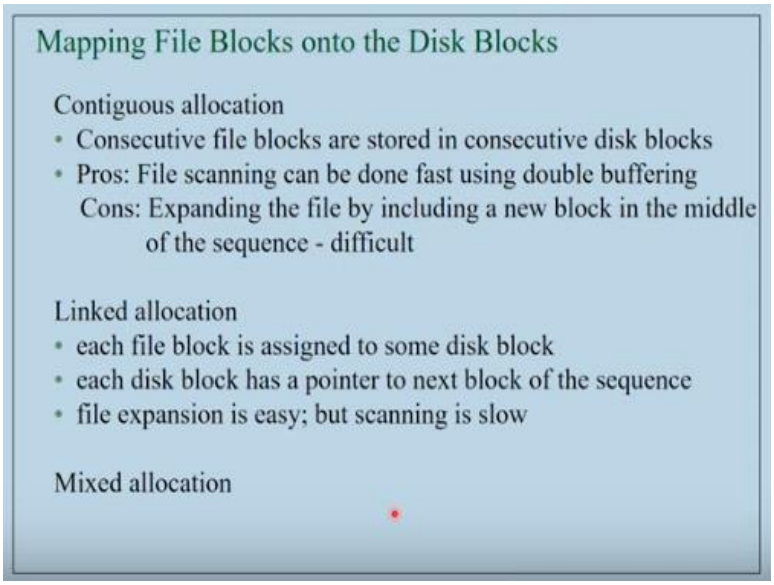
Introduction to Database Systems
Prof. Sreenivasa Kumar
CS & E Department
Indian Institute of Technology-Madras

Lecture - 26
File Organization Methods

Yeah, so let us just start it off. So in the last lecture, I was talking to you about the structure of disks.

(Refer Slide Time: 00:27)



The standard you know option for storing data in a non-volatile and basically what we observe is that disks will offer a good option where we have random access to the blocks. A block is a unit of transfer, data transfer from the disk main memory. And these are random access devices. So if you give a block address and the disk controller will be able to give you a block of data, which is usually something like half a kilobyte to about 8 kilobytes of data into memory buffers.

Now the we also seen that the amount of access time that is required to access this block involves the so called seek time where you have to position the read/write head onto the cylinder from which you are reading the block. And then the rotational delay for reading off a particular block from the surface when it is spinning, okay. So we have seen all this.

Now what we are doing is to see that if when there is a file, the file has records and we need to now store this file onto the disk. So what we will do is to pack these records into block units, and then map those, then we will call we will get one sequence of you know record units. So we call them as file blocks. And then these file blocks need to be mapped to the disk.

And so we were looking at what are the mapping policies that are possible here. So one of them is to do what is called contiguous allocation, where we will try where we will map the consecutive file blocks on to consecutive disk blocks, consecutive file blocks onto the consecutive disk blocks. Now the advantage of this is that if you want to do a scanning of the file, it can be done very fast because there the file blocks are all on the consecutive disk blocks.

So we have seen the notion of consecutivity. Basically consecutive means from an access time point of view, these blocks that are there on a certain one particular cylinder are said to be close to each other. And then the next cylinder etc., next adjacent cylinder.

Now but the downside of this kind of a contiguous allocation is that when you are trying to include a new block, into a file block actually into a file data, then that file block has to go into a since our policy of you know storing consecutive file blocks and consecutive disk blocks somewhere in the middle that new block has to be inserted.

So if you have to insert a new block somewhere inside, you know in between several other blocks on the disk, then you know if you are roughly trying to insert it in the middle, then it involves reading off half the blocks of the data file and writing them back which is a very costly affair, right. So first of all reading itself is yeah so and then reading so many blocks is a lot of cost, okay.

Then the other and the other extreme spectrum, we have this link allocation where each of your file blocks is actually stored on some random disk block. And then in order to find out where the next block is you know of next block of the file block is

lying, from each of those disk blocks we will have a pointer to the next block where the next file block can be found, okay.

So that is what is called each field block is, file block is assigned to some disk block and from each of these disk blocks, we have a pointer to the next block that contains the next you know file block in the sequence. Of course, the file expansion will be very easy in this case right? How, so all that you have to do is to actually do standard linked list insertion.

So in some sense, what is happening here is that the file block, which is a array of, you know blocks is actually you know made into a linked list on the disk, okay. There is a singly linked list on the disk. So that the pointer is the disk pointer. So that a disk pointer is nothing but the disk address. You know the disk address has surface number, cylinder number and block number.

