File Organisation in DBMS

Unit 4

Red: indicate important

Physical Database Design Issues

- Database design involves two processes :

1. Logical design : This is done with the help of E-R diagram, normalization, etc...
2. Physical design : This is done after the logical design. Physical database design translates the logical description of data into the technical specifications for storing and retrieving data for the DBMS.

- The key issues in the physical database design are :

- The purpose of physical database design is to translate the logical description of data into the technical specifications for storing and retrieving data for the DBMS.
- Its main goal is to create a design for storing data that will provide adequate performance and ensure database integrity, security and recoverability.

- Some of the basic inputs required for Physical Database Design are :
- Normalised relations
- Attribute definitions
- Data usage: retrieved, entered, deleted, updated
- Requirements for security, backup, recovery, retention, integrity
- DBMS characteristics
- Performance criteria such as response time requirement with respect to volume estimates.

- Physical Database Design Decisions that are to be taken are :
- Optimising attribute data types.
- Modifying the logical design.
- Specifying the file organization
- Choosing indexes.

- Designing the fields in the data base :
- Choosing data type
- Coding, compression, encryption
- Controlling data integrity
- Default value
  - ❑ Range control
  - ❑ Null value control
  - ❑ Referential integrity
- Handling missing data
  - ❑ Substitute an estimate of the missing value
  - ❑ Trigger a report listing missing values

- In programs, ignore missing data unless the value is significant.

- Physical Records
- Records that are stored in the secondary storage devices.
- For a database relation, physical records are the group of fields stored in adjacent memory locations and retrieved together as a unit.
- Data page is the amount of data read or written in one I/O operation to and from secondary storage device to the memory and vice-versa.
- Blocking factor is defined as the number of physical records per page.

- Issues relating to the design of the physical database files :
- Physical file is a file stored on the disk.
- Main issues relating to physical files are :
  - Constructs to link two pieces of data :
    - Sequential storage
    - Pointers
  - File organization : How the files are arranged on the disk ?
  - Access method : How the data can be retrieved based on the file organization?

Storage Of Database On Hard Disks

- A file organization refers to the organization of the data of a file into records, blocks, and access structures. This includes the way records and blocks are placed on the storage medium and interlinked.
- Access method refers to the way how the data can be retrieved based on the file organization.
- The databases are stored persistently on the magnetic disks for the following reasons :
- Databases are very large; may not fit completely in the main memory.
- Storing the data permanently using the non-volatile storage and provide access to the users with the help of front end applications.
- Primary storage is considered to be very expensive and in order to cut short the cost of the storage per unit of data to substantially less.

- Each hard drive is composed of a set of disk platters. Each disk platter has a layer of magnetic material deposited on its surface. The entire disk can contain large amount od data, which is organized into small packages called BLOCKS or pages.

  One block = 1KB of data =1024 Bytes

- A block is the smallest unit of data transfer between the hard disk and processor of the computer. Each block has a fixed, assigned address. The computer processor will submit a read/write request, which will contain the address of the block and address of the Ram in the computer memory area called buffer or cache where the data must be stored / taken from. The processor then reads and modifies the buffer data as required, and, if required writes the block back to the disk.

How are tables stored on disk ?

- Each record of a table can contain different amounts of data. This is because in some records, some attribute values may be 'null', or some attributes may be of type varchar(), and therefore may have a different length string as the value of the attribute.

- Therefore, the record is stored with each subsequent attribute separated by a special ASCII character called a field separator. Each block may contain many records. Each record is separated by another special ASCII character called the record separator.

File Organization And Its Types

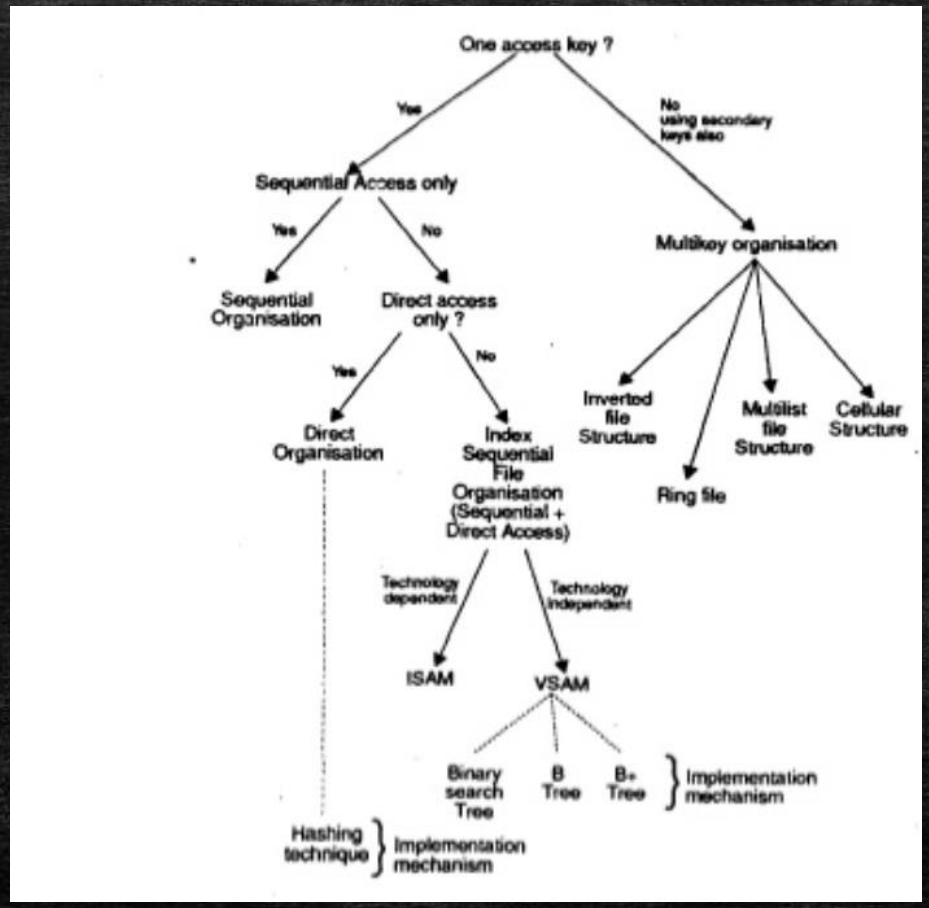
- It's a technique to organize data in the secondary memory.
- It's a way of arranging the records in a file when the file is stored on the disk.
- 2 reasons to organize files :

1. To facilitate access to records
2. To ensure efficient storage

- Selection of file organization depends on two factors :

1. Typical DBMS applications need a small subset of the DB at any gien time.
2. When a portion of the data is needed it must be located on disk, copied to memory for processing and rewritten to disk if the data was modified.

