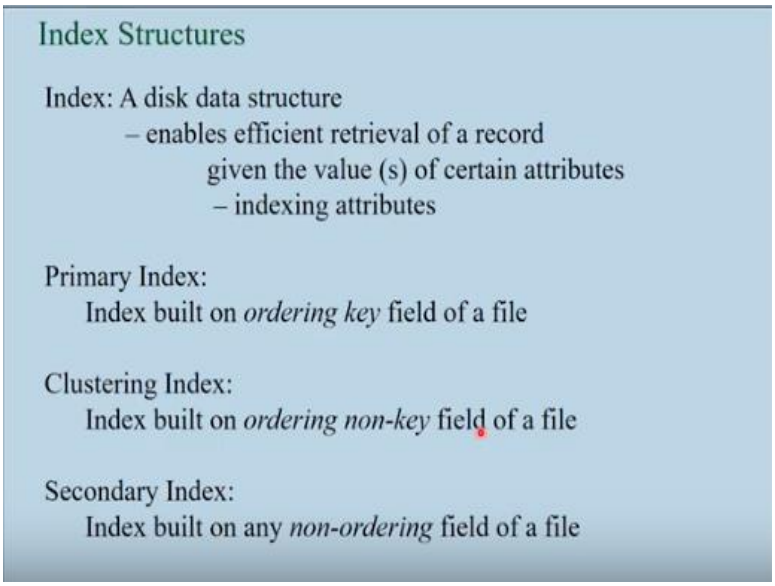
Introduction to Database Systems
Prof. Sreenivasa Kumar
CS & E Department
Indian Institute of Technology-Madras

Lecture - 28
Index Structures

Right. So in the last few lectures we have seen the structure of disk files, right. And then we have also seen what are the primary, possible primary organizations for the files.

(Refer Slide Time: 00:35)



And continuing our discussion on how to handle data on secondary storage, stable storage. In this lecture we will focus on what are called index structures okay. So basically in index is a disk data structure which will enable very efficient retrieval of a record given the value of a certain field on which we have constructed that index. So those attributes on for on which we construct this index are usually called the indexing attributes, okay.

So given the value of these indexing attributes, this data structure will help us retrieval the records in the data file that have values matching the given values. There may be multiple records, there may be one record, etc., okay. So essentially how to get hold of those records, given these values for the indexing attributes. And what are the various ways ideas that are available for us to build these indexes, what kind of indexes are there?

How do we build indexes? In fact, you can actually use the idea of hashing also to kind of construct indexes, okay. So In this lecture, we will basically focus on you know index structures where we are going to put the physical address, physical address of the data block in which the record is present in the index, in the index. So when you consult the index, the index will finally give you a disk block address or a few disk block addresses where you will find those records, okay.

So directly you can go to those disks, disk blocks and then pick up your records. It is also possible for you to kind of organize this in such a way that finally the index actually will give you some something like the key of the you know record so that you can go to the you can exploit the primary organization of the data file on which you know whatever it is the primary organization and then get hold of the actual records, okay.

So that is also possible. So there are, so where I am going to discuss a few ideas, and I expect you to kind of mix and match these ideas, right? Okay, so let us look at what kind of indexes are available. So we call something as a primary index if the index is built on the ordering key field of the file. That means a file is actually an ordered file. It is an ordered file. And the ordering field actually is also a key, right?

And if you build an index on that, then such an index is usually called as a primary index. Now a clustering index is an index built on ordering non-key field of a file. The field is still an ordering field. That means the data is available in the ascending order or descending order of the particular attributes values. But it turns out that it is not accurate key, right. So that is one possibility.

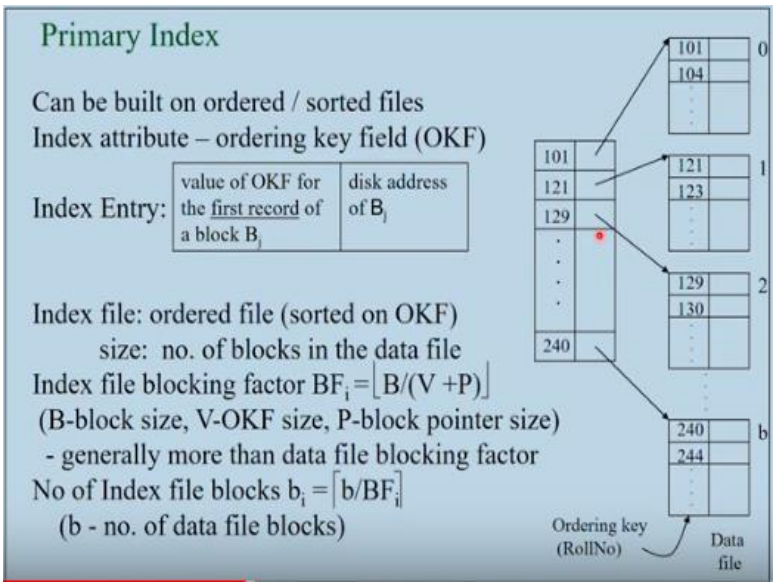
The other one is called the secondary index where the index is actually built on some arbitrary field. See if the data file is ordered on a particular field right, all the other fields are non-ordering fields, right. So if you want to build a index on one of these non-ordering fields, so you will such a kind of index is what is called a secondary index. Now, so mostly we are actually focusing on ordered files, ordered files.