So this and these numbers are so it takes about 10 bytes or something that is the disk address. So this disk address will be stored in each of these file blocks so that you will be able to find out what is there at the next file block lying on the disk. Now the expansion of course, is very easy because you know that inserting a new element in the linked list is easy, right. You can do pointer manipulation and then insert the block.

Whereas, but the downside of this is that if you have to access the first hundred blocks of a file for some reason then you know it will involve chasing hundred disk pointers, right? So each of them might potentially include again seek time rotational delay. If your file blocks are consecutive disk blocks, then you would hope that the disk accesses have less amount of seek time involved because they are on the same cylinder, right.

So this scanning can become pretty costly operation. So you can also of course, think about some kind of a mixed allocation where we say that okay, I will take first 25 blocks of the file sequence and then put them on consecutive blocks. And then the next 25 I will put in some other conservative place and put a pointer from the first 25 to the next 25.

You can take a policy like that next allocation where you use contiguous chunk of disk blocks for some part of the file and use again, put a pointer and so then before (()) (8:22) when you are trying to insert a new file block then it is kind of easier. You do not have to read and write lots of the disk stuff. You only have to read the disk blocks that are involved in that particular segment in this new block, okay.

So mixed allocation is also possible. Now this is how the sequence of records, in our course we have, the files that we are dealing with are essentially sequences of records, right. So these records of course can be need not be directly the tuples of the relations that we are talking about. They could be something, you know which may have some extra information or they could be longer than, these records could contain multiple you know information about multiple tuples, it is possible.

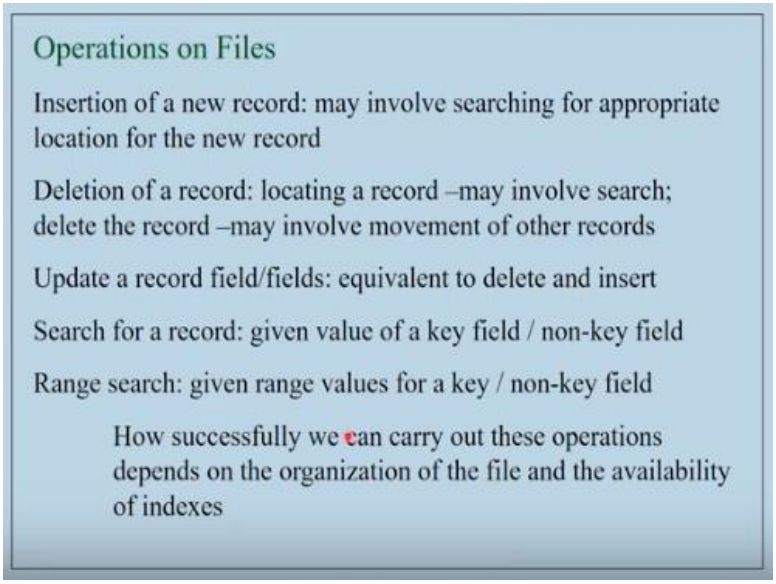
But so you remember that we long time back we discussed what is called physical data independence. So the ability to kind of change the organization without affecting the logical view of the disk. Now so all this would be possible if we have appropriate mapping information as to how those relations are realized on actual files of records, okay. So that information as to what is the file and you know whether the file has what is the information about that file with regards its configuration.

Is it one file or is it two files. Is this tuple sequence. So all that information will be there in a in some sense of you know metadata kind of store and so we have the concern that before we can actually make use of disk files for actual data access. So first the file headers will be returned, and then you get this information about what is there in the file and then whether there are what we call access structures additional.

So we are going to discuss what are called disk data structures. We have seen normal data structures, memory data structures and data structures. Now we are going to see disk data structures. And so these data structures can completely lie in the disk and so you need to know as to whether there is a available data structure or faster access for the cause of this file and then make use of it and all that will come into picture.

Now the first thing is let us look at what are the various operations that we may have to do on files from the database perspective.

(Refer Slide Time: 11:40)



Obviously, inserting a new record, if there is a bunch of already a bunch of records are there, you want to insert a new record into the file. Now depending on how the particular file is organized, which we are going to discuss in a short while now, this may involve actually an appropriate position to involve to appropriate location where this new record has to go into the slot, okay. So we will see.

