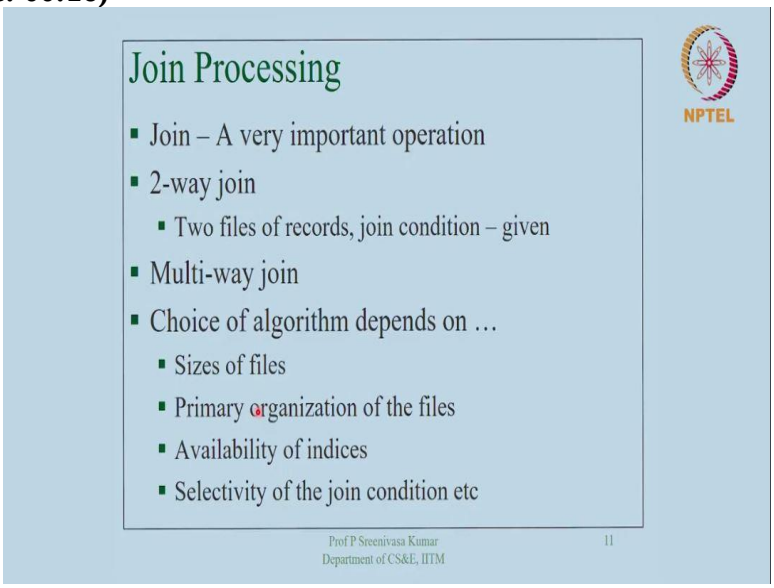
Database Systems
Prof. Dr. Sreenivasa Kumar
Department of Computer Science and Engineering
Indian Institute of Technology Madras

Lecture-34
Join operator processing algorithms

(Refer Slide Time: 00:16)



So let us continue our discussion on algorithms for relational algebra operators. Join is a very important operation as we have seen earlier in query formulation and all that. So, in the last lecture, I was, you know, talking to you about algorithm in a slightly you know how it manner. So, I thought I would start off with joint processing and then let people synchronize with this. Earlier to this, we just discussed external sorting and various issues in doing other operators like select all those.

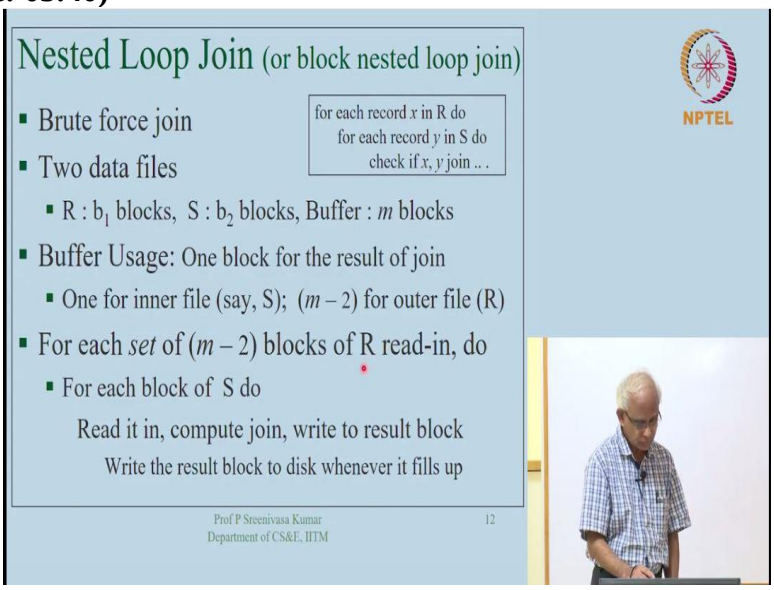
So let us focus on join today join, we will consider 2 way join, where we have 2 files of records and then we are given a joint condition and we need to join the records and produce the output. Now what is the minimum that you can, the minimum time that you can you need in order to produce join is the sum of the total of blocks in file 1 and file 2. You have to look at all the if you do not have indexes, If you do have indexes, then we are slightly better off.

