Chapter 7 Storage Management and Indexing

7.1 Disk and Storage

7.2 Organization of records into block

7.3 File Organization

7.4 B+ Tree index

7.5 Hash Index

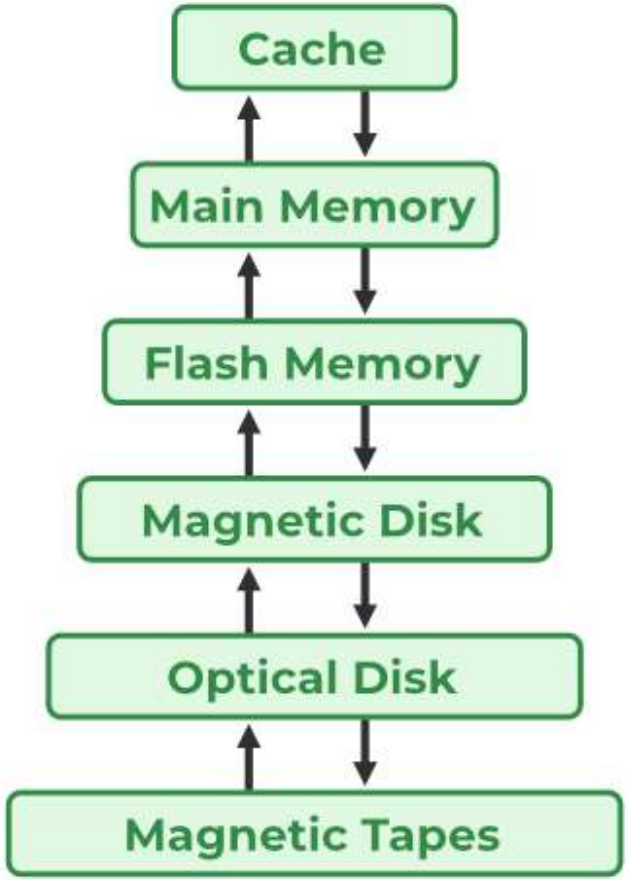
7.1 Disk and Storage

- Databases are stored in file formats, which contains records.
- At physical level, actual data is stored in electromagnetic format on some device capable of storing it for a longer amount of time.
- Several types of data storage exist in most computer systems. Among the media typically available are these:
  - Cache.
  - Main memory.
  - Flash memory
  - Magnetic-disk storage.
  - Optical storage.
  - Tape storage.

Write about each storage(Assignment)

7.1 Disk and Storage

Storage Device Hierarchy



7.2 Organization of records into block

- A user can store the data in files in an organized manner.
- These files are organized logically as a sequence of records and reside permanently on disks.
- Each file is divided into fixed-length storage units known as Blocks.
- These blocks are the units of storage allocation as well as data transfer.
- Although the default block size in the database is 4 to 7 kilobytes, many databases allow specifying the size at the time of creating the database instance.

7.2 Organization of records into block

- Usually, the record size is smaller than the block size.
- But, for large data items such as images, the size can vary.
- For accessing the data quickly, it is required that one complete record should reside in one block only.
- It should not be partially divided between one or two blocks.
- In RDBMS, the size of tuples varies in different relations.
- Thus, we need to structure our files in multiple lengths for implementing the records.

7.2 Organization of records into block

- In file organization, there are two possible ways of representing the records:
  - Fixed-length records
  - Variable-length records

7.2 Organization of records into block

Fixed length Records

- Fixed-length records means setting a length and storing the records into the file.
- If the record size exceeds the fixed size, it gets divided into more than one block.
- Due to the fixed size there occurs following two problems:
  - Partially storing subparts of the record in more than one block requires access to all the blocks containing the subparts to read or write in it.
  - It is difficult to delete a record in such a file organization. It is because if the size of the existing record is smaller than the block size, then another record or a part fills up the block.

7.2 Organization of records into block

Fixed length Records

- However, including a certain number of bytes is the solution to the above problems.
- It is known as File Header. The allocated file header carries a variety of information about the file, such as the address of the first record.
- The address of the second record gets stored in the first record and so on. This process is similar to pointers.
- The method of insertion and deletion is easy in fixed-length records because the space left or freed by the deleted record is exactly similar to the space required to insert the new records.
- But this process fails for storing the records of variable lengths.

7.2 Organization of records into block

Variable length Records

- Variable-length records are the records that vary in size.
- It requires the creation of multiple blocks of multiple sizes to store them.
- These variable-length records are kept in the following ways in the database system:
  - Storage of multiple record types in a file.
  - It is kept as Record types that enable repeating fields like multisets or arrays.
  - It is kept as Record types that enable variable lengths either for one field or more.

7.2 Organization of records into block

Variable length Records

- In variable-length records, there exist the following two problems:
  - Defining the way of representing a single record so as to extract the individual attributes easily.
  - Defining the way of storing variable-length records within a block so as to extract that record in a block easily.

7.2 Organization of records into block

Variable length Records

- Thus, the representation of a variable-length record can be divided into two parts:
  - An initial part of the record with fixed-length attributes such as numeric values, dates, fixed-length character attributes for storing their value.
  - The data for variable-length attributes such as varchar type is represented in the initial part of the record by (offset, length) pair. The offset refers to the place where that record begins, and length refers to the length of the variable-size attribute. Thus, the initial part stores fixed-size information about each attribute, i.e., whether it is the fixed-length or variable-length attribute.

7.3 File Organization

- File organization is a way of organizing the data in such way so that it is easier to insert, delete, modify and retrieve data from the files.
- The method of mapping file records to disk blocks defines file organization, i.e. how the file records are organized.
- File Organization refers to the logical relationships among various records that constitute the file, particularly with respect to the means of identification and access to any specific record.
- In simple terms, Storing the files in a certain order is called File Organization.

7.3 File Organization

The Objective of File Organization

- It helps in the faster selection of records i.e. it makes the process faster.
- Different Operations like inserting, deleting, and update on different records are faster and easier.
- It prevents us from inserting duplicate records via various operations.
- It helps in storing the records or the data very efficiently at a minimal cost

7.3 File Organization

Types of File Organizations

- Sequential File Organization
- Heap File Organization
- Hash File Organization
- Clustered File Organization

7.3 File Organization

Sequential File Organization

- The easiest method for file Organization is the Sequential method. In this method, the file is stored one after another in a sequential manner.
- There are two ways to implement this method:
  - **Pile File Method**
  - **Sorted File Method**

7.3 File Organization

Sequential File Organization

1. Pile File Method

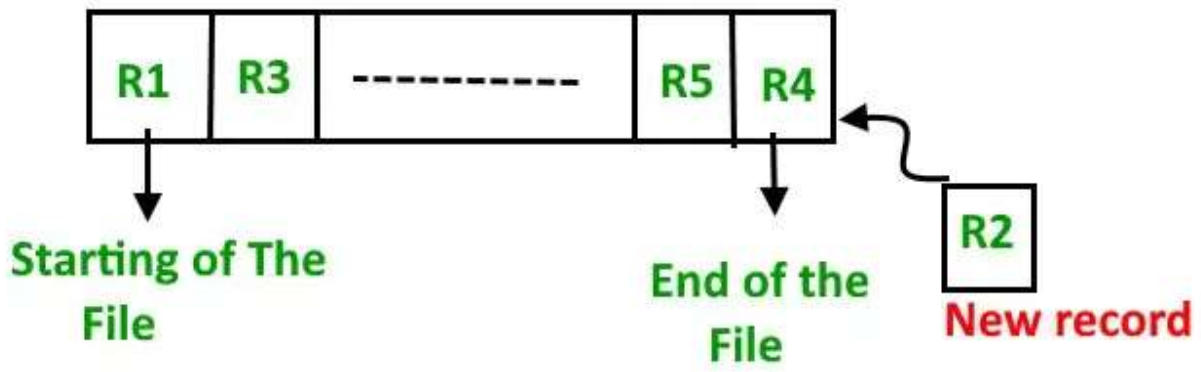
- This method is quite simple, in which we store the records in a sequence i.e. one after the other in the order in which they are inserted into the tables.

7.3 File Organization

Sequential File Organization

1. Pile File Method

- Insertion of the new record: Let the R1, R3, and so on up to R5 and R4 be four records in the sequence. Here, records are nothing but a row in any table. Suppose a new record R2 has to be inserted in the sequence, then it is simply placed at the end of the file.