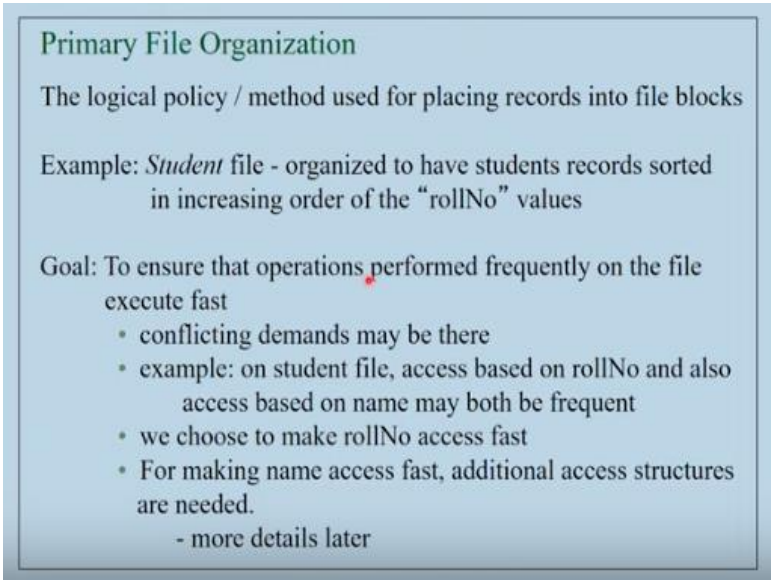
So this will become clear when we discuss what are called file organizations, how are files organized. Deletion of a record obviously. So deletion of a record involves first locating the record, where is it exactly, which block. So that will might involve some kind of search and then delete that particular record from that block. It may involve movement of other records in case there is a policy over they can take this allocation or something like that. Updating.

Updating a record field or fields is in some sense equivalent to deleting the old record and inserting it. And then search for a record. So given some value of the aerobic key field or the non key field etc. Given values of some of these fields, you have to locate either a record or records that have the specific value for those fields. This is a very often required operation on files. And then range search.

So we may get two values for the aerobic key field or the non key field and then we are requested to get all those records that have values between these two minimum and maximum values that are being given the range has been given. So let us call it a range search. Now how successfully we will be able to actually carry out these operations it crucially actually depend on something called the organization of a file.

And also whether there are any indexes available for the file. Indexes or additional access structures that we are going to discuss whether any indexes are available, and what is the organization of the file. How are records actually placed inside the file. So now depending on how the records are placed inside the file, some of these operations actually might be easy, some of them might be tough, etc. So we will see examples of that, okay.

(Refer Slide Time: 15:02)



So that brings us to what we call as primary file organization. The primary file organization is basically some kind of a logical policy that we use and method that we use for placing records into file blocks. How does it matter? Let us look at for example a student file. We might want to organize this file as a sorted file in increasing order of the roll number values.

We know that roll number is a key for the student records. So we might take a logical policy here of organizing these records of the students in increasing order of the roll number values. So this is an example of file organization. So this is a sorted file because they are in the increasing order of the roll number values. In general, the goal

of a file organization is essentially that how do we ensure that the operations performed frequently on this particular file execute fast, okay.

So as you can see for example in this case, suppose we have organized a file like this. Then there may be a so roll number based access will be pretty fast because we can exploit the sorted nature of the file and then get hold of the record that has a specific roll number or given roll number, we will be able to quickly get that. What if, you know you ask a, you give a name and ask for a record that matches this particular name of the student.

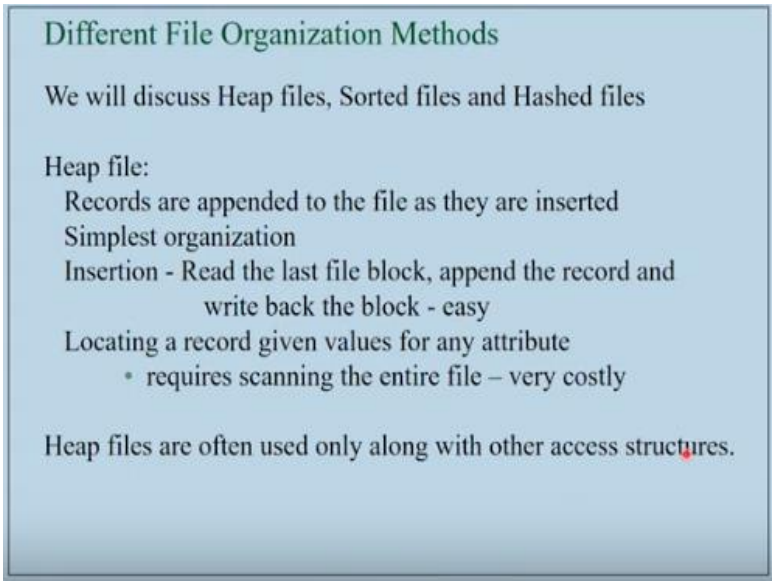
So we do not have any other, we do not have any means other than scanning the file. And then you know start from the first data record and then keep on searching for this particular name, and then find out all those student files that have matching values in this name. So if you now say that okay my access to the student records by roll number and by name is more frequent, and I would like this organization to be in such a way that you know both of them actually are pretty fast.