So for an ordered file, you can only have one primary index because there is only one ordering field, right. So one primary index is can be built. And if that ordering field happens to be not a key field, then you will build one clustering index. So either a primary index or a clustering index you will build. But on the same file, you can actually build several secondary indexes.

If you want faster access to the data records on fields, which are not the ordering field, then you can actually build several for each of those fields, you can build a secondary index, okay. So these, how do we build these indexes and where do we store them? Obviously, these are all disk files again. So in index will again generate a file. So that file has to be linked to the data file.

And then the availability of the what is the primary index for the file? And how many secondary indexes are available? If there are there where are there, all that information has to be there in the header of the data file, okay. So let us look at these areas. These are very interesting ideas and they will definitely help us get the, you know very fast access to the index to the data records, okay. Let us start off with the primary index.

(Refer Slide Time: 06:36)



So the primary index is a very simple idea. We have a data file versus this is the data file, right. So the data file has records and let us assume that this is the field on which we want to build the index. So this is an ordered file. We can see that these are the individual records. These individual records are in the increasing order of this

particular value. Let us assume that it is something like a roll number. And it is a key also.

So there are no two records with the same values, okay. So but each of these, each of these things is a block, you know this thing is a block. So it will have a bunch of records, okay. So we have a sequence of these blocks. And what we do here in order to build a primary index is keep the value of the first block and a pointer to the first block as the first entry in the index.

Keep the value of the first record of the second block and a pointer to that in the second entry and so on like that. So in general the index these are called index entries and this is the index file. So the index file will have value of the ordering key field for the first record of the block B j, okay and then the disk address of the block B j. Now these first records of each of the blocks are actually called block anchors.

You can call them as they are called as block anchors. They are the first especially when the data file is a ordered file we call them as block anchors. So for the block anchors, we have the values and the disk pointer to the block. So that will constitute the index, okay. Now let us look at the index file. The index file, this index file obviously itself is an ordered file. It is ordered on this values.

It is sorted on this ordering key field values. And what will be its size? The size will be the number of blocks that are there in the data file. So compared to the data file, the index file will be small, because it has one entry per block of the data file. And remember the block has huge number of records and the maximum number of records that is blocked in whole is called the blocking factor, the blocking factor.

So now you can calculate this index file itself has to be on the disk. So it has to be accommodated in a few blocks. So now we can calculate the index file blocking factor or the blocking factor for the index file. So blocking factor for the index file is the block size divided by the length of the record. Because that is the maximum number of records that you can hold in a block.

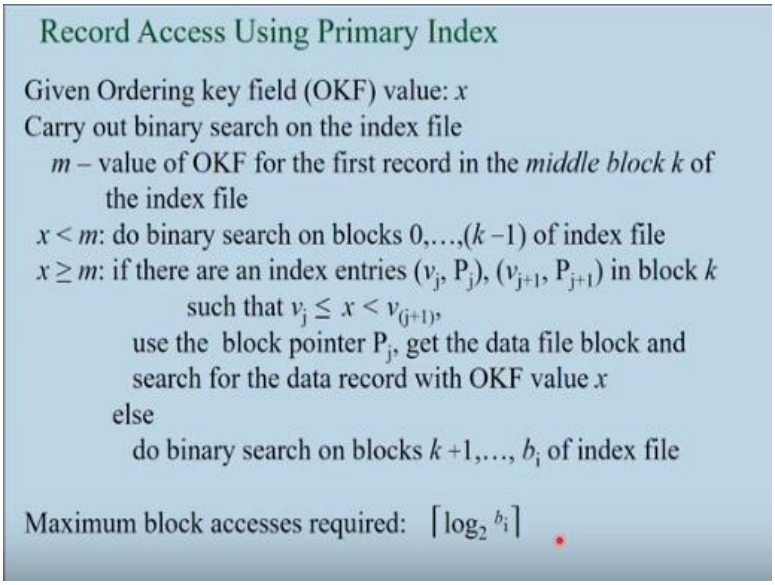
The blocking factor is nothing but the maximum number of records that you can hold in a block. So now the records are of length V + P, V is the length of the value. I mean ordering key field size and P is a block pointer, whatever is the block pointer size. It is usually about 6 to 8 bytes. So block pointer size. So B divided by V + P will give us the blocking factor for the index file.

Now this is the number of index entries that can be held in a particular block. The block size is same because we are using the same disk. The block size is same. Usually this blocking factor for the index file will be much larger because the index entry is much smaller compared with data record. Data records are usually long. The index entries are small. So the blocking factor for the index file will be high.

And the number of blocks that the index occupies will be low, you get it? So generally, the blocking factor is much higher for the data than the data file blocking factor. And now you can calculate what is the number of index file blocks that are needed. So let us call this b i the and let us call b as the number of data file blocks. So these are the data file blocks 0, 1, 2, 3 actually you should probably number them as 1, 1, 3 up to b. So b is the number of data file blocks that we have.

b divided by the blocking factor BF i for the index file will give us how many number index file blocks. Now what is the advantage of this thing? If you want to have a access to a particular record given this ordering key field value, what you can do is to do a, you can do a binary search. You can do a binary search on this and then get hold of the record. Go to the block where that particular record would possibly be there and then fetch the record. So that is the advantage.