- A file of record can be accessed and modified in different ways, and different ways of arranging the records enable different operations over the file to be carried out efficiently.
- A DBMS supports several file organization techniques.
- DBA is responsible to choose a good organization for each file depending on its use.
- Particular organization suitable for any application will depend on the following factors :
- Kind of external storage
- Types of queries allowed
- Number of keys
- Mode f retrieval
- Mode of update



- Some of the file organization techniques :

1. Heap files (unordered file)
2. Sequential file organization
3. Indexed file organization
4. Hashed file organization

Sequential File Organisation

- It's the most basic way to organize collection of records in a file .
- Here each file/records are stored one after the other in a sequential manner. Its stored by the primary key field values. They are accessible only in the order they are stored. Here each file / record is stored in a sorted manner.
- This kind of file organization works well for tasks that need to access nearly every record in a file e.g. payroll.
- In a sequentially organized file records are written consecutively when the file is created and must be accessed consecutively when the file is later used for input.
- Sequential file maintains the records in the logical sequence of its primary key values.
- Sequential files are inefficient for random access, however is suitable for sequential access.
- A sequential file can be stored on devices like magnetic tape that allow sequential access.

- On an average, to search a record in a sequential file would require to look into half of the records of the file.
- It allows binary search which enhances the speed of access.
- Updating a sequential file usually creates a new file to maintain the record sequence on primary key is maintained. The update operation first copies the records till the record after which update is required into the new file and then the updated record is put followed by the remainder of records. Thus method of updating a sequential file automatically creates a backup copy.
- Additions in the sequential files is similar to updating. Adding a record requires shifting of all records from the point of insertion to the end of file to create space for the new record.
- Deletion in sequential files requires compression of the file space.

- Main advantage of sequential file is sequential processing because the next record is easily accessible without any data structure.
- A single update is expensive as new file must be created. To reduce the cost per update, all updates requests are sorted in the order of the sequential file. This updated file is then used to update the sequential file in a single go. The file containing the updates is called transaction file.
- This process is called batch mode of updating. Before updating begins, the **old master file** must be in sequential order according to the **key** field. The **transaction file** should then be sorted on the **same key** as the **master file**. The computer compares the key of the first master file record with the key from the first record in the transaction file. If the keys do not match, the computer writes the record to the **new master file** as is, and then it reads the next record on the **master file**. When a match between the master and transaction records occurs, the computer updates the **new master file** record. Sometimes if a transaction record has no matching master record, the computer may generate an error message. Some transactions may add a new record, some may modify a record, while others may delete an existing record.A record that is to be deleted is not written to a new master file. Thus a new updated master file will be created from the transaction file and old master file.

- Update, insertion and deletion of records in a sequential file require a new file creation. We can reduce the creation of this new file if the original sequential file is created with holes which are empty record spaces as shown below.
- Thus, a reorganization can be restricted to only a block that can be done very easily within the main memory.
- Thus, holes increase the performance of sequential file insertion and deletion.
- This organization also supports a concept of overflow area, which can store the spilled over records if a block is full. This technique is also used in index sequential file organization.

A file with empty spaces for record insertion :

- Advantages of Sequential file organisation :
- It is fast and efficient when dealing with large volumes of data that need to be processed periodically (batch system).
- Disadvantages of Sequential file organisation :
- Requires that all new transactions be sorted into the proper sequence for sequential access processing.
- Locating, storing, modifying, deleting, or adding records in the file require rearranging the file.
- This method is too slow to handle applications requiring immediate updating or responses.

Indexed File Organisation

- Also known as indexed sequential file organization.
- It organizes the file like a large dictionary.
- Here records are stored sequentially in order of primary key in the file. For each primary key, an index is kept. The index permits a type of direct access.
- In this method, if any record has to be retrieved, based on its index value, the data block address is fetched and the record is retrieved from memory.
- Its an inefficient and time consuming technique. This is because the retrieval of a record from sequential file, on average, requires access to half the records in the file.
- To improve the query response time of a sequential file an indexing technique is used. An index is a set of index value, address pairs. Indexing associates a set of objects to a set of orderable quantities. Thus an index is a mechanism for faster search.

- Indices and data blocks are kept together physically but they are logically distinct.
- A sequential file that is indexed on its primary key is called an index sequential file. The index allows for random access to records, while the sequential storage of the records of the file provides easy access to the sequential records.
- It has a feature over flow area. The overflow area provides additional space for record addition without the need to create.

Hashed File Organisation

- Hashing is the most common form of random access to a file or database.
- Its also used to access columns that do not have an index as an optimization technique.
- Hash functions calculate the address of the page in which the record is to be stored based on one or more fields in the record. The hash function can be any simple or complex mathematical function. Records are randomly distributed in the available space in a hash file . This method is also known as Direct or Random file organization.
- It requires some hashing algorithms and hashing techniques. Hashing algorithms convert primary key into record address. Most popular hashing techniques is division hashing with chained overflow.
- It is suitable for online transaction systems like online banking, ticket booking system etc.

- Advantages :

- Insertion or search on hash-key is fast.
- Its best if equality search is needed on hash-key
- Records need not be sorted after any of the transaction. Hence the effort of sorting is reduced in this method.
- Since block address is known by hash function, accessing any record is very faster. Similarly updating or deleting a record is also very quick.

Disadvantages :

- It's a complex file organization technique.
- Search is slow.
- It suffers from disk space overhead. Disk overhead refers to the disk space required for information that is not data but is used for location and timing.
- Unbalanced buckets degrade performance.
- If we are searching for range of data, then this method is not suitable. Because, each record will be stored at random address.
- Since all the records are randomly stored, they are scattered in the memory. Hence memory is not efficiently used.
- Hardware and software required for the memory management are costlier in this case. Complex programs needs to be written to make this method efficient.

Types of Indexes

- An index or database index is a data structure which is used to quickly locate and access the data in a database table.
- Index in databases is defined as the <index value, address> pair.
- Databases usually contain very large amount of records. It will be difficult to locate an attribute value in a large database, if index on that value is not provided. In such a case the value is to be searched record-by-record in the entire database. This is time-consuming and cumbersome. For a large database the entire records cannot be kept in the main memory at a time, thus data needs to be transferred from the secondary storage device which is time consuming. Thus, without any index it may be difficult to search a database.
- An index contains a pair consisting of index value and a list of pointers to disk block for the records that have the same index value. An index contains such information for every stored value of index attribute. An index file is very small compared to the data file that stores a relation.

- Also index entries are ordered, so that an index can be searched using an efficient search method like binary search. In case an index file is very large, we can create a multi-level index, i.e. index an index.
- Different types of indexes those are categorised as :
- Primary index
- Secondary index
- Clustering index
- Single level index
- Multi-level index
- Sparse index
- Dense index

- Primary index :
- Its defined on the attributes in the order of which the file is stored.
- This field is called the ordering field.
- A primary index can be on the primary key of a file.
- Clustering index :
- If an index is on the attributes other than candidate key fields then several records may be related to one ordering field value. This is called clustering index.
- There can be only one physical ordering field. Thus, a field can have either the primary index or clustering index, not both.

- Secondary indexes :
- Secondary indexes are defined on the non-ordering fields.
- Thus, there can be several secondary indexes in a file, but only one primary or clustering index.

Primary index :

- A primary index is a file that contains a sorted sequence of records.
- It has two columns: the ordering key field; and a block address for that key field in the data file.
- The ordering key field for this index can be the primary key of the data file.
- Primary index contains one index entry for each value of the ordering key field.