So in some sense it is a if you organize it by roll number values, and this is a conflicting kind of a data access, right. So if you choose to make roll number access fast, then the access based on name might be slow. So what we do typically is to first choose some one primary file organization using which we can make one of these access as fast like for example roll number and then to make the other access fast say name access, we will construct a additional access structure okay.

So we will basically make use we will construct a separate data structure that will take this name using which we can actually use the name and then search through that structure and then get hold of the roll number or get hold of the pointer to the particular block in which the set of students are, okay. So those things are what are called index structures and we will see the details later.

So let us first look at what are the various you know options we have for file organizations, okay. So this is the let us look at the options that we have.

(Refer Slide Time: 19:16)



So we will discuss what are called the heap files and sorted files and hashed files, okay. Heap file is actually a file that does not really have much of an organization. It is like records are appended to the file as they are inserted. This is you know just like append only block file. So this is the simplest of organizations. You do not have to think anything about it.

You get a new record, go and add it at the end of the list of records that we actually have. That means you can keep a pointer to the last file block where it is in the list and then read that file block, add this new record and write it back, okay. So inserting a new record is done pretty quickly. You have to basically read the last file block, append this record and write back the block.

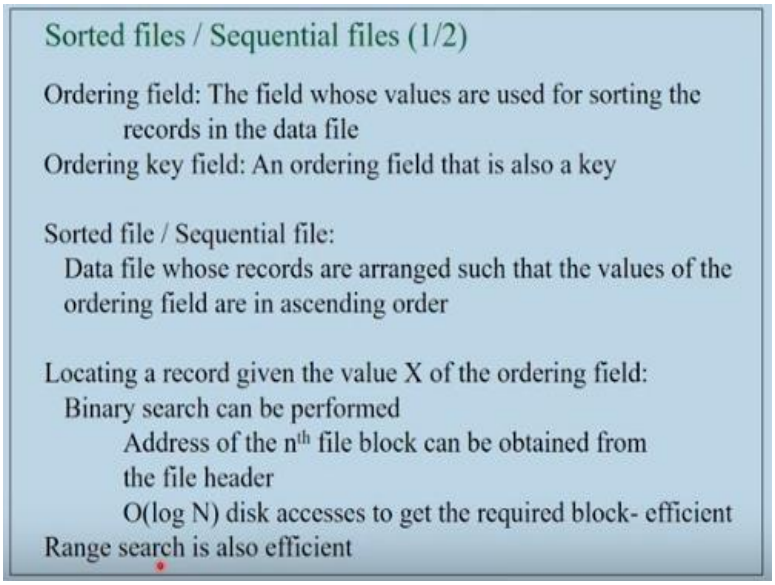
But then locating a record given some values of any of these attributes would be a nightmare, because basically you just have to scan the entire file. You do not have any other means of getting hold of these records. So you have to scan the entire file. Now this kind of files are useful for organizing something like you know our somewhere unless you know there are other access structures we rarely use this kind of a file organization, okay.

And this is typically kind of can be used in for example, organizing the log data of some system you know log in you know as and when operations are happening, you just want to keep a log of what happened there. So you generate a log record. And so

most of the time you may not actually use the log. So if you want to actually use the log then you can get hold of the thing and search for the record.

So you may be willing to pay the price for searching. But most of them we do not use this heap file organization unless we actually also build other access structures that will help us locate the file, okay.

(Refer Slide Time: 21:57)



So the next one is very interesting organization which is also called the sequential file or sorted file. So what we will assume here is that there is a field called the ordering field of the record that is chosen so that the records are sorted on that field, sorted on that field. So the field on whose values are used for sorting the records in the data file is what is called is ordering field and it could be in a key field or need not be a key field, okay.

So we call it ordering key field in case the ordering field is also a key. Otherwise, it is just called an ordinary field. So basically we store the file and the records are in the order of those ordering field values. And we call such as a file as a sorted file or a sequential file. Now what we can do here is on the given the value of this particular ordering field, locating a record is very somewhat easy, how do you do?

You know that you can do if you are a sorted array and then locating a value in that you will do binary search, right. So you could actually do some kind of binary search here. So locating a record given the value x of the ordering field you can do binary

search. So but here we are doing binary search on the disk file. So what is the assumption here? The assumption, important assumption here is the address, the disk address of the nth block nth block of the file can be obtained from the file header.