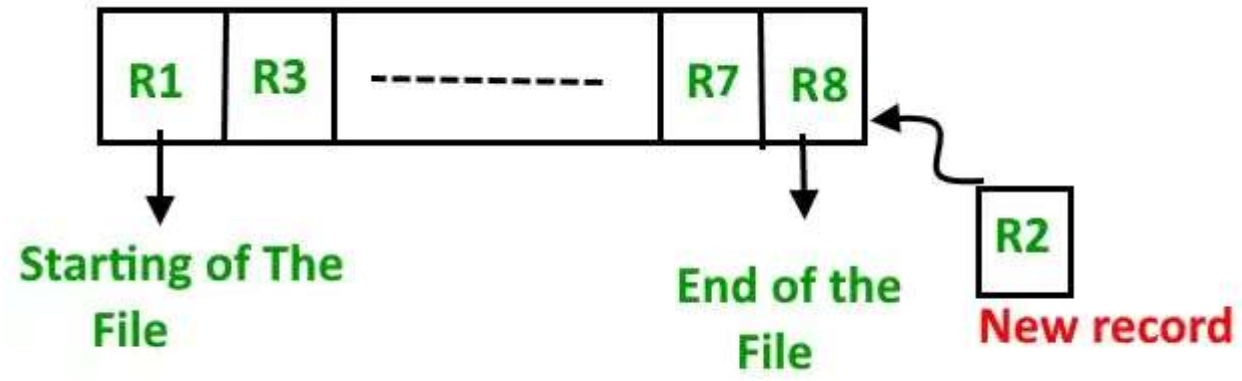


7.3 File Organization

Sequential File Organization

2. Sorted File Method

- In this method, As the name itself suggests whenever a new record has to be inserted, it is always inserted in a sorted (ascending or descending) manner. The sorting of records may be based on any primary key or any other key.

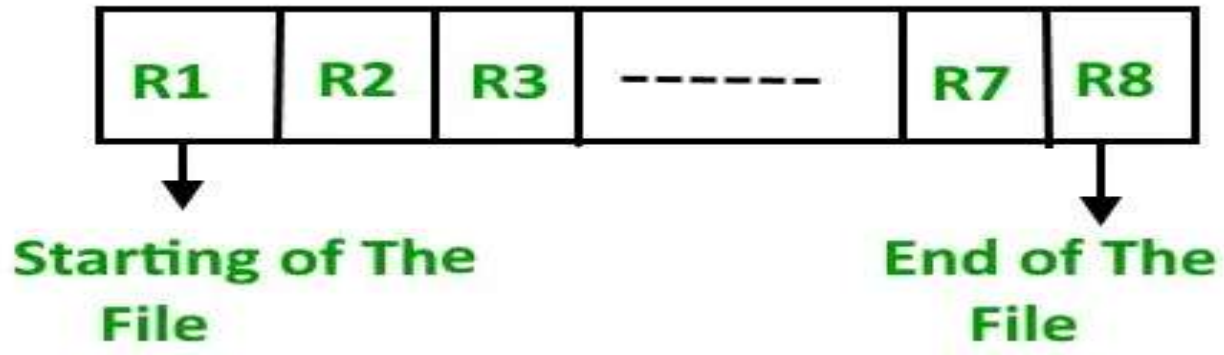


7.3 File Organization

Sequential File Organization

2. Sorted File Method

- Insertion of the new record: Let us assume that there is a preexisting sorted sequence of four records R1, R3, and so on up to R7 and R7. Suppose a new record R2 has to be inserted in the sequence, then it will be inserted at the end of the file and then it will sort the sequence.



7.3 File Organization

Sequential File Organization

Advantages of Sequential File Organization

- Fast and efficient method for huge amounts of data.
- Simple design.
- Files can be easily stored in magnetic tapes i.e. cheaper storage mechanism.

7.3 File Organization

Sequential File Organization

- **Disadvantages of Sequential File Organization**
  - Time wastage as we cannot jump on a particular record that is required, but we have to move in a sequential manner which takes our time.
  - The sorted file method is inefficient as it takes time and space for sorting records.

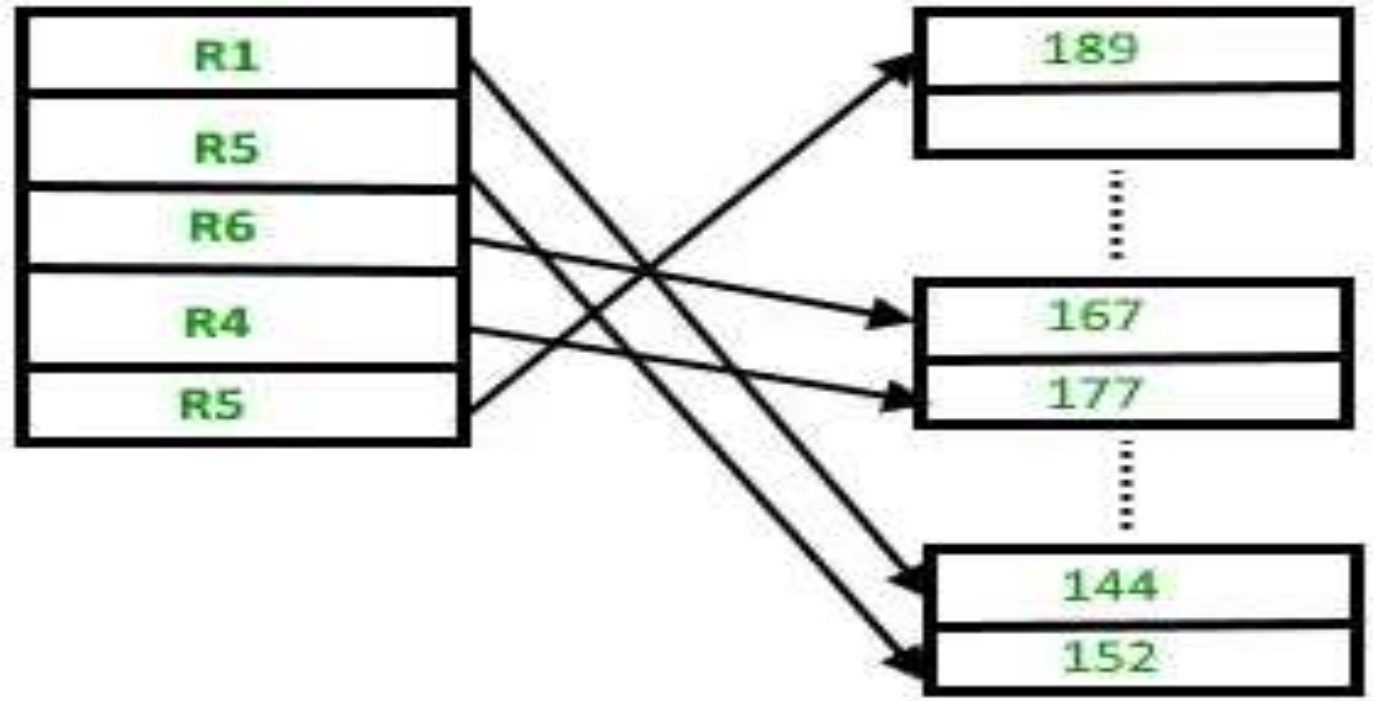
7.3 File Organization

Heap File Organization

- Heap File Organization works with data blocks. In this method, records are inserted at the end of the file, into the data blocks.
- No Sorting or Ordering is required in this method.
- If a data block is full, the new record is stored in some other block, Here the other data block need not be the very next data block, but it can be any block in the memory.
- It is the responsibility of DBMS to store and manage the new records.

7.3 File Organization

Heap File Organization



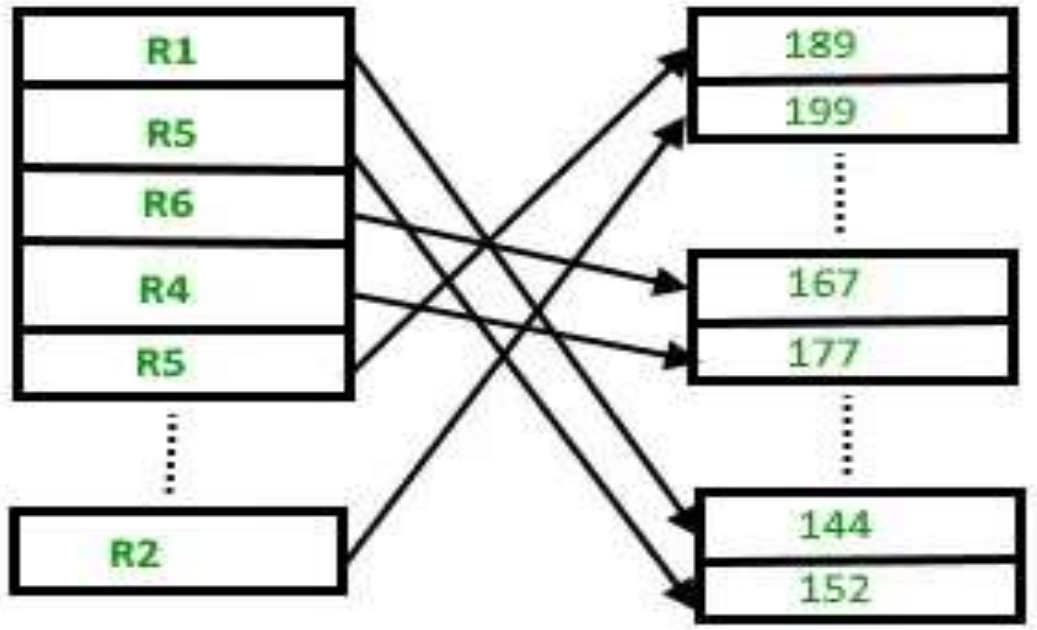
7.3 File Organization

Heap File Organization

- Insertion of the new record: Suppose we have four records in the heap R1, R5, R6, R4, and R3, and suppose a new record R2 has to be inserted in the heap then, since the last data block i.e data block 3 is full it will be inserted in any of the data blocks selected by the DBMS, let’s say data block 1.

