UNIT- V	File Concepts	(9Hrs)

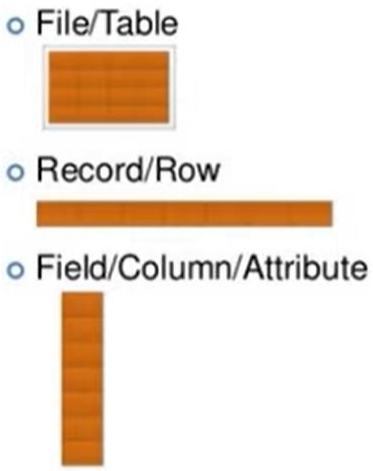
File Organisation – Sequential – Direct - Indexed Sequential - Hashed and various types of accessing schemes.

File Organization

What is File? What is File Organization?

✓ **File:**

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



✓ **File Organization:**

- File organization refers to the way data is stored in a file.

✓ **Objectives of File Organization:**

- It contains an optimal selection of records, i.e., records can be selected as fast as possible.
- To perform insert, delete or update transaction on the records should be quick and easy.
- The duplicate records cannot be induced as a result of insert, update or delete.
- For the minimal cost of storage, records should be stored efficiently.

INTRODUCTION

- File structures is the **organization of data** in secondary storage device
- Minimize the **access time** and the **storage space**.
- A File Structure is a combination of representations for data in files and of **operations for accessing the data**.
- A File Structure allows applications to **read, write and modify** data.
- Support **finding the data** that matches some search criteria or **reading** through the data in some **particular order**.

- **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.

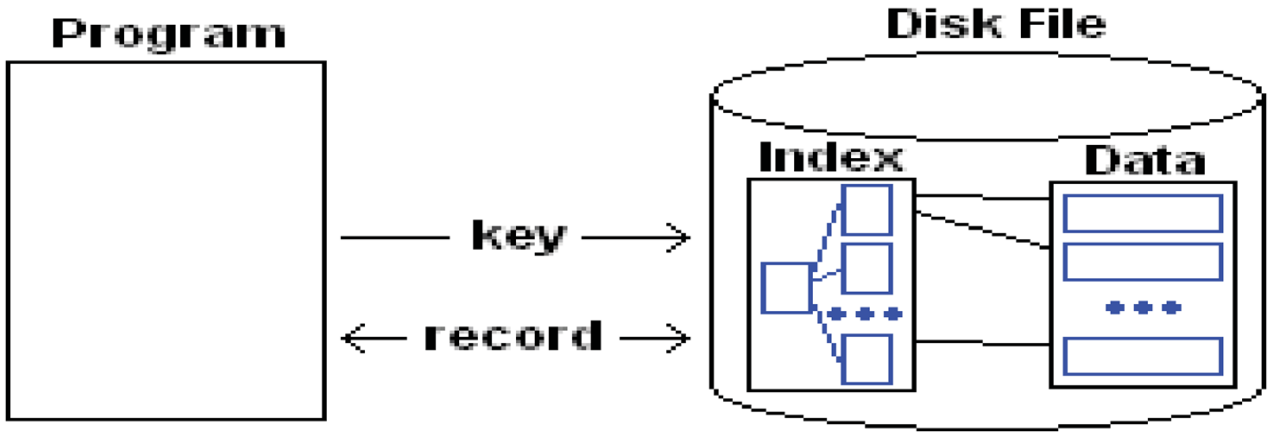
  File volatility addresses the properties of record changes. It helps to increase the efficiency of disk design than tape.

Record

- The identity of an attribute is given by the **position** of its attribute values within the record
- The data within a record is **registered sequentially** and has a definite **beginning** and **end**
- The record is divided into **fields** and the nth field carries the nth attribute value.

Disks and Indexes

- As files grew very large, unaided **sequential access** was not a good solution.
- Disks allowed for **direct access**.
- **Indexes** made it possible to keep a **list of keys and pointers** in a small file that could be searched very quickly.
- With the **key and pointer**, the user had **direct access** to the large, primary file.



Pointers

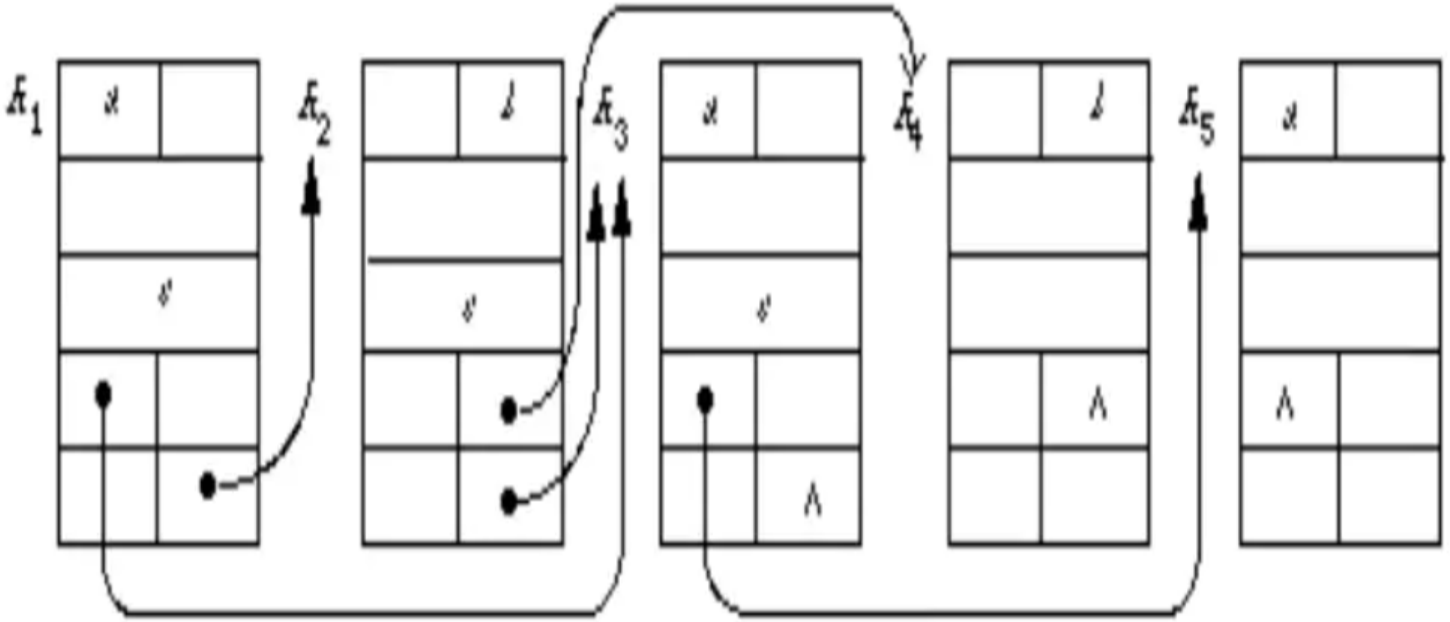


Figure 8.2: A demonstration of the use of pointers to link record

Internal and External Memory

- The data structures and algorithms considered thus far are all appropriate to the ***internal memory*** of a computer. This memory is also referred to as ***main*** memory or ***random access*** memory.
- A program can process only information stored in its internal memory. If the program is to process information stored in external memory, that information must first be read into its internal memory.
- Except for the instructions involved in this transfer of information, all program instructions refer to variables that name internal memory locations.
- By means of random access, it is possible to access *any* individual internal memory element in exactly the same length of time--typically one-millionth of a second or less. This access time is independent of the past history of accessed elements.
- However, random access memory is expensive and, although available in increasingly larger sizes, is still relatively small in size compared to the storage needs of many applications.

- ***External memory,*** also referred to as ***secondary storage,*** is so called because it is physically outside the computer itself.
- It is relatively inexpensive and is available in very large amounts, but access to an arbitrary record is slow compared to internal memory. ***Magnetic tapes, floppy disks,*** and ***magnetic disks*** (also known as ***hard disks***) are the usual devices for external storage of data.
- Characteristics of secondary storage devices are different from those of internal memory, the data structures and algorithms required to process the data they store are distinct from those for internal memory.
- However, many of the same concepts and techniques of good design and implementation of algorithms are still applicable and form the basis for representing and operating on information stored in external memory.

Types of File Organization

- File organization contains various methods.
- These particular methods have pros and cons on the basis of access or selection.
- In the file organization, the programmer decides the best-suited file organization method according to his requirement.

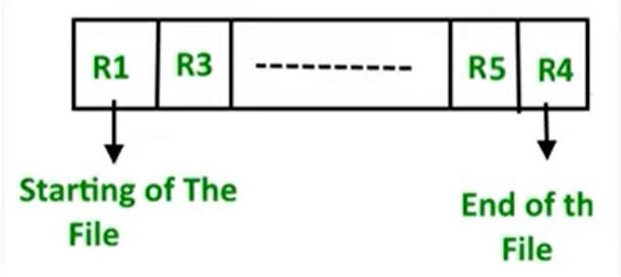
Types of File Organization:

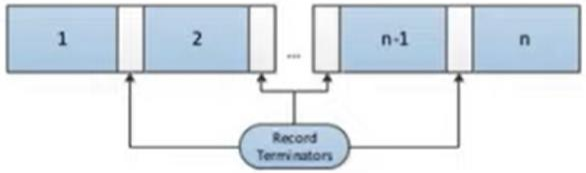
1. Sequential File Organization
2. Index Sequential File Organization
3. Direct Access File Organization
4. Cluster File Organization

Sequential

What is Sequential File Organization?

- The easiest method for file Organization is Sequential method.
- In this method the file are stored one after another in a sequential manner.
- 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.





Inserting New Records in Sequential F. O.

1. Inserting new Record:

- R1, R3 and so on up to R5 and R4 be four records in the sequence.
- Suppose a new record R2 has to be inserted in the sequence, then it is simply placed at the end of the file.

2. Sorting Method:

- The new record is always inserted at the file's end.
- Then it will sort the sequence in ascending or descending order.
- Sorting of records is based on any primary key or any other key.

Advantages & Disadvantages of Sequential File Organization

Advantages:

1. It is simple in design.
2. It requires no much effort to store the data.
3. Fast and efficient method for huge amount of data.
4. Files can be easily stored in magnetic tapes i.e cheaper storage mechanism.

Disadvantages:

1. Sequential file is time consuming process.
2. It has high data redundancy.
3. Random searching is not possible.
4. Sorted file method is inefficient as it takes time and space for sorting records.
5. Sequential records cannot support modern technologies that require fast access to stored records.

Application of Sequential File Organization

1. This method is used when most of the records have to be accessed like grade calculation of a student, generating the salary slip, etc.
2. This method is used for report generation or statistical calculations.

Indexed Sequential

What is Index Sequential File Organization?

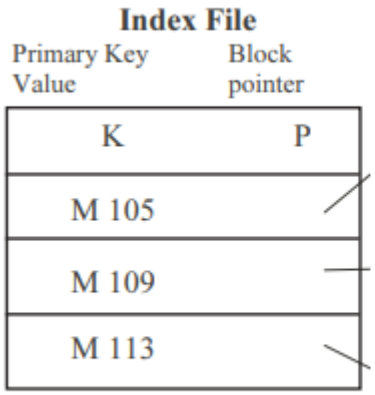
- ISAM method is an advanced sequential file organization.
- In this method, records are stored in the file using the primary key.
- An index value is generated for each primary key and mapped with the record.
- Index contains the address of the record in the file.
- The data can be access either sequentially or randomly using the index.
- The index is stored in a file and read into memory when the file is opened.
- If any record has to be retrieved based on its index value, then the address of the data block is fetched and the record is retrieved from the memory.