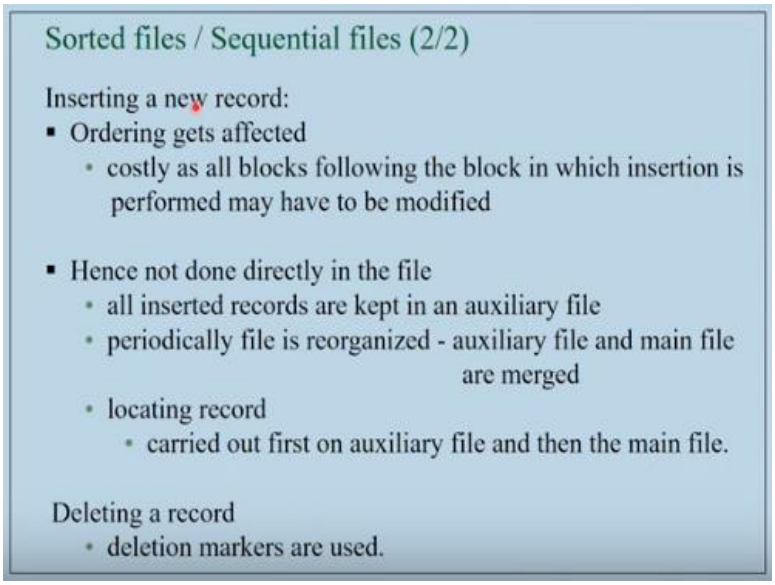
So the file header has all the disk addresses of all the blocks, okay. Then you can do binary search. So you can pick up if it has 1000 blocks, so you can take out the 500 blocks address and then you know you get 8 kilobytes of data or something like that. You get a block of data and then you look for this particular ordering record which has this value x.

And depending on the records that you get if you are getting records you know those values are much higher than X or much lower than X, you will have proper field link move. So decide whether you are go to the upper portion of the data file or the lower portion of the data file and then again ask for the make use of these file pointers, I am sorry, the disk pointers that are available in the file header, and then record of the next file block and then keep that in your file.

So what you do is about if there are N number of disk blocks, then about log N number of disk accesses would be required in order to get the block. So since log N is slowly rising function, so this is reasonably efficient. So we will later on see how to improve this further and range search is also efficient.

So given two values you can, v 1, v 2, you can locate v 1 first and then sequentially stamp the file till you exhaust all the records which are whose values are between v 1 and v 2.

(Refer Slide Time: 26:06)



So what about inserting a new record? Inserting a new record there is a problem because the ordering might get affected, right. You have already ordered the file on roll number and it is there on the disk. And if you are very unfortunate this new record that has to go will be in the very first block or something like first somewhere in the very first few blocks everything and then you know you have exceeded the number of records that can be kept in block.

So this you know the overflow block has to go into the next block and so on. All blocks will get affected, right. So that is messy, because if the file has stocking blocks then you have to put this following 100 blocks then all this remaining block you have to read and write back. So this is this can be pretty costly. One technique you can do, one technique you can adopt here is to, though is to actually not insert the data records into the data file at all.

What you do is you insert them into an auxiliary file, into an auxiliary file keep inserting the new records, whatever the records have come. And then the idea is of course after some time, so after several you know days of use of this file, then you can take some lean time off and then you know reorganize this file, insert all the records that are there in the auxiliary file into the main data file.

So you can merge the auxiliary file and the main data file and then get make the main file again into a sorted file. So if you are using an auxiliary file, then for locating a record, you first do a search on the auxiliary file first, because there are fewer number

of records there and they will probably be just fitting in a few blocks. So you will be able to quickly find out a new record you know the record that is being requested or is there in that.

If it is not there, then you go to the main data file and then do a standard search. Now inserting a new record we do it like this mainly because it might affect the ordered nature of the file. So is the case with the deleting actually. If we delete a file, delete a record request comes for deleting a record, it might affect the order, right? So what you do, of course you can just place a marker on the file on that particular record saying that that particular record actually has been deleted, without actually deleting okay.

