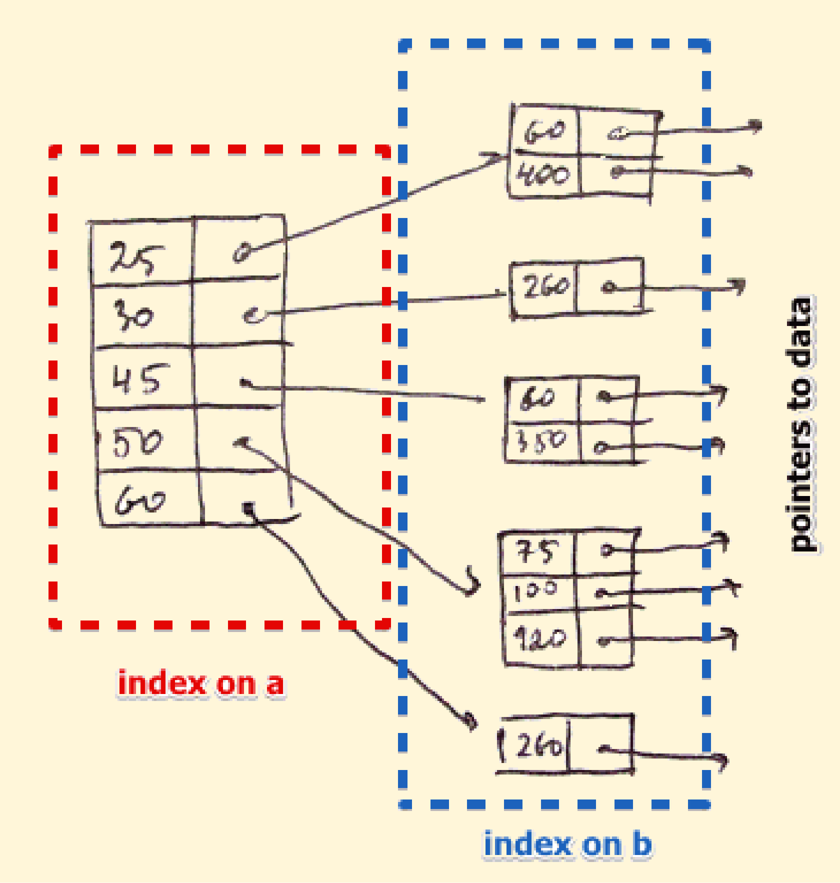
Indexing is used to speed up retrieval of records.

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.



TYPES OF INDEXING

There are primarily three methods of indexing:

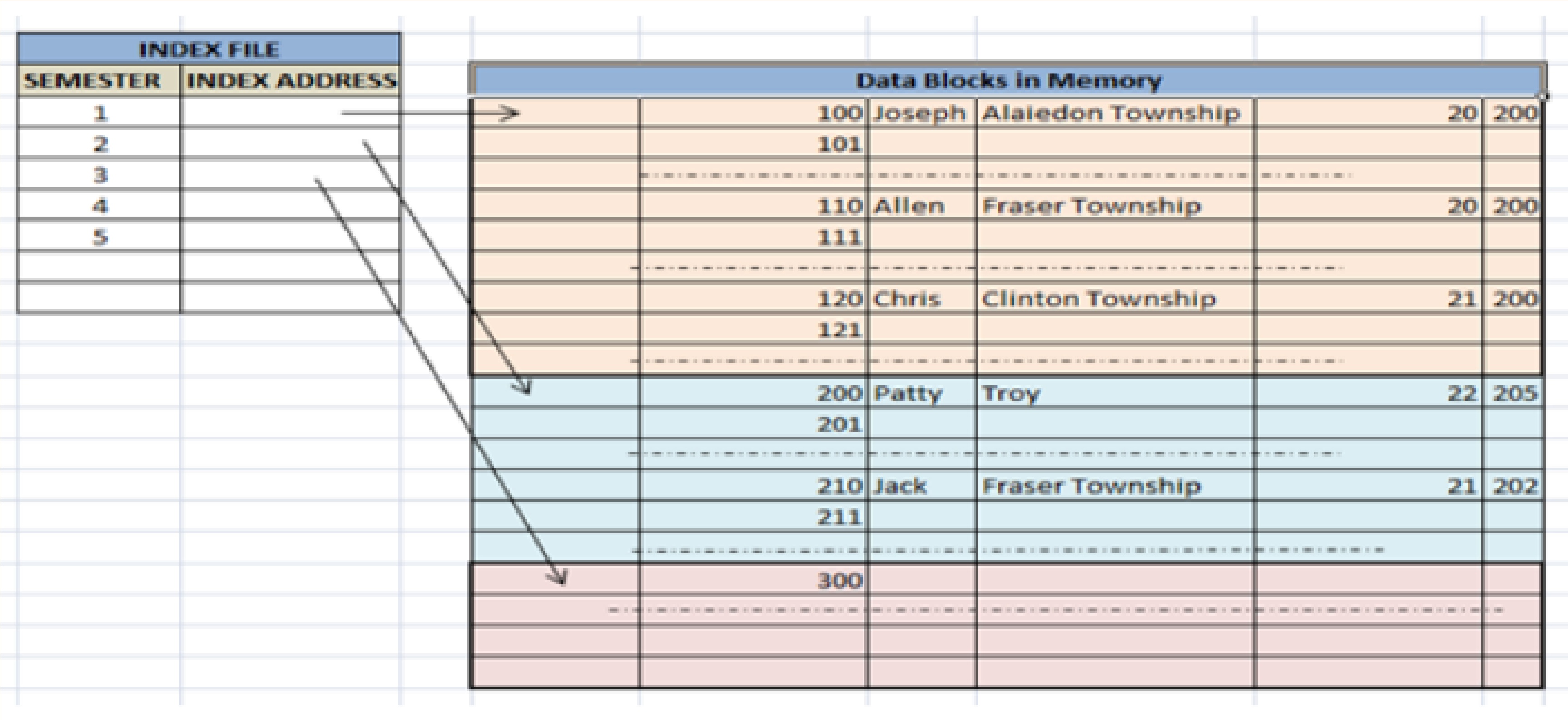
- Clustered Indexing
- Non-Clustered or Secondary Indexing
- Multilevel Indexing

CLUSTERING INDEXING

When more than two records are stored in the same file these types of storing known as cluster indexing. By using the cluster indexing we can reduce the cost of searching reason being multiple records related to the same thing are stored at one place and it also gives the frequent joing of more than two tables(records).

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.

For example, students studying in each semester are grouped together. i.e. 1st Semester students, 2nd semester students, 3rd semester students etc are grouped.



INDEX FILE

SEMESTER	INDEX ADDRESS
1	
2	
3	
4	
5	

Data Blocks in Memory

	100	Joseph	Alaiedon Township	20	200
	101				
	110	Allen	Fraser Township	20	200
	111				
	120	Chris	Clinton Township	21	200
	121				
	200	Patty	Troy	22	205
	201				
	210	Jack	Fraser Township	21	202
	211				
	300				