7.3 File Organization

Heap File Organization



7.3 File Organization

Heap File Organization

- **Advantages of Heap File Organization**
  - Fetching and retrieving records is faster than sequential records but only in the case of small databases.
  - When there is a huge number of data that needs to be loaded into the database at a time, then this method of file Organization is best suited.
- **Disadvantages of Heap File Organization**
  - The problem of unused memory blocks.
  - Inefficient for larger databases.

7.3 File Organization

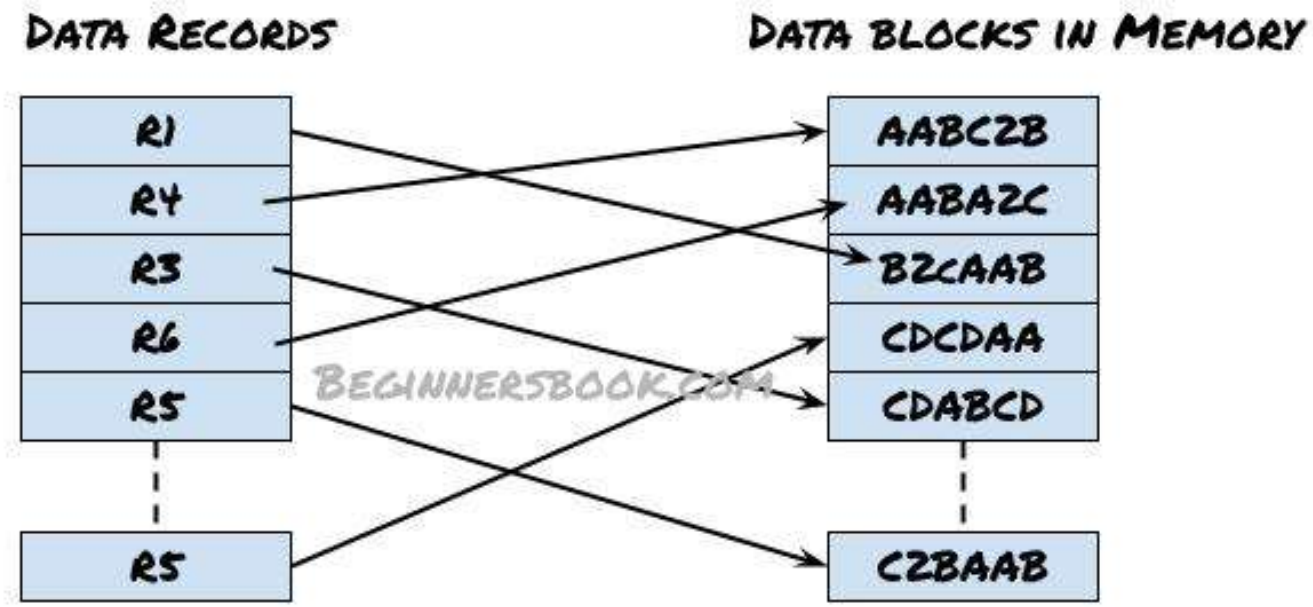
Hash File Organization

- In this method, hash function is used to compute the address of a data block in memory to store the record.
- The hash function is applied on certain columns of the records, known as hash columns to compute the block address.
- These columns/fields can either be key or non-key attributes.

7.3 File Organization

Hash File Organization

- In this method, hash function is used to compute the address of a data block in memory to store the record.



HASH FILE ORGANIZATION

7.3 File Organization

Hash File Organization

- These memory addresses are computed by applying hash function on certain attributes of these records.
- Fetching a record is faster in this method as the record can be accessed using hash key column. No need to search through the entire file to fetch a record.

7.3 File Organization

Hash File Organization

Advantages of Hash File Organization

- This method doesn’t require sorting explicitly as the records are automatically sorted in the memory based on hash keys.
- Reading and fetching a record is faster compared to other methods as the hash key is used to quickly read and retrieve the data from database.
- Records are not dependent on each other and are not stored in consecutive memory locations so that prevents the database from read, write, update, delete anomalies.

7.3 File Organization

Hash File Organization

Disadvantages of Hash File Organization

- Can cause accidental deletion of data, if columns are not selected properly for hash function. For example, while deleting an Employee "Steve" using Employee_Name as hash column can cause accidental deletion of other employee records if the other employee name is also "Steve". This can be avoided by selecting the attributes properly, for example in this case combining age, department or SSN with the employee_name for hash key can be more accurate in finding the distinct record.
- Memory is not efficiently used in hash file organization as records are not stored in consecutive memory locations.
- If there are more than one hash columns, searching a record using a single attribute will not give accurate results.

7.3 File Organization

Cluster File Organization

- Cluster file organization is different from the other file organization methods.
- Other file organization methods mainly focus on organizing the records in a single file (table).
- Cluster file organization is used, when we frequently need combined data from multiple tables.
- While other file organization methods organize tables separately and combine the result based on the query, cluster file organization stores the combined data of two or more frequently joined tables in the same file known as cluster. This helps in accessing the data faster.

7.3 File Organization

Cluster File Organization

EMPLOYEE TABLE

EMP_ID	EMP_NAME	EMP_ADD	EMP_DEP
1001	STEVE	ADDR1	D01
1002	JOHN	ADDR2	D02
1003	AJITH	ADDR2	D02
1004	RAM	ADDR4	D03
1005	CHAITANYA	ADDR1	D01

DEPARTMENT TABLE

EMP_DEP	DEPT_NAME
D01	SALES
D02	MARKETING
D03	HR

CLUSTER KEY

CLUSTER

EMP_DEP	DEPT_NAME	EMP_ID	EMP_NAME	EMP_ADD
D01	SALES	1001	STEVE	ADDR1
D01	SALES	1005	CHAITANYA	ADDR1
D02	MARKETING	1002	JOHN	ADDR2
D02	MARKETING	1003	AJITH	ADDR2
D03	HR	1004	RAM	ADDR4

7.3 File Organization

Cluster File Organization

- There are two types of cluster file organizations:
  - **Index based cluster file organization**
  - **Hash based cluster file organization**

7.3 File Organization

Cluster File Organization

- **Index based cluster file organization:** The example that we have shown in the above diagram is an index based cluster file organization. In this type, the cluster is formed based on the cluster key and this cluster key works as an index of the cluster.
- Since EMP_DEP field is common in both the tables, this becomes the cluster key when these two tables joined to form the cluster. Whenever we need to find the combined record of employees and department based on the EMP_DEP, this cluster can be used to quickly retrieve the data.

7.3 File Organization

Cluster File Organization

- **Hash based cluster file organization:** This is same as index based cluster file organization except that in this type, the hash function is applied on the cluster key to generate the hash value and that value is used in the cluster instead of the index.

- **Note:** The main difference between these two types is that in index based cluster, records are stored with cluster key while in hash based cluster, the records are stored with the hash value of the cluster key.

7.3 File Organization

Cluster File Organization

Advantages of cluster file organization

- This method is popularly used when multiple tables needs to be joined frequently based on the same condition.
- When a table in database is joined with multiple tables of the same database then cluster file organization method will be more efficient compared to other file organization methods.

7.3 File Organization

Cluster File Organization

Disadvantages of cluster file organization

- **Not suitable for large databases:** This method is not suitable if the size of the database is huge as the performance of various operations on the data will be poor.
- **Not flexible with joining condition:** This method is not suitable if the join condition of the tables keep changing, as it may take additional time to traverse the joined tables again for the new condition.
- **Isolated tables:** If tables are not that related and there is rarely any join query on tables then using this file organization is not recommended. This is because maintaining the cluster for such tables will be useless when it is not used frequently.

7.4 Indexing

- Indexing is used to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed.
- The index is a type of data structure. It is used to locate and access the data in a database table quickly.
- Indexes can be created using some database columns.