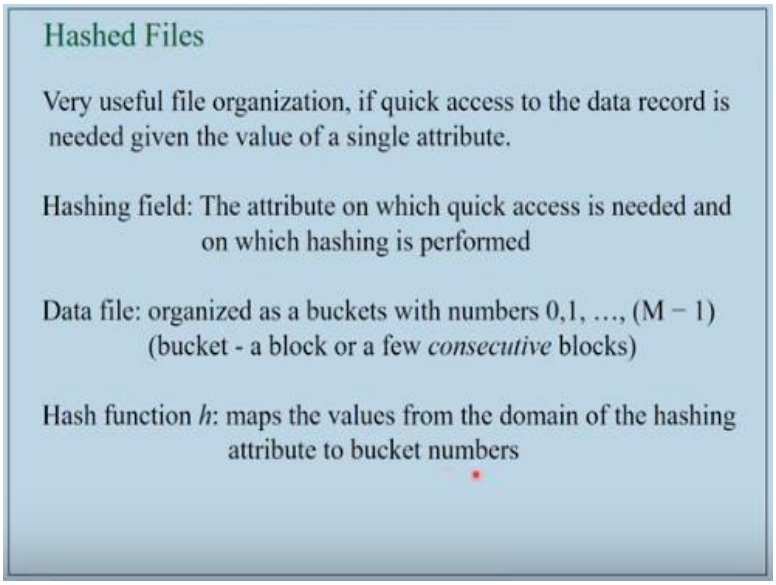
So there will be, in some sense an empty slot there. It is marked as deleted, but you actually do not delete it. So if you, the reason why you do not delete it is, is that if you delete and try to, you know make the blocks compact or something like that, then again there will be a lot of movement among the blocks which you want to avoid. So what you do is so both these techniques have to be used.

You use an auxiliary file to keep inserted records as the file is in our operation, use the auxiliary file to keep inserted records and use deletion markers to mark deletions. And then periodically we organize the whole scene and then insert the auxiliary file records into the main data file and ensure that all the marked up records are deleted and make the data file clean.

So you know how often you should reorganize will depend on how often the file is used, okay. So all that can be done. So on that particular ordering field, the access to the records is fast and we will sort it fast, okay. What about other fields? Other fields you basically have the files, okay. So what you typically do is on the other, if you we need access on the other files.

So you build an index, you build an index and then create a separate data structure using which you will then be able to find out a block address where the record is, okay.

(Refer Slide Time: 31:33)

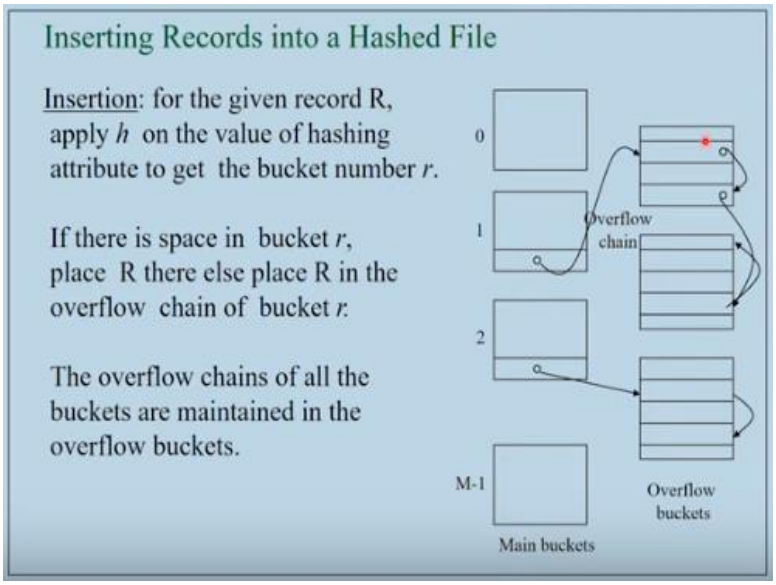


So the other thing, other primary organization is the hash file. You have studied hashing in data structure, so I am not going to repeat the details of hashing. But we will use the hashing idea and then use it for organizing the file. So this is a very useful file organization. If we really want a very quick access to data records given the value of one single attribute which is what we choose as a hashing attribute.

So the hashing field is the attribute on which we want quick access, okay and that is the attribute on which we perform hashing. So let us see how hash files are organized or how we use the idea of hashing to organize a file. So this is one of the primary organization methods, okay. So the data file is we use a term called bucket in this context. A bucket is either a block or a few consecutive blocks, two, three consecutive blocks is called as a bucket.

So the data file is organized in the form of buckets. So bucket numbers 0 through M - 1 are used, okay. And then we use a hash function. The hash function what the hash function does is map the values of the domain of the hashing attribute, some attribute were chosen. So the values of the domain in the domain of the hashing attribute will be hashed by this hashing function into bucket numbers. So you give the value and it will give a bucket number, okay.

