Database Management System

Dr. S. Srinath
Department of Computer Science & Engineering
Indian Institute of Technology, Madras

Lecture No. # 11

Indexing Techniques Single level

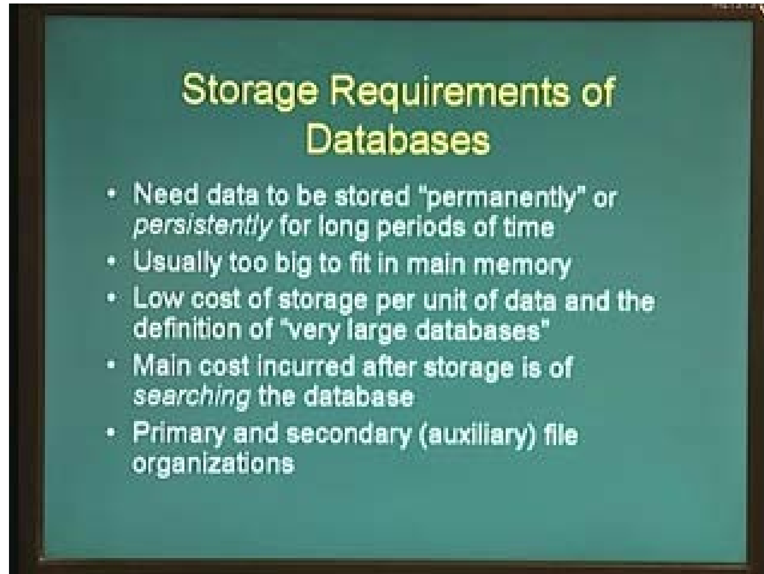
Hello and welcome to another session in database management systems. In our ongoing exploration of dbms, we have a kind of graduated from looking at a data management from a logical perspective to looking at a data management from a physical perspective. We have looked at how we can represent data conceptually using say the ER schema or in a more physical form that is in a way that is more friendly to the computer as in the relational data form and so on.

However all of these models were basically mathematical models that gave us a kind of formalism which told how we can represent data elements and how we can represent relationships among data elements and so on. We graduated from there to see how data is actually stored on the computer, what kinds of overheads do we incur when we use one kind of storage method versus another, which kind of storage method is easier in terms of lets us say insertion, easier in terms of updation, easier in terms of maintenance, easier in terms of searching and so on.

We looked at three basic kinds of file organizations and compared them in terms of their complexity of insertion, updation, search and so on. And in our session on storage structures, we have also mentioned that when we talk about storage files there are basically two kinds of files that we are concerned about, what is called as a primary file or the data file contains the actual data that is stored in the database system and then set of one or more secondary files are what are called the auxiliary files which contain metadata which help us in accessing data elements as efficiently as possible.

In today's session we are going to be concentrating on these secondary files. these secondary files are called as index files which provide one or more index structures that can help us in accessing whichever data element we need as efficient enough as possible. So let us look into index methods by firstly, briefly surveying or briefly summarizing what we have learnt about storage structures.

(Refer Slide Time: 03:27)



First of all what are the storage requirements of databases, what kinds of requirements are we looking at here? Databases need data to be stored in a permanent fashion or what is termed as persistent fashion for long period of time. It shouldn't be the case that once power is switched off, all your data is lost. It should be there for much longer period of time than the typical user session on a computer. And usually the amount of data that we are talking about is too large to fit in memory.

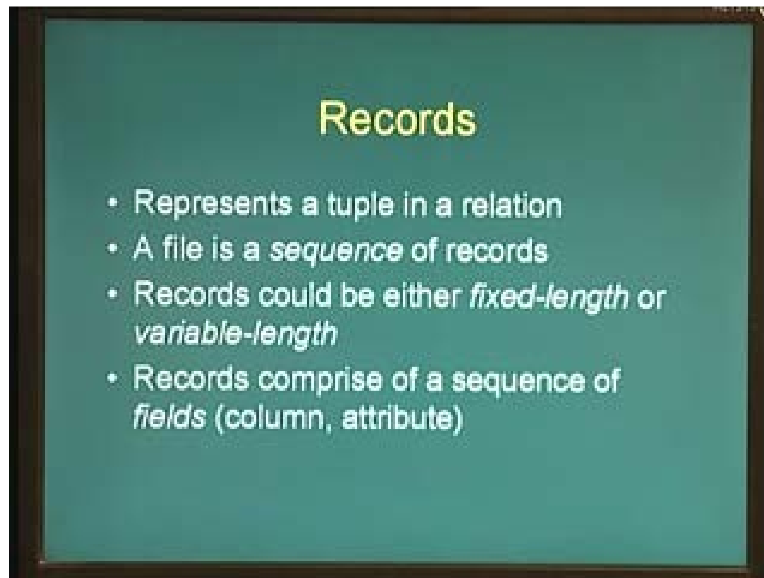
We saw how the definition of very large databases has been changing over the years. Initially the term very large databases was used to mean hundreds of megabytes of data and now we are talking about peta bytes of data which is actually 10 to power 15 bytes of data. This is especially true in databases like web log, I mean web databases as in web search engines like Google, AltaVista or so on. They routinely deal with peta bytes of data and in fact we also saw how the storage technology has been beating Moore's law in the sense that larger and larger amounts of storage is now possible in a smaller and smaller surface area and also at lower and lower prices.

Therefore storage by itself is not a big problem. Storage is cheap, secondary storage is especially is cheap, quite easily available. However the bigger problem now is to search for databases. We have anyway stored peta bytes of data but how do we search the relevant data items or whatever data that we need in as efficient fashion as possible. Imagine how would it be if you used a search engine like say goggle and you gave a web search and it gave you a request to come back after two days to look at your search results.

It is unthinkable, we are looking at response time that is interactive in nature at most a few seconds before which the user gets bored or the user cannot wait beyond that. Therefore a search engine like this has to search potentially a data space of peta bytes of data before giving results for a given request. So in order to efficiently handle these data

structure or this vast amounts of data, what is getting more important now is the set of secondary or auxiliary file structures using which we can try to efficiently access the data that is stored in primary databases.

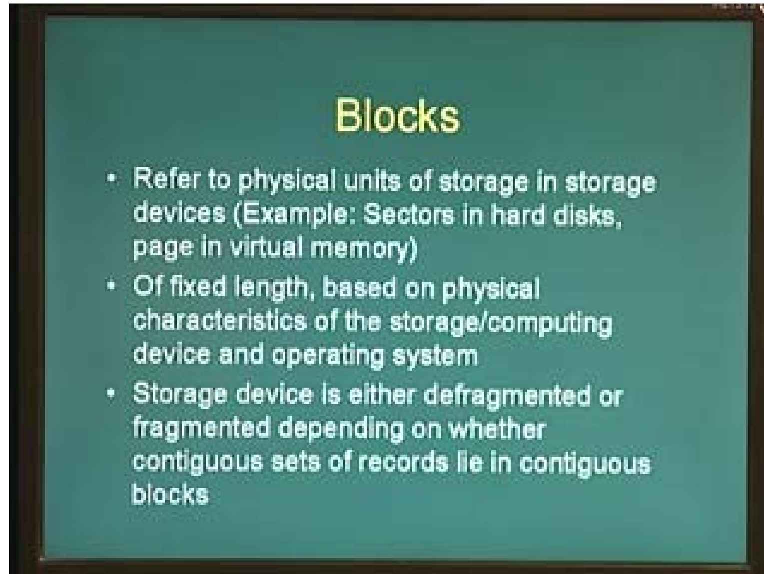
(Refer Slide Time: 06:25)



Let us briefly go through some of the definitions that we studied in storage structures. Let us review some of them because they are again important when we are talking about indexing methods. First, the notation of a record. A record is the physical counter part of what could be termed a tuple in a relation or a row in sql parlance and on disk a data is stored in terms of files and file is treated as a sequence of record. So, one might analogously say that a file stores a given relation or a given table. Although it need not exactly be the case because sometimes files are used to store more than one relations or relation is sometimes spread across different files due to some physical considerations like maximum file length that is allowed by the operating system and so on.