(Refer Slide Time: 12:37)



So record access using this primary index. Given the ordering key field value x versus some value x carry out a binary search on the index file, okay. So let m be the value of the ordering key field for the first record in the middle block k of the index file. So let us consider the cases now. So x is strictly less than m then obviously you know that this data record will not be there in the portion in the lower portion of the data block.

So you have to do a binary search on the blocks 0 through k - 1 of the index file, k is the middle block, k is the middle block. How do you get the middle block? Again the index file is a like a file. So it will have its own file headers and it will have the number of blocks that are present in the file and then block pointers for all of the blocks. Let us assume that, okay.

See in the index file header, you have what are the number of blocks and information about what are the number of blocks available and for each of the block where is the block, the block pointers are available. Okay, so you can find out what is the middle blocks thing and get the value of m or the first record in the middle block. So you have already done one block access here.

Then you do a binary search on 0 through k - 1 of the index file blocks. Otherwise, if x is either equal to m or greater than m, then you basically have got hold of this middle block k. So in that middle block you search for this entries v j and v j + 1 okay

two entries v j, v j + 1 that contain this x. So x is either less than or equal I mean between v j and v j + 1, strictly less than v j + 1 okay.

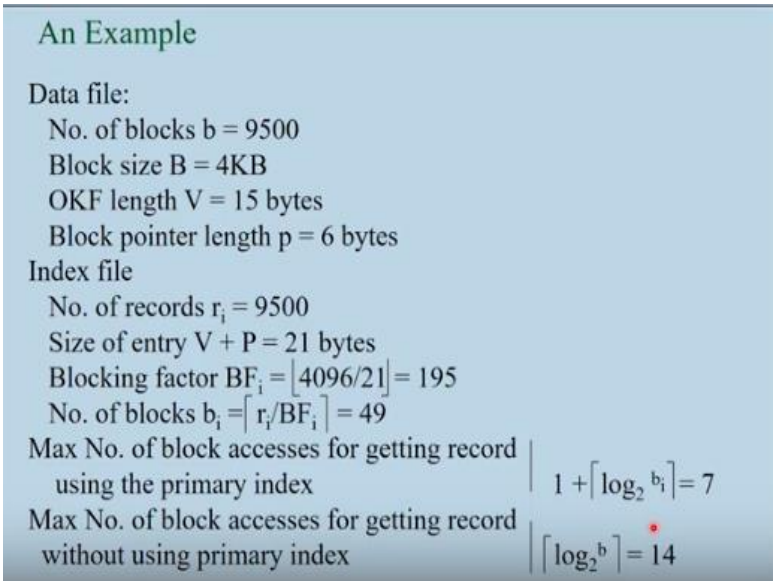
So if this is the case, then the block pointer P j will give you the data file block where this particular record is present, okay. So that is the so remember this block of the index file, block of the index file has several disk pointers, right? Remember, I will show you this picture again. So this has been blocked I mean it has been divided into blocks. So each of the blocks will have several disk pointers.

So that is why you need to search through here and then figure out between what two values is this particular value that you are searching for is present and then pick up the appropriate data file pointer. So use the block pointer P j to get the data file block and first search for the actual data record in that block, in the data block.

Otherwise, if x is not present here, if such index entries are not present, then you know that the record that you are looking for does not have a pointer in the middle block. So you go and do binary search in k + 1 through b i. So like this you can figure out actually where the data file block is that contains the record that has this value x. These is only one record, right.

So the maximum number of block accesses required will be log b i to the base 2 because you are doing binary search. So let us look at some sample values and get an idea of this this figures.

(Refer Slide Time: 16:36)



Let us say the data file has some 9500 blocks and the block size is about 4 kilobytes and the ordering key field length is about 15 bytes then block pointer is about 6 bytes, okay. So now the index file will have 9500 records. It will have exactly as many records as there are blocks in the data file. So 9500 records will be there. And the size of the entry will be 21 bytes, 15 + 6 21 bytes.

So now you can calculate the blocking factor for the index file that is 4 kilobytes divided by 21 equal to 195 index entries will fit into one block. Now you can see the actual number of blocks that are required will be 9500 divided by this 195. We will see that about 49 blocks will be required to hold the index file itself. And so basically, you are going to do search on this blocks.

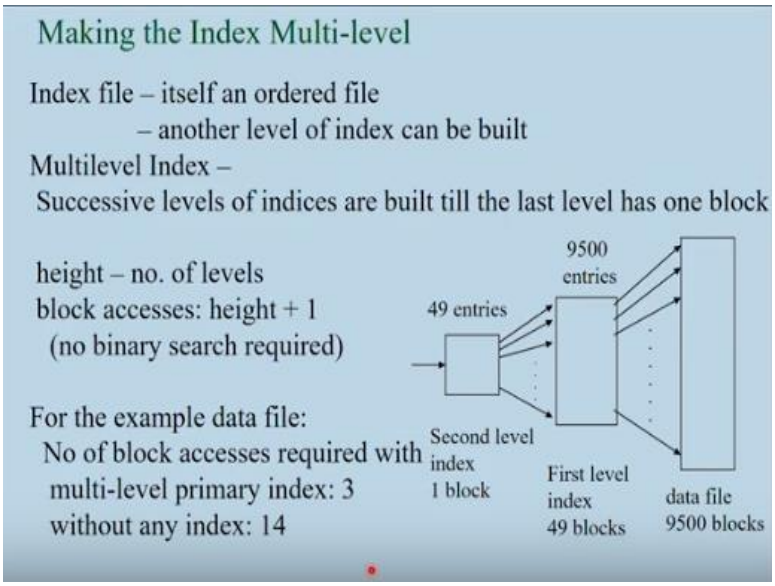
So the maximum number of block accesses for getting a record using the primary index will be log b i to the base 2 c + 1. Because this is the number of block accesses on the index file to get hold of the actual index entry that gives the block pointer. Then finally you have to access the data file using that pointer and then one more access is required. So if you do not have this primary index then the you can of course do the same binary search on the data file itself.