(Refer Slide Time: 33:36)



So to insert to insert a new record into this kind of a hash file, what you do so this is the, this is how the main data buckets main data is laying here. And these are actually overflow buckets. So we will see what exactly overflow buckets are. So each of these, each of these main file buckets can potentially have a, an overflow chain, an overflow chain attached to it, okay.

An overflow chain will be like this. It is a pointer to one of the records, and that record has another pointer, etc. So that is an overflow chain alright. So why do we need overflow chains we will see that. So given some record, you have the value of the hashing field. So apply the hash function on this on that hashing attribute value and then you get a bucket number go there to that bucket, bucket number.

So this bucket numbers where these buckets are, this point is of all these buckets will be stored in the file header. So bucket number i you know the disk pointer, so you get hold of this bucket and then go and insert. So there are two possibilities here. This bucket might be full, okay. There is no guarantee that the hashing function actually distributes the records evenly, right? It might be skewed, you do not know, right.

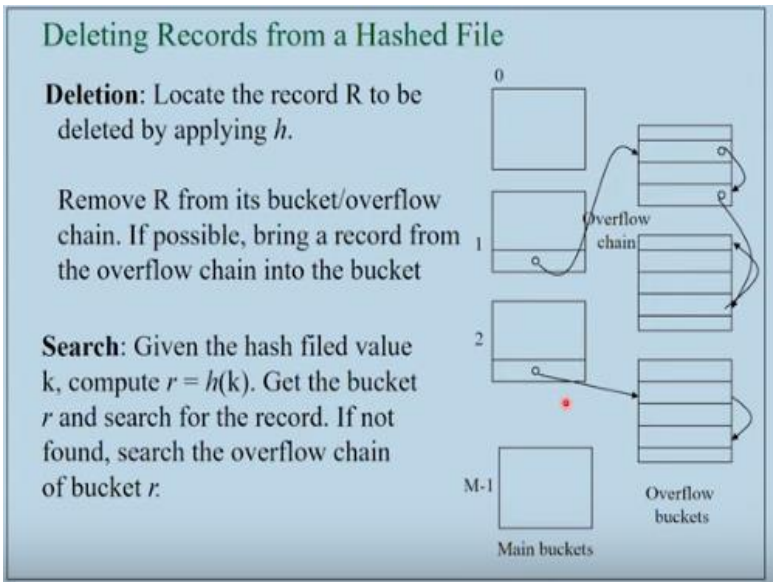
So this bucket actually might be might actually be full. If that is the case, then you know insert the new record into the overflow chain. If it is not this one space here, where you can if there is a screen where you can put this insert this data record well and good insert it. Otherwise, we will chase this overflow chain and then basically, this is something like a buddy list, right.

So you have you can take a free slot from here, you can take a free slot from here, put your record there, adjust this overflow chain to point to that and then from that, you can put a pointer to the next record that was there originally in overflow chain like slicing a new record into this overflow chain. So you can do that right in the beginning by using one of these free slots that are available in the overflow buckets.

So you would have probably studied in data structure course as to how to maintain the free list the available free slots can be can themselves be actually arranged as a list. So you can take out one item from that list and insert that frame into this overflow chain and put that record in that place. So that is how we can handle overflows into main data buckets, okay.

So the overflow chains of all these buckets are all maintained in this overflow buckets, okay. So in the file header there will be a pointer to the overflow bucket and in the free list actually, the free list so that you can in case necessary borrow an item from the previous music to put your record and then adjust the field, okay.

(Refer Slide Time: 38:00)



Now deleting record is also similar. All that you have to do is first use the hash function then get hold of the locate the record, which has to be deleted by applying this hash function then remove that R from the bucket and or the overflow chain wherever it is. It is possible that supposing there is no overflow chain then it is there in the bucket, we can just remove it.

If there is a overflow chain for the bucket, then there are two possibilities. The record might be here or might be in the overflow chain. So basically, you will have to follow the overflow chain and then follow the record and then adjust the chain so that the record actually is not anymore there in the overflow chain. It is actually that slot can be actually sliced back to the previous, okay. So that is how you remove.

So bring the record yeah if possible bring the record from the overflow chain into the main bucket. That is also possible. Yeah, for example, if you if a slot becomes empty here because of deletion, you might actually bring in the first record that is there in the overflow chain into the main bucket and then cut short the overflow chain because having this overflow chain is actually a burden.

So because you have to follow it. So search can be done given the hash field value k, compute this r which is a hash of the hash k. Get the bucket r and basically search for the record. Get the bucket and so the main cost is getting the bucket. That means get hold of those consecutive file blocks. And once you get them into the you know main memory the searching is fast so you can search in there.