However in a general sense we can consider a file to be representing or to be storing or to be the physical counter part of a relation. Records could be of either fixed length or of variable length. Fixed length records are easier to handle in terms of quickly finding their location in a file for example finding the offset of a given record in a file. However not all data elements can be amenable to fixed length records especially when we have to store data elements in form of text where a text can range from few words to thousands of words. So if we allocate a large amount of memory for the text field, it would be unduly wasteful when we are using fixed length records in which cases we use variable length records. And records themselves comprise of a sequence of fields. A field is analogous to a column in sql parlance or an attribute in the relational algebra parlance.

(Refer Slide Time: 08:25)



We dint saw the concept of blocks. A block is a physical unit of storage in storage devices for example sectors in hard disk or page in virtual memory and so on. They are the smallest unit of data that are been transferred between the storage device and the computer. And usually we deal with block storage devices. When we are talking about databases, we rarely deal with character devices where the unit of information transferred is a single character. Blocks are usually are almost always of fixed length, they are not of variable length and the length of a block is based on several characteristics that are beyond the scope of a typical dbms. For example they are based on a consideration that deal with what is the storage capacity of the storage device that we are talking about, what is the operating system that we are using, what is the size of the data bus in the machine and so on.

Therefore the database management system has little or no control over the size of a block and a contiguous block in a storage device may or may not correspond to the same file. If they correspond to the same data file then it is well and good, in the sense that there is lesser overheads in accessing a file. We don't need too many seeks, seek is the set of operations that is performed on any storage device like disk in order to find the correct block that we want from the device. So if contiguous blocks belong to same file, we do not require too many seek operations when accessing a data file. On other hand if they do not belong to the same file then we may incur some overheads in the seek time of the storage device. So a storage device is said to be defragmented, if contiguous blocks belong to the same file and it said to be fragmented if the blocks are distributed all across the storage device.

(Refer Slide Time: 10:30)

Blocking Factor

The number of records that are stored in a block is called the "blocking factor". Blocking factor is constant across blocks if record length is fixed, or variable otherwise.

If B is block size and R is record size, then blocking factor is:

$$bfr = \lfloor B/R \rfloor$$

Since R may not exactly divide B, there could be some left-over space in each block equal to:

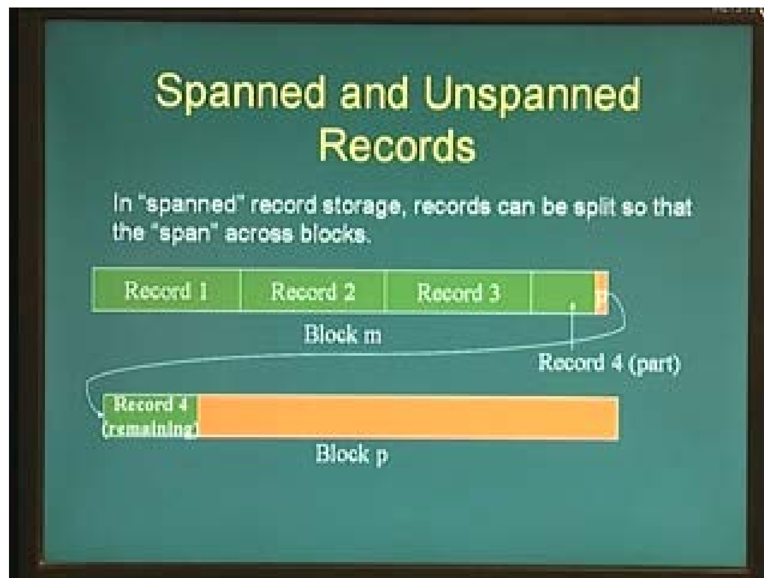
$$B - (bfr * R) \text{ bytes.}$$

When we are taking about blocks, there is an important term that we used the notion of a blocking factor. The blocking factor is simply the number of blocks per record that are stored in the database. That is if I have a record size of R and I have block size of B, blocking factor is simply B divided by R, the floor of this function B divided by R. If B is greater than R then blocking factor is greater than 1, that means there can be more than one records per block. However if R does not divide B that is if B is not a pure multiple of R then there is some extra space that is wasted which is given by the reminder of this division. So how do you deal with this extra space?

(Refer Slide Time: 11:23)

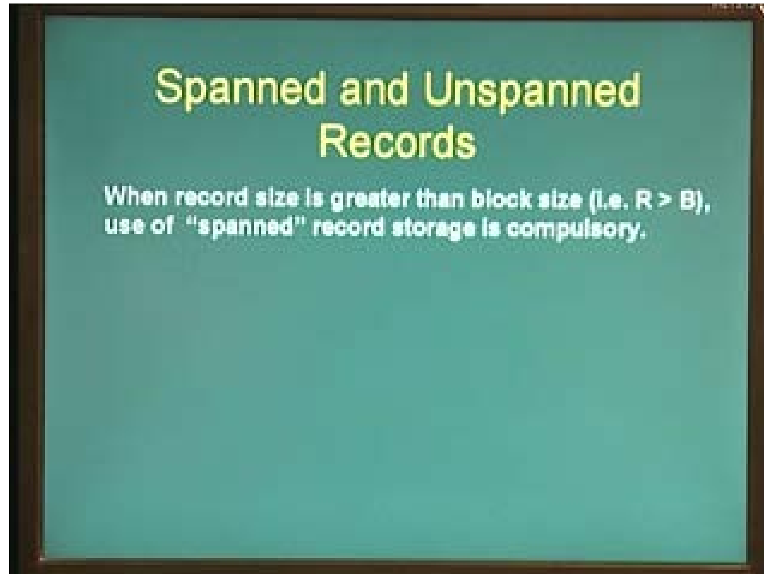
There are two varieties of dealing with this extra space. We saw that records could be either of the kind of unspanned records or they could be spanned records. Unspanned records are those which do not span across different blocks, in such cases if we have some extra spaces as shown in the slide here they just are unused. We can't do anything about it, that is we just leave that extra space unused which results in certain amount of wastage of space. However it helps in easy accessing of records from blocks.

(Refer Slide Time: 12:35)



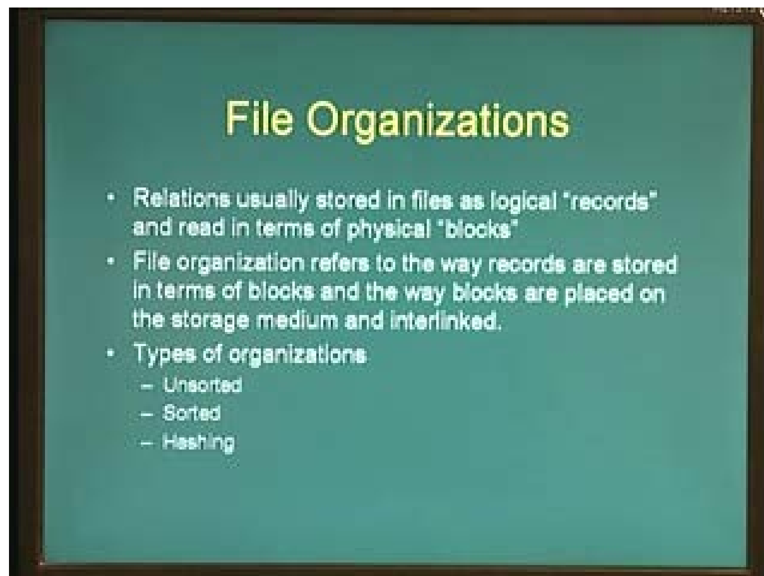
On the other hand we could think of spanned records were records can be split so that they are stored across different blocks. This slide shows such a diagram where three records are stored in their entirety in a given block m and the fourth record is split between block m and block p. And of course at the end of each block we should have some kind of a pointer that points to the next logical block, next block in logical sequence in the disk, so that we can know which block to access next in order to find the remaining part of the fourth record.

(Refer Slide Time: 12:38)



And of course whenever B is less than R that is whenever record size is greater than the block size then we have to use unspanned record storage.