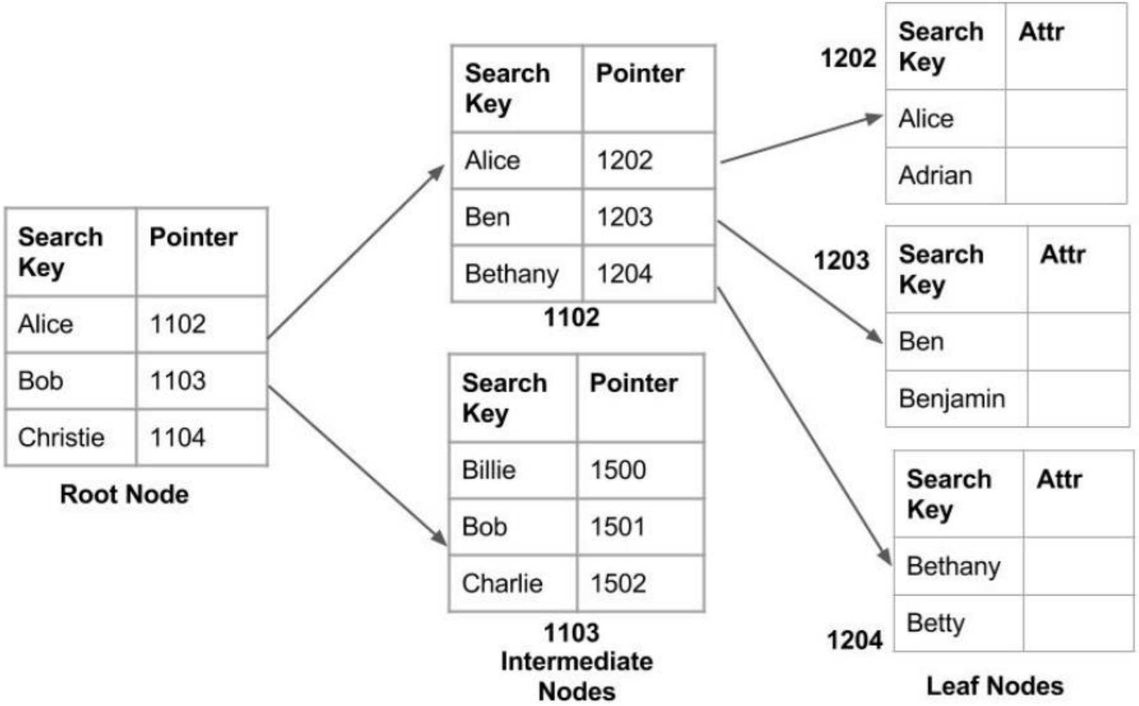
PRIMARY INDEXING

This is a type of Clustered Indexing wherein the data is sorted according to the search key and the primary key of the database table is used to create the index. It is a default format of indexing where it induces sequential file organization. As primary keys are unique and are stored in a sorted manner, the performance of the searching operation is quite efficient.

NON CLUSTER OR SECONDARY INDEXING

A non clustered index just tells us where the data lies, i.e. it gives us a list of virtual pointers or references to the location where the data is actually stored. Data is not physically stored in the order of the index. Instead, data is present in leaf nodes. For eg. the contents page of a book. Each entry gives us the page number or location of the information stored. The actual data here(information on each page of the book) is not organized but we have an ordered reference(contents page) to where the data points actually lie. We can have only dense ordering in the non-clustered index as sparse ordering is not possible because data is not physically organized accordingly.

It requires more time as compared to the clustered index because some amount of extra work is done in order to extract the data by further following the pointer. In the case of a clustered index, data is directly present in front of the index.



