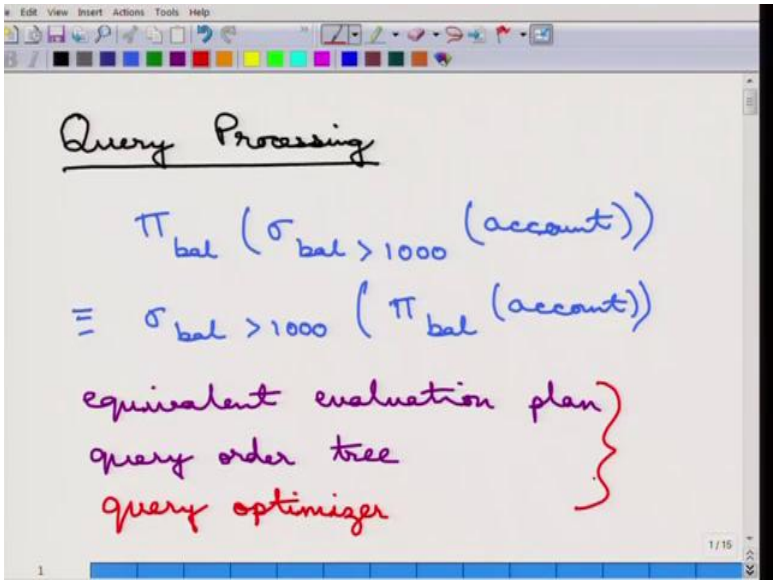
Fundamentals of Database Systems
Prof. Arnab Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 20
Query Processing: Selection

Welcome, today we will start with Query Processing.

(Refer Slide Time: 00:13)

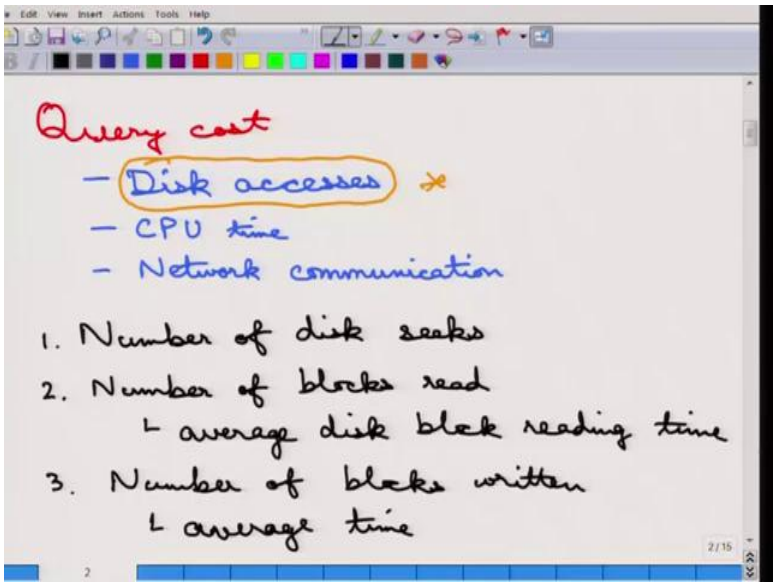


So, what do we mean by query processing is that when a query comes, for example an SQL query comes, how does the database actually answer it. So, the first thing that database system does is to parse the SQL query and it figures out, what the query is trying to do, then it evaluates the different expressions that are equivalent to the SQL query. So, what do we mean by that is that we will go over these examples in the form of a relational algebra, because SQL is finally broken down in the form of relational algebra, so let us for example take this query.

So, we all understand that for this query, the aim of this query to find out the balance of all accounts, where the balance is greater than 1000. Now, once we write this query this has an equivalent expression of the following manners, so both of these queries are the same and will result in the same output. So, two things are done when a query is first, first of all the query, so all the equivalent action plans are generated.

So, these are called equivalent evaluation plans and for each of this evaluation plan, a query order tree is created and then, the query order tree is evaluated. So, which equivalent action plan will be evaluated which is depending on the query optimizer, so there is a query optimizer that we will cover in the next chapter. The query optimizer will decide out of the different equivalent action plans, which one is going to be faster. So, based on all of these finally, the query code is generated and that is being executed.

(Refer Slide Time: 02:44)



So, what are the different things that affect the query cost? The cost of running a query can be affected by different things, where of course, as we saw the most important thing is the disk access as we have been arguing from the last time. The number of random and sequential disk IO is the more important time, also CPU time is a factor and if there is some network communication, then the network communication can also be a factor.