But if you do not have indexes on either of the file, then the lower bound on the amount of time that you need to do for processing join is that you basically have to look at all this records. So, we will discuss a few algorithms for doing join processing and the and then so, at a particular for a particular join, you have to choose one of these operations, choose one of these algorithms and then apply that algorithm.

So, the choices algorithm depends on the sizes of the files that are involved in the join and what are the what is the primary organization of those files that are there. And what are the available indices, all the join attributes the indices have to be on the join algorithm. So are their indices on join attributes. And we will introduce an idea called selectivity of the join condition and even this has to be taken into consideration.

Before you choose the appropriate algorithm for joining process so, we will basically discuss 2 3 of these join algorithms and then the choice of course, depends on many factors the actual join is being used for a specific relational of the operation.

(Refer Slide Time: 03:40)



So, we will talk briefly talked about nested loop join yesterday. So, let me repeat this idea. This is a brute force join we have 2 data files R and S R with b 1 blocks and S with b 2 blocks. And we always assume that there is a certain amount of memory that is available to us RAM space. And we will take it as m blocks of RAM space. So we have m blocks of RAM space. And the algorithm is very simple. It is a double loop. So for each record, it is a nested loop.

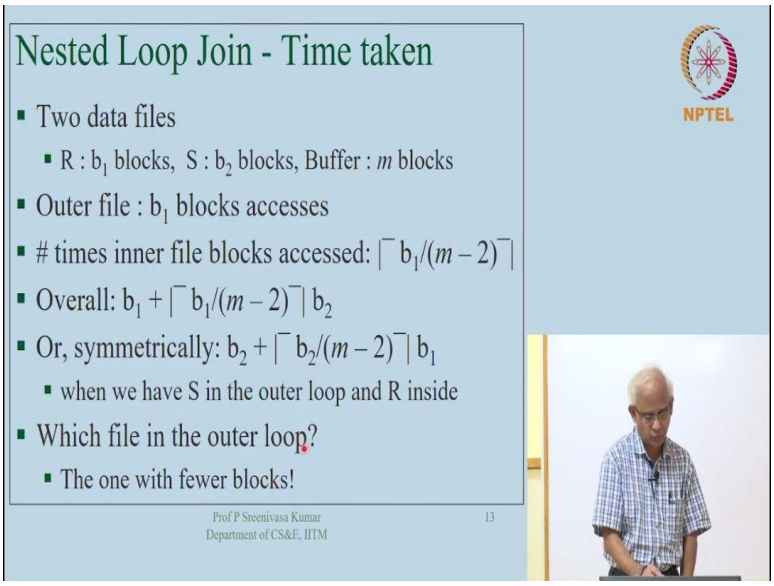
So, that is why this is also called nested loop join block nested loop join. So for each record x in R, do this thing. Which is again, a loop for each record y in S, check if y joints with the x and if so, produce the join process. What are the join with basically check the join condition, whatever. So in most of the discussion here, we will simply assume that we are doing an equally join, but the issues are similar for if we have other conditions.

So, as far as the buffer usage is concerned, we have m blocks. So, one of these blocks once the space of about one block will be reserved for storing the result of the query result of the join, join. And for the inner file, which is S we will use m - 2 we will use one block for the inner file, and m - 2 for the outer file. And the basic idea here is that since we are using external files, we will read as much as large chunk of the auto file as possible into the main memory and then make use of it fully before you move on to the next chunk of the auto file.

So, that is the basic idea. So for each set of m - 2 blocks of R that are read in from the disk file of (())(06:02) we will do this. So, we have reserved one block space for S. So, read the records of S the inner file into the one block by block you read the record of S. So, when you read it, then you have access to a bunch of records, then you can find out whether they join with any of these records of R and then it can produce a result.

So, when once you read a block, compute the join and write the result to the we have kept one block for writing the result the joint result. So, we will write records into that block. And as soon as the block is full, write it to the disk. It is a buffer step, we fill it and then I keep writing database so this is what we will do. So essentially instead of for each record x in R we are not taking a chunk of records or and then reading the entire S file and then producing the joining's. So the chunk is m - 2 blocks.