You can do binary search on the data file also, right. Because the data file itself is ordered, okay. So and it has 9500 records, so log b to the base 2 flow that will give you 14. So from 14 block access, you are able to reduce it to 7 block accesses. And then you will be able to get hold of your data record. Now this is what it is called

primary index. It is constructed for data files that are ordered and on the ordering field, okay?

If the ordering field is not a key, then similar thing can be done, we will call that as a clustering index.

(Refer Slide Time: 19:29)



Now here is a again another interesting idea. The index file is itself an ordered file anyway. So why do we not try and construct the index for the index file? Why do we not construct an index for that index file. So we have data file which has a 9500 blocks and we have first level index which has 49 blocks and 9500 entries. And this itself is a you can treat this as a data file which is ordered.

Now obviously, I can also construct an index for this. An index for this will have 49 entries, okay. So I can convert this into a multilevel index. So there can be a successive levels of indices built till the last level has one block. This is last level is this one. So now this 49 entries will be there because there are 49 blocks here in the index file. So there will be 49 index entries here and these 49 index will fit into one block because the blocking factor for the data file is 195.

You can fit a large number of records. So this is one block. Now for the search, you actually now do not need a binary search at all. I mean, disk binary search you do not need. You can get hold of this block first, okay. That will give you access to locate an

appropriate index entry given the value here. So you can of course do in memory binary search, but we do not count in memory binary search at all.

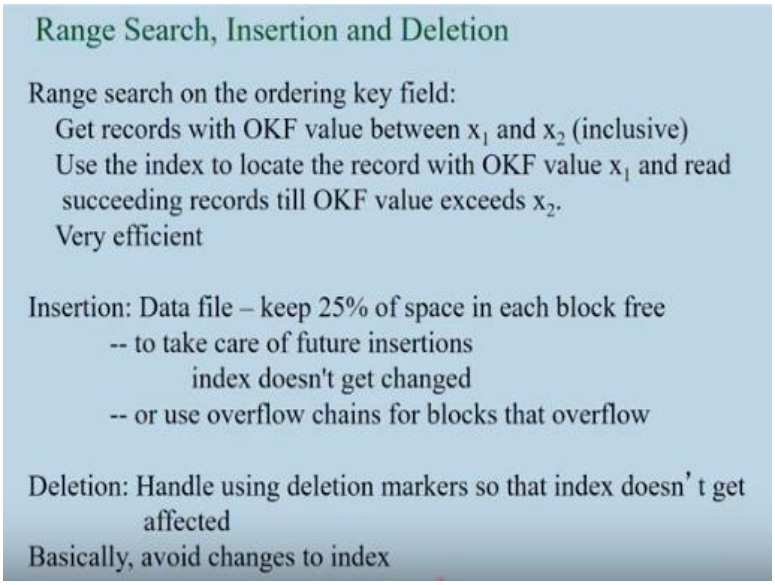
Our complexity is in terms of block accesses. So memory operations we will simply ignore. They are so fast that in compared to disk block accesses, we will not count them, okay. So you can get hold of this the index for this index file, and then do a search to record to locate an appropriate record here, okay. And then get hold of the data pointer and then go to the actual data file get hold of the record.

So now you can see, so for such a kind of a multilevel, multilevel index, we call the height as the number of levels. So this is one level first level, this is a second level. So the number of levels is 2 here, so we will call it as you know the height is 2 plus so yeah so height is 2. And the number of block access required is height plus 1 because height, number of block accesses are required to navigate through these index levels.

And then finally you will get a data file pointer. So plus one will be required, one more block access is required to get the actual data block. Okay, so the number of block accesses required with this primary, this kind of a index will now be just 3. We will reduce it from 7 to 3. So one here one here. And finally one here. So without index it is 14. Now we have brought it down to something like 3.

So three block accesses especially when you keep this particular 4 kilobytes of you know data in the memory, then you do not have. See if you are frequently accessing this data file, you might actually buffer this and then keep it in memory. So in 2 block accesses you usually get your record, which is not bad compared to 14 block accesses that we had to do earlier. Okay, so this is the idea of turning indexes into multilevel indexes.

(Refer Slide Time: 23:51)



Okay, now let us see how you can, you can do range search pretty efficiently now. On this data file, you can do range search pretty efficiently because all that you have to do is to say supposing you are to get hold of records ordering key field value between x 1 and x 2 inclusive, then you can locate the record which has value x 1 using the index and then keep on reading successive records from the data file till the ordering key field value exceeds x 2.