Search Key	Pointer
Alice	1102
Bob	1103
Christie	1104

Root Node

Search Key	Pointer
Alice	1202
Ben	1203
Bethany	1204

1102

Search Key	Pointer
Billie	1500
Bob	1501
Charlie	1502

1103

Intermediate Nodes

1202

Search Key	Attr
Alice	
Adrian	

1203

Search Key	Attr
Ben	
Benjamin	

Search Key	Attr
Bethany	
Betty	

1204

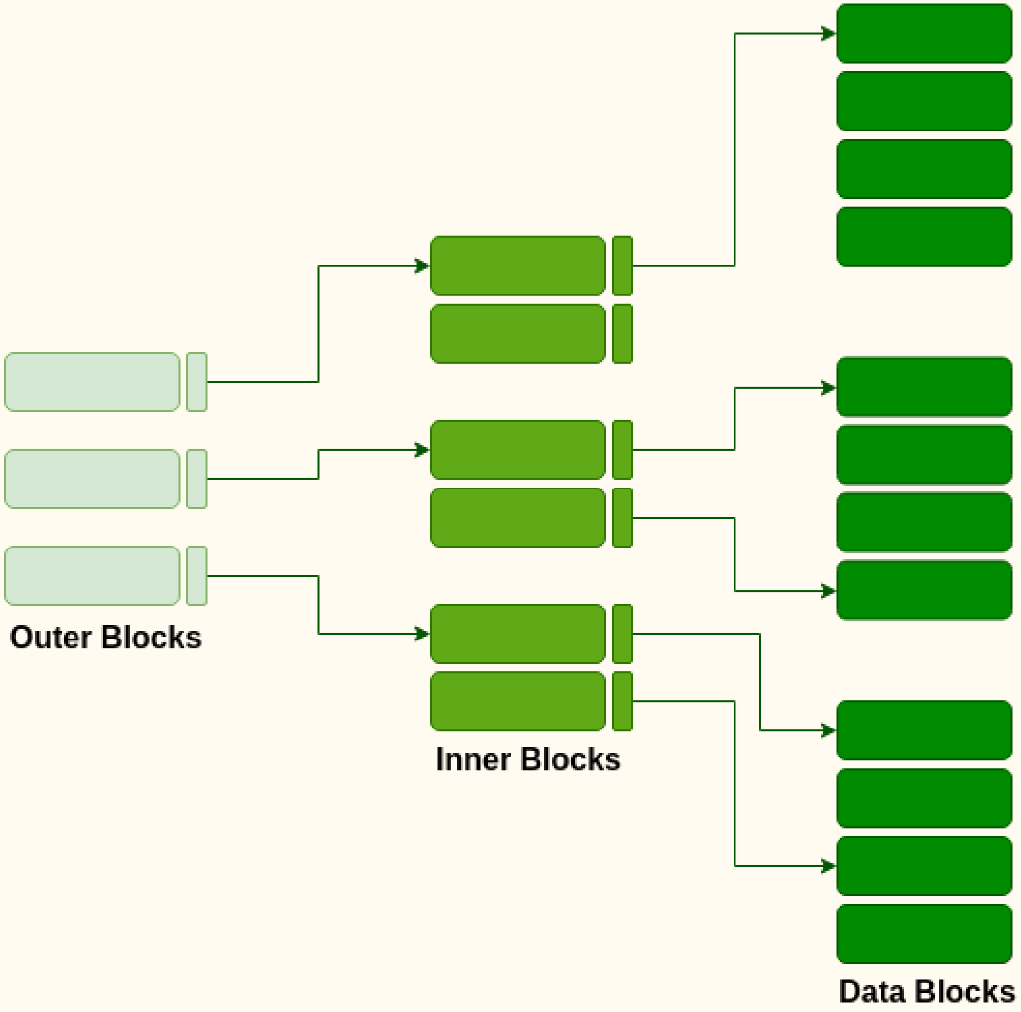
Leaf Nodes

Non clustered index

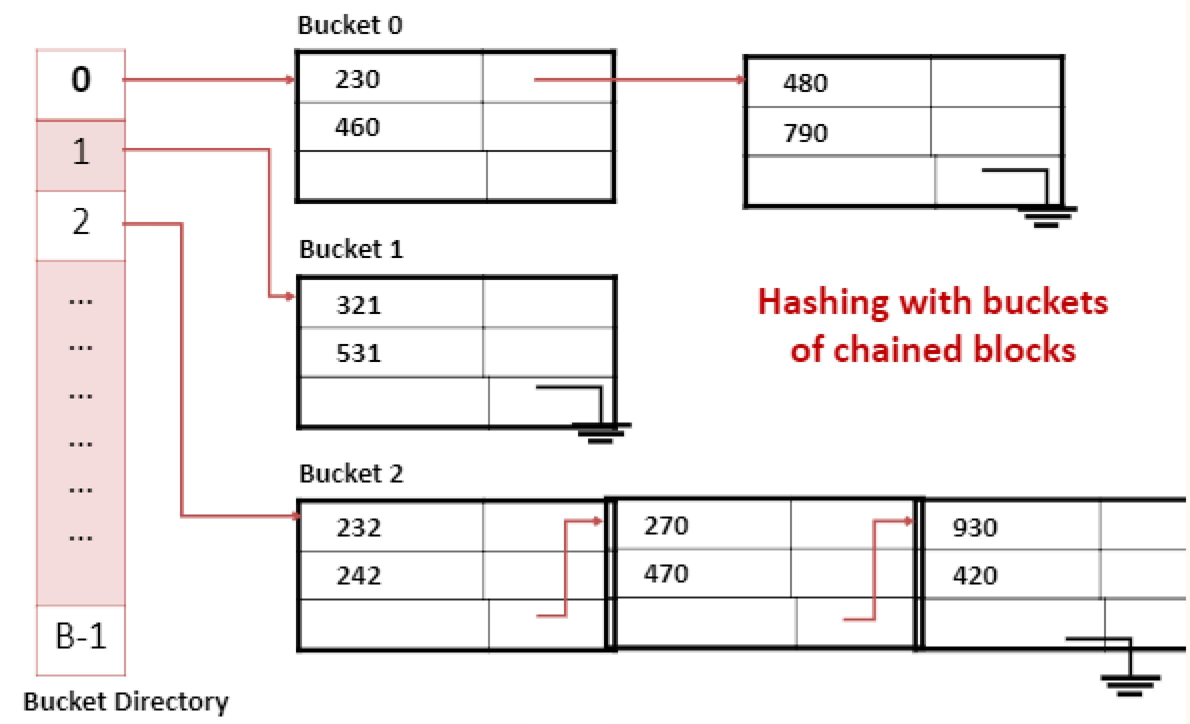
AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