7.4 Indexing

Index structure:

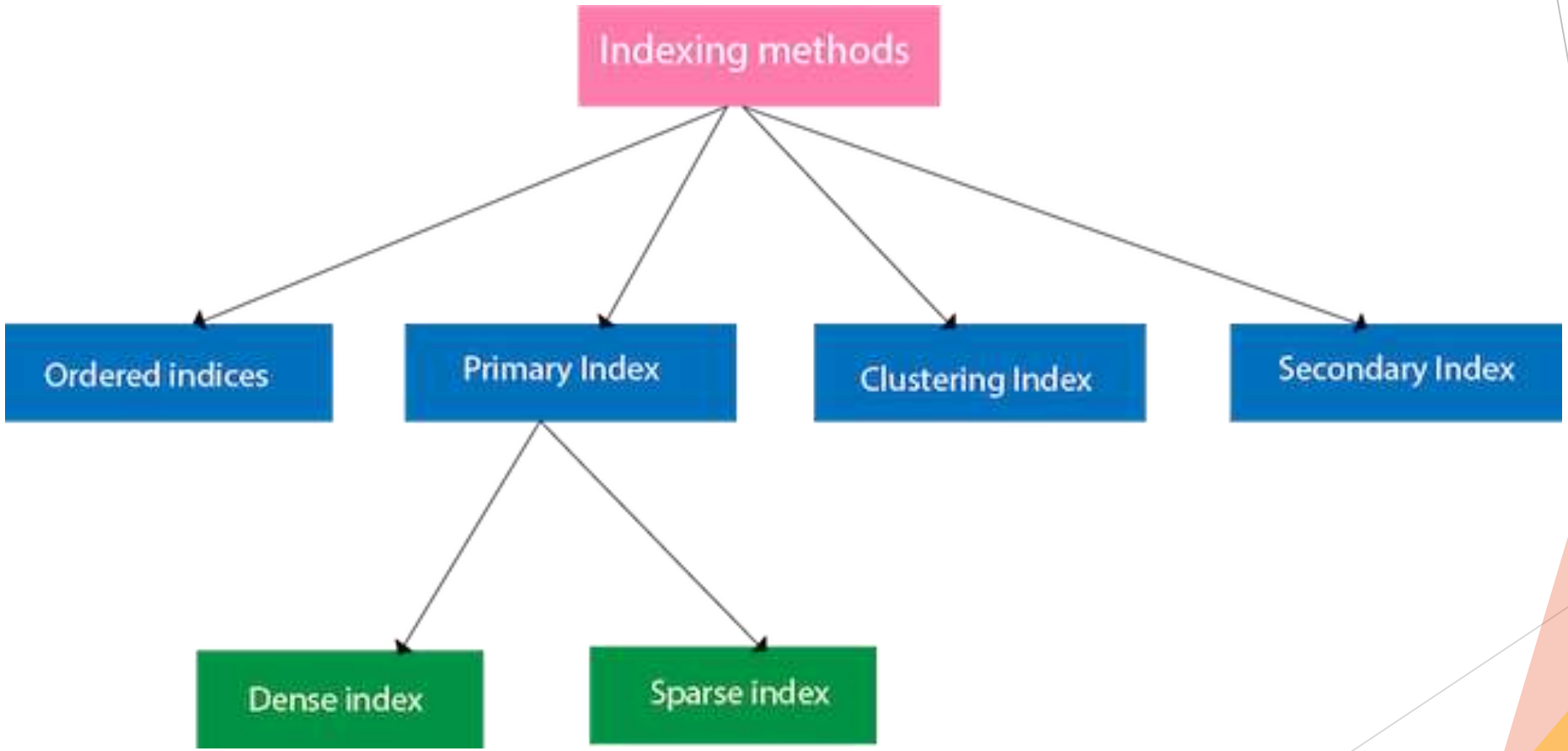
- The first column of the database is the search key that contains a copy of the primary key or candidate key of the table. The values of the primary key are stored in sorted order so that the corresponding data can be accessed easily.
- The second column of the database is the data reference. It contains a set of pointers holding the address of the disk block where the value of the particular key can be found.

Search key	Data Reference

Fig: Structure of Index

7.4 Indexing

- Types of indexing



7.4 Indexing

Ordered indices

- The indices are usually sorted to make searching faster. The indices **which are sorted** are known as ordered indices.
- **Example:** Suppose we have an employee table with thousands of record and each of which is 10 bytes long. If their IDs start with 1, 2, 3....and so on and we have to search student with ID-543.
  - In the case of a database with no index, we have to search the disk block from starting till it reaches 543. The DBMS will read the record after reading 543*10=5430 bytes.
  - In the case of an index, we will search using indexes and the DBMS will read the record after reading 542*2= 1074 bytes which are very less compared to the previous case.

7.4 Indexing

Primary index

- If the index is created on the basis of the **primary key of the table**, then it is known as primary indexing.
- These primary keys are unique to each record and **contain 1:1 relation** between the records.
- As primary keys are stored in **sorted order**, the performance of the searching operation is quite efficient.
- The primary index can be classified into two types:
  - **Dense index**
  - **Sparse index**

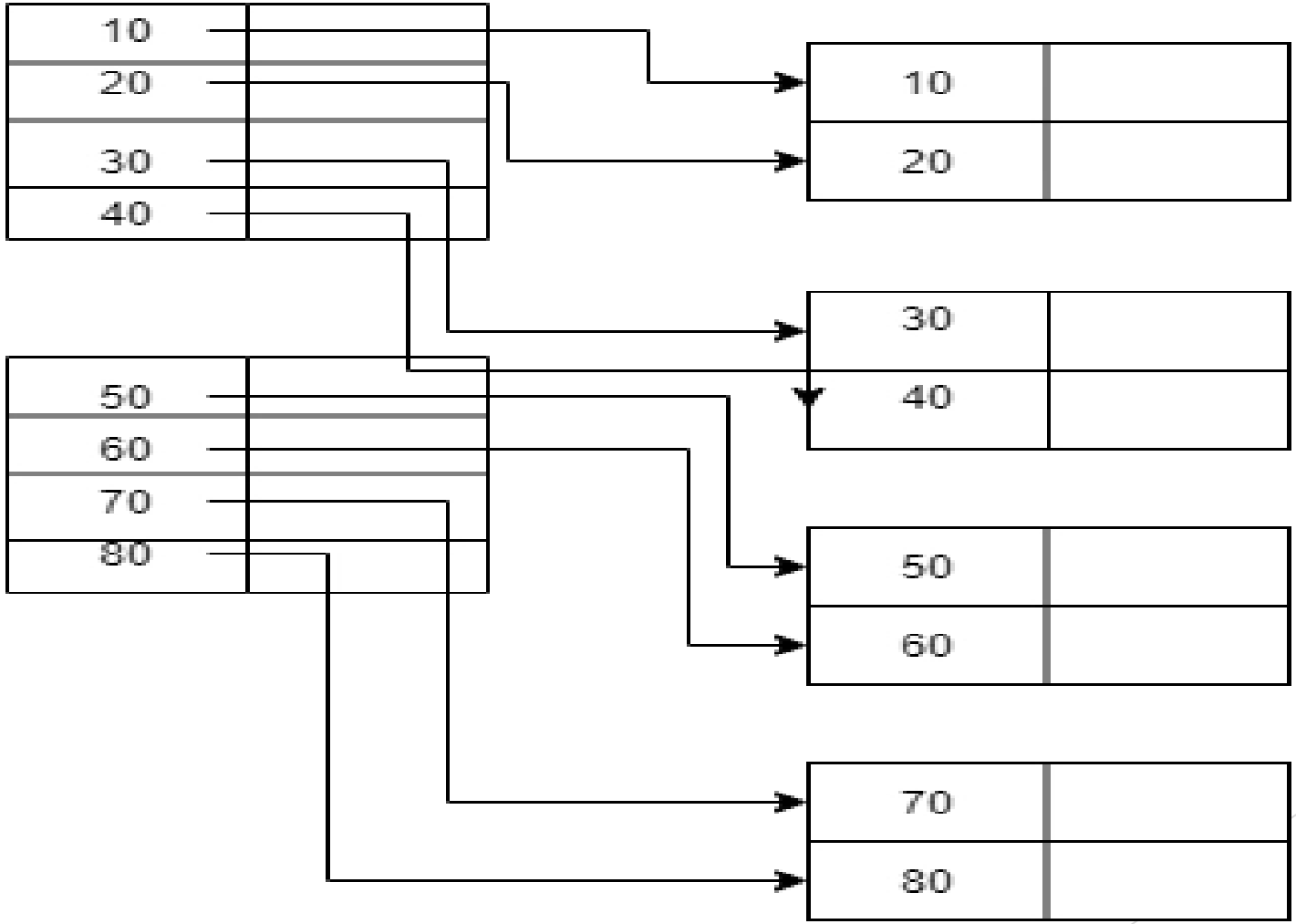
7.4 Indexing

Dense index

- The dense index contains an index record for every search key value in the data file. It makes searching faster.
- In this, the number of records in the index table is same as the number of records in the main table.
- It needs more space to store index record itself.
- The index records have the search key and a pointer to the actual record on the disk.