So it will be very efficient. It is as good as locating x 1. From a block access point of view, it is locating x 1 and then reading as many data file blocks as required. So that of course depends on the range that has been given. So range search will be very efficient. What about insertions and deletions? What about insertions and deletions?

So imagine that you have a data file and then you have constructed this first level index, and then constructed another index for that maybe in this case it is stopped with 2, maybe there is one more level, etc. So there is some h number of indexes. And then now somebody want to come and insert a few records into your data file. And it is an ordered file. And so there can be movement of records across blocks.

And if records move across blocks, then the whole of your structure goes for a toss, right? We are in great trouble if you have to insert new records into the data file. So if the data file is, you know relatively static, you do not have too many insertions to be performed in the data file. Then this idea of you know constructing indexes and then making them multilevel and all that will be very useful.

But in practice, there can be some insertions but how do you handle them? So what do we do? There are multiple ideas to take care of insertions. One idea is to keep some something like 24% of the space in each block free anticipating that there will be data records coming into the, into that particular block. The moment you have some free space in each block, then the block anchors do not change.

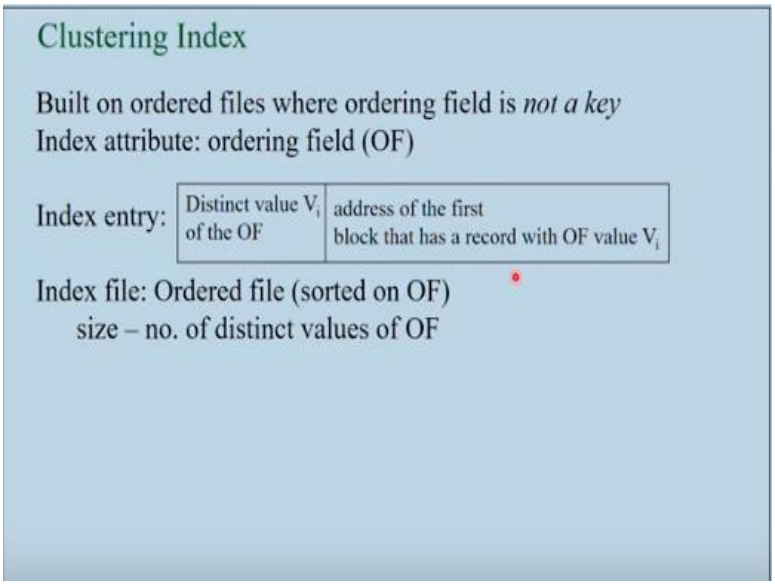
If a new record comes into the picture if it comes to the first version, the block anchor will change, otherwise you know nothing much happens. So it is not a major issue. So to take care of future insertions, you can keep some 25 to 30% of the block space empty or you have to use overflow chains for the blocks. Basically have some bunch of overflow blocks where you can hold this overflow records.

And then but essentially the idea is do not change the, do not change the ordering among the records of the data files. If you change them, then you have to change not only the first level index and then probably other indexes you will have to and deletion is always handled using deletion markers. So for each of the record you have a field, you create a field where you mark whether it is valid or invalid, in case it is deleted.

So you can turn that you know flag off on in case it is to be deleted. And essentially, we will if you are organizing data files like this we will try and avoid changes to the index. Okay but this is of course, this will, there are there would be files like you know which we will create say once in a year or something like that. The all the, you know students admitted into the university in that particular year, you create a data file.

And you keep accessing that all through the year or you know probably for 4, 5 years. So for such kind of files, this is a kind of a perfect situation where you can access them on say roll number and things like that. You want to have quick access to the records.

(Refer Slide Time: 28:39)



Now the idea of a clustering index is similar. The clustering index is similar. Only thing is the clustering index is actually built on the ordering field, which is not a key. The ordering field is not a key. So in that case, the index attribute index entries will be like this distinct value of the ordering field and what is the address of the first block that has a record with this particular value V i.

So distinct value V i and what is the address of the first block that has a record with value V i. In fact the first record of V i because there may be multiple records having the same value V i, okay. So what is the address of the first block. So that is that will be the index entry. They of course index entries are small in size so the index will fit into fewer number of records, fewer number of blocks compared to the data file.

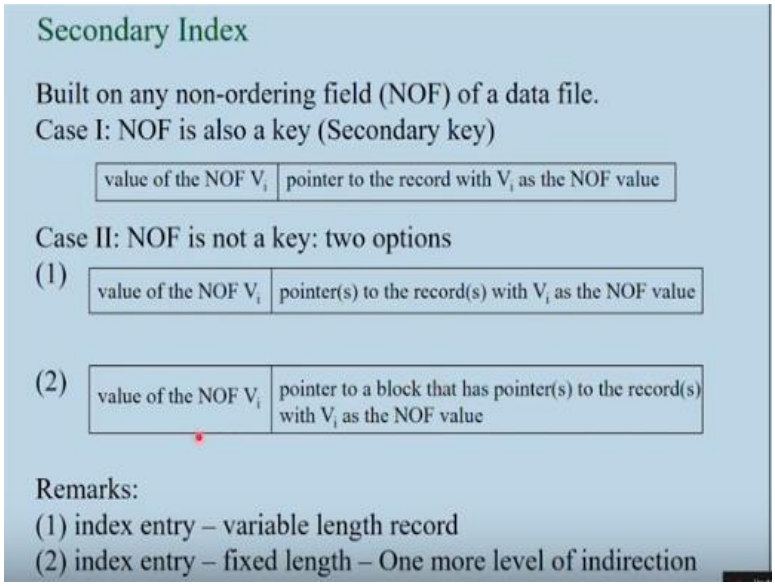
Now the index file itself of course is an ordered file and the number of entries in that is the number of distinct values of the ordering field. And you can actually construct even you can even convert this into also a multilevel index. Remember that the first level index entry first level index is an ordered file and this ordering field is accurately fill in that file because we have taken distinct values of V i.