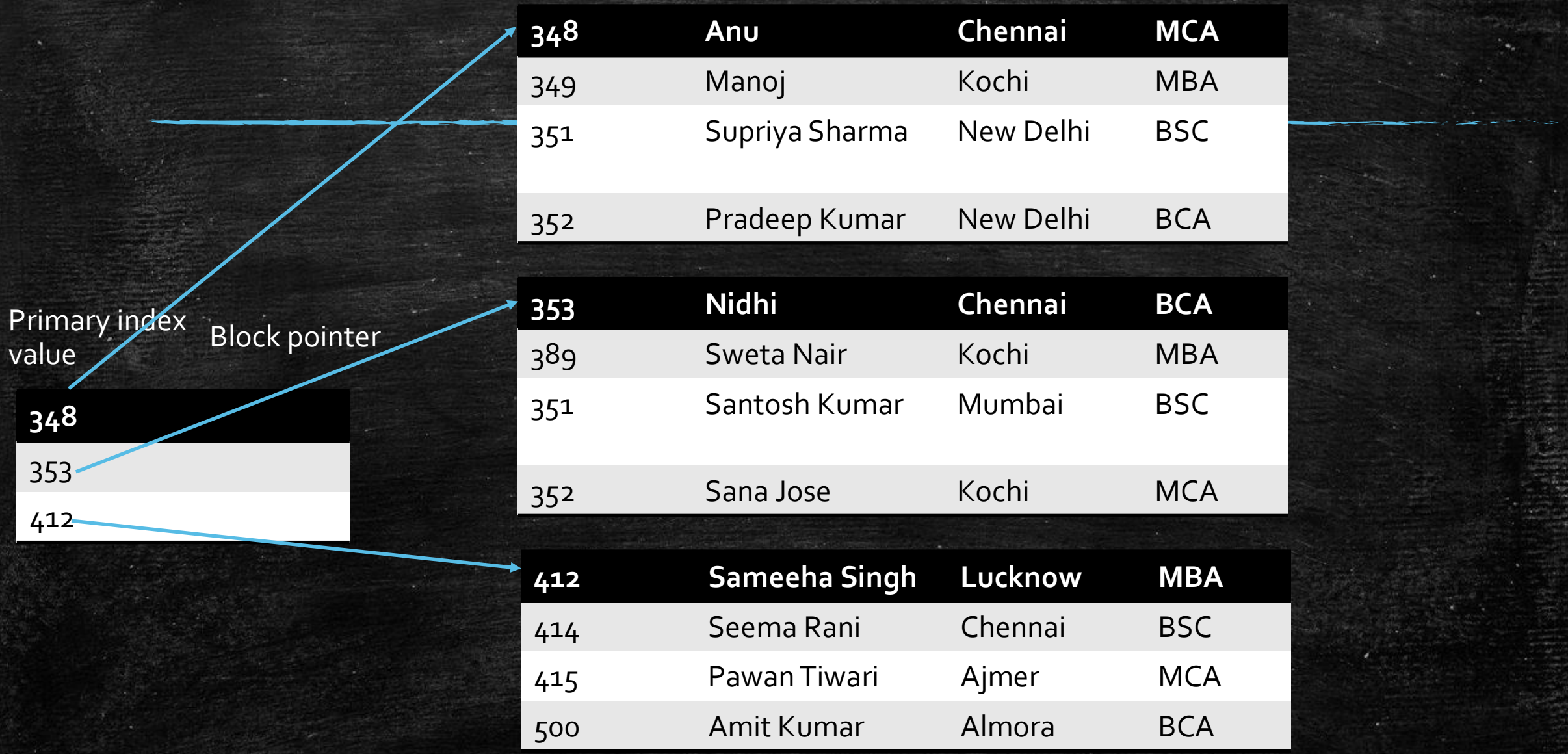
- An entry in primary index file contains the index value of the first record of the data block and a pointer to that data block.
- Example :

Consider a student file stored in the order of enrolment number.

The primary index on this file would be on the ordering field – enrolment number.

Figure 1 :

	Enrolment Number	Name	City	Programme
Block 1	348	Anu	Chennai	MCA
	349	Manoj	Kochi	MBA
	351	Supriya Sharma	New Delhi	BSC
	352	Pradeep Kumar	New Delhi	BCA
Block 2	353	Nidhi	Chennai	BCA
	389	Sweta Nair	Kochi	MBA
	390	Santosh Kumar	Mumbai	BSC
	411	Sana Jose	Kochi	MCA
Block 3	412	Sameeha Singh	Lucknow	MBA
	414	Seema Rani	Chennai	BSC
	415	Pawan Tiwari	Ajmer	MCA
	500	Amit Kumar	Almora	BCA



- Please note the following points in the above slide :
- An index entry is defined as the attribute value, pointer to the block where that record is stored. The pointer physically is represented as the binary address of the block.
- We have used the first key value stored in the block as the index key value. This is also called the anchor value . All the records stored in the given block have ordering attribute value as same as the same or more than this anchor value.
- The number of entries in the primary index file is the same as the number of disk block in the ordered data file. Therefore, the size of the index file is small. All the records need to have an entry in the index file. This type of index is called non-dense index. Thus, the primary index is non-dense index.
- To locate the record of a student whose enrolment number is 390, we need to find two consecutive entries of indexes such that value1 < 390 < index value 2. For example, in the above figure second and third index values are : 353 and 412 . Thus, the required student is in block 2.

- Primary index enhances efficiency of searching. Let us explain this with the help of an example.
- Example : An ordered student file (ordering field is enrolment number) has 20,000 records stored on a disk having the block size as 1K. Assume that each student record is of 100 bytes, the ordering field is of 8 bytes, and block pointer is also of about 8 bytes, find how many block accesses on average may be saved on using primary index.

Answer:

Number of accesses without primary index :

Number of records in the file = 20000 Block size = 1024 bytes Record size = 100 bytes

Number of records per block = integer value of [1024/100]= 10

Number of disk blocks acquired by the file = [Number of records/ records per block] = 20000/10 = 2000

Assuming a block level binary search, it would require $\log_2 2000$ = about 11 block accesses

Number of accesses with primary index :

Size of an index entry = 8+8= 16 bytes

Number of index entries that can be stored per block = integer value of [1024/16] = 64

Number of index entries = number od disk blocks = 2000

Number of index blocks = ceiling of [2000/64] = 32

Number of index block transfers to find the value in index blocks = $\log_2 32 = 5$

One block transfer will be required to get the data records using the index pointer after the required index value has been located. So total number of block transfers with primary index = 5+1 = 6

Thus, the primary index would save about 5 block transfers for the given case.

- Disadvantage of using primary index :

A primary index requires the data file to be ordered, this causes problems during insertion and deletion of records in the file.

This problem can be solved by selecting a suitable file organization that allows logical ordering only.

Clustering index :

- An index that is created on an ordered file whose records of a file are physically ordered on a non-key field (i.e. the field does not have a distinct value for each record) is called a clustering index.
- Figure 2 and 3 shows the clustering indexes in the same file organised in different ways.

Figure 2: A clustering index on Programme in the Student file

Programme	Index block pointer
BCA	.
BSC	.
MBA	.
MCA	.