(Refer Slide Time: 12:54)



We also saw three different kinds of file organizations and a file organization is simply an organization mechanism by which data is stored in files so that they can be efficiently accessed. We saw three different kinds of file organizations which we termed as unsorted files or pile files, sorted files and hashing files. Unsorted files are those files where you just input records into the file or just append new records at the end of the file without any consideration to the data that is present in the record or in the file.

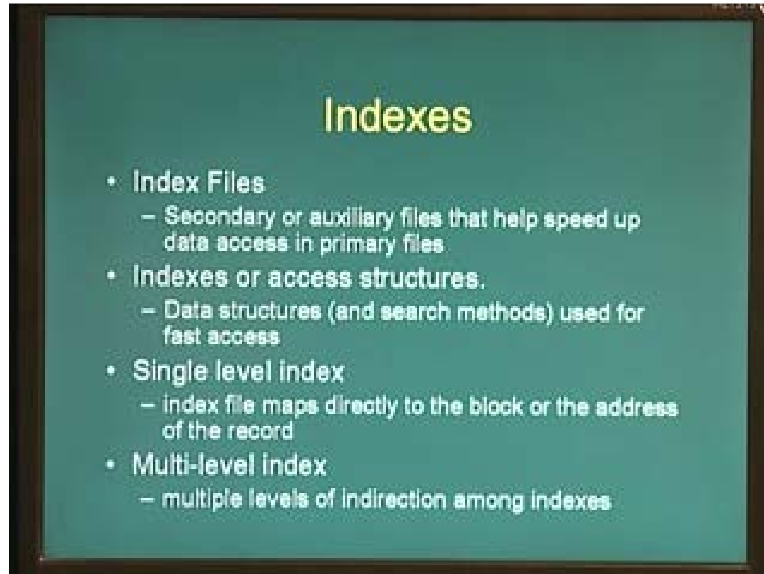
Unsorted files are very efficient when it comes to inserting new records. You don't have to do any kind of searching, you don't have to do any kind of reorganization, you just have to append it to the end of file. However it is very inefficient when it comes to either deletion or modification of a data or searching, especially in searching of data. In the worst case we may have to search the data file in a sequential fashion. And this can be a tremendous overhead when the file is extremely large in size, when it is giga bytes or tera bytes of data stored in one unsorted file.

A sorted file on the other hand is a file organization where the file is physically sorted on the disk based on the some field in a record which is called the ordering field. Therefore a physical sorting of records helps us in easy access of the file. We can use a search technique called binary search that can reduce or search space by half in each iteration so that usually what is termed as log N order of time, we can find whatever record that we are looking at. However the sorted file organization incurs a lot of overhead whenever insertion or deletion happens in the file.

Whenever a new record is inserted, we should ensure that the file remains physically sorted that means we may have to records in order to accommodate the new records in its place. And similarly the case for deletion, in order to remove any kind of fragmentation we have may to move records so that the overall file remains sorted. The last kind of file organization we saw was the hash files. A hash file uses a hashing function which hashes or that is which is a function that can take a value of a key field in a record and transform it into one of several bucket addresses.

We saw two different kinds of hashing, static hashing and dynamic hashing where in static hashing the number of buckets are fixed and we have to deal with, we have the problem of contending with overflows especially if the data is queued. That is if the data set or the distribution of keys is queued, we may have a small number of buckets overflowing and I have to be dealt with techniques like open addressing or chaining and so on which poses terrible overheads during searching and we could have a large number of buckets that are empty because none of the keys were hashed to those functions. We also saw a remedy to this problem namely the notion of dynamic hashing where the number of buckets could actually grow or shrink in size whenever records are added or deleted from the database.

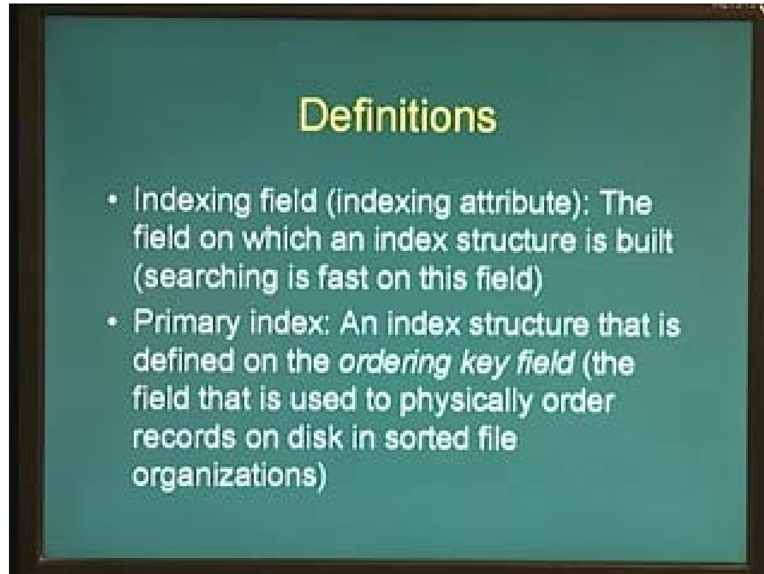
(Refer Slide Time: 16:40)



Let us move on to auxiliary file structures, the main topic of concern today, the notion of indexes. Let us first go through a few definitions before we look at actual index structures. Firstly, the notion of an index file. An index file is a secondary or auxiliary file that contains meta data or data that helps us in accessing the required data elements from the database. An index or an access structure is the data structure that is used inside the auxiliary files that helps us in searching for our data elements as efficiently as possible. And of course data structure are augmented with their corresponding search methods in order to search for our data record.

We can think of two kinds of indexes what are called as single level indexes and multi-level indexes. Single level indexes have just one level of index structures that is in addition to the primary file there is just one secondary file and the index file maps directly to block or record addresses in the primary file. A multi-level index on the other hand has multiple levels of indirection where one level of index structure may point to another level of index structure and so on. And finally the last level would point to block addresses or record addresses in the primary file.

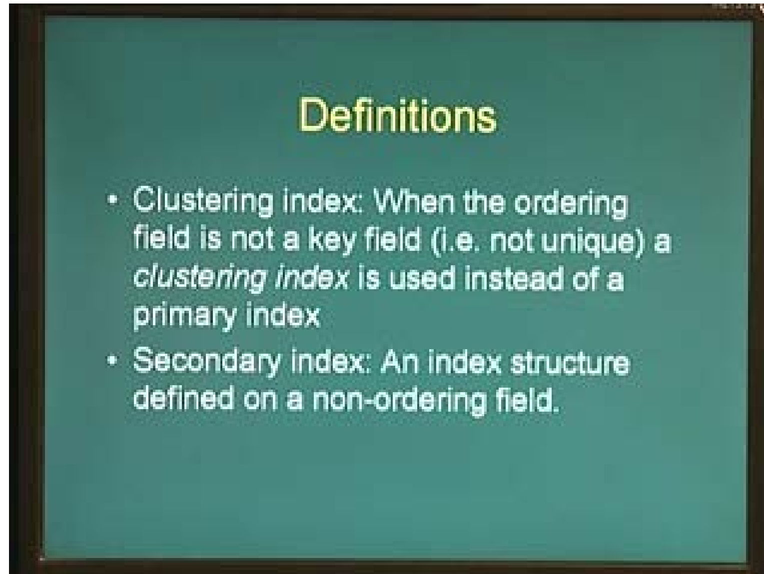
(Refer Slide Time: 18:15)



Let us look at a few more definitions regarding pertaining to index structures. The notion of an indexing field or an indexing attribute is the field on which the index structure is built. That is searching is efficient whenever a search is given on this field that is on whichever field an index is maintained. For example, we saw the notion of the ordering field which is a field based on which records are ordered on disk in sorted file organizations. An indexing field is analogous in the sense that this is the field on which an index structure is build.