So from the first level onwards it becomes the field on which we are searching, in fact is a key. Only for the data file it is not key, okay. So if you are constructing converting a clustering index into a multilevel index, you can again for the second level onwards you can again exploit block anchors and then have entries only for the first records of the blocks in the subsequent levels of the index, okay.

So it is possible that we can convert the clustering index also into a entire multilevel index and then benefit from the fast access to the data file, a data block that contains the record whose value is being given. In fact in this case, there will be a bunch of records, which you will have to retrieve from that particular block or the consecutive block.

It is possible that a certain value you know in the ordering starts at one block and then there will be a few more few records in the subsequent block also in which case you start from that data block and then access successive blocks in order to get hold of all the records, okay. So is this idea of clustering index clear? It is very similar to the primary index except that we have the index entries will have distinct values of the ordering field, okay.

(Refer Slide Time: 32:44)



Now let us move on to secondary indexes. The secondary indexes are actually very useful. If you do not have secondary index for certain data files you know there will see if you do not have primary index you can still do binary search on the ordered file and then get hold of the record in 14 block access for example in that example that we have seen, okay.

But supposing you have been given the value of some field that is not the ordering field of that particular data file. Then what else what is the option you have? You have to basically do a file scan where files can read all the data blocks of that

particular data file and get hold of the records that have these matching values. So the alternate option is so let us see in that situation, what can we do?

So this index is built on some field, which is the obviously it is a non-ordering field of a data file. Okay, if you take a non-ordering field there are again two possibilities here. That field could be a key, could be a key. For example, in a data file, you have ordered the data file of student records on roll number values, okay. And somewhere there, there is a field which says it is a Aadhaar number of the student.

Aadhaar number obviously is a key, okay. But you have ordered the data file on increasing values of the roll number. And so the data records will not be there in the increasing order of the Aadhaar number obviously, okay. So the Aadhaar number is a secondary key. We call them as secondary key. Remember, when we are defining keys, we call them one of them is chosen as the primary key.

The reason why we choose a primary key is that we will organize the primary index for the data file using that key. And the other one is called the secondary key. So now we are looking at how to construct the secondary indexes, okay. So that non-ordering field could be a key in which case we would have called them as secondary keys.

And so the index entry in this case will be the value of that particular non-ordering field V i and the pointer to the record with V i as the NOF value. Take Aadhaar number for example, Aadhaar number here pointed to the student record with that Aadhaar number. It will be exactly one. Now here we are first time we are using pointers to the records ordering the pointer to the record.

We have been always having block pointers, okay. A pointer to the record is nothing but a block pointer plus some offset value, which will give you what is the position of that particular record within the block that is all. So it is some small couple of bytes you know which will give you an integer telling you the position of the record in your in the block.

So pointer of the so you can so that is how the index entries for the for this case, if the non-ordering field is also a key, then we will have this kind of index entries. Now the

other case is that the field on which we are constructing this index is not a key, okay. Like for example, you want to have fast access of the names of the students. Names typically are not keys.

There may be students which have same name values. So in this case, we have actually a couple of options as to how to organize the index. First thing is we can organize the index entry like this. The value of that particular field. And then pointers to the records which have that value. There will be multiple records which have that particular value, collect together all those pointers, and then put them in this record itself.

Now the moment you do this, this particular record, actually is not a fixed length record. Because you do not know how many number of records will have this value V i. And so how many pointers are going to be there in this field you do not know, right? So in some index entries there may be 5 pointers. In some index entries they may be 10 pointers, okay. So you will get hold, you will see that this particular record is a variable length record, okay.

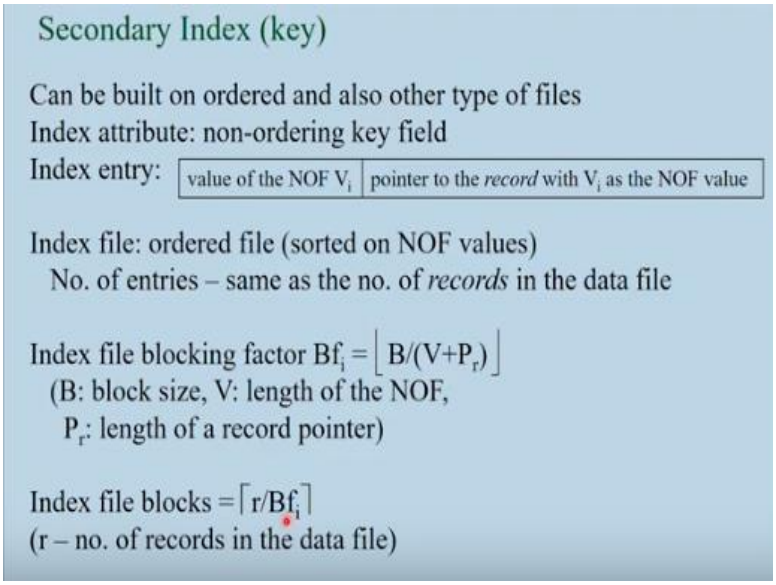
The other option is do one more level of index indirection. What you do is to kind of make this index entry as a fixed length record. What you can do is to keep the first field as it is. In the second field instead of putting all these pointers, put a pointer to a block and in that block keep all these pointers, okay. You have a bunch of pointers 5 or 10 or whatever, record pointers.