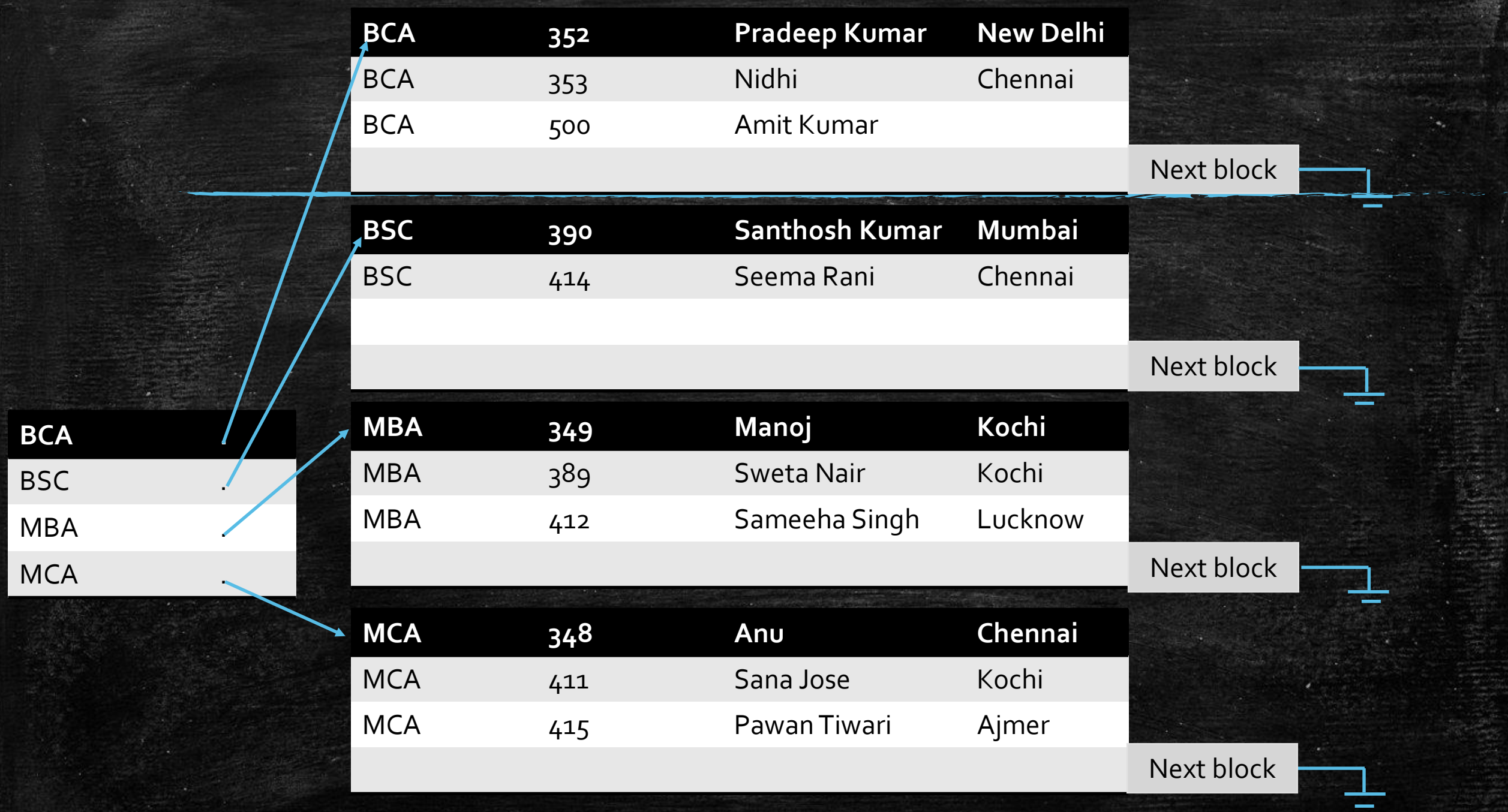
Programme	Enrolment Number	Name	City
BCA	**352**	**Pradeep Kumar**	**New Delhi**
BCA	353	Nidhi	Chennai
BCA	500	Amit Kumar	Almora
BSC	351	Supriya Sharma	New Delhi
BSC	**390**	**Santhosh Kumar**	**Mumbai**
BSC	414	Seema Rani	Chennai
MBA	349	Manoj	Kochi
MBA	389	Sweta Nair	Kochi
MBA	**412**	**Sameeha Singh**	**Lucknow**
MCA	348	Anu	Chennai
MCA	411	Sana Jose	Kochi
MCA	415	Pawan Tiwari	Ajmer

- Please note the following points about the clustering index as shown in figure 2.

- The clustering index is an ordered file having the clustering index value and a block pointer to the first block where that clustering field value first appears.

- Clustering index is also a sparse index. The size of clustering index is smaller than primary index as far as number of entries is concerned.

In the above figure the data file have blocks where multiple Programme students exist. We can improve upon this organization by allowing only one programme data in one block. Such an organization and its clustering is shown in the following figure 3 (next slide) :



- Please note the following points in the tables :
- Data insertion and deletion is easier than in the earlier clustering files, even now it is cumbersome.
- The additional blocks allocated for an index entry are in the form of linked list blocks.
- Clustering index is another example of a non-dense index as it has one entry for every distinct value of the clustering index field and not for every record in the file.

Secondary indexes :

- A secondary index is a file that contains records containing a secondary index field value which is not the ordering field of the data file, and a pointer to the block that contains the data record.
- A file can have many secondary indices.
- This is used when the database is to be searched or accessed in the alphabetical order of names. This search is very time consuming. This search on an average would require reading of half of the total number of blocks. So we need secondary indices in database system.
- Secondary index can be defined on an alternate key or non-key attributes.
- A secondary index that is defined on the alternate key will be dense while secondary index on non-key attributes would require a bucket of pointers for one index entry.

A dense secondary index on a non-ordering key field of a file

Name	Block pointer
Abhishek K	.
Abhishek R	.
Amit	.
Binoy	.
:	
:	
:	
:	
Supriya Swami	.
Tanuja Sethi	.
Vimal Kishore	.

Programme	Enrolment Number	Name	City
BCA	**351**	**Vimal Kishore**	**Kochi**
BCA	411	Amit	New Delhi
BCA	014	Daven Singhal	Mumbai
BCA	015	S. Srivastava	Mumbai
BCA	**616**	**Nidhi**	**Ajmer**
BSC	422	Kamlesh Kumar	Mumbai
BSC	414	Supriya Swami	New Delhi
BSC	017	Shwetha Singh	New Delhi
MBA	**216**	**Tanuja Sethi**	**Lucknow**
MCA	617	Jagmeet Singh	Lucknow
MCA	623	Arun Kumar	Chennai
MCA	666	Abhishek R	Mumbai

- Please note the following in the above figure :
- The names in the data file unique and thus are being assumed as the alternate key.
- The pointers are block pointers, thus are pointing to the beginning of the block and not a record.
- This type of secondary index file is dense index as it contains one entry for each record / distinct value.
- The secondary index is larger than primary index as we cannot use block anchor values here as the secondary index attributes are not the ordering attribute of the data file.
- To search a value in a data file using name, first the index file is (binary) searched to determine the block where the record having the desired key value can be found. Then this block is transferred to the main memory where the desired record is searched and accessed.