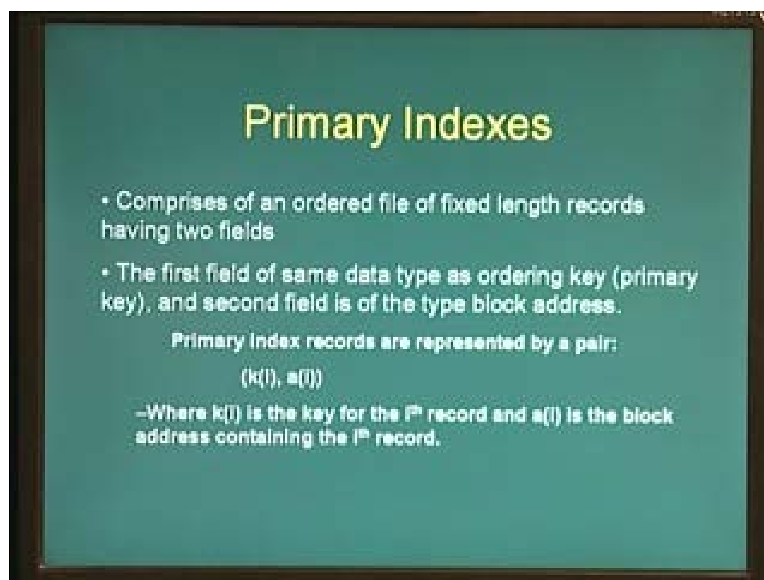
Usually the ordering field and the indexing field are the same and usually they are also the primary key that is the primary key is usually the ordering filed and by default an index structures is built on the primary key that is used in the records. A primary index is an index structure that is defined on the ordering key field that is the field that is used to physically order records on file in sorted file organizations. And in many cases the ordering field is the same as the primary key.

(Refer Slide Time: 19:36)



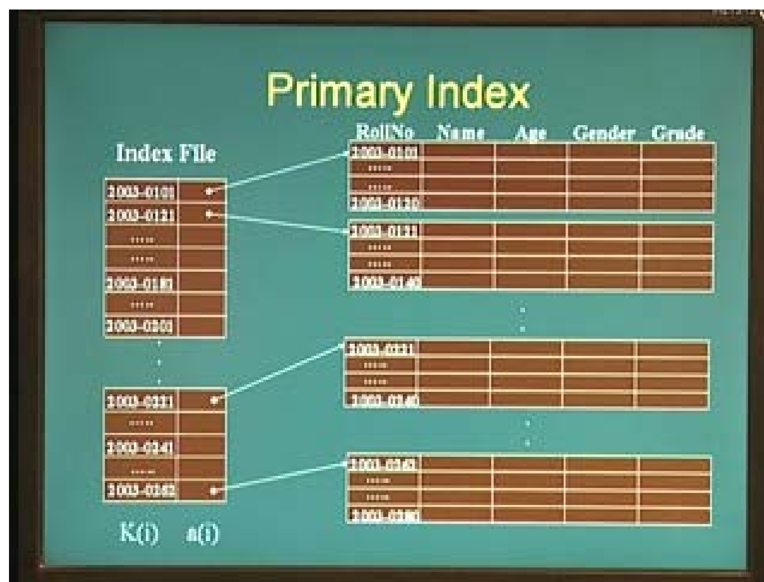
We also define a notion of clustering index where these are index structures that are defined on an ordering key field. However in cases where the ordering field is not the primary key or is not even a key field that means the ordering filed need not be unique. Remember that all key fields have the unique constraint, that are posed on them and whenever ordering is performed on a non-key field, it is not unique. So an index structure on such a non-unique field is called a clustering index. We also define the notion of a secondary index which is an index structure that is defined on a non-ordering field and even not necessary the key field.

(Refer Slide Time: 20:27)



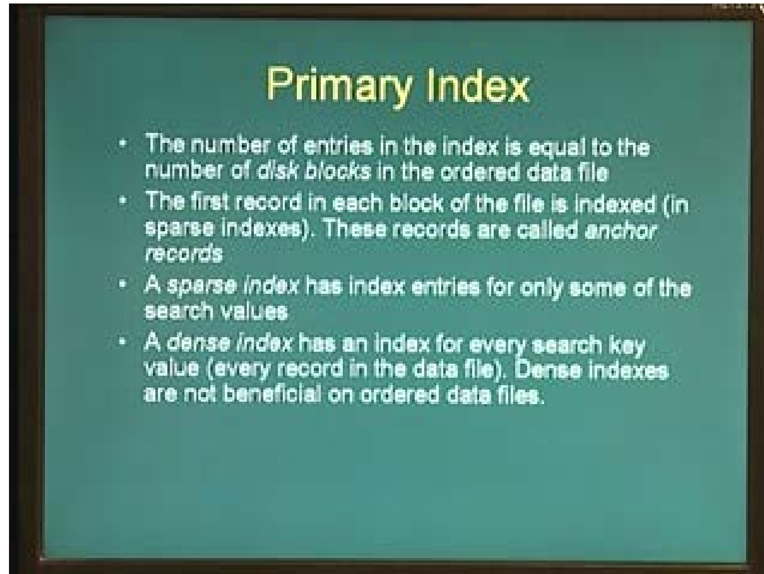
Let us have a look at primary index to begin with. Like we mentioned before a primary index is an index structure that is defined on the ordering field of records and that to when the ordering field is a key field. Usually it is a primary key on which records are ordered on disk. A primary index comprises of an ordered file, note that even a primary index is a sorted file and it comprises of fixed length records and has two fields. These two fields are shown in the slide as k of i as you can see here. That is there is a pair called k of i which is the key for the i th record and a of i is the block address containing the i th record.

(Refer Slide Time: 21:21)



The figure in this slide shows an example of a primary index. Here on the right hand side of the figure, the primary data file is shown where records are divided into several blocks. So each block begins with a particular field number and of course this is a sorted file organization in the sense that records are physically sorted in these blocks. Now if you can notice here, the first record or the first field in each block for example the first field in the first block is 2000 3 0 1 0 1, the roll number of a student and of the second block is 2000 3 01 21 and of some other block here is 2003 02 21 and so on. The first field in these blocks are indexed in the index file that is they appear in the index file here. These the first field in the index file is K of i which we saw earlier that is the key value that is maintained in the index. The second field contains the pointer to the block or the block address that contains this record.

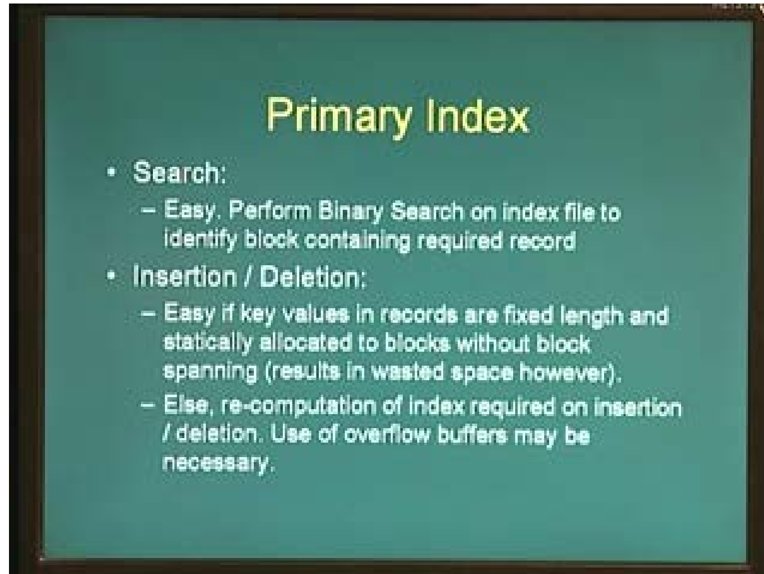
(Refer Slide Time: 22:39)



What is some of the properties of primary indexes? If you have noticed, the number of entries in the index is equal to the number of disk blocks that comprise of the primary file that is the primary or the data file. The first record in each block of the file is indexed that is the first record in each block of the file appears as 1 of k of i's in the index file. These records are called anchor records because this is the anchor by which other records in the or other key values in the block are accessed.