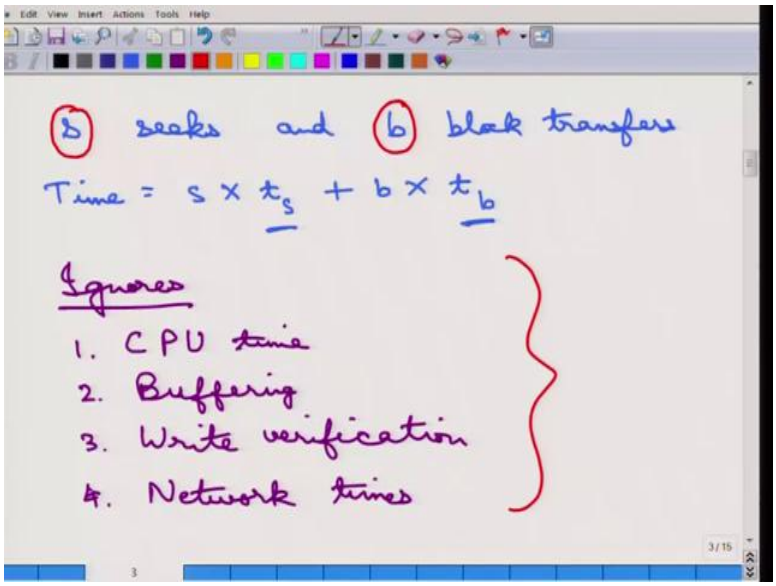
If; however, the database is in the same machine, then there is no network communication, but in general it can be done. As we said disk accesses is the biggest factor, so that is what it is being, so the all the query optimization plans generally optimize only on the disk access. Now, how can the disk access be estimated? Disk accesses can be estimated based on simply the random IO and the sequential IO cost.

So, the number of disk seeks, these include the disk rotation as well and then, the number of, so this number of disk seeks is one parameter. The number of blocks that is read, number of blocks read as a part of this of course, is the average block reading time that is

assumed, average disk block reading time and similar to the number of blocks read, there can be a number of disk blocks written.

Now, generally writing time is more than the reading time, because once the database writes it also checks whether whatever has been written is correct, so there is a reading time hidden into it. And again the average time for each of this block read, so these are the three things that can be done, but very simply it is estimated as the following manner.

(Refer Slide Time: 04:58)



So, suppose there are s number of disk seeks and there are b number of block transfers. So, generally we will ignore the reading and writing difference, you will just say b number of block transfers. So, the query cost, the time is simply estimated as s times t s, this is the average time for a disk seek plus b times t b, this is the average seek time and this is the average block time.

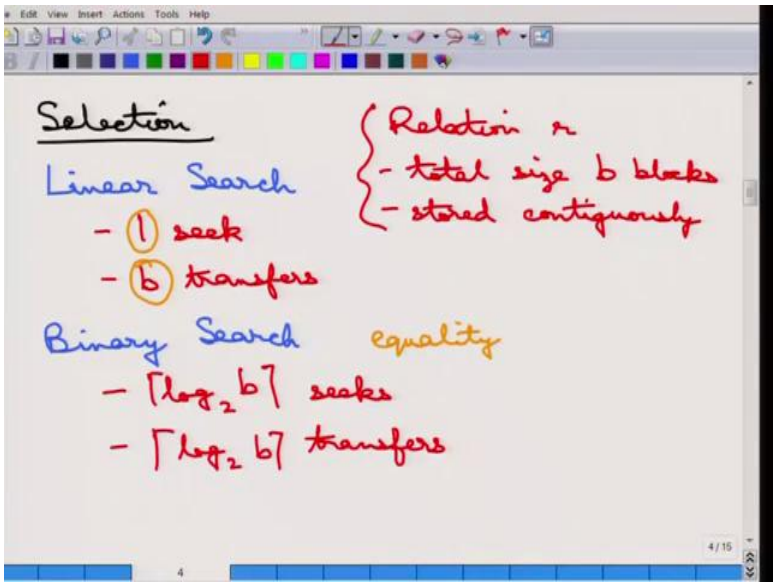
So, the two important parameters that are there is the number of seeks and the number of block transfers. So, all of the query processing algorithms that we will study will try to estimate, what is the number of seeks and what is the number of block transfers, because everything else is a constant and can be figured out there.

So, these does ignores CPU time, so very importantly we should note, what it ignores. In actual system this does play a role, it ignores the CPU time, it ignores the buffering time, as we said the writes or buffer, etcetera. It ignores the write verification time generally

and it assumes there is no network communication, so network time seeks ignores So, these are the important assumptions that one must understand when while we are analyzing this query cost.

So, once more the only important thing is the number of seeks and the number of block transfers. So, we will go over some of the algorithms in detail and we will try to analyze each one of them.

(Refer Slide Time: 06:43)



So, the first thing is the selection, remember this selection is in the relational algebra sets, not by SQL, so this is the selection as in the relational algebra. So, how can selection be done? So, selection, just to recap, selection for every tuple it tries to asserting whether the tuple can be part of the answer set or not whether the tuple should be selected or not. So, the first thing the very basic algorithm that can be employed to do it is a linear search.