- A secondary index file is usually larger than that of primary index because of the large number of entries. However, secondary index improves the search time to a greater proportion than that of primary index. This is because of the fact that if primary index does not exist even then we can use binary search on the blocks as the records are ordered in the sequence of primary index value. However, if a secondary key does not exist then you may need to search the records sequentially. Proof example 2

Example 2 : An unordered student file has 20,000 records as stored on a disk having the block size as 1 K. Assume that each student record is of 100 bytes, the secondary index field is of 8 bytes, and block pointer is also of 8 bytes, find how many blocks accesses on average may be saved on using secondary index on enrolment number.

Answer :

Number of accesses without using secondary index :

Number of records in the file = 20000

Block size = 1024 bytes

Record size = 100 bytes

Number of records per block = integer value of [1024/100] = 10

Number of disk blocks acquired by the file = [Number of records / records per block] = 20000/10 =2000

Since the file is unordered any search on an average will require about half of the above blocks to be accessed. Thus, average number of block accesses = 1000

Number of accesses with secondary index :

Size of an index entry = 8+8 = 16 bytes

Number of index entries that can be stored per block= integer value of [1024/16] = 64

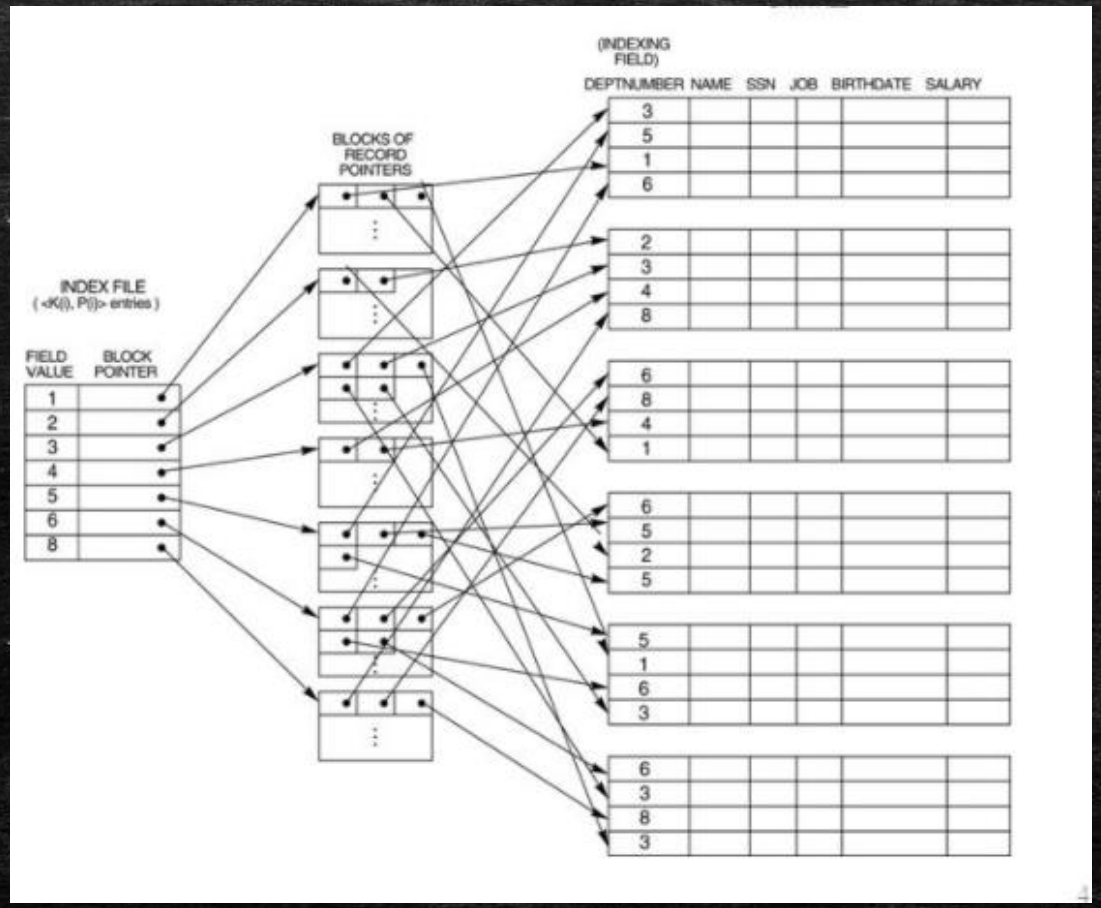
Number of index entries = number of records = 20000

Number of index blocks =ceiling of [20000/64] = 320

Number of index block transfers to find the value in index blocks = ceiling of $[\log_2 320] = 9$

One block transfer will be required to get the data records using the index pointer after the required index value has been located. So total number of block transfers with secondary index = 9 + 1 = 10.

Thus the secondary index would save about 1990 block transfers for the given case.



- A secondary index that needs to be created on a field that is not a candidate key can be implemented using several ways. In the above figure we have shown here the way in which a block of pointer records is kept for implementing index. This method keeps the index entries at a fixed length. It also allows only a single entry for each index field value. This method creates an extra level of indirection to handle the multiple pointers. The algorithms for searching the index, inserting and deleting values on an index are very simple. This is the most popular scheme for implementing such secondary indexes.

Sparse and Dense indexes :

- An index is defined as the ordered <index value, address> pair.
- These indexes are same as the indexes used at the back of the book.
- Key ideas of indexes are :

➢ They are sorted on the order of the index value (ascending or descending) as per the choice of the creator.

➢ The indexes are logically separate files.

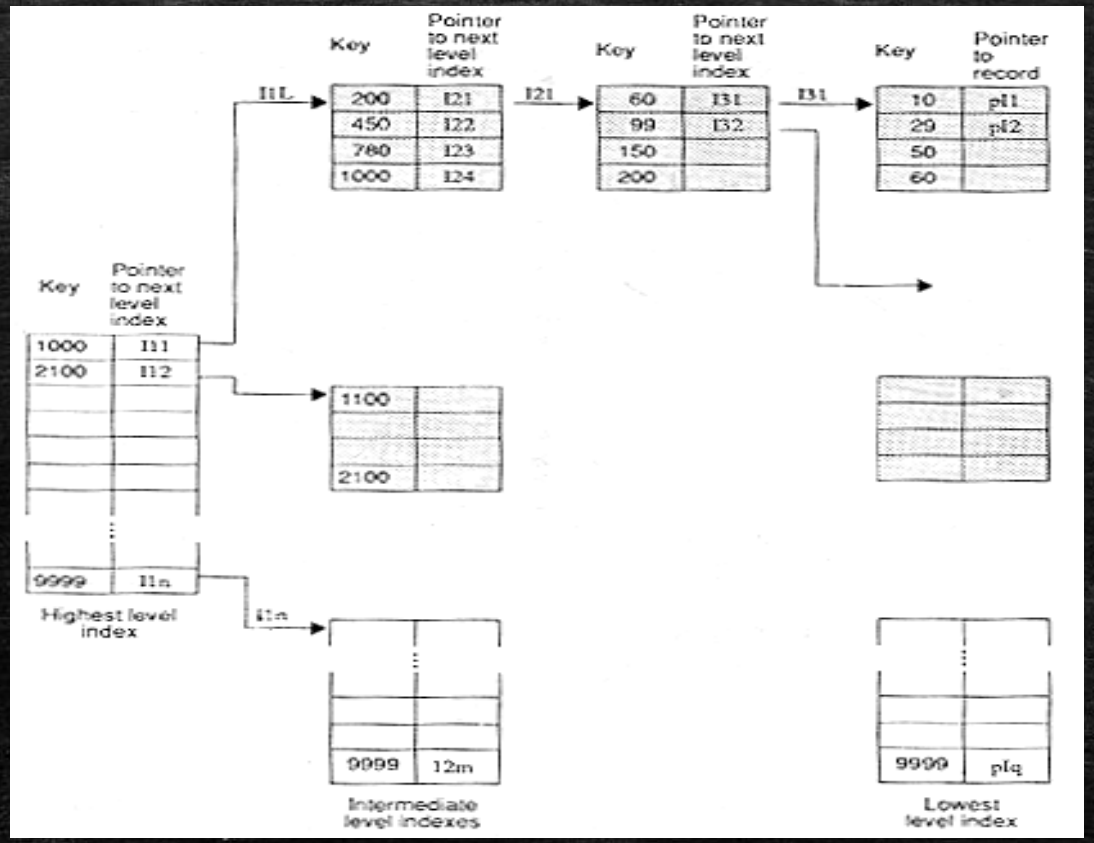
- An index is primarily created for fast access of information.
- The primary index is the index on the ordering field of the data file whereas a secondary index is the index on any other field, thus more useful.