Therefore if you are searching for a key value of let us say 2003 01 20 as shown in the slide here, we perform a binary search on the index file and we come to a point where we realize that it has to lie in the first block itself because 2003 01 23 is greater than 2003 01 01 and lesser than 2003 01 21. So we have to find that block address which or that anchor address which is lesser than or lesser than or equal to the given key and the next block address would be greater than the given key. Such an index structure in which not all ordering or key attributes are indexed is called a sparse index. The primary index that we saw here is a sparse index. A sparse index essentially means that not all attributes or not all possible key values are indexed. More specifically in a primary index, we are only indexing one key value per block. On the other hand a dense index or is an index structure where each key or each search key value that appears in the primary data file is indexed in the indexed file.

(Refer Slide Time: 24:50)

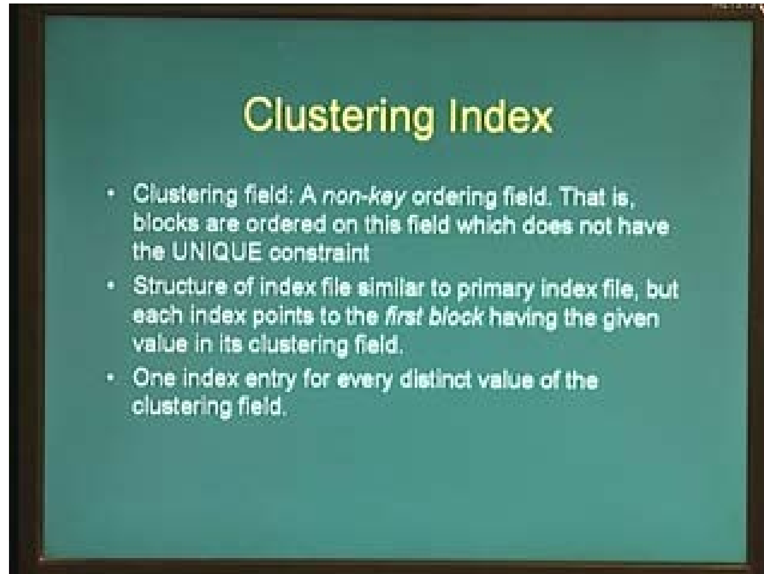


How do we search using a primary index? Search is easy, we saw just a few moments ago that we can search using binary search, so we just have to perform a binary search whenever we have to search using the key value and then find such a record or a key value that is less than or equal to the key value that we are looking for, such that the next key value that appears in the index is greater than the key value that we are looking for. What about insertion and deletion? Insertion and deletion is easy if records are of course one fixed length and they are statically allocated to blocks, without block spanning. What is this mean? This means that suppose I allocate, suppose I know the set of all, the entire range of values of a given set of keys and I statically allocate a given sub range or subset of keys to particular blocks.

For example let us consider that a student roll number can range from 0 1 0 1 to 0 1 5 0. Let us say there are 150 students who can range from 0 1 0 1 to 0 1 5 0 and then we say that each block contains exactly 20 records. Therefore we store 0 1 0 1 to 0 1 1 9 or 0 1 2 0 rather in the first block and 0 1 2 1 to 0 1 4 0 in the second block and so on. Therefore we statically allocate each record to block address. If we do that insertion and deletion are easy. However they may result in wasted space, especially if not all records of this data set may be available at any given point in time.

For example if we have only let us say 0 1 0 1 and 0 1 5 0, we still have to have a number of blocks wasted because each address is stored in a particular block, we cannot store any other address in other blocks. On the other hand if we don’t want this wastage of space then we have to contend with re-computation of blocks that is moving records between blocks and also re-ordering of the index structure whenever insertion and deletion takes place. That is this is because primary index is based on a sorted file that is a file on which the entire data set is sorted and it has to remain sorted whenever insertion and deletion take place.

(Refer Slide Time: 27:37)



The next kind of index structure that we look at is the clustering index. Remember the definition of clustering index. A clustering index is an index structure that is used when a sorted file organization is used and the ordering key or the key on which a file is sorted is not the key field. That is ordering field is not a key field. What is the implication of saying that the ordering field is not a key field? The implication is that when a field is not a key field, it means that there is no unique constraint on the field. That means there could be repetitions, that is the same key value in the k of i, a of i model, the given k of i value may point to multiple addresses or multiple block addresses which store data values pertaining to the same key.

The structure of a clustering index file is similar to that of a primary index file in the sense that even this stores k of i and a of i addresses except that, except to this small change that only distinct values of the ordering field are stored. That is suppose we are ordering a primary data file based on the student names rather than roll numbers. We just store one index entry for each distinct student name and not every occurrence of student name. And we store the block address of the first occurrence of any given student name that is the first occurrence of the particular ordering field is what we are storing rather than all addresses of a given field.

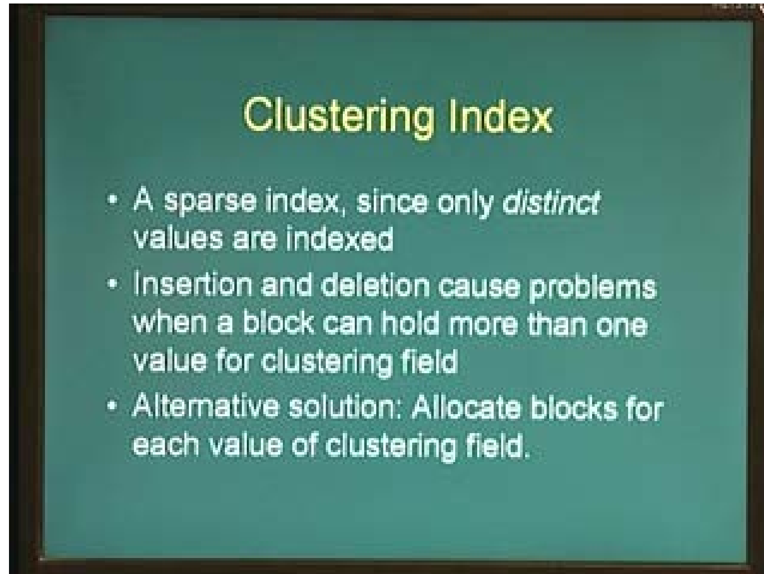
(Refer Slide Time: 29:38)

Let us look at an example. The slide in this example shows a data file which is a sorted data file of course and where the ordering field is the department number and there is no unique constraint on the department number. That is department number may repeat over several records. We see here in this example that there are three records pertaining to department number 1 and three records pertaining to department number 2 and two records pertaining number 3 and so on. And we also see that the record pertaining to department number 2 has spanned or this key or this ordering field department number 2 has spanned over to two different blocks that is the first block and second block. The left hand side of the slide shows the clustering index which contains the usual k of i and a of i fields where k of i is the value of the ordering field or the indexed field and a of i is the block address.

Note the first two entries in the indexed file. The first entry says that for the value one, department number 1 look up block number one. The second record says that or the second indexing record says that for the value of department number 2 look up block number 1 again. Why is this so? This is because the value two appears for the first time in block number one. What about the other values of two, how do we search for the other values of two. Note that the file or the primary data file is a sorted file that means to say that if we know the first occurrence, if we know the block number of the first occurrence of a given ordering field, it is sufficient because the file is sorted and we can start looking at the next logical blocks in sequence, until we finish all the set of records having the same ordering key.

Therefore if we are searching for a particular data record having department number 2, we have to first start from block number 1 and proceed in the next logical block in sequence which in this case is the block number 2 until we exhaust all records having department as 2.