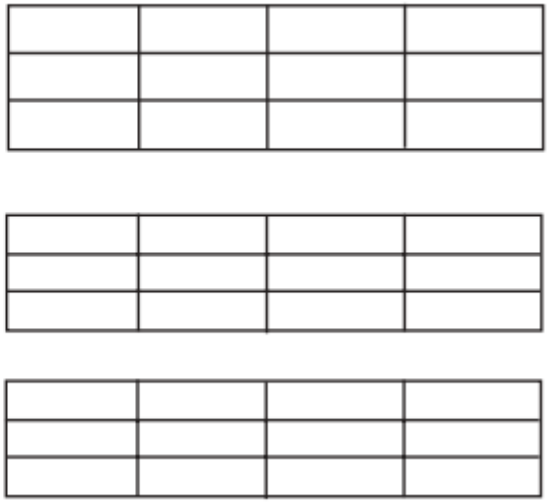
Types of Indexes

- An index is a table which stores the key values and the corresponding addresses of the records in a file.
- Given a key value, its address is located in the index and the corresponding record can be accessed using this address.
- The idea behind an index structure is similar to the one used commonly in textbooks.
- In a textbook index important terms are listed at the end of the book in alphabetic order. Along with each term, a list of page numbers or locations where the term appears is also given.
- We can search the index to find a list of addresses – page numbers in this case. This page number is a ‘direct pointer’ to the page in the book where the term appears. One can go directly to the specified page and find the material there.
- an index is an exact indication of where each term occurs in the book.

- A considerable amount of time is saved, but extra space is needed to store the index. The use of index in locating a record is similar to the use of a card index in a library.
- An index is usually defined on a single field of a file, called an Indexing Field. The index typically stores each value of the index field along with a list of pointers to all disk blocks that contain a record with that field value.
- The values in the index are ordered and the index file is much smaller than the data file. There are several types of indexes.
- These include primary index, clustering index, secondary index etc.

- A primary index is an index specified on the primary key of a data file.
- Primary key is a field which contains unique values and uniquely identifies each record. By knowing a primary key field's value we can access its record.
- The primary key is also known as the ordering key er the file





- A primary index is an ordered file with two fields. The first field is of same data type as the primary key field and the second is a pointer to a disk block – a block address
- An index file contains one entry (record) for each block in the data file. To find a record using the primary index, first the primary key value is located and then the indicated address is used to find the record.
- Thus, whenever a record is accessed, first it looks at the index, identifies the block and then it searches sequentially within the block.

- A clustering index stores data similar to a phone directory where all people with the same last name are grouped together.
- A clustering index is specified on a field that does not have a distinct value, for each record. These records are then stored in ascending or descending order according to the data values in this field.
- A table/database can have only one clustering index. e.g., if a clustering index is build on “state” columns of the “Authors” table in the Publisher database, the data will be ordered based on the values of “state” – in either ascending or descending order.

- A third type of index, called a secondary index, can be specified on any field other than the primary key of the file.
- A secondary key is any field other than the primary key that is used to uniquely identify a record in a table.

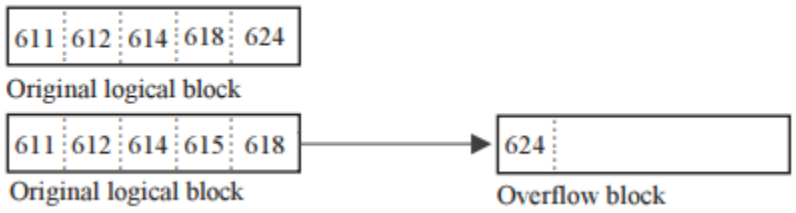
Use of indexed files

- Indexed files are used mainly in areas where timeliness of information is highly critical. Examples are found in airline reservation systems, job banks, military data systems, and other inventory type applications.
- Here data are rarely processed sequentially, except for the occasional, stock taking.
- When some information is obtained, e.g., a free seat on a certain flight, the data should be correct at that point of time, and if the information is updated, e.g., a seat is sold on that flight, this fact should be immediately known throughout the system.
- Indexed files are also desirable at places where data are highly variable and dynamic.

Structure of Index Sequential Files

- In Indexed-sequential organisation the records are grouped into fixed-length data block and each record is identified by a primary key.
- The physical sequence of the file is in ascending order according to the primary key.
- An index is built, separate from the data records. This index contains key values together with pointers to the data records themselves.
- Thus the indexed sequential fileorganisation allows both sequential and random processing. This index permits accessing individual records at random without accessing other records.
- For example, the white pages of an ordinary telephone directory represent an indexed sequential organisation. In the upper left-hand corner of each page is the name of the first person listed on that page

- An index-sequential file is not a single file but it consists of the data plus one or more levels of indexes. The data file contains the actual data items.
- When inserting a record, we have to maintain the sequence of records and this may necessitate shifting subsequent records. For a large file this is a costly and inefficient process.
- Instead, an overflow area is provided so that the records that overflow their logical area are shifted into a designated overflow area and a pointer is provided to it to the overflow location.
- This is illustrated below (Fig.) Record 615 is inserted in the original logical block causing a record to be moved to an overflow block.



- An index-sequential file is made up of the following components:
  - a) A primary data storage area. This contains the data records of the file.
  - b) Overflow area(s). This is used for records that are added to the file but will not fit in the prime area.
  - c) Indexes. These indexes enable access to any given record.
- There are two access method for the indexed sequential files.
  - i) Index Sequential Access Method (ISAM)
  - ii) Virtual Storage Access Method (VSAM)
- These two also represent the two basic implementation techniques of the Indexed sequential organization. ISAM is hardware dependent and VASM is hardware independent technique.

Index Sequential Access Method (ISAM)

- The ISAM file is a special type of indexed file. In the indexed-sequential file the records are physically stored on the disk in groups.
- Within each group the records are stored sequentially by primary key.
- When each group corresponds to a physical subdivision of the disk e.g., a track, the file type is called Index Sequential Access Method (ISAM).
- A record field or a combination of fields is called a key. A key may be unique or non-unique. It may be primary or secondary.
- Usually a file may have only one primary key, giving unique names to the records of the file. Any key other than the primary key is a secondary key.

- There may be one or more secondary keys defined for a file. The unique primary key is the basis for the Direct and ISAM files, while secondary keys are the basis for any other type of index.
- Records in an ISAM file may be indexed in two different ways:
  - 1) The entire file is read/searched sequentially using the primary key; and
  - 2) the record is accessed via the index using the key value provided in each index entry, the address is located within the index, and the record is retrieved. A combination of the two types of search may also be used, as with CDS-ISIS and WINISIS database searches.

- ISAM file combines some of the good features of indexed and sequential files.
- The ISAM file requires relatively smaller disk storage space, as it maintains records in the sequence on each track of the file, as mentioned above.
- A primary key is the basis of constructing an ISAM file. There may be several secondary indexes defined on an ISAM file.
- The major disadvantage of the index-sequential organisation is that as the file grows, performance deteriorates rapidly because of overflows and consequently there arises the need for periodic reorganisation.
- Reorganisation is an expensive process and the file becomes unavailable during reorganisation.

Virtual Storage Access Method (VSAM)

- The virtual storage access method (VSAM) is IBM's advanced version of the index-sequential organisation that avoids these disadvantages mentioned above. It is more powerful and flexible than ISAM.
- The VSAM files are made up of two components: the index and data. However, overflows are handled in a different manner.
- In a VSAM file, the basic indexed data block is called a control interval (or a virtual track). Each time a data block overflows it is divided into two blocks. Appropriate changes are made to the indexes to reflect this division.
- Indexed-sequential files of the basic type discussed above are in common use in modern commercial processing. They are used especially in on-line or terminal-oriented access, where the files have to be updated within very short time frame.

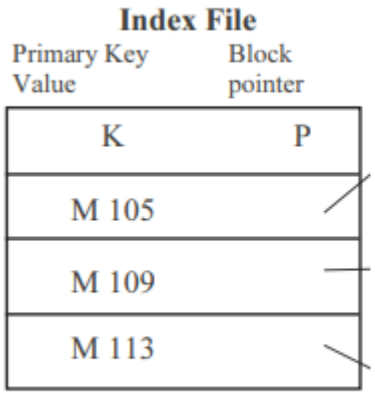
- An indexed-sequential file can, for instance, be used to produce an inventory listing on a daily basis. Indexed-sequential files are also commonly used to handle inquiries, like billing inquiries based on account numbers.
- VSAM offers many advantages over ISAM. In VSAM, periodic file reorganisation is not required as the file can grow indefinitely by means of the splitting process.
- The VSAM file organisation is also independent of hardware characteristics.

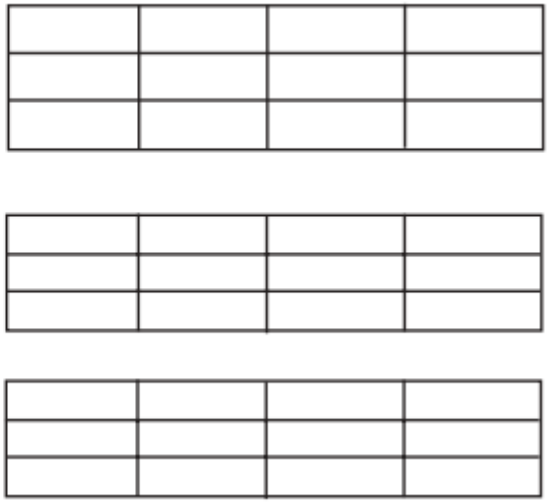
Types of Indexes

- An index is a table which stores the key values and the corresponding addresses of the records in a file.
- Given a key value, its address is located in the index and the corresponding record can be accessed using this address.
- The idea behind an index structure is similar to the one used commonly in textbooks.
- In a textbook index important terms are listed at the end of the book in alphabetic order. Along with each term, a list of page numbers or locations where the term appears is also given.
- We can search the index to find a list of addresses – page numbers in this case. This page number is a ‘direct pointer’ to the page in the book where the term appears. One can go directly to the specified page and find the material there.
- an index is an exact indication of where each term occurs in the book.

- A considerable amount of time is saved, but extra space is needed to store the index. The use of index in locating a record is similar to the use of a card index in a library.
- An index is usually defined on a single field of a file, called an Indexing Field. The index typically stores each value of the index field along with a list of pointers to all disk blocks that contain a record with that field value.
- The values in the index are ordered and the index file is much smaller than the data file. There are several types of indexes.
- These include primary index, clustering index, secondary index etc.

- A primary index is an index specified on the primary key of a data file.
- Primary key is a field which contains unique values and uniquely identifies each record. By knowing a primary key field's value we can access its record.
- The primary key is also known as the ordering key er the file





- A primary index is an ordered file with two fields. The first field is of same data type as the primary key field and the second is a pointer to a disk block – a block address
- An index file contains one entry (record) for each block in the data file. To find a record using the primary index, first the primary key value is located and then the indicated address is used to find the record.
- Thus, whenever a record is accessed, first it looks at the index, identifies the block and then it searches sequentially within the block.

- A clustering index stores data similar to a phone directory where all people with the same last name are grouped together.
- A clustering index is specified on a field that does not have a distinct value, for each record. These records are then stored in ascending or descending order according to the data values in this field.
- A table/database can have only one clustering index. e.g., if a clustering index is build on “state” columns of the “Authors” table in the Publisher database, the data will be ordered based on the values of “state” – in either ascending or descending order.

- A third type of index, called a secondary index, can be specified on any field other than the primary key of the file.
- A secondary key is any field other than the primary key that is used to uniquely identify a record in a table.

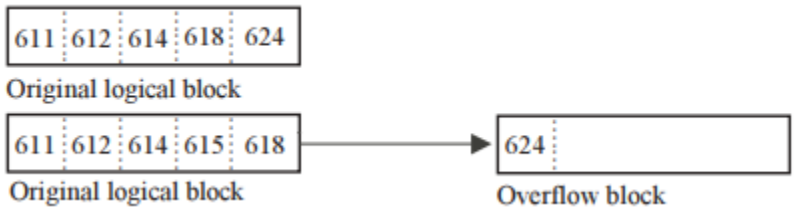
Use of indexed files

- Indexed files are used mainly in areas where timeliness of information is highly critical. Examples are found in airline reservation systems, job banks, military data systems, and other inventory type applications.
- Here data are rarely processed sequentially, except for the occasional, stock taking.
- When some information is obtained, e.g., a free seat on a certain flight, the data should be correct at that point of time, and if the information is updated, e.g., a seat is sold on that flight, this fact should be immediately known throughout the system.
- Indexed files are also desirable at places where data are highly variable and dynamic.