7.4 Indexing

Dense index



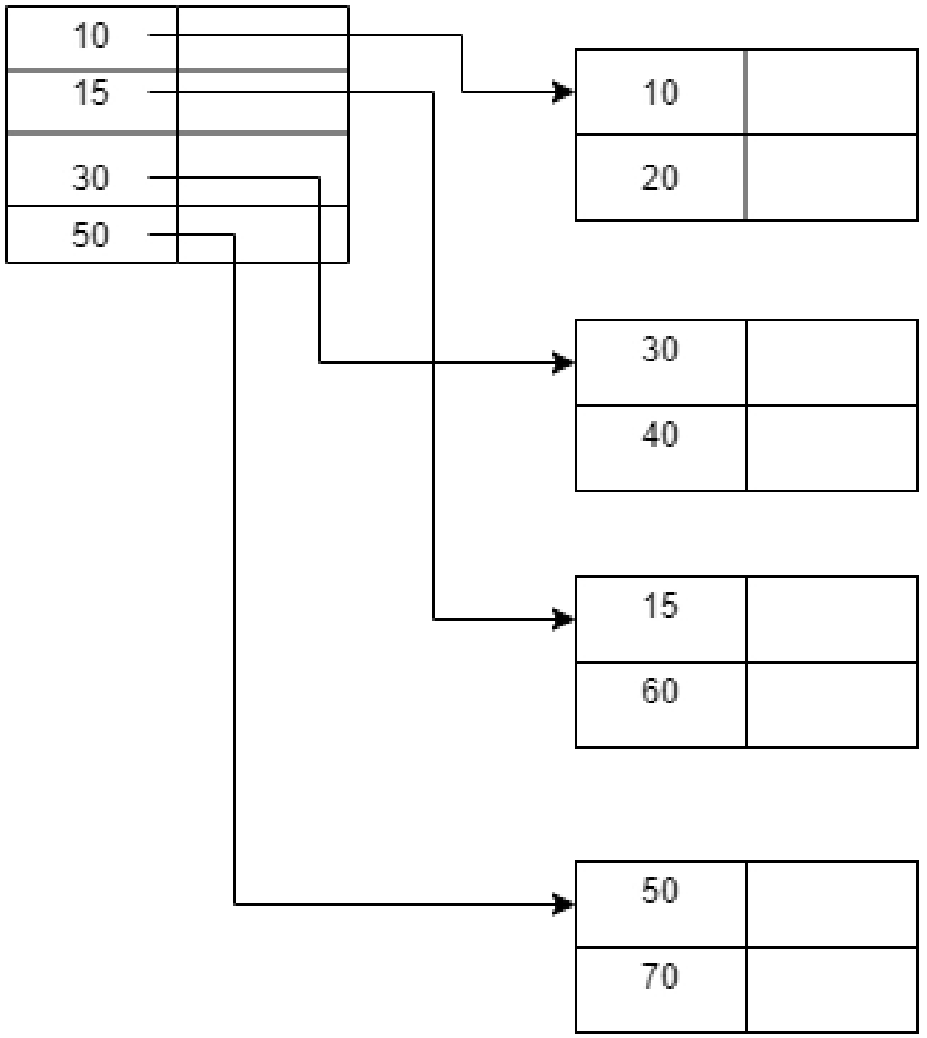
7.4 Indexing

Sparse index

- Only a few of the data file's items-each of which points to a block-have an index record.
- Instead of pointing to every record in the main database, the index in sparse indexing only points to the entries in the gap.

7.4 Indexing

Sparse index



7.4 Indexing

Clustering index

- A clustered index can be defined as an **ordered data** file. Sometimes the index is created **on non-primary key columns** which may not be unique for each record.
- In this case, to identify the record faster, we will **group two or more columns** to get the unique value and create index out of them. This method is **called a clustering index.**
- The records which have similar characteristics are grouped, and indexes are created for these group.

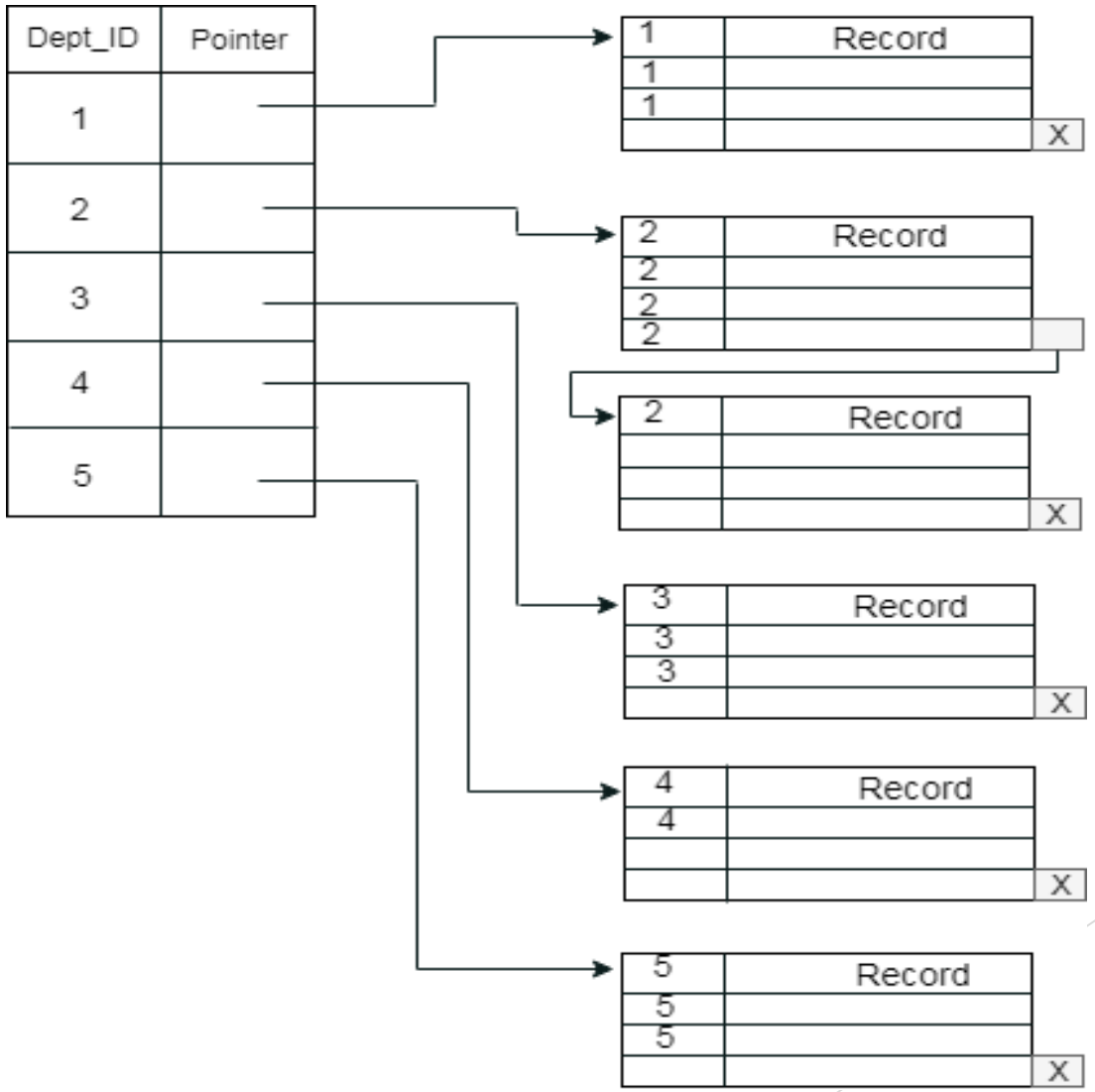
7.4 Indexing

Clustering index

- **Example:** suppose a company contains several employees in each department. Suppose we use a clustering index, where all employees which belong to the same Dept_ID are considered within a single cluster, and index pointers point to the cluster as a whole. Here Dept_Id is a non-unique key.