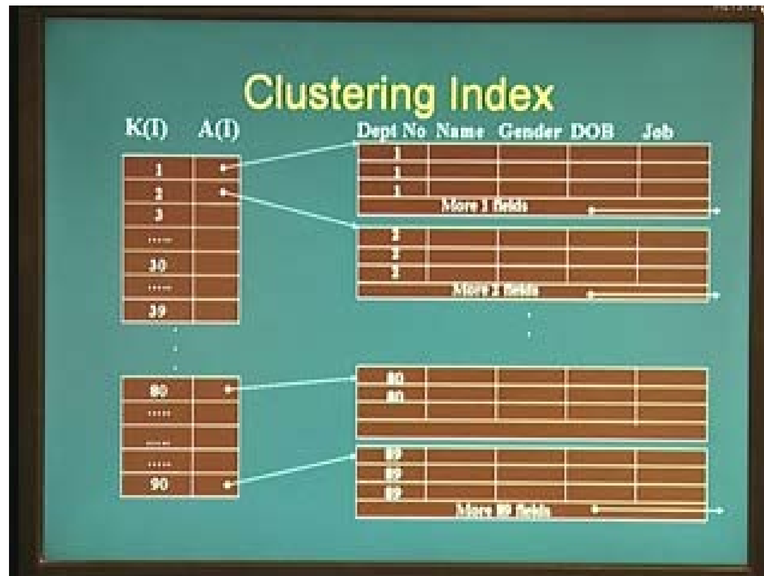
(Refer Slide Time: 32:07)



A clustering index is also a sparse index. Why is this so? This is because only distinct values that are appearing in the ordering field are indexed. In the previous slide even though department number 1 appeared three times and the same thing would be for department number 2. Only a single entry existed in the clustering index field for cluster indexing file for both of these records. Insertion and deletion in clustering index may cause problems because of the well-known problem of sorted files that is the files have to be sorted and when we have to retain this sorted mechanism of or sorted form of files, this may impact on the clustering index.

In the previous example suppose we inserted a new record having department number as 1 then we have to move all the records having department number 2 that is the last record in block number 1 to the next block. That changes the corresponding address, the corresponding a of i value in the clustering index field that is a of i cannot, can no longer point to the first block but instead it has to point to the second block. Therefore we need re-orderings, whenever we perform insertions and deletions. There is an alternative solution for handling this problem of insertions and deletions that is to allocate blocks for each distinct value of the clustering field. This is shown in the example in the next slide.

(Refer Slide Time: 33:53)



This slide shows an example, where a clustering index is used on a primary file that is ordered on a non-key attribute. However there is a very specific organization of the non key of the primary file in the sense that each distinct value of the non key attribute of the ordering field is allocated a separate block. Have a look at the right hand side of the figure more carefully. In the first block there are three records having department number 1, in case more records are inserted having department number 1, they are not allocated to next logical block in sequence. However they are allocated to separate blocks and a separate pointer is maintained to these blocks, so that we can access more records having department number 1.

Have a look at the third block that is seen in this slide here. There are two records having department number 80 and the block can accommodate three records of this particular size. However even though other records exist for example department number 89, they are not put into to this block; they are given a separate block by themselves. Therefore this kind of block organization results in certain kinds of wasted space because especially since, especially if there are not many repetitions of the non key attribute. However insertion and deletion are much more simpler in such an organization this is because we don’t have to worry about any changes in either the block structures or in the index structure itself.

(Refer Slide Time: 36:02)

Secondary Index

- Used to index fields that are neither *ordering fields* nor *key fields*.
- Many secondary indexes possible on a single file.
- One index entry for the every record in the data file (dense index), containing the value of the indexed attribute, and a pointer to the block / record.

The next kind of index that we are exploring today is what is called as the secondary index. A secondary structure or a secondary index file is an index file that is used to index fields that are neither ordering fields nor key fields, that is there is no assurance from the primary file that the file is organized or ordered along these fields and they are also not key fields. That is there is no assurance that the values in these fields are unique. There could be many secondary indexes possible on a single file that is depending on how many fields that are there in a given record. A secondary index maintains one index entry for each record in the file that is if you remember the definition of a dense index, a secondary index is a dense index.

(Refer Slide Time: 37:10)

Secondary Index on Key Field

K(I), A(I)

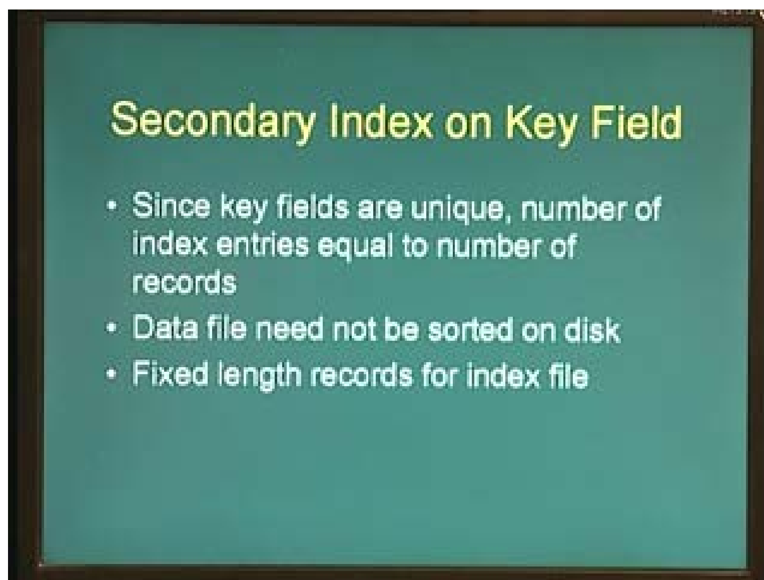
2003-0101
2003-0102
2003-0103
2003-0104
2003-0105
2003-0106
2003-0107
...

RollNo Name Age Dept No Job

Has as many index entries as the number of records...

In contrast to a sparse or a non-dense index where not all values of the indexed field are indexed. This slide shows an example of a secondary index. Assume that roll number is no longer the key field and it need not even be unique. So the left hand side of this slide shows a dense index where each and every field or each and every value of the roll number field that appears in the primary data file also appears in the index file. And there is a corresponding pointer from each of these key values to the particular record addresses directly. Note that we don't have to maintain block addresses here because this is a dense index, we can directly dereference or directly refer to the record address that is the block address and the offset within a block where the record begins. And note that the file also need not to be ordered based on this secondary index and the pointers or the record pointers are arbitrarily shaped when they point to the primary data file.

(Refer Slide Time: 38:22)

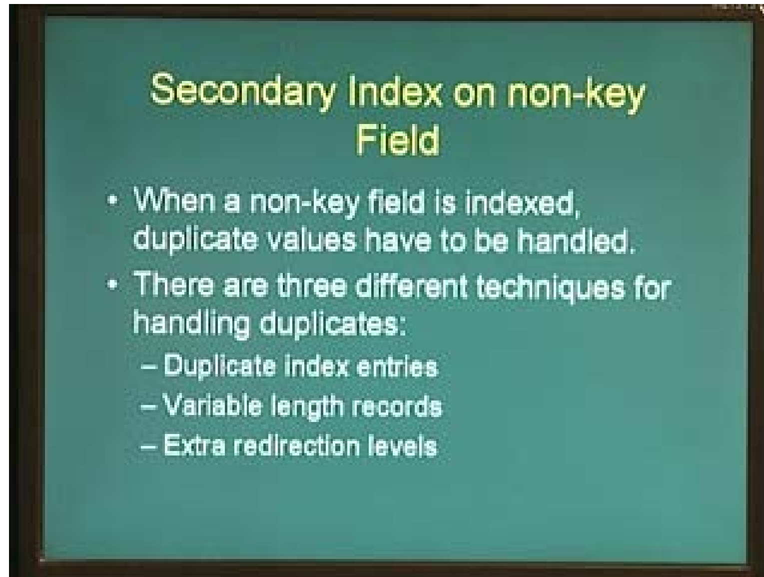


What are some of the properties of this secondary index? Since if I am performing a secondary index on a key field, let us look at some properties of secondary index, maintaining secondary index on a key field. In previous example roll number could be a key field in the sense that it need not be an ordering field but it could be a key field. The data file could be ordered on some other field let us say name or grade something like that. But the index is maintained on the secondary or on the key field which is unique that means that since each field or each key field is unique, there are as many secondary index entries, as there are records in the data file because it is dense index and each index entry or each key field has to be indexed.

However the data file need not to be sorted on to the disk that is it need not have, there is no need for that because we are directly referencing it's a dense index and directly pointing to the particular record address. And because it is key field and because key fields are unique, we can maintain or we can be sure that we can use fixed length records for the secondary index because we know the length of the key field and we know the length of the address of a given record.

Therefore the length of a key field plus the address of the given record in a primary file forms a length of the cluster of the secondary indexes.