Structure of Index Sequential Files

- In Indexed-sequential organisation the records are grouped into fixed-length data block and each record is identified by a primary key.
- The physical sequence of the file is in ascending order according to the primary key.
- An index is built, separate from the data records. This index contains key values together with pointers to the data records themselves.
- Thus the indexed sequential fileorganisation allows both sequential and random processing. This index permits accessing individual records at random without accessing other records.
- For example, the white pages of an ordinary telephone directory represent an indexed sequential organisation. In the upper left-hand corner of each page is the name of the first person listed on that page

- An index-sequential file is not a single file but it consists of the data plus one or more levels of indexes. The data file contains the actual data items.
- When inserting a record, we have to maintain the sequence of records and this may necessitate shifting subsequent records. For a large file this is a costly and inefficient process.
- Instead, an overflow area is provided so that the records that overflow their logical area are shifted into a designated overflow area and a pointer is provided to it to the overflow location.
- This is illustrated below (Fig.) Record 615 is inserted in the original logical block causing a record to be moved to an overflow block.



- An index-sequential file is made up of the following components:
  - a) A primary data storage area. This contains the data records of the file.
  - b) Overflow area(s). This is used for records that are added to the file but will not fit in the prime area.
  - c) Indexes. These indexes enable access to any given record.
- There are two access method for the indexed sequential files.
  - i) Index Sequential Access Method (ISAM)
  - ii) Virtual Storage Access Method (VSAM)
- These two also represent the two basic implementation techniques of the Indexed sequential organization. ISAM is hardware dependent and VASM is hardware independent technique.

Index Sequential Access Method (ISAM)

- The ISAM file is a special type of indexed file. In the indexed-sequential file the records are physically stored on the disk in groups.
- Within each group the records are stored sequentially by primary key.
- When each group corresponds to a physical subdivision of the disk e.g., a track, the file type is called Index Sequential Access Method (ISAM).
- A record field or a combination of fields is called a key. A key may be unique or non-unique. It may be primary or secondary.
- Usually a file may have only one primary key, giving unique names to the records of the file. Any key other than the primary key is a secondary key.

- There may be one or more secondary keys defined for a file. The unique primary key is the basis for the Direct and ISAM files, while secondary keys are the basis for any other type of index.
- Records in an ISAM file may be indexed in two different ways:
  - 1) The entire file is read/searched sequentially using the primary key; and
  - 2) the record is accessed via the index using the key value provided in each index entry, the address is located within the index, and the record is retrieved. A combination of the two types of search may also be used, as with CDS-ISIS and WINISIS database searches.

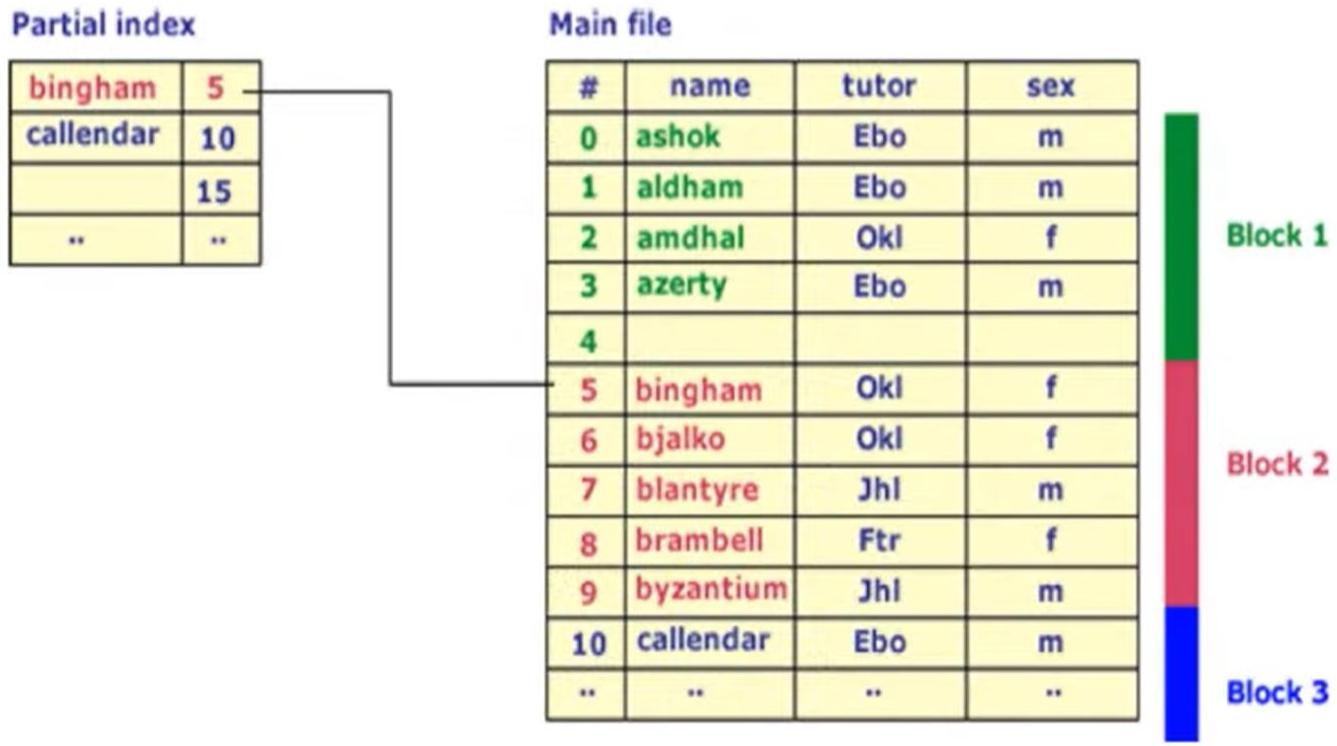
- ISAM file combines some of the good features of indexed and sequential files.
- The ISAM file requires relatively smaller disk storage space, as it maintains records in the sequence on each track of the file, as mentioned above.
- A primary key is the basis of constructing an ISAM file. There may be several secondary indexes defined on an ISAM file.
- The major disadvantage of the index-sequential organisation is that as the file grows, performance deteriorates rapidly because of overflows and consequently there arises the need for periodic reorganisation.
- Reorganisation is an expensive process and the file becomes unavailable during reorganisation.

Virtual Storage Access Method (VSAM)

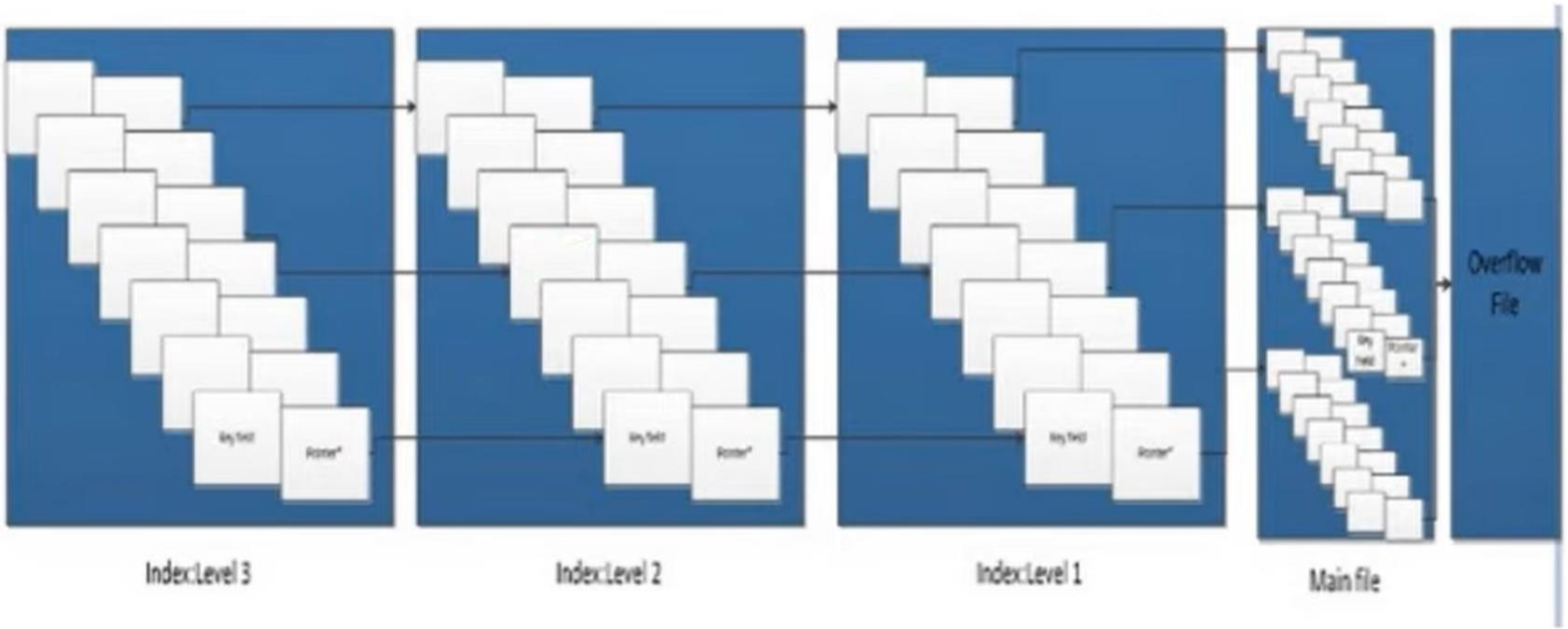
- The virtual storage access method (VSAM) is IBM's advanced version of the index-sequential organisation that avoids these disadvantages mentioned above. It is more powerful and flexible than ISAM.
- The VSAM files are made up of two components: the index and data. However, overflows are handled in a different manner.
- In a VSAM file, the basic indexed data block is called a control interval (or a virtual track). Each time a data block overflows it is divided into two blocks. Appropriate changes are made to the indexes to reflect this division.
- Indexed-sequential files of the basic type discussed above are in common use in modern commercial processing. They are used especially in on-line or terminal-oriented access, where the files have to be updated within very short time frame.

- An indexed-sequential file can, for instance, be used to produce an inventory listing on a daily basis. Indexed-sequential files are also commonly used to handle inquiries, like billing inquiries based on account numbers.
- VSAM offers many advantages over ISAM. In VSAM, periodic file reorganisation is not required as the file can grow indefinitely by means of the splitting process.
- The VSAM file organisation is also independent of hardware characteristics.

Index Sequential File Organization



Index Sequential File Organization



Advantages of Index Sequential File Organization

1. In indexed sequential access file, sequential file and random file access is possible.
2. It accesses the records very fast if the index table is properly organized.
3. Searching a record in a huge database is quick and easy.
4. The records can be inserted in the middle of the file.
5. It provides quick access for sequential and direct processing.
6. It reduces the degree of the sequential search.
7. It has less time complexity as compare to Sequential File Organization.

Disadvantages of Index Sequential File Organization

1. Indexed sequential access file requires unique keys and periodic reorganization.
2. Indexed sequential access file takes longer time to search the index for the data access or retrieval.
3. It requires more storage space.
4. When the new records are inserted, then these files have to be reconstructed to maintain the sequence.
5. When the record is deleted, then the space used by it needs to be released. Otherwise, the performance of the database will slow down.

Applications of Index Sequential File Organization

1. Mainly in Database. E.g. Storing Student Records.
2. Index of Online Books.
3. Hyperlinks on Google for fast searching purpose.