What are parse index and dense index?

- Sparse indices are those indices that do not include all the available values of a field. An index groups the records as per the index values. A sparse index is the one where the size of the group is one or more, while in a dense index the size of the group is 1.
- In a dense index contains one index entry for every value of the indexing attributes, whereas a sparse index is also called non-dense index contains few index entries out of the available indexing attribute values. For example, the primary index on enrolment number is sparse, while secondary index on student name is dense.

Multilevel indexing Scheme

- Consider the indexing scheme where the address of the block is kept in the indes for each record, for a small file, this index would be small and can be processed efficiently in the main memory. However, for a large file the size of index can also be very large. In such a case, one can create a hierarchy of indexes with the lowest level index pointing to the records, while the higher level indexes point to the indexes on indexes. The following figure shows this scheme.



- The lowest level index points to each record in the file; thus is costly in terms of space.
- Updating, insertion and deletion of records require changes to the multiple index files as well as the data file. Thus, maintenance of the multi-level indexes is also expensive.

Implementation of indexes :

The indexes are implemented through B trees.

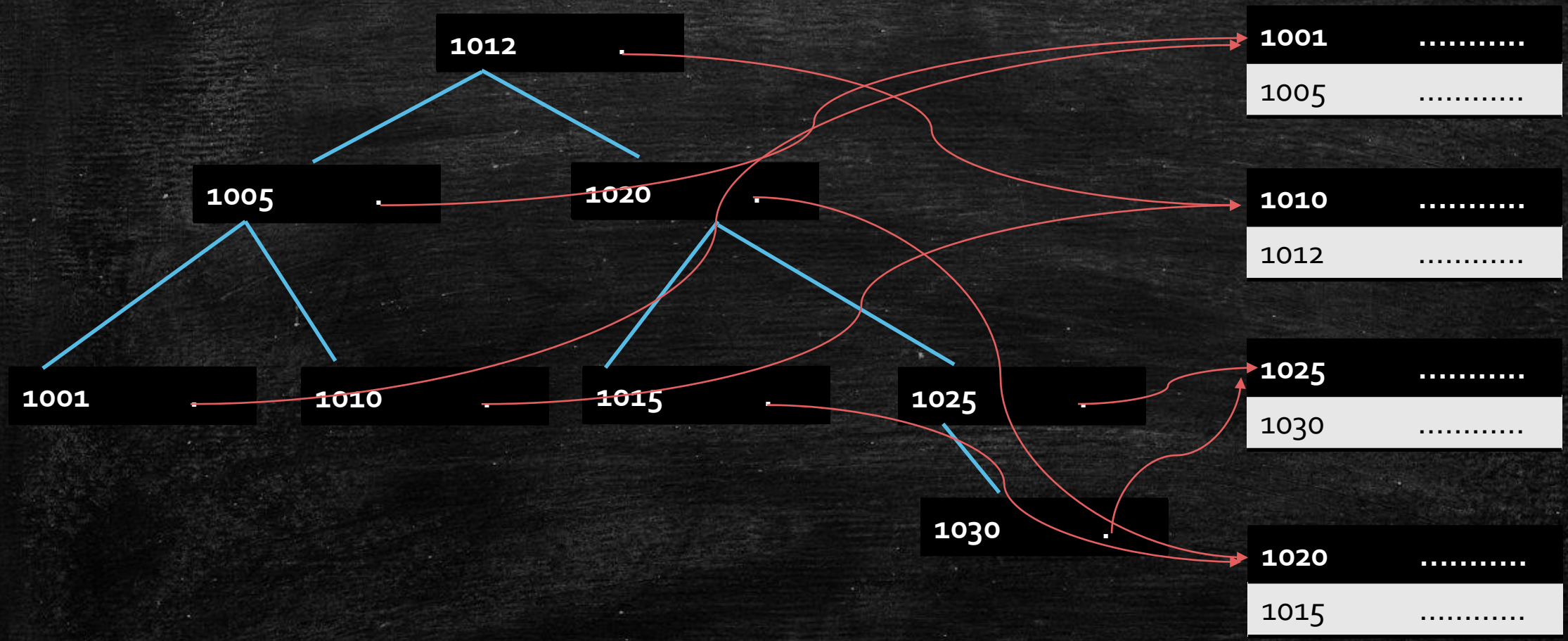
Index and Tree Structure

Can we use Binary Search Tree (BST) as indexes?

A binary search tree (BST) is a data structure that has a property that all the keys that are to the left of a node are smaller than the key value of the node and all the keys to the right are larger than the kay value of the node.

To search a typical key value, you start from the root and move towards left or right depending on the value of key that is being searched. Since an index is a <value, address> pair, thus while using BST, we need to use the value as the key and address field must also be specified in order to locate the records in the file that is stored on the secondary storage devices.

The index structure using binary search tree (BST)



- In the figure above a key value is associated with a pointer to a record. A record consists of the key value and other information fields. We do not store these information fields in the binary search tree, as it would make a very large tree. To speed up searches and to reduce the tree size, the information fields of records are commonly stored into files on secondary storage devices. BST structure is key value, address pair.
- A BST as a data structure is very much suitable for an index, if an index is said to be contained completely in the primary memory. However, indexes are quite large in nature and require a combination of primary and secondary storage. As far as BST is concerned it might be stored level by level on a secondary storage which would require the additional problem of finding the correct sub-tree and also it may require a number of transfers, with the worst condition as one block transfer for each level of a tree being searched. This situation can be remedied if we use b-tree as data structure.