So put all of them in one block, put all of them in one block and put a pointer to that block here. So pointed so a block that has pointers to the records which have this particular value, okay. The advantage of this option two is that the index entry becomes fixed length. So this is usually a good option because if you have fixed length records it is easier to handle them, okay.

But the downside of this is that there is one more block access that is required because you will only get a pointer to the block that has the pointers to the data records. So you have to first access that block and then you will get hold of your pointers to the

actual data records, okay. Is that clear? Is there some issue with this? Okay, so these are the two options that we have while constructing secondary indexes.

(Refer Slide Time: 39:59)

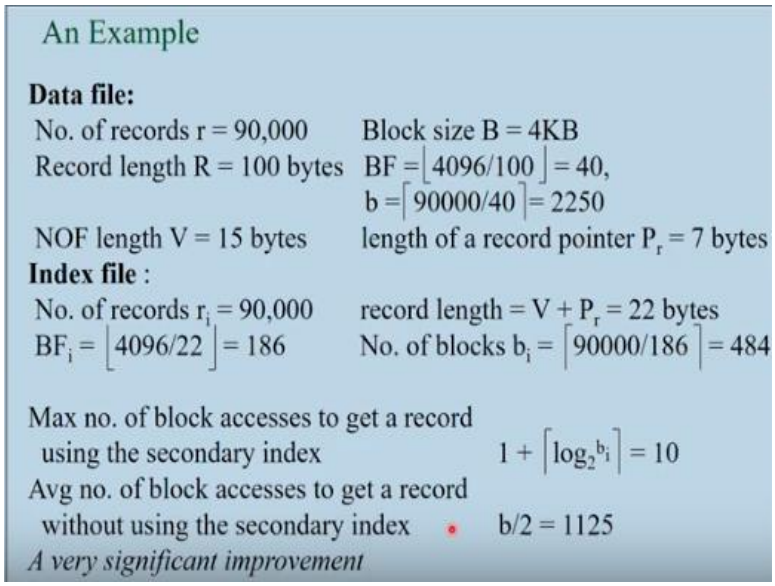


Now let us look at the in detail secondary index with a key option, the secondary index, the field on which we are building the index also happens to be key like the Aadhaar number for the students. Okay, so in this case, the index entry will be the value of that particular field and the pointer to the record. Now let us consider what is the index file. The index file number of entries will be the number of records in the data file.

Remember this, this is different from the primary index because in the primary index, we had number of entries in the index as the number of blocks in the data file. Whereas now the number of entries in the index file will be equal to the number of records in the data file. Because there are as many so many Aadhaar numbers. For each of the Aadhaar numbers you need a pointer. You know that fellow is lying somewhere in your data file, okay.

So number of entries is same as the number of records in the data file. So the number of index entries have increased now by a factor of blocking factor, right. The number of index entries are huge. So the index file blocking factor is as usual, you know calculated like this B divided by V + P r where P r is the length of the record pointer. And the number of index file blocks that are required will be number of data file records divided by the blocking factor, okay.

(Refer Slide Time: 41:50)



So let us look at some example scenario. Let us say we have 90,000 students, 90,000 students each of them some 100 bytes of information we are storing for each of these people. And the Aadhaar number is about 15 bytes. The block size for the disk is 4 kilobytes and the blocking factor for the data file, blocking factor for the data file is the block size divided by the record.

And the number of blocks in the data file is about 90,000 divided by 40. Because that is how much backing you can do. So 2250 blocks and let us say 7 bytes is the record pointer. Now you construct a index for this. So the index file will have 90,000 records, right. Index file will have 90,000 records, exactly as many records as there are there. And this is Aadhaar number plus the record pointer of 22 bytes.

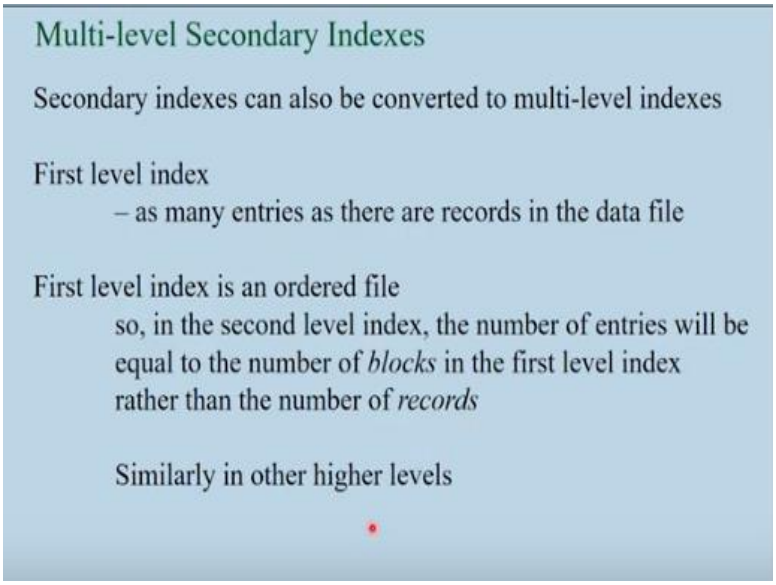
And now the blocking factor for the index file is about the 4 kilobytes divided by this size 22. It is about 186. So from 40 it has come to 186. 40 is the blocking factor for the data file and 86 is the blocking factor for the index file. So the number of blocks here will be this this 90,000 entries divided by 186, 186 per block. So you get about 484 blocks for the index file.