Direct

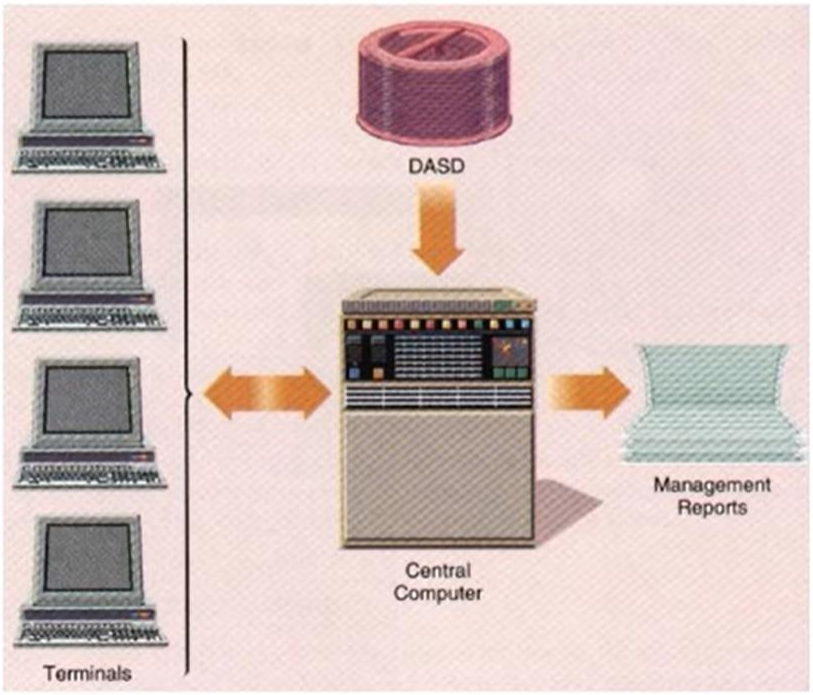
What is Direct Access File Organization?

- Direct access file is also known as random access or relative file organization.
- In direct access file, all records are stored in direct access storage device (DASD), such as hard disk.
- The records are randomly placed throughout the file.
- The records does not need to be in sequence because they are updated directly and rewritten back in the same location.
- This file organization is useful for immediate access to large amount of information.
- It is used in accessing large databases.
- It is also called as hashing.

- A filed-length logical record that allows the program to read and write record rapidly. in no particular order.
- The direct access is based on the disk model of a file since disk allows random access to any file block.
- For direct access, the file is viewed as a numbered sequence of block or record. Thus, we may read block 14 then block 59 and then we can write block 17.
- There is no restriction on the order of reading and writing for a direct access file.
- A block number provided by the user to the operating system is normally a *relative block number*, the first relative block of the file is 0 and then 1 and so on.

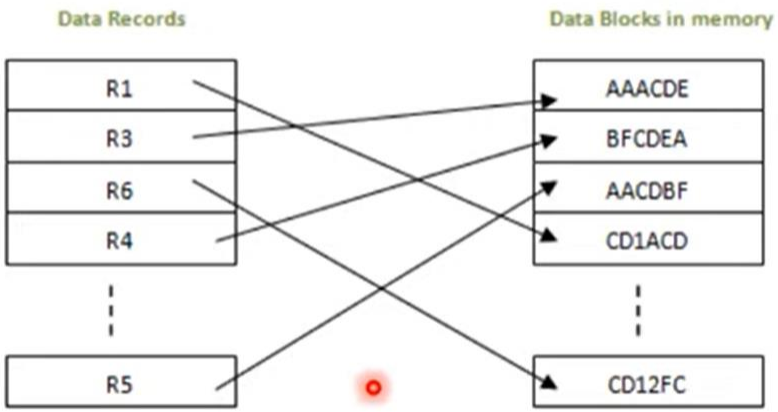
Direct Access File Organization

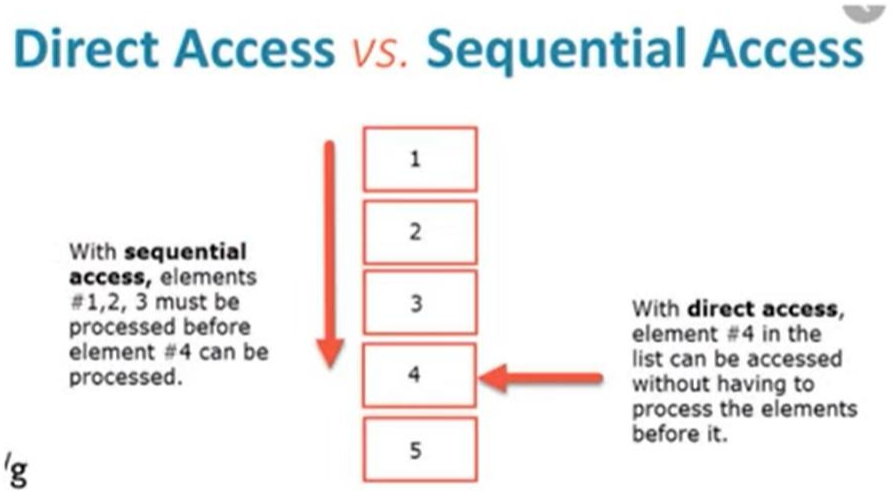
1. A hashed file uses a hash function to map the key to address.
2. Eliminates need of extra file i.e index file.



Direct Access File Organization Structure

Structure:





/g

Advantages of Direct Access File Organization

1. Direct access file helps in online transaction processing system (OLTP) like online railway reservation system.
2. In direct access file, sorting of the records are not required.
3. It accesses the desired records immediately.
4. It updates several files quickly.
5. It has better control over record allocation.
6. Most suitable for interactive online applications.

Disadvantages of Direct Access File Organization

1. Direct access file does not provide back up facility.
2. Expensive hardware and Software are required.
3. File updating is more difficult when compared to that of sequential method.
4. Less efficient use of Storage Space.
5. Data may be accidentally erased or over written unless special precautions are taken.

Applications of Direct Access File Organization

1. Airline seat Reservations, Current information about available flights and seats must be available at all times so that flights are not overbooked.
2. Railway Reservation System.
3. OTP(One Time Password).

Comparison's of Sequential, Index Sequential & Direct Access File

Sr. No	Parameters	Sequential File Organization	Index Sequential File Organization	Direct Access File Organization
1	**Called as**	QSAM(Queued Sequential Access Method)	VSAM(Virtual Storage Access Method)	VSAM(Virtual Storage Access Method)
2	**Data Entry Order**	Data entered in Sequential Order.	Data entered in Key Sequential Order.	Data is entered in RRN(Receiver Registration Number) order. RRN = Mob No., Account No., OTP.
3	**Sorted Order**	Not sorted.	Sorted order based on key.	Sorted order based on RRN number.
4	**Duplicate Data**	Allowed	Not Allowed	Not Allowed
5	**Delete**	Not Applicable	Applicable	Applicable
6	**Access**	Slow	Faster	More Faster
7	**Key**	Not Available	Available, User Define & Part of record.	Available, System generated & Not part of record.

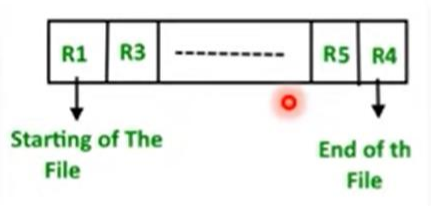
Comparison's of Sequential, Index Sequential & Direct Access File

Sr. No	Parameters	Sequential File Organization	Index Sequential File Organization	Direct Access File Organization
8	Storage Allocation	Data stored in Tape/Disc.	Data stored in Disc only.	Data stored in Disc only.
9	Applications	1. Grade calculation of a student, 2. Generating the salary slip. 3. Report generation 4. Statistical calculations.	1. Mainly in Database. E.g. Storing Student Records. 2. Index of Online Books. 3. Hyperlinks on Google for fast searching purpose	1. Airline seat Reservations, Current information about available flights and seats must be available at all times so that flights are not overbooked. 2. Railway Reservation System. 3. OTP(One Time Password).

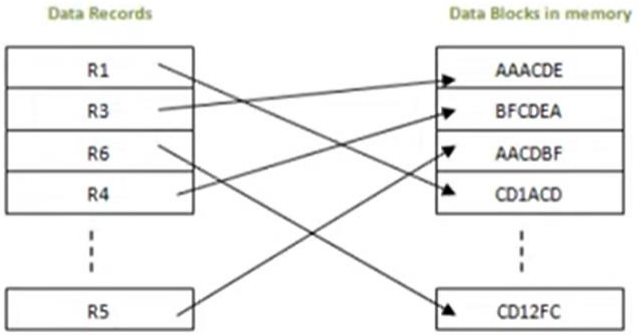
Comparison's of Sequential, Index Sequential & Direct Access File

10. Structure:

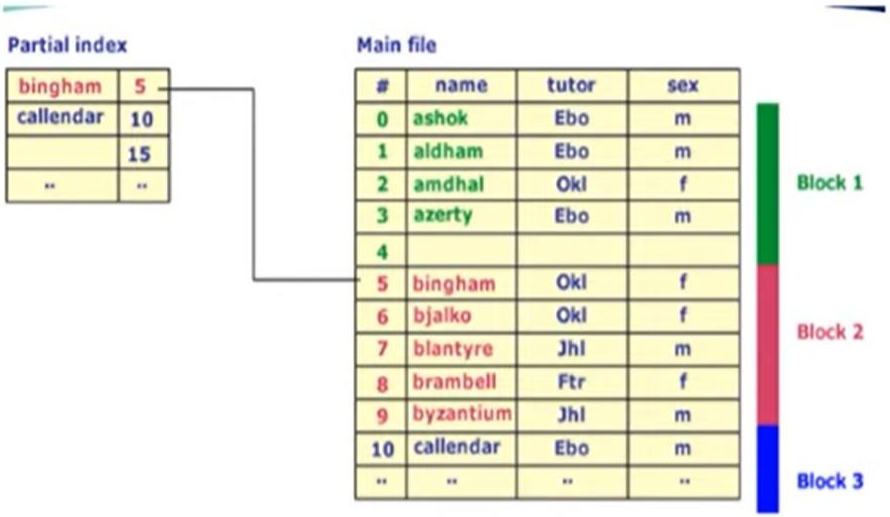
1. Sequential File Organization:



2. Direct Access File Organization



3. Index Sequential File Organization:



What is Cluster File Organization?

- When the two or more records are stored in the same file, it is known as clusters.
- These files will have two or more tables in the same data block.
- Key attributes which are used to map these tables together are stored only once.
- This method reduces the cost of searching for various records in different files.
- The cluster file organization is used when there is a frequent need for joining the tables with the same condition.
- These joins will give only a few records from both tables.
- We are retrieving the record for only particular field by using key attribute.

Cluster File Organization Structure

EMPLOYEE

EMP_ID	EMP_NAME	ADDRESS	DEP_ID
1	John	Delhi	14
2	Robert	Gujarat	12
3	David	Mumbai	15
4	Amelia	Meerut	11
5	Kristen	Noida	14
6	Jackson	Delhi	13
7	Amy	Bihar	10
8	Sonoo	UP	12

DEPARTMENT

DEP_ID	DEP_NAME
10	Math
11	English
12	Java
13	Physics
14	Civil
15	Chemistry

Cluster Key

DEP_ID	DEP_NAME	EMP_ID	EMP_NAME	ADDRESS
10	Math	7	Amy	Bihar
11	English	4	Amelia	Meerut
12	Java	2	Robert	Gujarat
12		8	Sonoo	UP
13	Physics	6	Jackson	Delhi
14	Civil	1	John	Delhi
14		5	Kristen	Noida
15	Chemistry	3	David	Mumbai

In this method, we can directly insert, update or delete any record. Data is sorted based on the key with which searching is done. Cluster key is a type of key with which joining of the table is performed.

Types of Cluster File Organization

1. **Index Cluster:**

- In indexed cluster, records are grouped based on the cluster key and stored together.
- The above EMPLOYEE and DEPARTMENT relationship is an example of an indexed cluster.
- Here, all the records are grouped based on the cluster key- DEP_ID and all the records are grouped.

2. **Hash Cluster:**

- In hash cluster, instead of storing the records based on the cluster key,
- We generate the value of the hash key for the cluster key and store the records with the same hash key value.