7.4 Indexing

Clustering index



7.4 Indexing

Secondary index

- In the sparse indexing, as the **size of the table grows, the size of mapping also grows.**
- These mappings are usually kept in the primary memory so that address fetch should be faster.
- Then the secondary memory searches the actual data based on the address got from mapping.
- If the **mapping size grows then fetching the address itself becomes slower**.
- In this case, the sparse index will not be efficient. To overcome this problem, **secondary indexing is introduced.**

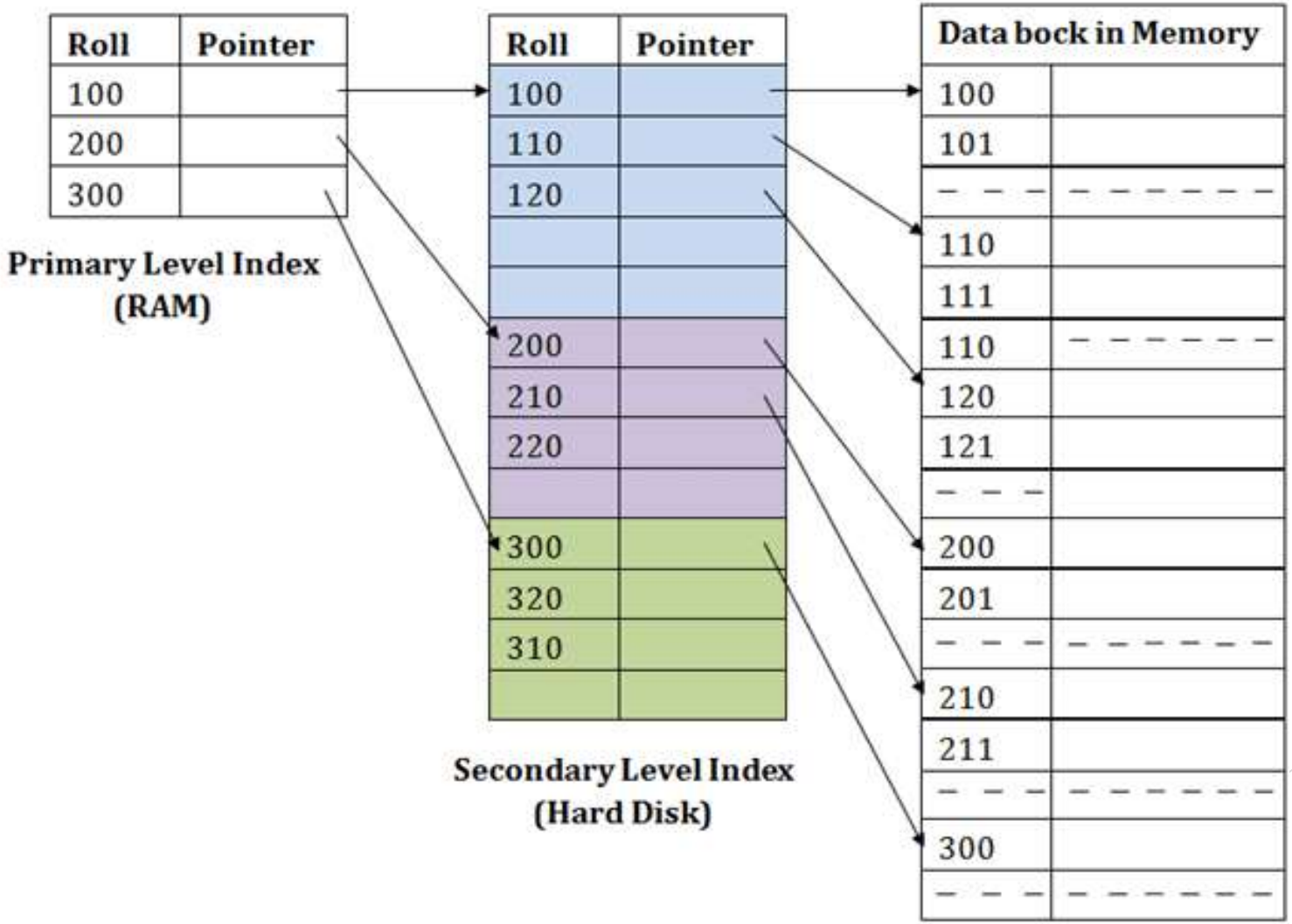
7.4 Indexing

Secondary index

- In secondary indexing, to reduce the size of mapping, another level of indexing is introduced.
- In this method, the huge range for the columns is selected initially so that the mapping size of the first level becomes small.
- Then each range is further divided into smaller ranges.
- The mapping of the first level is stored in the primary memory, so that address fetch is faster.
- The mapping of the second level and actual data are stored in the secondary memory (hard disk).

7.4 Indexing

Secondary index



7.4 Indexing

Secondary index

For example:

- If you want to find the record of roll 111 in the diagram, then it will search the highest entry which is smaller than or equal to 111 in the first level index. It will get 100 at this level.
- Then in the second index level, again it does max (111) <= 111 and gets 110. Now using the address 110, it goes to the data block and starts searching each record till it gets 111.
- This is how a search is performed in this method. Inserting, updating or deleting is also done in the same manner.

7.4 Indexing

Multilevel index

- In this method, we can see that index mapping growth is reduced to considerable amount.
- But this method can also have same problem as the table size increases.
- In order to overcome this, we can introduce multiple levels between primary memory and secondary memory.
- In this method number of secondary level index is two or more.

7.4 Indexing

B+ index

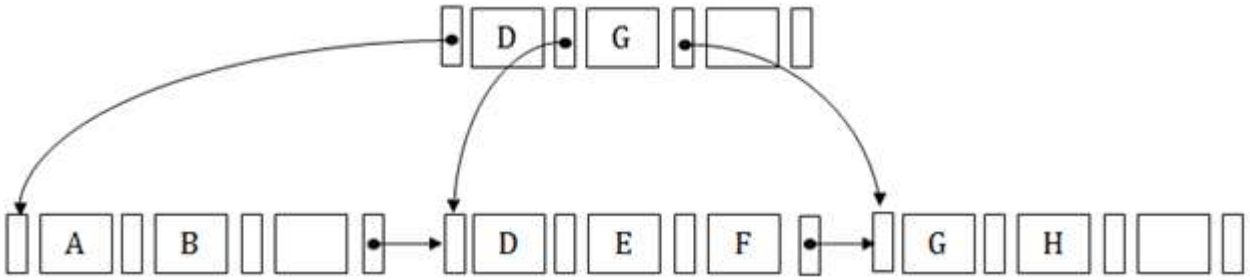
- The B+ tree is a balanced binary search tree. It follows a multi-level index format.
- In the B+ tree, leaf nodes denote actual data pointers. B+ tree ensures that all leaf nodes remain at the same height.
- In the B+ tree, the leaf nodes are linked using a link list. Therefore, a B+ tree can support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

7.4 Indexing

B+ index

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.



7.4 Indexing

B+ index

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 and n key values.
- Every leaf node of the B+ tree contains one block pointer P to point to next leaf node.

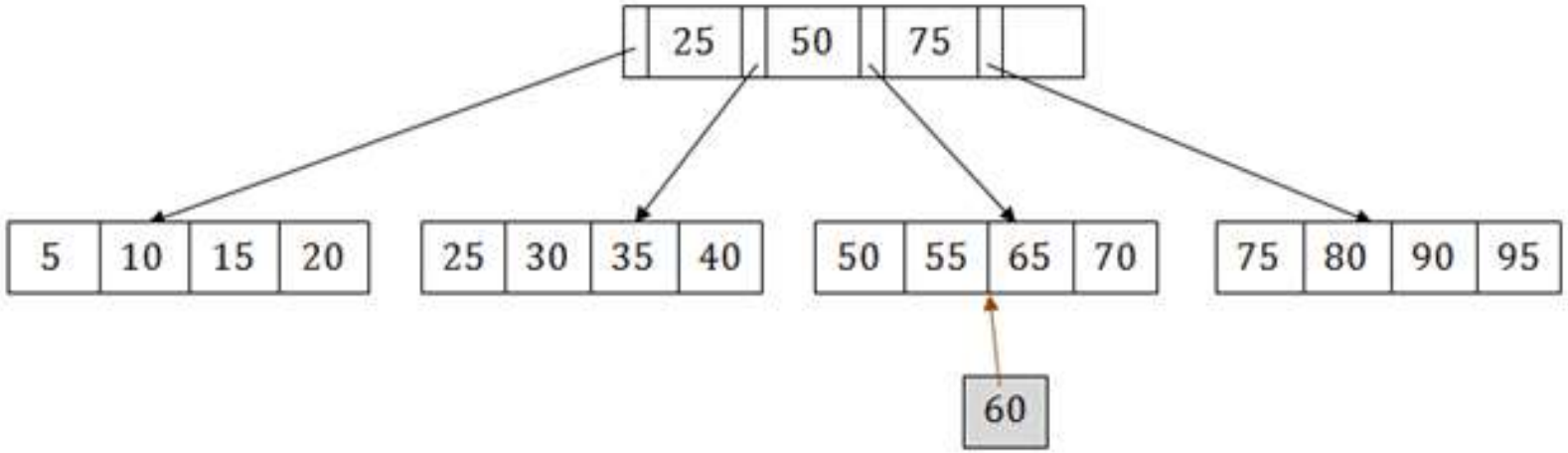
7.4 Indexing

B+ index 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.