(Refer Slide Time: 40:11)



What happens if the secondary index is maintained on a non-key field that is where the field that is being indexed may have duplicate values. That is it need not be unique and it can have many number of value. What is the implication of having duplicate values? As we saw in the clustering index, a given value of the indexed field may point to multiple records or multiple records in the primary file. There are three different techniques for handling the duplicates in secondary index. Note the difficulty that we encounter in secondary index that is not there in a clustering index. In a clustering index we have the assurance that even though the indexing field that we are using is not a key field, the primary file is ordered or is physically sorted according to this ordering field on this non-key field.

Therefore if we just know the first occurrence of a given particular value of this field, it is sufficient using which we can access all other values that are present in the database. On the other hand when we are using a secondary index and a secondary index can be used on fields which need not be ordering fields. We don't have any assurance of that sort that is this is a non-key field, so therefore there could be duplicates and there is no assurance as to how these records are distributed in the primary file itself. So handling duplicates becomes much more difficult in a secondary index rather than in a clustering index.

There are three different varieties of handling secondary index duplicates, one is to use duplicate index entries. Duplicate index entries means that we use fixed length records however and of course use a sorted file or physically sorted file for the secondary index file and maintain as many index records as there are different values of a given non-key field.

The second one, the second approach is to use variable length records that is have one value of K of i and many values of a of i that is more than one values of a of i that point or that in turn point to different record addresses or the third approach is to use extra re direction levels which will see in more detail shortly where the first level points to a block of record addresses and so on.

(Refer Slide Time: 43:04)

Duplicate Index Entries

K(I)	A(I)
2003-0101	
2003-0102	
2003-0102	
2003-0102	
2003-0102	
2003-0103	
2003-0103	
...	

Index entries are repeated for each duplicate occurrence of the non-key attribute.

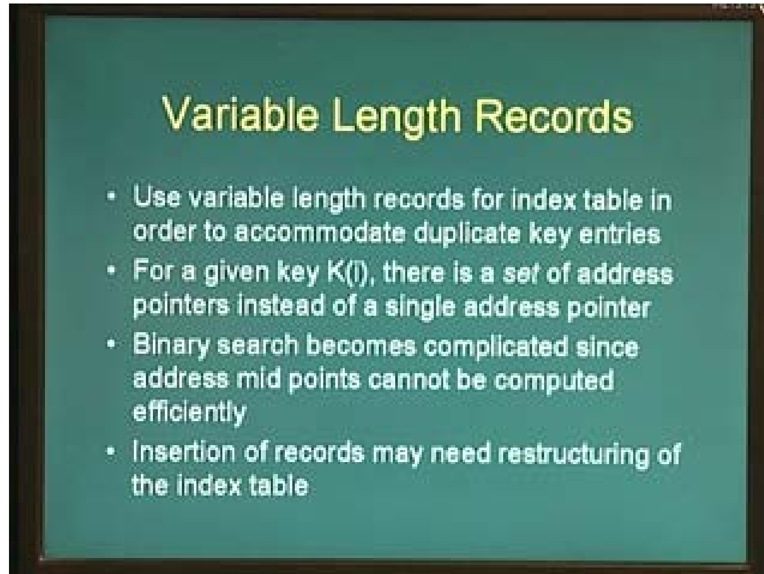
Binary search becomes more complicated. Mid-point of a search may have duplicate entries on either side.

Insertion of records may need restructuring of index table.

The first option is to use duplicate index entries, that is index entries are repeated for each duplicate occurrence of the non key attribute. The example shows here that the term 2003 01 02 is repeated 4 times and 2003 01 03 is repeated twice in the secondary index file. The advantage of such a scheme is that we can still use fixed length records however the searching of this data file becomes a bit more complicated. Binary search becomes more complicated. Why is this so? Because remember how binary search works. Binary search starts with the entire space or the entire set of indexes, as the search space to begin with that is the lower bound for the binary index is to begin with the first record and the upper bound is the last record. And then we compute a mid-point of the lower bound and the upper bound and compare our key, the key that we are searching with the mid-point.

Now what happens if we compute a mid-point in a data file or in an index file where key values could be repeating. When we compute a mid-point that is when you compute lower plus upper divided by two, we are not sure that or we cannot say that all records having this particular key will appear either to the left or to the right that is there could be repetitions or there could be duplicate entries of the particular key at the midpoint on either sides of the midpoint. So we have to search both sides in order to retrieve our particular or in order to make the next decision about the next iteration. And of course insertion of records would require restructuring of the index table because the index table is always sorted and maintained in a sorted order.

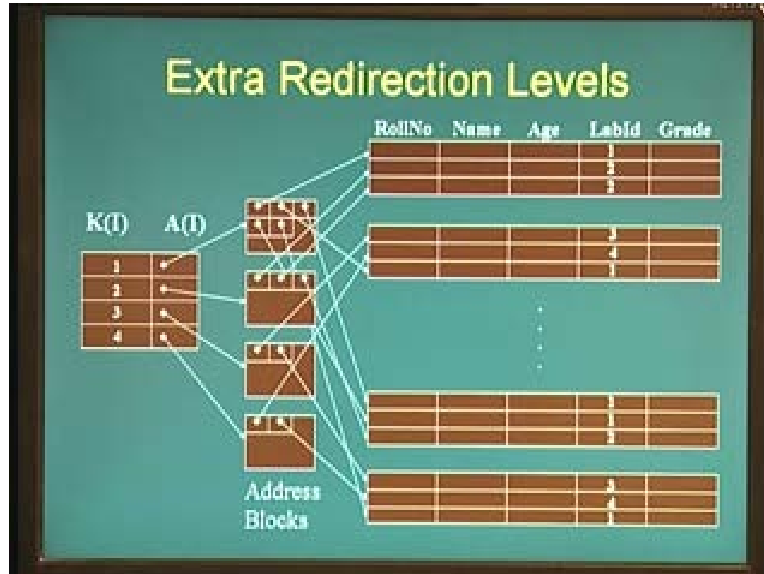
(Refer Slide Time: 45:10)



The second approach to handling secondary indexes was to use variable length records. In a variable length recording record schemes we use a, given K of i value that is if you look at the previous slide here for a given K of i value, the size of the a of i field is not fixed. That is there could be one or several address fields for the given key index. What are the problems or advantages of this approach? One advantage is that binary search, well it is still little bit complicated. However it does not suffer from the complications of repeating multiple keys that is if we know exactly the block addresses and block addresses are stored in a way that we can find out the next block address very quickly, we don't have to worry about whether a given entry appears on both either side of the mid-point.

However it becomes a complicated in the sense that if variable length records are stored in a single index file, the midpoint may not pertain to or may to point to a valid block address or a valid block address of the index file. And insertion and deletion of the records may require restructuring of the index table. So restructuring in the sense that we may have to add more fields in the a of i or we may have add more addresses, add more values a of i field which in turn may affect the next fields that appear in the database. And of course there is also this problem of spanning and non-spanning of records that is we may have to use spanning records in order to in order to allocate them into blocks and associated problems of a searching and retrieval which we saw in the session on storage structures also hold for this kind of indexing scheme.