Advantages of Cluster File Organization

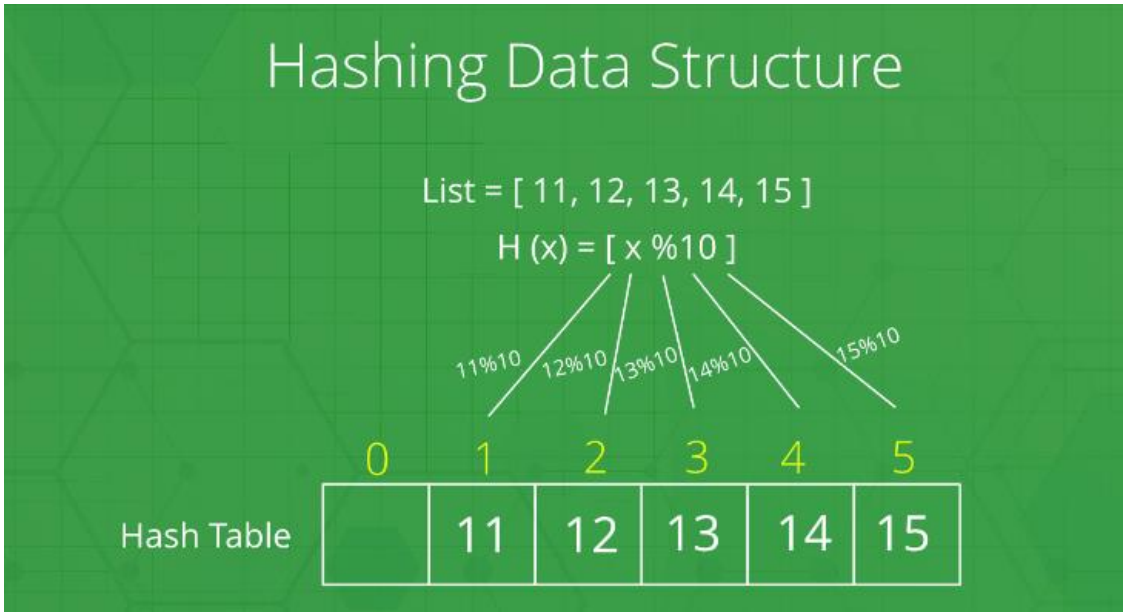
1. The cluster file organization is used when there is a frequent request for joining the tables with same joining condition.
2. It provides the efficient result when there is a 1:M mapping between the tables.

Drawbacks of Cluster File Organization

1. This method has the low performance for the very large database.
2. If we change the condition of joining then traversing the file takes a lot of time.
3. This method is not suitable for a table with a 1:1 condition.

Hashed and various types of accessing schemes

- **Hashing** is a fundamental data structure that efficiently stores and retrieves data in a way that allows for quick access.
- It involves mapping data to a specific index in a hash table using a **hash function** that enables fast retrieval of information based on its key.
- This method is commonly used in databases, **c**aching systems, and various programming applications to optimize search and retrieval operations.



What is Hashing in Data Structure?

- **Hashing** is a technique used in data structures to store and retrieve data efficiently. It involves using a **hash function** to map data items to a fixed-size array which is called a **hash table**.

Hash Table in Data Structure

- A **hash table** is also known as a hash map. It is a data structure that stores **key-value pairs**. It uses a **hash function** to map **keys** to a fixed-size array, called a **hash table**. This allows in faster **search**, **insertion**, and **deletion** operations.

How does Hashing work?

Suppose we have a set of strings {“ab”, “cd”, “efg”} and we would like to store it in a table. Our main objective here is to search or update the values stored in the table quickly in O(1) time and we are not concerned about the ordering of strings in the table. So the given set of strings can act as a key and the string itself will act as the value of the string but how to store the value corresponding to the key?

- **Step 1:** We know that hash functions (which is some mathematical formula) are used to calculate the hash value which acts as the index of the data structure where the value will be stored.
- **Step 2:** So, let’s assign
  - “a” = 1,
  - “b”=2, .. etc, to all alphabetical characters.
- **Step 3:** Therefore, the numerical value by summation of all characters of the string:
- “ab” = 1 + 2 = 3,
- “cd” = 3 + 4 = 7 ,
- “efg” = 5 + 6 + 7 = 18
- **Step 4:** Now, assume that we have a table of size 7 to store these strings. The hash function that is used here is the sum of the characters in **key mod Table size**. We can compute the location of the string in the array by taking the **sum(string) mod 7**.
- **Step 5:** So we will then store
  - “ab” in 3 mod 7 = 3,
  - “cd” in 7 mod 7 = 0, and
  - “efg” in 18 mod 7 = 4.

0	1	2	3	4	5	6
cd			ab	efg		

Mapping key with indices of array

Hash functions:

1. Division Method.
2. Mid Square Method.
3. Folding Method.
4. Multiplication Method.

1. Division Method:

This is the most simple and easiest method to generate a hash value. The hash function divides the value k by M and then uses the remainder obtained.

Formula:

h(K) = k mod M

Here,

k is the key value, and

M is the size of the hash table.

It is best suited that **M** is a prime number as that can make sure the keys are more uniformly distributed. The hash function is dependent upon the remainder of a division.

Example:

k = 12345

M = 95

h(12345) = 12345 mod 95

= 90

k = 1276

M = 11

h(1276) = 1276 mod 11

= 0

2. Mid Square Method:

The mid-square method is a very good hashing method. It involves two steps to compute the hash value-

1.Square the value of the key k i.e. k^2

2.Extract the middle **r** digits as the hash value.

Formula:

h(K) = h(k x k)

Here,

k is the key value.

The value of **r** can be decided based on the size of the table.

Example:

Suppose the hash table has 100 memory locations. So r = 2 because two digits are required to map the key to the memory location.

k = 60

k x k = 60 x 60

= 3600

h(60) = 60

The hash value obtained is 60

3. Digit Folding Method:

This method involves two steps:

1.Divide the key-value **k** into a number of parts i.e. **k1, k2, k3,....,kn**, where each part has the same number of digits except for the last part that can have lesser digits than the other parts.

2.Add the individual parts. The hash value is obtained by ignoring the last carry if any.

Formula:

k = k1, k2, k3, k4, ….., kn

s = k1+ k2 + k3 + k4 +….+ kn

h(K)= s

Here,

s is obtained by adding the parts of the key **k**

Example:

k = 12345

k1 = 12, k2 = 34, k3 = 5

s = k1 + k2 + k3

= 12 + 34 + 5

= 51

h(K) = 51

4. Multiplication Method

This method involves the following steps:

1.Choose a constant value A such that $0 < A < 1$.
2.Multiply the key value with A.
3.Extract the fractional part of kA.
4.Multiply the result of the above step by the size of the hash table i.e. M.
5.The resulting hash value is obtained by taking the floor of the result obtained in step 4.

Formula:

h(K) = floor (M (kA mod 1))

Here,

M is the size of the hash table.

k is the key value.

A is a constant value.

Example:

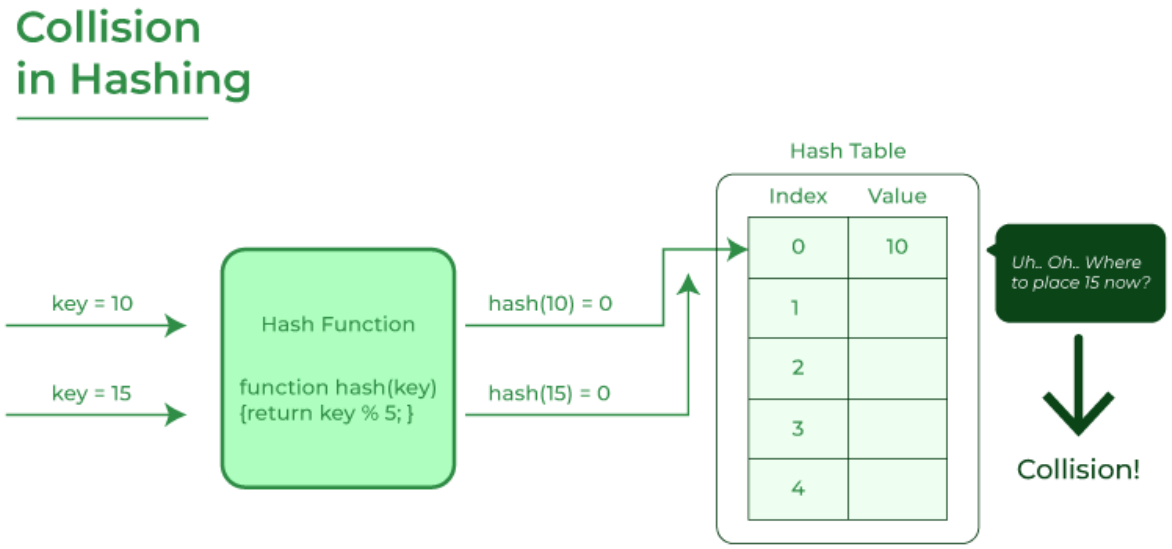
k = 12345
A = 0.357840
M = 100
h(12345) = floor[100 (12345*0.357840 mod 1)]
= floor[100 (4417.5348 mod 1)]
= floor[100 (0.5348)]
= floor[53.48]
= 53

Problem with Hashing

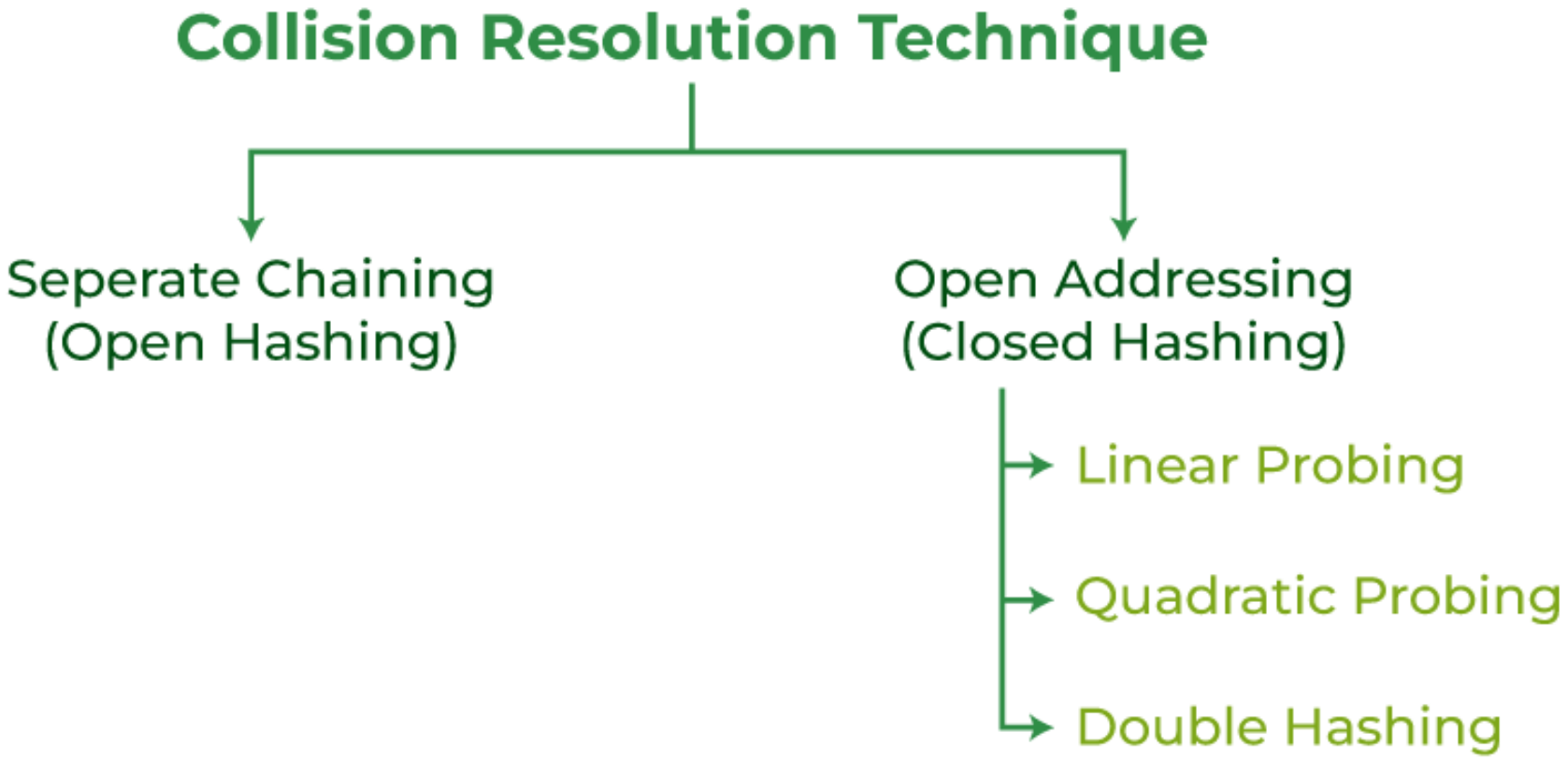
- If we consider the above example, the hash function we used is the sum of the letters, but if we examined the hash function closely then the problem can be easily visualized that for different strings same hash value is begin generated by the hash function.
- For example: {“ab”, “ba”} both have the same hash value, and string {“cd”,”be”} also generate the same hash value, etc. This is known as **collision** and it creates problem in searching, insertion, deletion, and updating of value.