7.4 Indexing

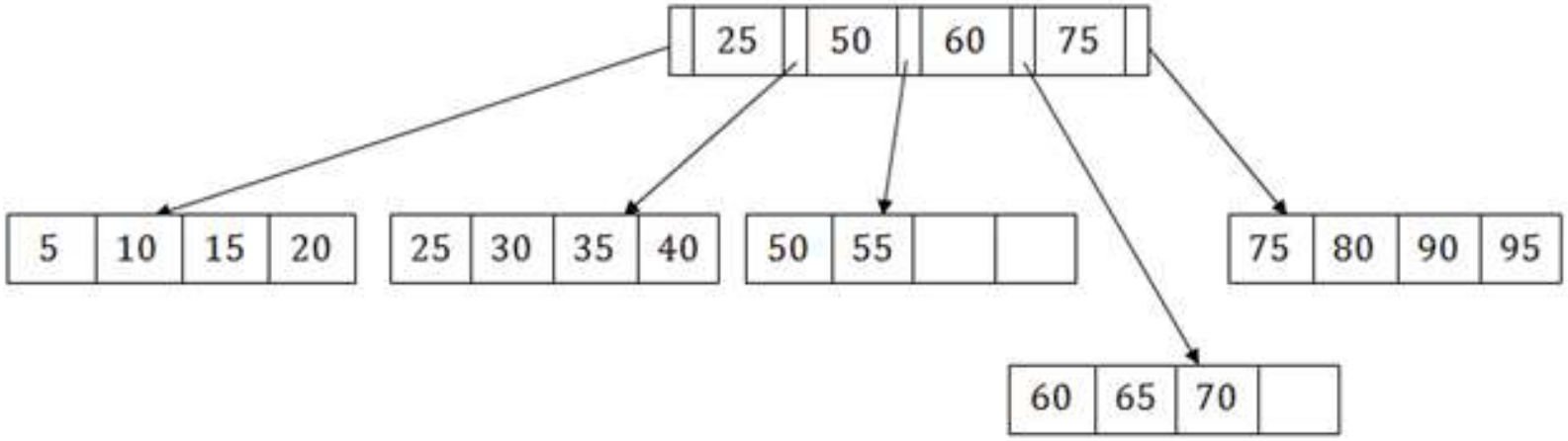
B+ index Insertion



- 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.

7.4 Indexing

B+ index Insertion



- 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.

7.4 Indexing

Hash index

- Hashing technique is used to calculate the direct location of a data record on the disk without using index structure.
- In this technique, data is stored at the data blocks whose address is generated by using the hashing function.
- The memory location where these records are stored is known as **data bucket or data blocks.**

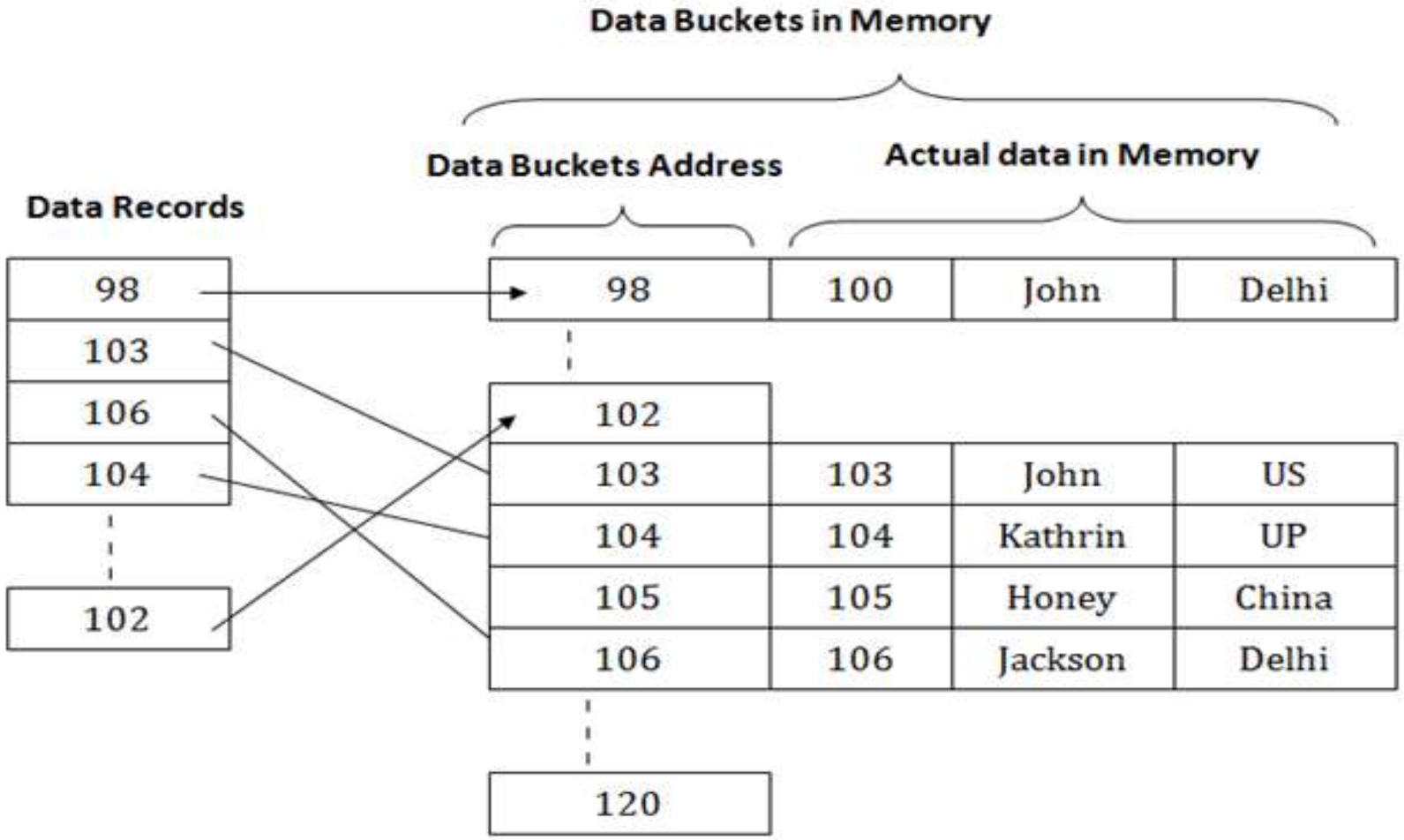
7.4 Indexing

Hash index

- In this, a hash function can choose any of the column value to generate the address.
- Most of the time, the hash function uses the primary key to generate the address of the data block.
- A hash function is a simple mathematical function to any complex mathematical function.
- We can even consider the primary key itself as the address of the data block. That means each row whose address will be the same as a primary key stored in the data block.