What does a linear search algorithm does? It goes over all the tuples, applies the predicate the selection predicate on it if it passes, then it is fine, it is output; otherwise it is not. So, linear search is therefore, always applicable, no matter what the selection predicate is, it can be always applicable and it scans all the files and test each of these records. So, what is the cost for a relation? So, suppose there is a relation r, whose total size is b block, so the entire relation takes b blocks, total size of this thing is b blocks.

So, what is the cost for linear search? It is essentially one seek, because it needs to go to the beginning of the, where the relation is stored. So, here the assumption is that it is stored contiguously; the entire relation is stored contiguously. So, these are the assumptions that we will go over, that we will assume for all the algorithms, this is one seek plus b transfers that is it. So, these are the two important parts about linear search, there is only one seek and there are b transfers; that is all for the linear search, because it goes over the entire relation and test each record one by one.

The next important algorithm of course, is the binary search. Now, binary search although it sounds more useful, etcetera and faster, have certain background be needed. So, it is applicable only when the relation is sorted according to the attribute that is being compared. So, the selection it cannot be applied for all kinds of selection predicates, it must be only on the predicate on which the relation is sorted.

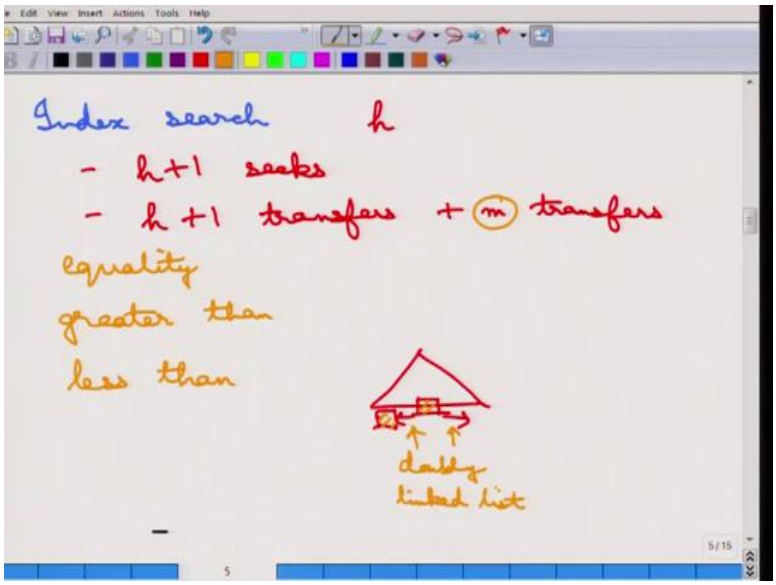
Once it is done the cost can be broken down into the following manner, suppose it is just an equality thing, so you might just want to find the equality, so find the record which is equal to a particular thing. Then, the answer to that it is about log b base 2 seeks and that many transfers, because these are random jumps, so that many seeks for each of these blocks we require a seek and a transfer, so that is the answer.

However, different cases need to be handled. So, suppose the equality condition is on a predicate, where all the tuples, where the equality is satisfied does not fit into one block. Then, the next block needs to be searched etcetera, so this is kind of a conservative estimate, that, that many seeks and transfers are needed this is kind of a average estimate. Moreover, suppose the condition is now changed to not equality, but it is let us say greater than a particular value. Then, what it is being done is that the first block is identified and from then onwards, everything else to the end of the file is being scanned.

So, the first block is identified, then after that everything else is read. So, everything else is read; that means, there were so many block transfers that needs to be done, because these are read sequentially, so that many block transfers need to be added, so that is for the greater than case. For the less than case it is a little bit more interesting, the previous strategy of handling the greater than case cannot be applicable completely on the less than, because the file cannot be scanned in the backward manner.

So, what it is being done is the other way around, the file is scanned from the beginning of it and it goes on the scan goes on till the value for which this is less than is being sort. So, it is only one seek plus all the block transfers that is needed and these are the two ways that this binary search can handle.

(Refer Slide Time: 11:09)



There is another important kind of search, which is called the index search. So, what exactly is an index search it is that we use that index that way saw earlier the b plus tree and the search is done using the b plus tree index, so it is a searching using that primary index, which is the b plus tree. Now, suppose the height of the b plus tree is h, so this means there are h plus 1 seeks and then, there are h plus 1 transfers to go to the correct thing.

So, why you use h plus 1 the leaf has to be searched, again this is assuming that the entire answer fits into one block if not there are some m transfers that are added. So, if the numbers of answers do not fit into one and then, there are m more transfers needed to be done, but it is essentially equivalent to the height of the b plus tree. So, for these of course, it is assuming the m is assumed to be 0, so there is no more transfers that are needed.