What is collision?

- The hashing process generates a small number for a big key, so there is a possibility that two keys could produce the same value. The situation where the newly inserted key maps to an already occupied, and it must be handled using some collision handling technology.



What is Collision in Hashing

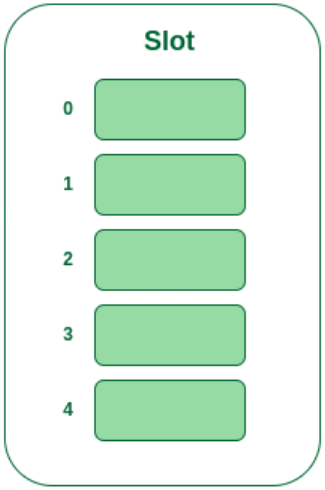


Collision resolution technique

1) Separate Chaining

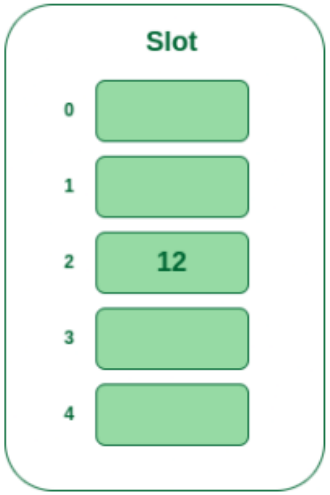
- The idea is to make each cell of the hash table point to a linked list of records that have the same hash function value. Chaining is simple but requires additional memory outside the table.
- Example: We have given a hash function and we have to insert some elements in the hash table using a separate chaining method for collision resolution technique.
- Hash function = key % 5,
- Elements = 12, 15, 22, 25 and 37.

Step 1: First draw the empty hash table which will have a possible range of hash values from 0 to 4 according to the hash function provided.



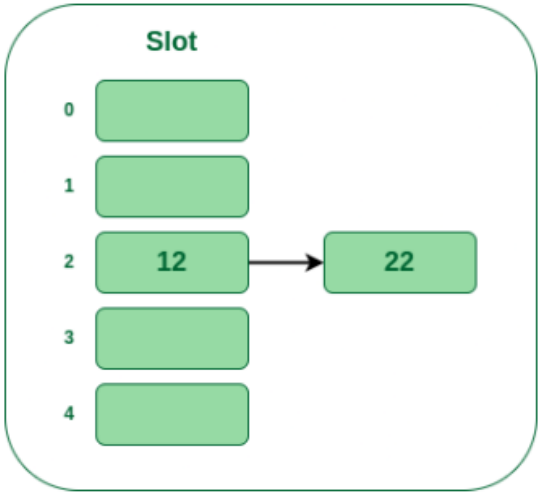
Hash table

- **Step 2:** Now insert all the keys in the hash table one by one. The first key to be inserted is 12 which is mapped to bucket number 2 which is calculated by using the hash function 12%5=2.



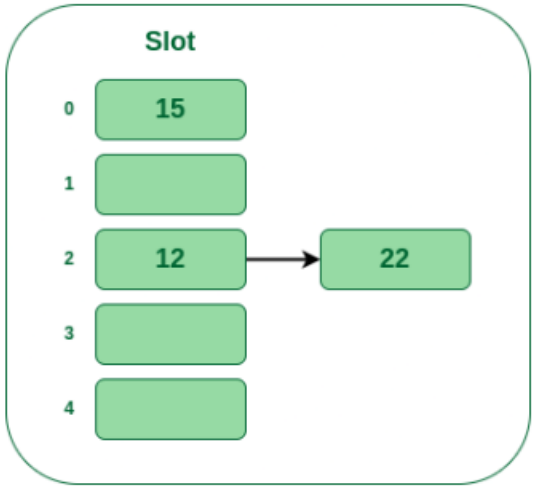
Insert 12

- **Step 3:** Now the next key is 22. It will map to bucket number 2 because 22%5=2. But bucket 2 is already occupied by key 12.



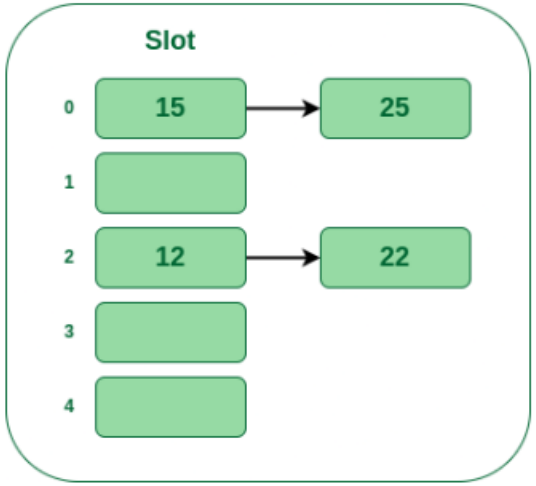
Insert 22

- **Step 4:** The next key is 15. It will map to slot number 0 because 15%5=0.



Insert 15

- **Step 5:** Now the next key is 25. Its bucket number will be 25%5=0. But bucket 0 is already occupied by key 25. So separate chaining method will again handle the collision by creating a linked list to bucket 0.



Insert 25

2) Open Addressing

In open addressing, all elements are stored in the hash table itself. Each table entry contains either a record or NIL. When searching for an element, we examine the table slots one by one until the desired element is found or it is clear that the element is not in the table.

2.a) Linear Probing

In linear probing, the hash table is searched sequentially that starts from the original location of the hash. If in case the location that we get is already occupied, then we check for the next location.

Algorithm:

1.Calculate the hash key. i.e. **key = data % size**

2.Check, if **hashTable[key]** is empty

- store the value directly by **hashTable[key] = data**

3.If the hash index already has some value then

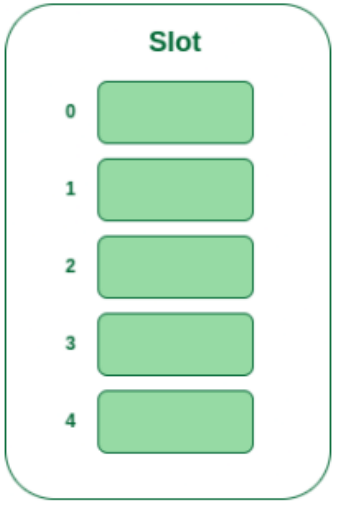
1. check for next index using **key = (key+1) % size**

4.Check, if the next index is available hashTable[key] then store the value. Otherwise try for next index.

5.Do the above process till we find the space.

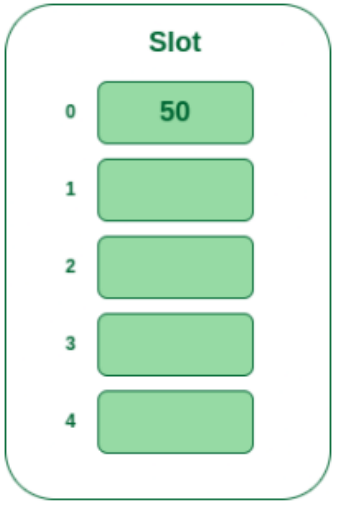
Example: Let us consider a simple hash function as “key mod 5” and a sequence of keys that are to be inserted are 50, 70, 76, 85, 93.

- **Step 1:** First draw the empty hash table which will have a possible range of hash values from 0 to 4 according to the hash function provided.



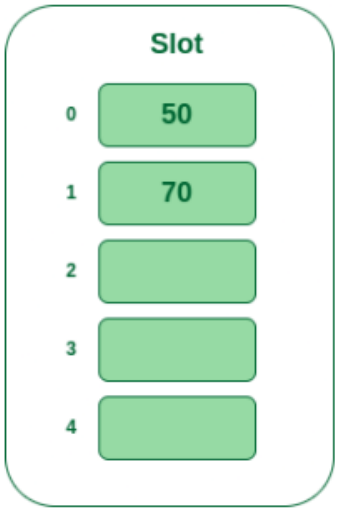
Hash table

- **Step 2:** Now insert all the keys in the hash table one by one. The first key is 50. It will map to slot number 0 because 50%5=0. So insert it into slot number 0.



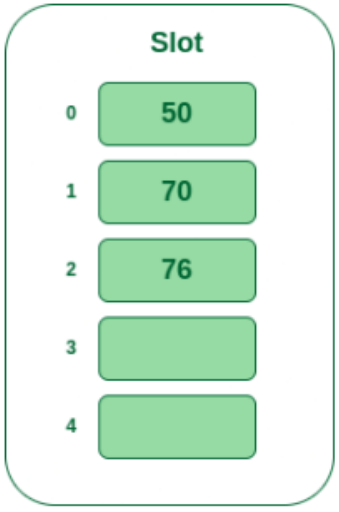
Insert 50 into hash table

- **Step 3:** The next key is 70. It will map to slot number 0 because 70%5=0 but 50 is already at slot number 0 so, search for the next empty slot and insert it.



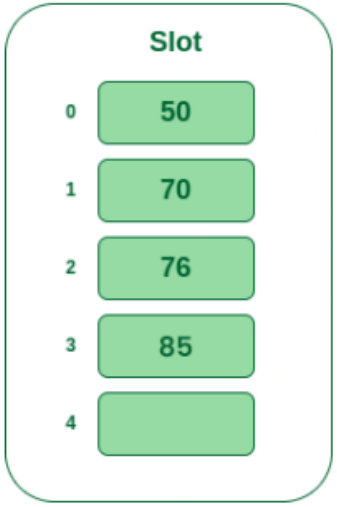
Insert 70 into hash table

- **Step 4:** The next key is 76. It will map to slot number 1 because 76%5=1 but 70 is already at slot number 1 so, search for the next empty slot and insert it.



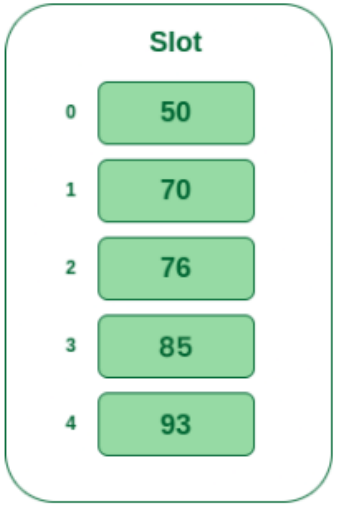
Insert 76 into hash table

- **Step 5:** The next key is 85 It will map to slot number 3 because 85%5=0, but 50 is already at slot number 0 so, search for the next empty slot and insert it. So insert it into slot number 3.



Insert 85 into hash table

- **Step 6:** The next key is 93 It will map to slot number 4 because 93%5=3, but 85 is already at slot number 3 so, search for the next empty slot and insert it. So insert it into slot number 4.



Insert 93 into hash table

2.b) Quadratic Probing

Quadratic probing is an open addressing scheme in computer programming for resolving hash collisions in hash tables. Quadratic probing operates by taking the original hash index and adding successive values of an arbitrary quadratic polynomial until an open slot is found.