So if you are unlucky then it is not there in the bucket, it is actually there in the overflow chain. Then we will have to really and that operation will be little costly because you have to do several disk accesses, okay. So on the average you would have seen the analysis of hashing. On the average you would only you would most of the time you will get your record in the main bucket itself.

The overflow chains will be very, you know small length overflow chains. If your hashing function is chosen in such a way that the records that hash down to uniformly across the data buckets and there is enough amount of space in all these main buckets then the overflow change will be small, if at all they are there. Okay, the length of the overflow chains will be small.

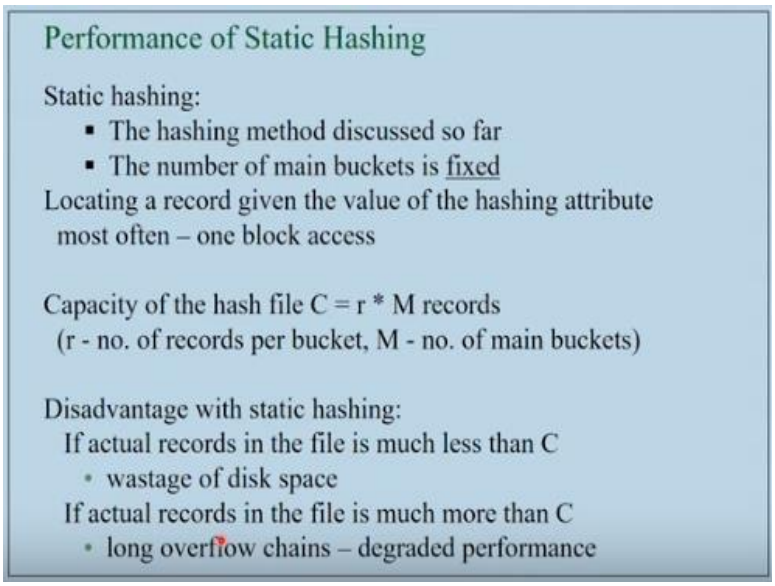
And so most of the time you will get your data access you will get your record the first time you access the main file, okay. So you compute the hash the bucket number and then go access that bucket you are most likely to get the data. So that is likely

pretty fast because if you have organized your file to have say just two buckets for two blocks per bucket or something like that.

So you are just making couple of block access and you are getting one of your data records irrespective of how many number of actual blocks are there in the immediate file. So this is one of the isolations that gives you very quick access to records which have which I mean which will have the value of the hashing adjective okay.

Now of course there is a so basically now we have looked at what is called a primary file organization, which is the logical method by which we organize records into a file, okay. And one of them is sorted files, the other one is hash files. This particular hash file organization basically makes use of what is called static hashing.

(Refer Slide Time: 43:08)



The problem with static hashing is that so we have assumed that the number of main buckets is in some sense fixed, right. So but locating a and the positive thing is that locating the record given the value of the hashing field most often will happen in one block access. And the capacity of the hashed file is something like if M is the number of main data buckets, r is the number of records per bucket, then this will be the capacity of the hash files.

Now if the actual file, actual records in the file so you have designed it for some capacity, right. You have designed it for some capacity. And you are expecting that

many records to actually come into the file and so you have designed it and kept ready for use. The actual records that go into the file is, let us say is much less than C.

Then there is going to be some wastage of disk space because you have already allocated M number of main data buckets and kept that space ready for use. And the actual number of records is much more than C, then obviously, you know the overflow chains will become longer where you are hashing lot more records into the structure than it was designed for.

And so obviously, main file buckets will overflow. And so overflow buckets have to be used and so overflow chains will be long and so the data access will suffer, okay. So data access will become costly. So these are the issues with hashed files. So these are this of course, similar kind of issues also are there with sorted files, okay.

So we have to look at we have to look at organizations that can actually help us grow the file or shrink the file and then the organization, you know can adjust itself but still give you a good performance. So that is another aspect we will see if the file is not a static file, but it keeps growing and shrinking how do we organize that kind of files? So static hashing is good for files, whose file or whose size, you have some good idea about.

And then you do not expect more than say 10% of excess records into the file, okay. So in the next lecture we will see how exactly we can use this hashing idea and then make files dynamic or handle dynamic files and dynamic files. By dynamic files what I meant is that you really do not have much of an idea about how big the file is going to become.

It might start small and then grow big and then after some time it might actually shrink back to some (()) (46:52) like that. If that is the case, how do you argue, okay. So we will stop here for today.