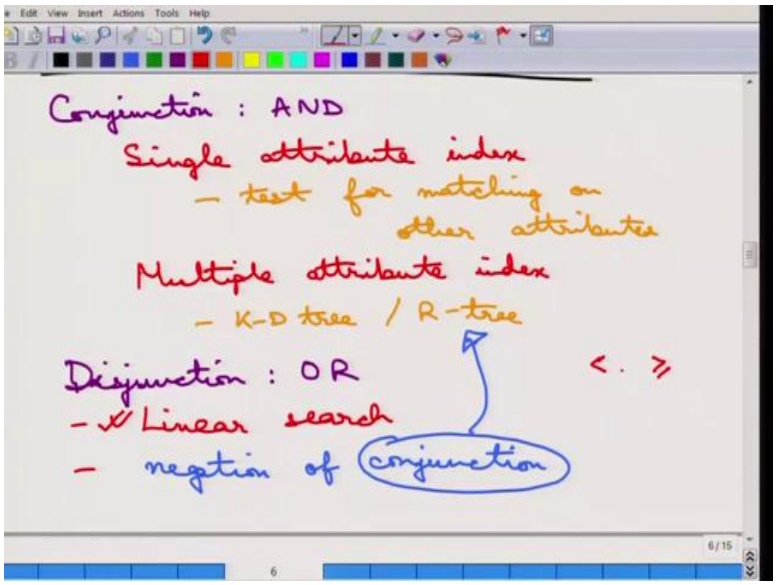
So, this is when the b plus tree is built on the index, for which b plus tree this is for when the b plus tree is build on the attribute, on which the predicate is searched, otherwise it cannot be done. So, this if this is a primary index, then this is all the cost is if this is a

secondary index, then there may be more seeks and transfers, because the attributes may be stored in different blocks etcetera the tuples that match the attributes to be stored in different blocks etcetera.

So, this is again assuming this is an equality join equality search if it is a greater than search. So, the predicate is greater than, then what happens is kind of the same as what the binary search did, then the same thing, what the binary search does is being followed. So, the first locating block using the index is located and then, everything from the index is being searched. Remember, the b plus tree generally the leaves are linked as in a doubly link list, so it can go either way and if it is doubly link list the simply the next leaves are followed.

The same kind of a thing is done for a less than if it is a doubly link list the block that matches this particular value is being searched. And since, these are doubly link list it can be traversed in both the ways and that can be done to search the other blocks that match the less than condition. So, this is the sibling leaf pointer this is the property of the b plus tree that these are doubly link list these are between the leaves and these are connected by doubly link list, so this is a doubly link list; that is what the property of a doubly linked list is being used, so this is about the index search. Now, this all we have been assuming that the selection is on a single attribute and then, we have three algorithms the linear search the binary search and the index search.

(Refer Slide Time: 14:25)



Now, suppose the selection is not a single attribute, but on multiple attributes. So, the selection on multiple attributes. So, suppose the condition is a conjunction, so this is an and condition the conjunction of the AND of selection on two attributes, then there can be different ways of solving it first of all suppose there is a single index. So, there is a index built on a single attribute.

Then everything that matches the first attribute it is being tested for matching on the other attributes that can be done, because this is on and the first one should be matched. So, everything that matches the first test for matching on the other attributes can, then be handled, so that is one way of doing it. The other is an index may be built on a multiple attribute index and this there can be different ways of building it and these are some advanced topics, that we have not covered there can be trees such as K-D tree or R tree etcetera the index itself is build in multiple attributes and then, that index can be used directly to build that.

So, this is about the conjunction if it is about disjunction OR and R condition, then it is a little the story is a little different, so disjunction OR of attributes, then the story is different. Then, if some of the attributes of the disjunction do not have an index built on it, then there nothing can be done, then the best way of doing it is simply linear scan. So, the first thing is simply linear search; that is the only thing that can be done, if there is no index built on one of these.

Otherwise, what can be done is a trick that can be done, so disjunction can be written down as a negation of conjunction. So, everything that the conjunction does not pass is part of the disjunction. So, then this conjunction can be done in the manner just like the previous case and everything whenever it is outputting it is the output in the other way around, so the negation of the conjunction can be done. But, mostly it is the linear scan that is the most useful algorithm in place of disjunction.

Because, the negation of the conjunction can be applied sometimes, but not always in the sense that it can be always applied, but it may not be beneficial the linear search may be better. So, there are two ways of doing this, this is one way the other is this way, because negation of the conjunction meaning negating the comparison and negating the comparison is just essentially another comparison for example, if less than is negative then it essentially becomes greater than, so that is about this select.