7.4 Indexing

Hash index



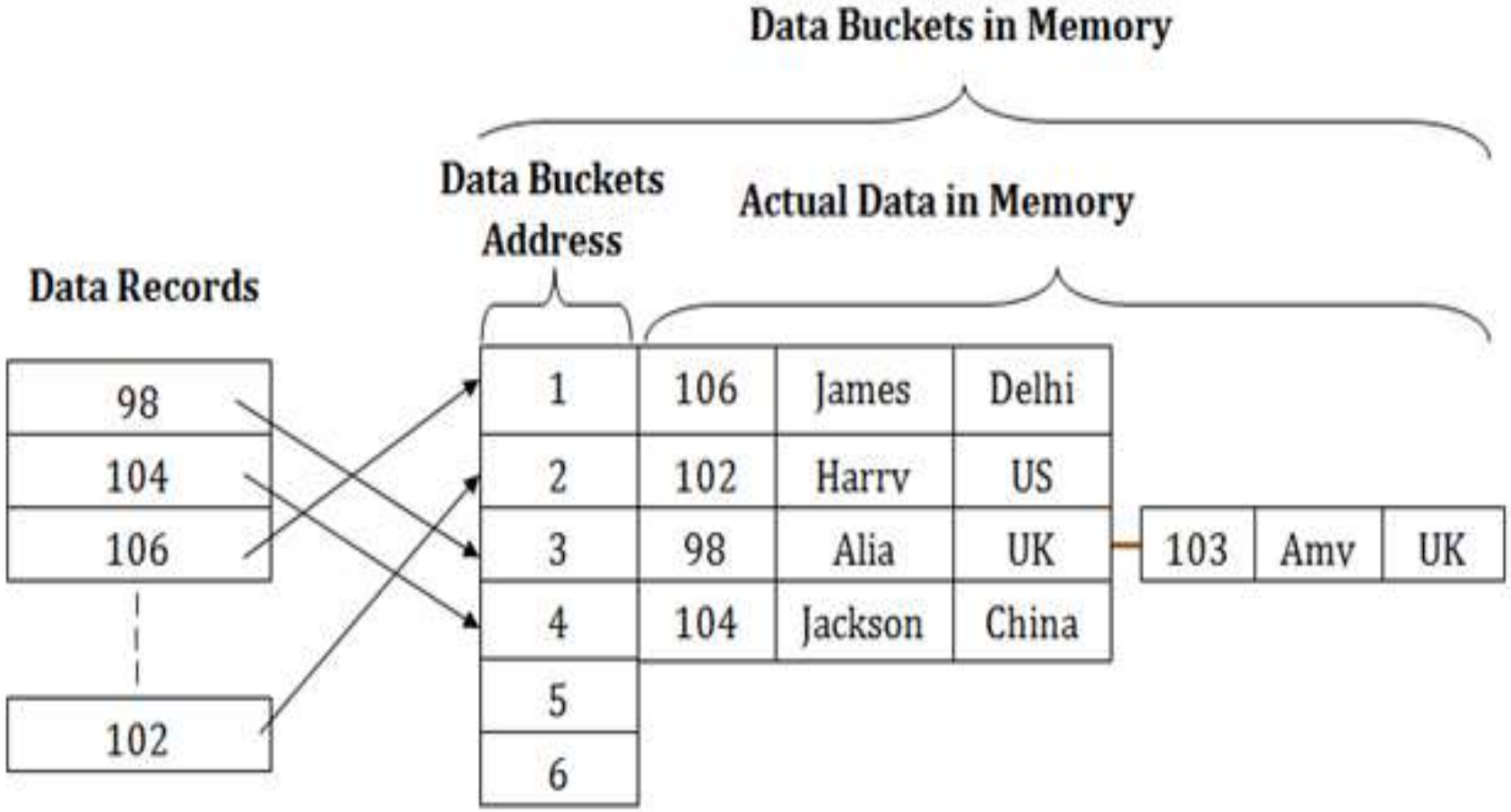
7.4 Indexing

Hash index

- The above diagram shows data block addresses same as primary key value.
- This hash function can also be a simple mathematical function like exponential, mod, cos, sin, etc.
- Suppose we have mod (5) hash function to determine the address of the data block. I
- n this case, it applies mod (5) hash function on the primary keys and generates 3, 3, 1, 4 and 2 respectively, and records are stored in those data block addresses.

7.4 Indexing

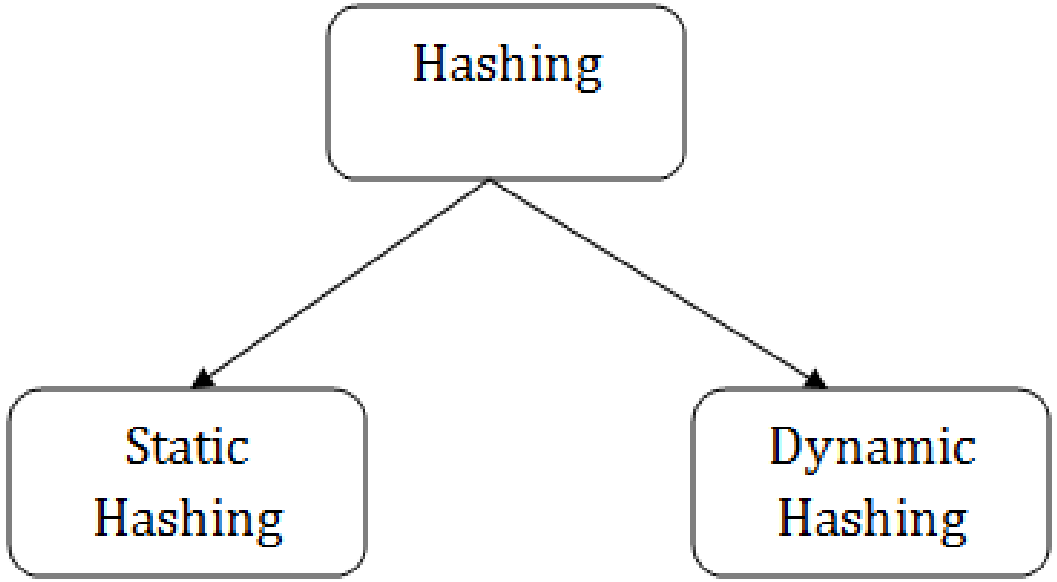
Hash index



7.4 Indexing

Hash index

Types of Hashing:



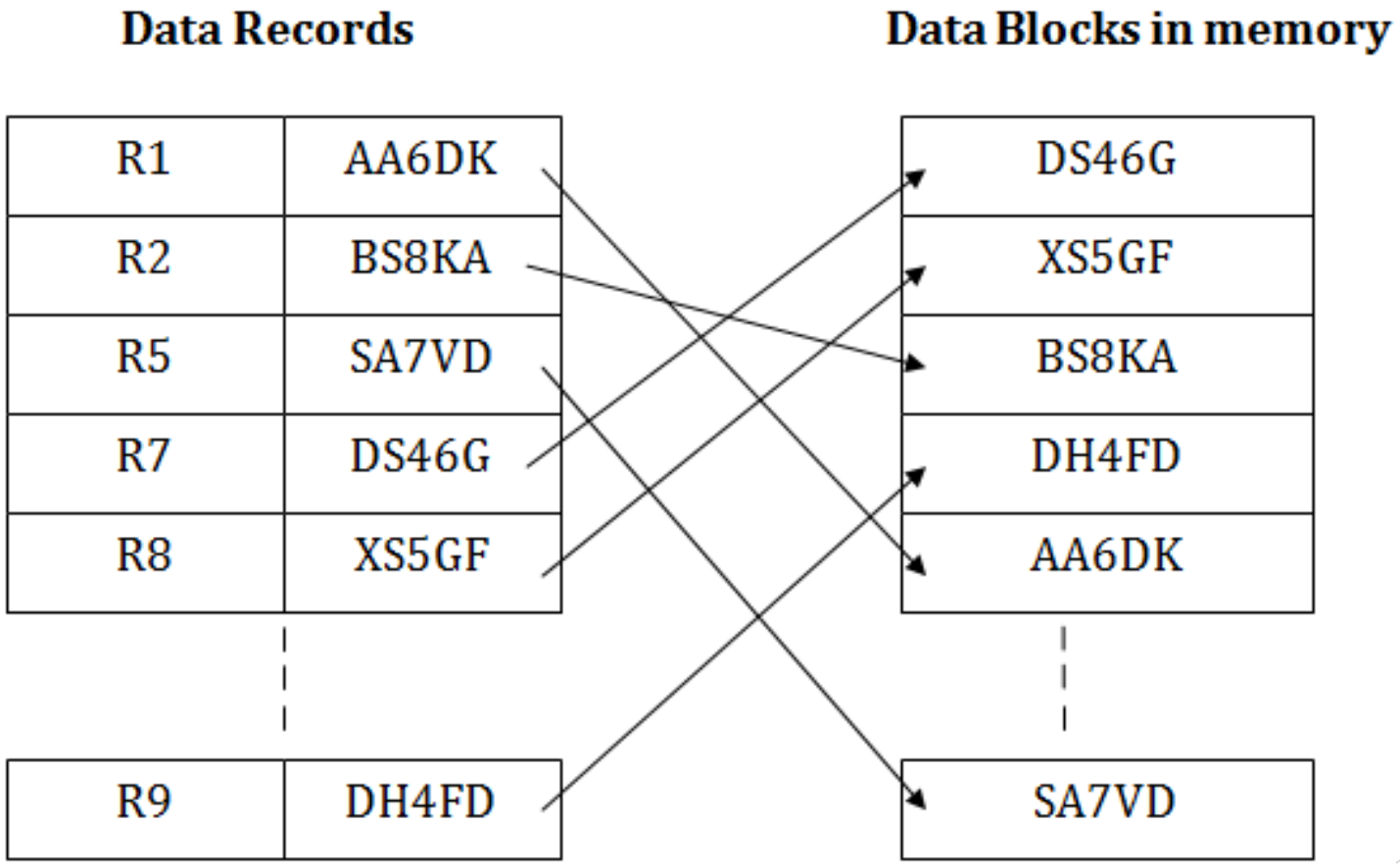
7.4 Indexing

Indexed sequential access method (ISAM)

- 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. This index contains the address of the record in the file.
- 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.

7.4 Indexing

Indexed sequential access method (ISAM)



7.4 Indexing

Indexed sequential access method (ISAM)

Pros of ISAM:

- In this method, each record has the address of its data block, searching a record in a huge database is quick and easy.
- This method supports range retrieval and partial retrieval of records. Since the index is based on the primary key values, we can retrieve the data for the given range of value. In the same way, the partial value can also be easily searched, i.e., the student name starting with 'JA' can be easily searched.

7.4 Indexing

Indexed sequential access method (ISAM)

Cons of ISAM:

- This method requires extra space in the disk to store the index value.
- When the new records are inserted, then these files have to be reconstructed to maintain the sequence.
- When the record is deleted, then the space used by it needs to be released. Otherwise, the performance of the database will slow down.

Assignment

- Types of Hashing
- Advantages and Disadvantages Indexing
- Non clustering Index

This is the end of the lecture!
I hope you enjoyed it.
Thank You