B-tree as an index has two advantages :

- Its completely balanced.
- Each node of B-tree can have a number of keys. Ideal node size would be if it is somewhat equal to the block size of secondary storage.
- The basic structure of B-tree was discovered by R. Bayer and E. McCreight (1970) of Bell Scientific Research Labs and has become one of the popular structures for organizing an index structure.

- The B-tree is a useful balanced sort-tree for external sorting. The techniques of organizing a file and its index called B-tree is the standard organization for indexes in a database system.
- A B-tree of order N is a tree in which :
- Each node has a maximum of N children and a minimum of the ceiling of [N/2] children. However, the root node of the tree can have 2 to N children.
- Each node can have one fewer keys than the number of children, but a maximum of N-1 keys can be stored in a node.
- The keys are normally arranged in an increasing order. All keys in the sub tree to the left of a key are less than the key, and all the keys in sub tree to the right of a key are higher than the value of the key.

- If a key is inserted into a full node, the node is split into two nodes, and t median value is inserted in the parent node. If the parent node is the roo created.

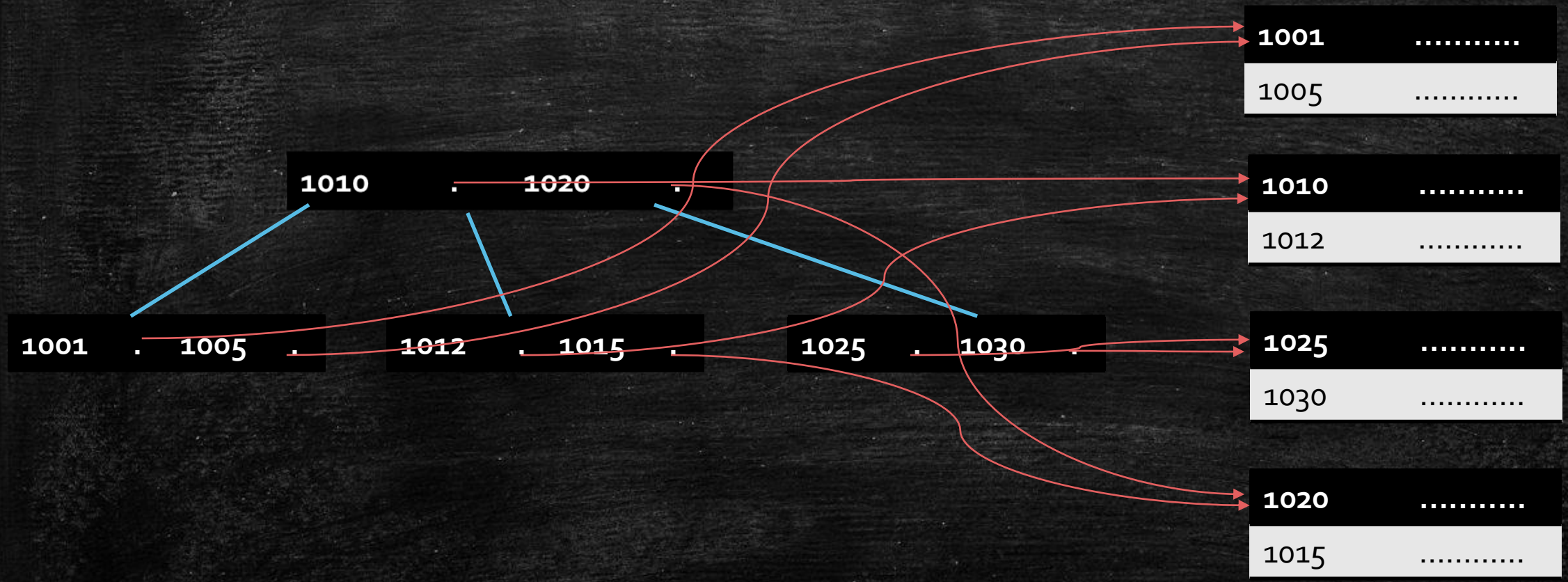
1001	.....
1005	.....

- All the leaves of he B-tree are on the same level. There is no empty sub-tree above the level of the leaves. Thus a B-tree is completely balanced.

1001	.....
1005	.....

1001	.....
1005	.....

1001	.....
1005	.....



- The B-tree has a very useful variant called B+ tree, which have all the key values at the leaf level also in addition to the higher level. These lowest level leaves are linked through pointers. Thus B+ tree is a very useful structure for index sequential organization. j

Multi-key file organization :

- File organization schemes that allow records to be accessed by more than one key field, thus, allowing multiple access paths each having a different key. These are called multi-key file organizations. These file organization techniques are at the heart of database implementation.
- Two of the commonest techniques for this organization are :
- Multi-list file organization
- Inverted file organization

Need for multiple access paths :

- Most of the online information system require the support of multi-key files. For example, in a banking database application having many different kind of users such as :
- Teller
- Branch manager
- Account holders
- All these users access the bank data in a different way. Let us assume a sample data format for the Account relation in a bank as :

Account Number	Account Holder Name	Branch code	Account type	Balance	Permissible loan limit

- A teller may access the record above to check the balance at the time of withdrawal. S/he needs to access the account on the basis of branch code and account number.
- A branch manager may need to find the top ten most preferred customers in each category of account so may access the database in the order of account type and balance.
- The account holder will be interested in his/her own record.
- Thus, all these applications are trying to refer to the same data but using different key values. Thus, all the applications as above require the database file to be accessed in different format and order.
- The faster ways to provide faster access to all such applications can be done in two approaches :
- By replicating data
- By providing indexes

- Replicating data :

In this approach, efficient access to different applications is done by providing access to each of the applications from different replicated files of the data. Each of the file may be organised in a different way to serve the requirements of a different application. All of these files differ in the organisation and would require different replica for different applications. The data replication brings in the problems of inconsistency under updating environments. Therefore, a better approach for data access for multiple keys has been proposed.

- Support by adding indexes :

Multiple indexes can be used to access a data file through multiple access paths. In such a scheme only one copy of the data is kept, only the number of paths is added with the help of indexes. Two important approaches in this category are : Multi-List file organisation nd inverted file organisation.

Multi-list file Organisation :

- Multi-list file organisation is a multi-index linked file organization.
- A linked file organisation is a logical organisation where physical ordering of records is not of concern.
- In linked organisation the sequence of records is governed by the links that determine the next record in sequence.
- Linking of records can be unordered but such a linking is very expensive for searching of information from a file. Therefore, it's a good idea to link records in the order of increasing primary key. This will facilitate insertion and deletion algorithms. Also this greatly helps the search performance.
- In addition to creating order during linking, search through a file can be facilitated by creating primary and secondary indexes. All these concepts are supported in multi-list file organisation.

- Example 1: Consider the employee data. Empid is the key field of the data records.

Record Number	Empid	Name	Job	Qualification	Gender	City	Married/ Single	Salary
A	800	Jain	Software Engineer	B. Tech.	Male	New Delhi	Single	15,000/-
B	500	Inder	Software Manager	B. Tech.	Female	New Delhi	Married	18,000/-
C	900	Rashi	Software Manager	MCA	Female	Mumbai	Single	16,000/-
D	700	Gurpreet	Software Engineer	B. Tech.	Male	Mumbai	Married	12,000/-
E	600	Meena	Software Manager	MCA	Female	Mumbai	Single	13,000/-

Figure 13: Sample data for Employee file

- Primary key of the file is Empid, therefore the linked order of records should be defined as B(500), E(600), D(700), A(800), C(900).
- However, as the file size will grow the search performance of the file would deteriorate. Therefore

we can create a primary index on the file .