MULTI LEVEL INDEXING

With the growth of the size of the database, indices also grow. As the index is stored in the main memory, a single-level index might become too large a size to store with multiple disk accesses. The multilevel indexing segregates the main block into various smaller blocks so that the same can stored in a single block. The outer blocks are divided into inner blocks which in turn are pointed to the data blocks. This can be easily stored in the main memory with fewer overheads.


Outer Blocks
Inner Blocks
Data Blocks

HASHING FOR DIRECT FILES



HASHING FOR DIRECT FILES

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.

MULTI KEY FILE ORGANIZATION & ACCESS METHODS

This technique is used to sort a file based on multiple key values. Multi key file organization allows access to a data file by several different key fields. Example: Library file that requires access by author and by subject matter and title.

Scenario 1 can be like, sort the file based on roll number (this is key; you can also create clustered index). On an attribute may be other than roll number e.g. Class (MCA-I, MCA-II, MCA-III, …., MCA-VI) invert the file.

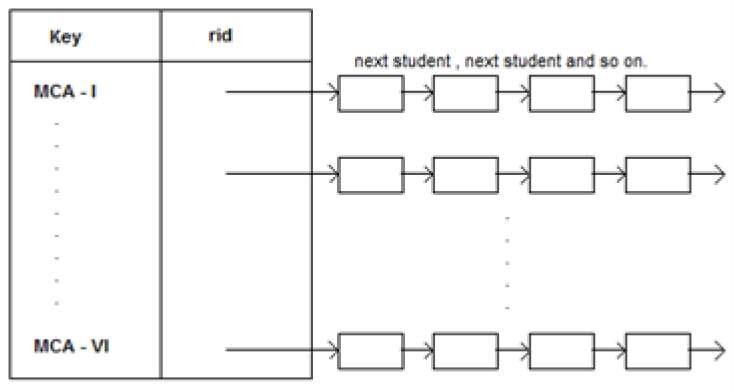
Now if I wish to access the records of students from MCA-VI, I will locate the first record and will go sequentially in the linked list. This technique works great if -

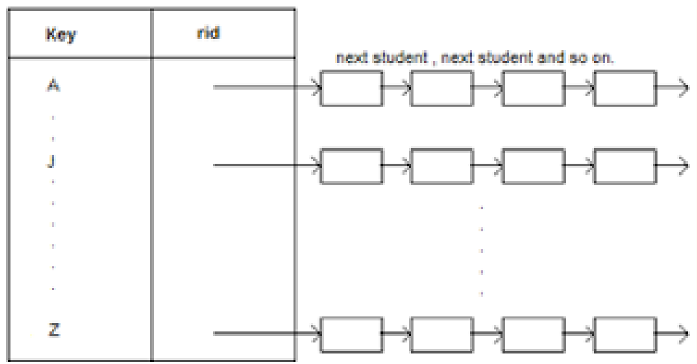
- Attribute on which I am inverting is primary key or having nearly unique values
- The need is to access everybody in a particular class

Longer the list, it is going to be a clumsy scan. The list can be flexible in size.

- Selection/Search – go in a particular order;
- Insertion – will be easy in append mode;
- Deletion – 1) Re-arrange pointer   2) Mark deleted & re-arrange pointer at later point of time.

Scenario 2, records are sorted based on Roll numbers; now requirement is to get names of students starting with ‘J’. Therefore create inverted index on names where names starting with each alphabet will be stored as linked list. Now go through the linked list of ‘J’ to get all the records





AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Thank you!