(Refer Slide Time: 47:41)

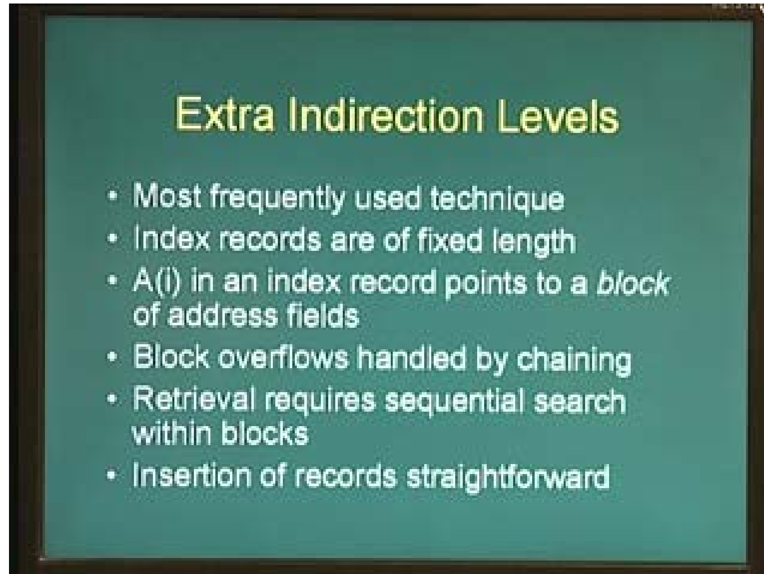


The third kind of scheme, secondary index scheme in order to handle duplicates or in non-key attributes is to use extra redirection levels. Have a look at the slide shown in this figure. This slide shows three different levels of files. That is the right most part of this slide shows the primary data file which contains blocks which in turn holds one or more records. There is a label id or a lab id field in this file which is the record or which is the field which is being indexed by the secondary index file. The lab id field is neither in sorted form in the primary data file nor is it a key field that is it need not have unique addresses, it can have repetitions. Because it can have repetitions, each repetition that is each distinct record address for a given value is given a separate block.

Have a look at the left most part of the slide here. The left most part is the usual secondary index file comprising of K of i and A of i fields where K of i contains distinct values of each of the distinct values that appear for the indexed fields which is a non-key field. Now, because this distinct value can pertain to several records or several records in the database, it is first the A of i address first points to a block that is reserved for storing addresses of this key value or of this index value. This block is of fixed length. Note that this is shown in the second field here. Note that even though a particular is not full, it is not used up for the second address. This is the block for the second value is a completely different block from the block for the first value even if it is not full.

That is a separate block is allocated for each distinct value of the index field and this block contains the set of addresses by which we can using which we can perform a sequential search or on the particular address that we require or retrieve the set of all records pertaining to the given key value that we are searching for. So in this session we looked at three kinds of indexed files or three kinds of single level index structures namely the primary index, the clustering index and the secondary index. Let us briefly summarize before we end this session and look at what are the different kinds of index structures that we saw in this session today.

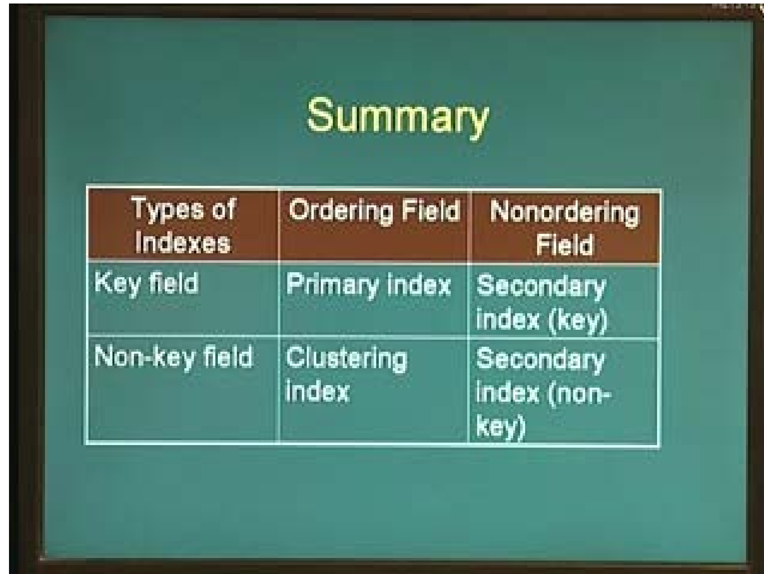
(Refer Slide Time: 50:59)



First of all let us revisit the extra indirection levels and look at some more properties of extra indirection levels. Before we summarize, the extra indirection level is the most frequently technique for handling duplicate record and the advantage of this is that the index records are of fixed length and it doesn't suffer from complications, binary search complications that we discussed earlier and we can use the usual binary search in order to search for a particular given address.

However there could be wastage of spaces because a complete block is allocated for a given value even though if it is not completely full. And what happens when blocks overflow? That is there are large number of records of the given index, in such cases block overflows are handled by chaining which is the same technique that we saw in the hashing technique and retrieval requires sequential search within blocks. However insertion and deletion of records are straight forward, we don't have worry about restructuring the index or restructuring the data file. We just have to insert the corresponding entries in the block file whenever insertion and deletion take place. So let us come back and summarize the different kinds of index structures that we saw today.

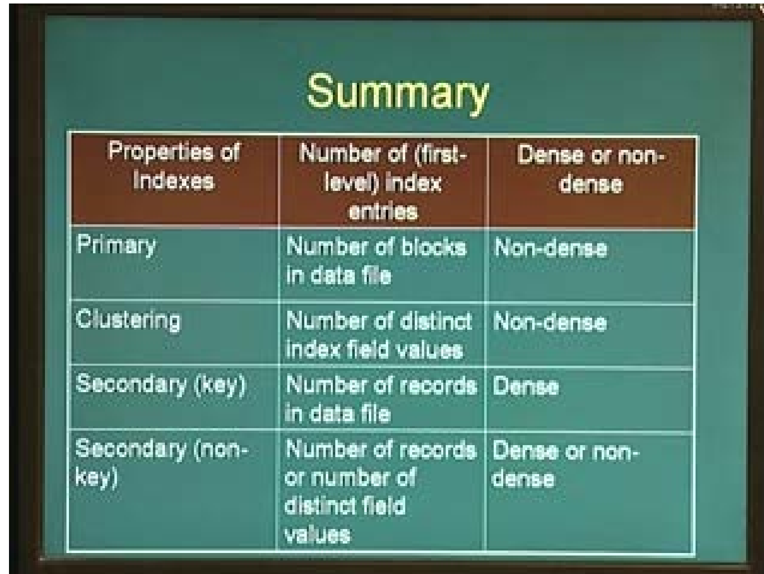
(Refer Slide Time: 52:31)



Firstly the type of indexes: If the index structure is on a key field and the ordering field is or rather if the index structure is on key field and the ordering field which is also an ordering field then such and index structure is called a primary index. That is usually on the primary key and the keys order. So primary index can afford to be sparse and point to and store and index only the anchor records in each blocks. On the other hand if the index is on a non-key field however which is a ordering field then it is called a clustering index. That means the data file or the primary data file is physically sorted based on the ordering field which is not a key field. That means there is no unique constraint for this field, in which case we can use the clustering index where we just store index structures for each distinct value of this field and store the address of the first occurrence of this distinct value. Because it is ordered, we don't have to worry about accessing the other values that exist in the database because they appear in logical sequence starting from the first occurrence.

If the key field is or if the indexing field is a key field, however it is not an ordering field then we use a secondary index for a key field. We saw that a secondary index on a key field is a dense index with fixed length records on which binary search can be used efficiently. If the field on which indexing is performed is not a key field and is also not a ordering field then we have to use secondary index of the non key variety that means we have to deal with duplicates in one of three different fashions either use duplicate index entries or use variable length records or use the most commonly used technique of extra indirection levels in order to handle duplicate address.

(Refer Slide Time: 54:48)



Properties of Indexes	Number of (first-level) index entries	Dense or non-dense
Primary	Number of blocks in data file	Non-dense
Clustering	Number of distinct index field values	Non-dense
Secondary (key)	Number of records in data file	Dense
Secondary (non-key)	Number of records or number of distinct field values	Dense or non-dense

And quickly what are some of the properties of the different index structures that we saw? The primary index, what are the number of index entries. The number of index entries for a primary index is the number of disk blocks in the primary file or in the data file. It is a non-dense index or a sparse index. A clustering index is also a sparse index which stores the number of distinct index field values that is the number of distinct values that appear in the index file. And a secondary index on a key attribute contains as many number of index records as there are number of records in the data file itself, it is a dense index.

And a secondary index on a non-key field may or may not be dense, may be either dense or sparse depending on whether the non key field is unique or not that is the repetition in the non key field or not. And the number of records contained in the index file is equal to the number of distinct values that appear in the indexing attribute. So this brings us to the end of this session where we saw different kinds of single level indexes. In the next session we shall be looking at multi level indexes where index structure can have several different auxiliary files.