- Let us create a primary index for this file having the Empid values in the range:

>=500 but < 700

>=700 but < 900

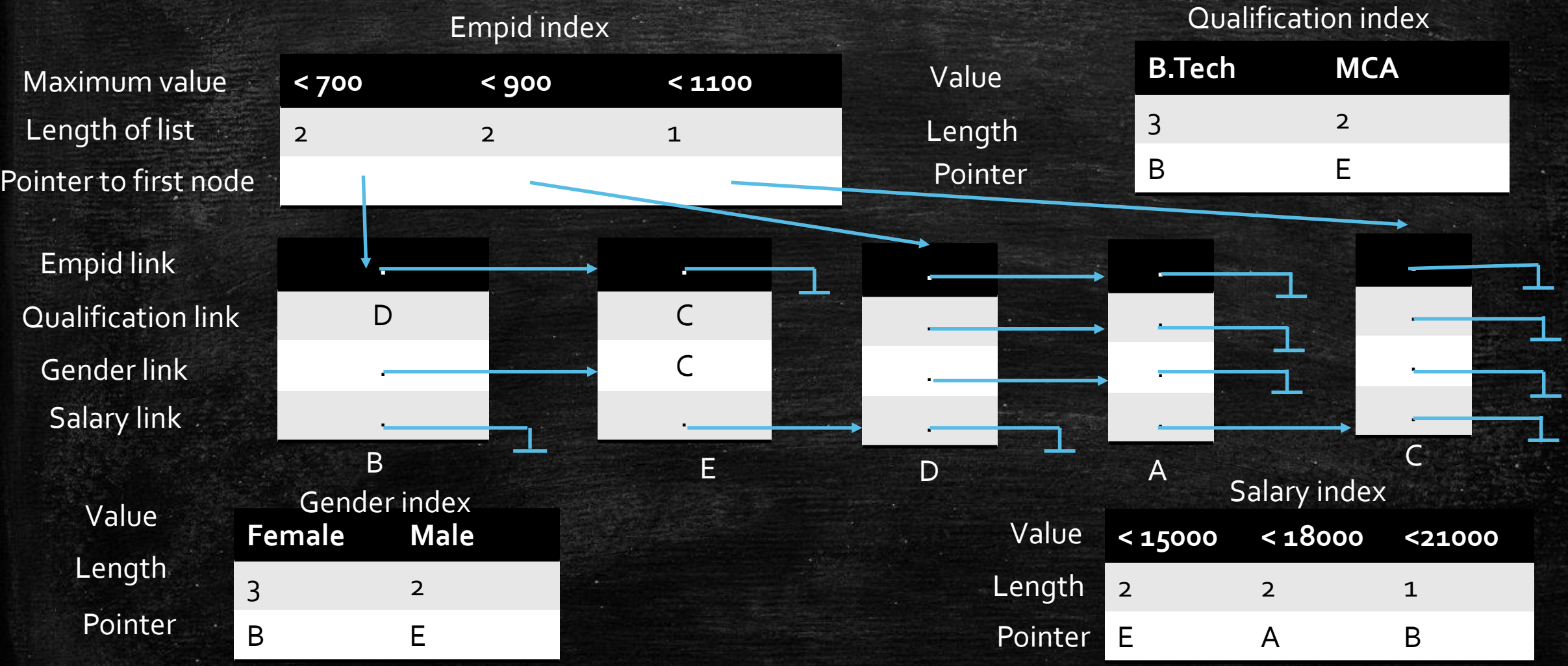
>=900 but < 1100

- The index file for the example data as per above figure is :

Linking together all the records in the same index value :

- In the above figure, records in the same index value range of Empid are linked together. These lists are smaller than the total range and thus will improve search performance.
- This file can be supported by many more indexes that will enhance the search performance on various fields, thus creating a multi-list file organization.
- The below figure shows various indexes and lists corresponding to those indexes.

Multi-list representation for above example



- Multi-list organisation is an indexing scheme in which an index entry contains the length of each sub-list, in addition to the index value and the initial pointer to the list. This information is useful when the query contains a Boolean expression.
- The information about the length of the list may help in reducing the search time in the complex queries.

Example : if you need to find the list of female employees who have MCA qualifications. This can be found out in two ways.

1) Go to the gender index, and search the female index list for MCA qualification.
2) Search the qualification index to find MCA list and in MCA list search for female candidates.

- Consider another situation when the female and MCA lists both are of about a length of 1000 and only 10 female candidates are MCA. To enhance the search performance of such a query a combined index may be the Cartesian product of the two attribute values.

- Insertion and deletion into multi-list is as easy/hard as is the case od list data structures. In fact, the performance of this structure is not good under heavy insertion and deletion of records. However, it's a good structure for searching records in case the appropriate index exist.

Inverted File Organisation :

- Inverted file organisation is one file organisation where the index structure is most important.
- In this organisation, the basic structure of file records does not matter much.
- Its similar to that of multi-list file organisation. The key difference between the two is that multi-list file organisation index points to a list, whereas in inverted file organisation the index itself contains the list. Therefore, maintaining the proper index through proper structures is an important issue in the design of inverted file organisation.

Inverted file organisation for example 1

Empid index

500	B
600	E
700	D
800	A
900	C

Qualification index

B. Tech	B,C,D
MCA	A,E

Gender index

Female	B,C,E
MCA	A,D

Salary index

500	B
600	E
700	D
800	A
900	C

- Please note following points for the inverted file organisation :
- The index entries are of variable lengths as the number of records with the same key value is changing, thus, maintenance of index is more complex than that of multi-list file organisation.
- The queries that involve Boolean expressions require accesses only for those records that satisfy the query in addition to the block accesses needed for the indices. For example, the query about Female, MCA employees can be solved by the gender and qualification index. We just need to take intersection of record numbers on the two indices.
- Similarities between multi-list and inverted file organisation :
- Both organisations can support :
  - An index for primary and secondary key
  - The pointers to data records may be direct or indirect and may or may not be sorted.

- Differences between multi-list and inverted file organisation :

The indexes in the two organisations differ as :

- In a multi-list organisation an index entry points to the first data record in the list, whereas in inverted index file an index entry has address pointers to all the data records related to it.
- A multi-list index has fixed length records, whereas an inverted index contains variable length records.

However, the data records do not change in an inverted file organisation whereas in the multi-list file organisation a record contains the links, one per created index.

Some of the implications of these differences are :

- An index in a multi-list organisation can be managed easily as it is of fixed length.
- The query response of inverted file approach is better than multi-list as the query can be answered only by the index. This also helps in reducing block accesses.

Importance of file organisation in database

- To implement a database efficiently file organisation is needed.
- There are several important consequences of an inappropriate file organisation being used in a database. Using replication would be wasteful of space besides posing the problem of inconsistency in the data.
- The wrong file organisation can also :
- Mean much larger processing time for retrieving or modifying the required record
- Require unique disk access that could stress the hardware