An example sequence using quadratic probing is:

H + 1^2, **H** + 2^2, **H** + 3^2, **H** + 4^2..... **H** + k^2

This method is also known as the mid-square method because in this method we look for i^2'th probe (slot) in i'th iteration and the value of i = 0, 1, . . . n – 1. We always start from the original hash location. If only the location is occupied then we check the other slots.

Let hash(x) be the slot index computed using the hash function and n be the size of the hash table.

If the slot hash(x) % n is full, then we try (hash(x) + 1^2) % n.

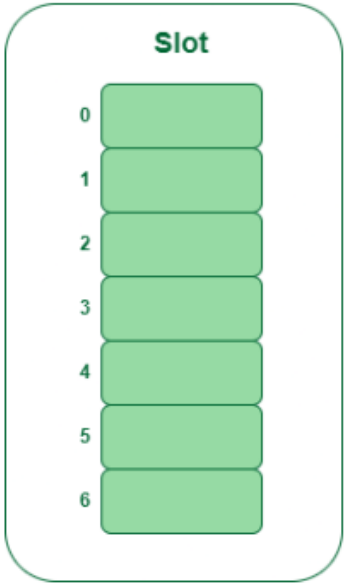
If (hash(x) + 1^2) % n is also full, then we try (hash(x) + 2^2) % n.

If (hash(x) + 2^2) % n is also full, then we try (hash(x) + 3^2) % n.

This process will be repeated for all the values of **i** until an empty slot is found

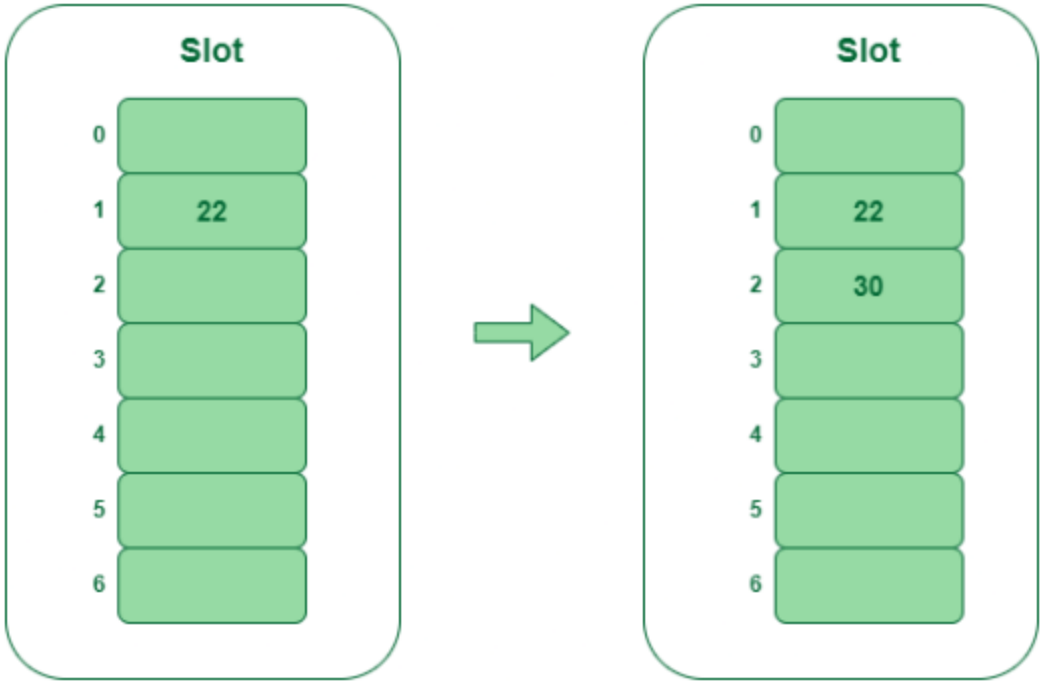
Example: Let us consider table Size = 7, hash function as Hash(x) = x % 7 and collision resolution strategy to be $f(i) = i^2$. Insert = 22, 30, and 50

- **Step 1:** Create a table of size 7.



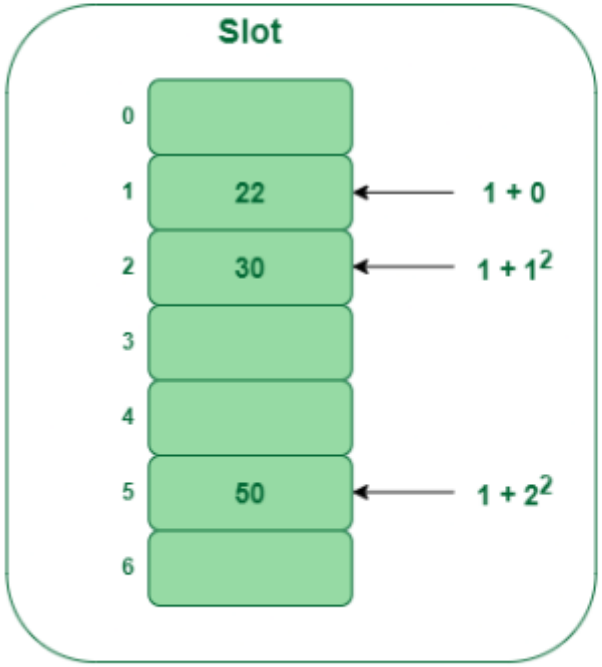
Hash table

- **Step 2** – Insert 22 and 30
  - Hash(22) = 22 % 7 = 1, Since the cell at index 1 is empty, we can easily insert 22 at slot 1.
  - Hash(30) = 30 % 7 = 2, Since the cell at index 2 is empty, we can easily insert 30 at slot 2.



Insert key 22 and 30 in the hash table

- **Step 3:** Inserting 50
  - Hash(50) = 50 % 7 = 1
  - In our hash table slot 1 is already occupied. So, we will search for slot $1+1^2$, i.e. 1+1 = 2,
  - Again slot 2 is found occupied, so we will search for cell $1+2^2$, i.e.1+4 = 5,
  - Now, cell 5 is not occupied so we will place 50 in slot 5.



Insert key 50 in the hash table

2.c) Double Hashing

Double hashing is a collision resolving technique in Open Addressed Hash tables. Double hashing make use of two hash function,

- The first hash function is **h1(k)** which takes the key and gives out a location on the hash table. But if the new location is not occupied or empty then we can easily place our key.
- But in case the location is occupied (collision) we will use secondary hash-function **h2(k)** in combination with the first hash-function **h1(k)** to find the new location on the hash table.

This combination of hash functions is of the form

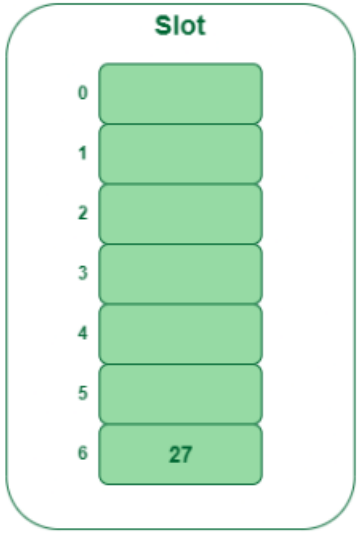
```
h(k, i) = (h1(k) + i * h2(k)) % n
```

where

- i is a non-negative integer that indicates a collision number,
- k = element/key which is being hashed
- n = hash table size.

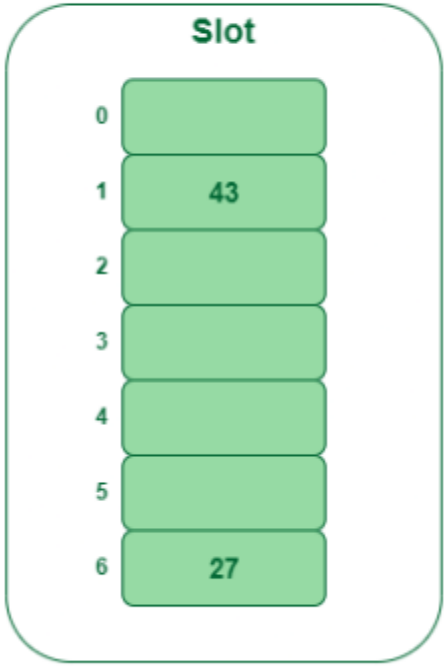
Example: Insert the keys 27, 43, 692, 72 into the Hash Table of size 7. where first hash-function is **h1(k) = k mod 7** and second hash-function is **h2(k) = 1 + (k mod 5)**

- **Step 1:** Insert 27
  - 27 % 7 = 6, location 6 is empty so insert 27 into 6 slot.



Insert key 27 in the hash table

- **Step 2:** Insert 43
  - 43 % 7 = 1, location 1 is empty so insert 43 into 1 slot.

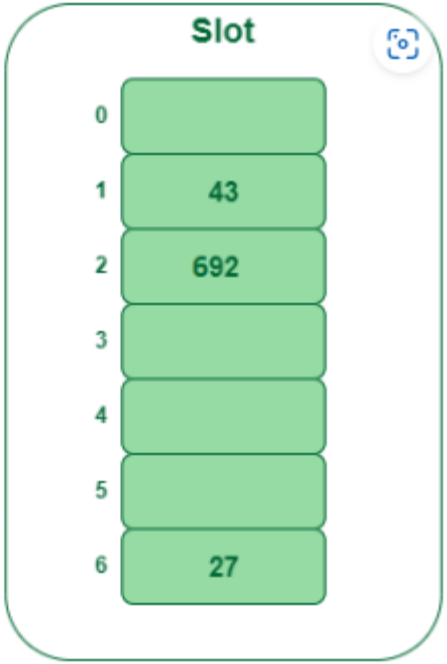


Insert key 43 in the hash table

- **Step 3:** Insert 692
  - 692 % 7 = 6, but location 6 is already being occupied and this is a collision
  - So we need to resolve this collision using double hashing.

```
h_new = [h1(692) + i * (h2(692)] % 7
= [6 + 1 * (1 + 692 % 5)] % 7
= 9 % 7
= 2

Now, as 2 is an empty slot,
so we can insert 692 into 2nd slot.
```

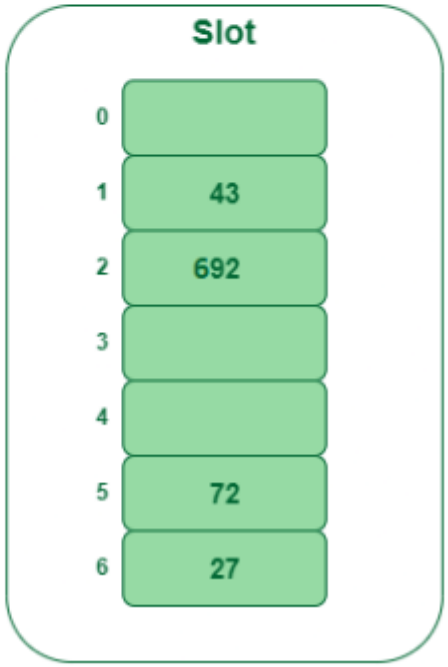


Insert key 692 in the hash table

- **Step 4:** Insert 72
  - 72 % 7 = 2, but location 2 is already being occupied and this is a collision.
  - So we need to resolve this collision using double hashing.

```
h_new = [h1(72) + i * (h2(72)] % 7
= [2 + 1 * (1 + 72 % 5)] % 7
= 5 % 7
= 5,

Now, as 5 is an empty slot,
so we can insert 72 into 5th slot.
```



Insert key 72 in the hash table

Applications of Hashing

- Hash tables are used in a wide variety of applications, including:
- **Databases:** Storing and retrieving data based on unique keys
- **Caching:** Storing frequently accessed data for faster retrieval
- **Symbol Tables:** Mapping identifiers to their values in programming languages
- **Network Routing:** Determining the best path for data packets

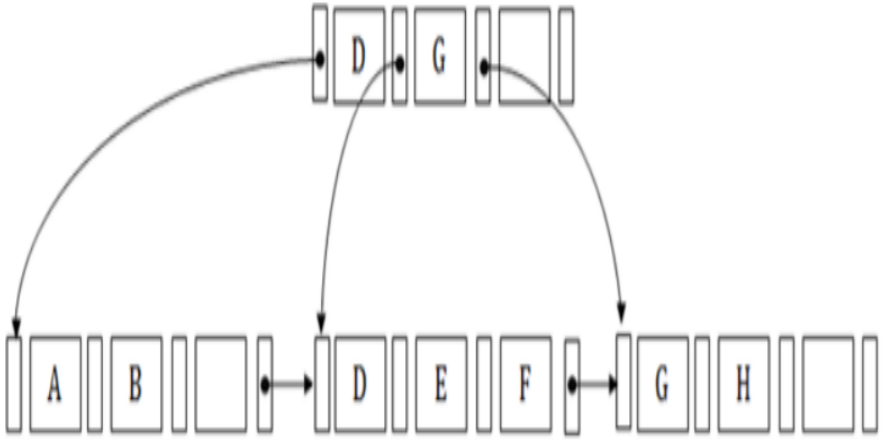
B+ Tree

- The B+ tree is a **balanced binary search tree**.
- It follows a **multi-level index** format.
- In the B+ tree, leaf nodes denote **actual data pointers**. B+ tree ensures that **all leaf nodes remain at the same height**.
- In the B+ tree, the leaf nodes are **linked using a link list**. Therefore, a B+ tree can support **random access as well as sequential access**.