Okay, now what will be the maximum number of block access to get a record using the secondary index? You can actually now take the index file and do binary search on this, binary search on this. So if you do binary search on this block, log b log b i to

the base 2 plus 1 will give you the record. So about 10 block accesses you are able to get your student record given the Aadhaar number, okay.

Of course, if you do not have this index, then you basically have to do a file scan. You basically have to do a file scan. So if you are lucky, halfway down, you will get hold of your record, which is very bad, like 11 so half of the number of block access, okay. So secondary index is very, very useful. Now the same idea of converting the index into a multilevel index can be done here too.

(Refer Slide Time: 45:06)



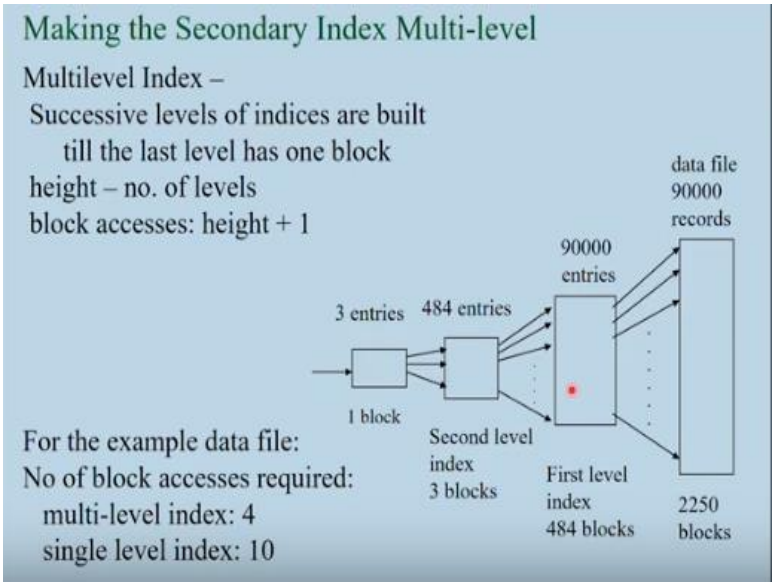
So we can convert this secondary index into a multilevel index, okay. So the first level index will have as many entries as there are records in the data file. Now that index, the first level index is an ordered file is an ordered file. So in the second index, the number of entries will actually be equal to the number of blocks in the first level index, okay. It is as good as constructing the primary index for the for this index.

What is this? So let us take the student case, student file case. This first level index is actually ordered by Aadhaar numbers along with the pointers to the student records, okay. So Aadhaar numbers pointers to the student records. Now obviously, we can order this file as on Aadhaar numbers and it becomes an ordered file where the and the ordering field is a key.

So now we can actually construct a something like a primary index for this particular field. So you can now use block anchors. So the number of entries into the second

level number of entries in the second level index will actually be equal to the number of blocks in the first level index. You get that idea? Rather than the number of records, even though there are 90,000 students.

(Refer Slide Time: 46:55)



So let us look at that. For example the data file has 90,000 students and 2250 blocks. And the first level index has 90,000 entries because you are getting each Aadhaar number and the pointer to the student, not necessarily in this order. They will actually be jumbled up right, okay. Now this one is an ordered file. Because this is an ordered file in the second level index we will have okay 484 entries.

Because this is occupying 484 blocks, it is occupying 484 blocks. So there will be 484 entries in the second level index occupying just 3 blocks. You can calculate that later. And then this of course index to this per block chains. So if your successive levels of indexes are built then the block accesses will be height plus 1. So in this case actually you can get in 4 block accesses the students record.

So you 1, 2, 3 and then you had got hold of the student record, okay? So that is pretty fast because that is approximately the same as the kind of access that we had on roll number right. So this is very good to have this kind of an index and convert that into. So of course, the same issues of insertions deletions will come into picture, so handle them in a similar way here also.

So the those files, which are actually ordered files with multilevel primary or clustering index, are actually in the industry called ISAM files. They are called Index Sequential Access Method Files, okay. So insertions are handled using overflow chains on this data file blocks. And deletions are handled using deletion markers. So if the files are relatively static, this will be most useful.

And if the files are actually dynamic, then we need to go for dynamic multilevel index structures. And those will be based on B plus trees actually. So in the next lecture, I will be talking about the B trees. B trees you would have studied in data structures course. Now we take that essential idea and then convert that into a disk data structure which will be able to handle insertions, deletions and then we will have an index for a dynamic file using B trees. Okay, now let me stop here for today. Thank you.