Structure of B+ Tree:

- In the B+ tree, every leaf node is at **equal distance** from the root node.
- The B+ tree is of the order n where n is fixed for every B+ tree.
- It contains an internal node and leaf node.

Structure of B+ Tree



Internal node:

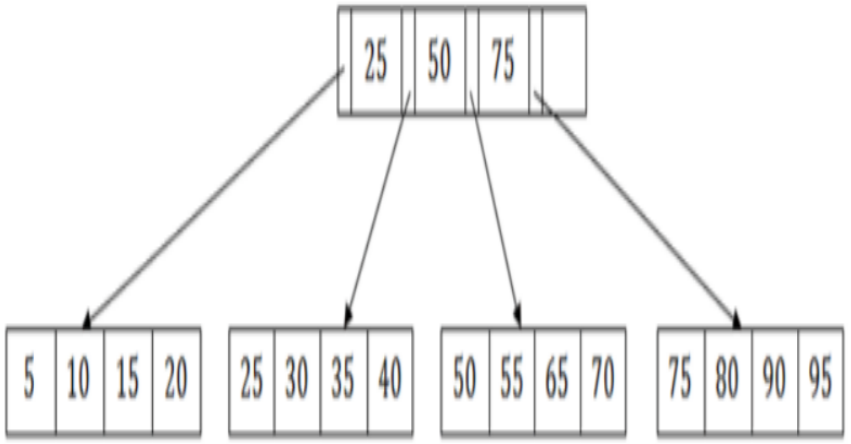
- An internal node of the B+ tree can contain at least **n/2 record pointers** except the root node.
- At most, an internal node of the tree contains **n pointers**.

Leaf node:

- The leaf node of the B+ tree can contain at least **n/2 record pointers** and **n/2 key values**.
- At most, a leaf node contains **n record pointer**

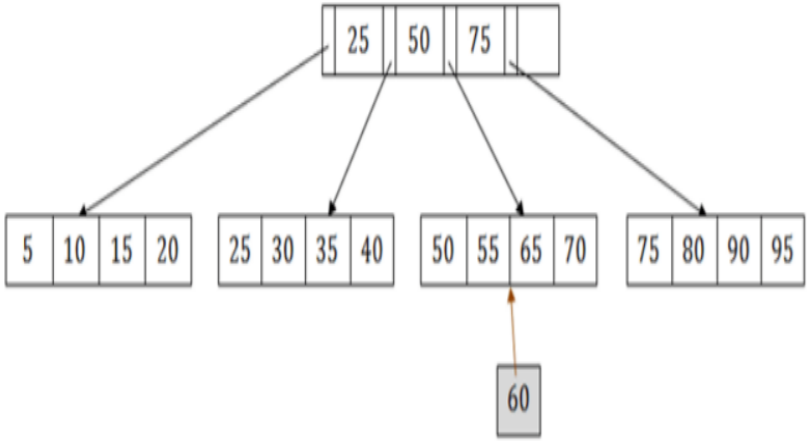
Searching a record in B+ Tree

- Suppose we have to search 55 in the below B+ tree structure.
- First, we will **fetch for the intermediary node** which will direct to the leaf node that can contain a record for 55.
- So, in the intermediary node, we will find a branch between 50 and 75 nodes.
- Then at the end, we will be redirected to the third leaf node.
- It will perform a sequential search to find 55.

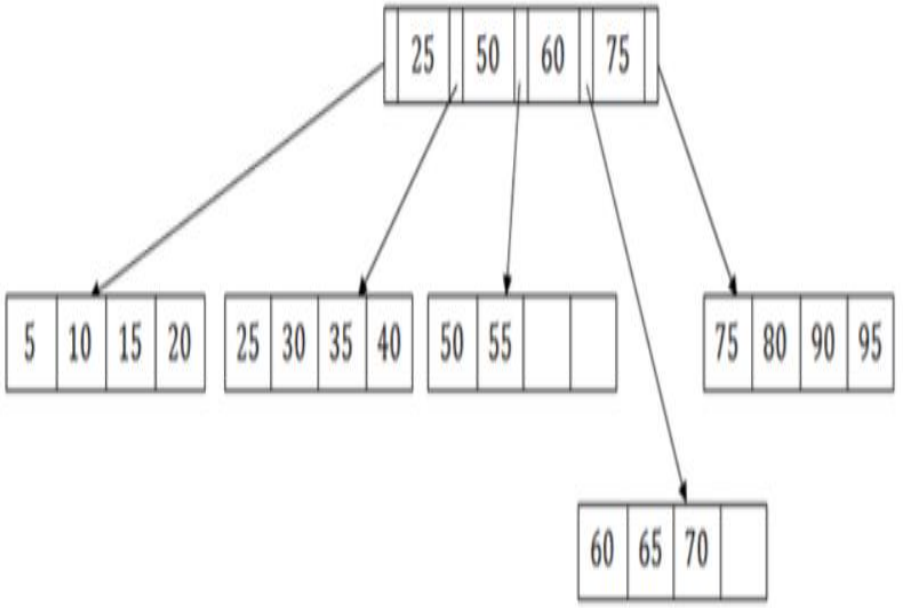


B+ Tree Insertion

- Suppose we want to insert a record 60 in the below structure. It will go to the 3rd leaf node after 55.
- It is a balanced tree, and a leaf node of this tree is already full, so we cannot insert 60 there.
- In this case, we have to **split the leaf node**, so that it can be inserted into tree without affecting the **fill factor, balance and order**.

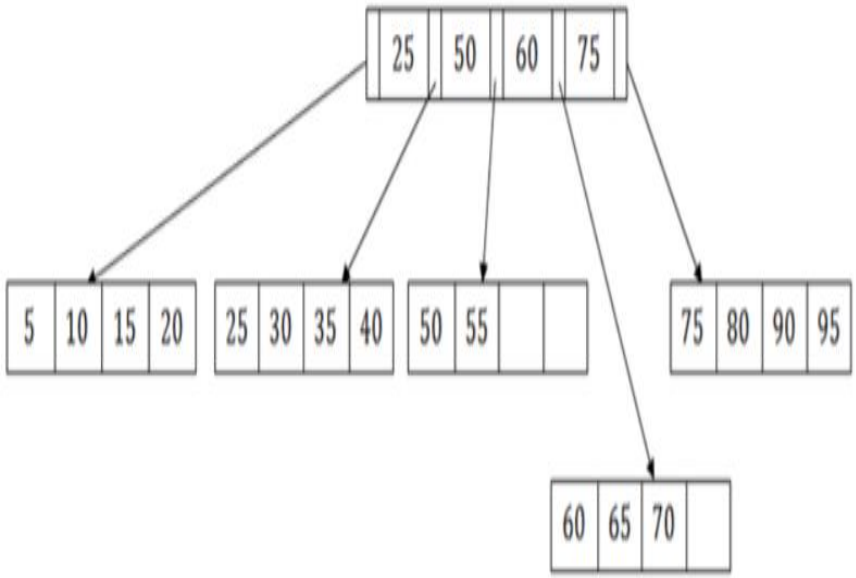


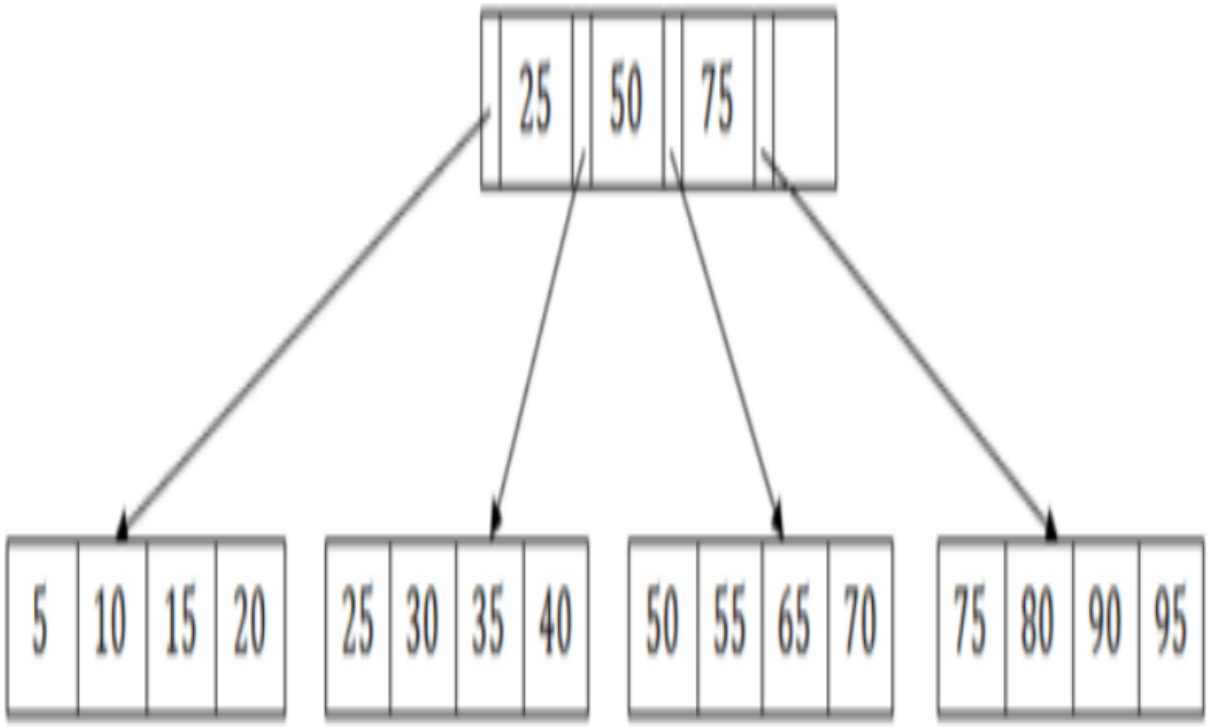
- The 3rd leaf node has the values (50, 55, 60, 65, 70) and its current root node is 50. We will split the leaf node of the tree in the middle so that its **balance is not altered**. So we can group (50, 55) and (60, 65, 70) into 2 leaf nodes. If these two has to be leaf nodes, the intermediate node cannot branch from 50. It should have 60 added to it, and then we can have pointers to a new leaf node.
- This is how we can insert an entry when there is overflow. In a normal scenario, it is very easy to find the node where it fits and then place it in that leaf node.



B+ Tree Deletion

- Suppose we want to delete 60 from the above example.
- In this case, we have to remove 60 from the intermediate node as well as from the 4th leaf node too.
- If we remove it from the intermediate node, then the tree will not satisfy the rule of the B+ tree.
- So we need to modify it to have a balanced tree. After deleting node 60 from above B+ tree and re-arranging the nodes, it will show as follows:




25
50
75
5
10
15
20
25
30
35
40
50
55
65
70
75
80
90
95