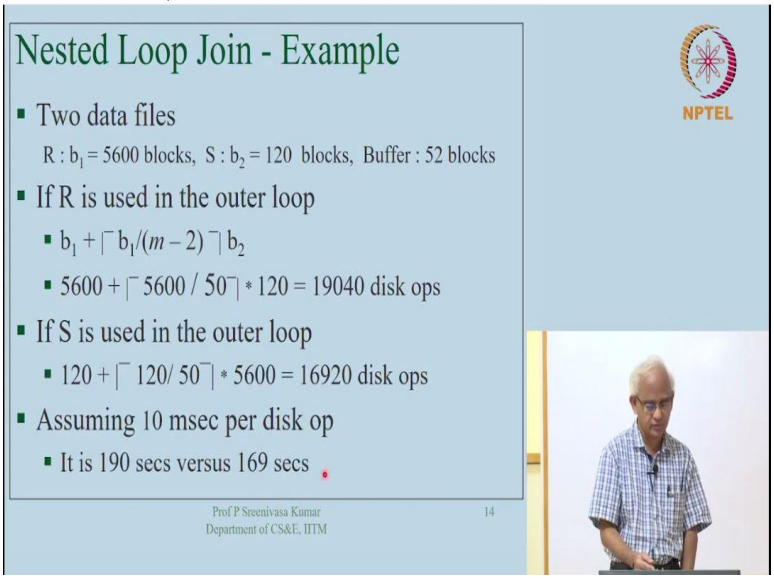
(Refer Slide Time: 08:02)



So, as far as the time taken is concerned, you can now see that if R has b 1 blocks S has b 2 blocks and buffer has m blocks, then the number of times that inner file has to be read is basically b 1 / m - 2 where b 1 you see number of blocks in the outer part R. So, overall, the complexity in terms of block accesses will be that the entire b 1 block have to be read. So, b 1 plus this many number of times you have to read the inner file.

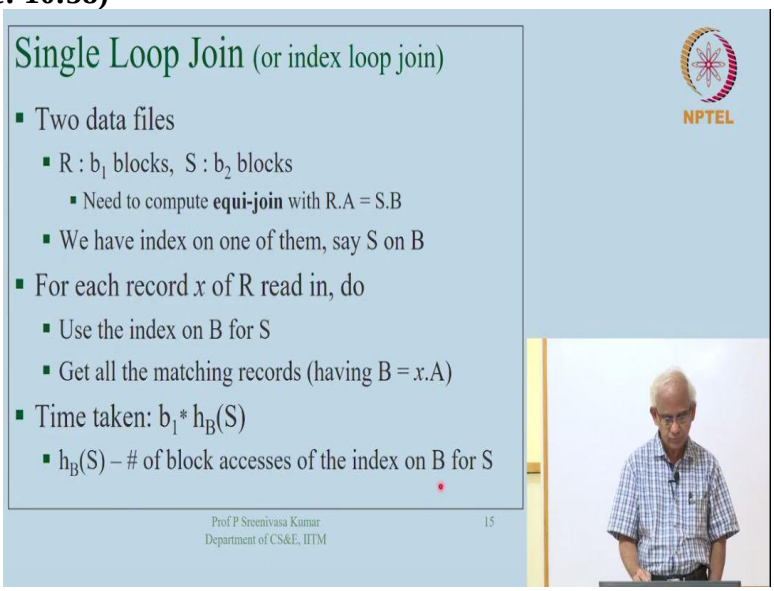
So, b 1 / m - 2 c times b. So, that is the overall complexity. You can see that there quadratic term that is coming up here, it is a multiplication of the 2 sizes of device. So, this algorithm is truly is a quadratic algorithm because it has 2 loops, you can see that it is quadratic terms. Now, you can choose the file S to be the outer file and R to be the inner file. So, by symmetry you will get this other expression. And so, of course, one small point here is that this factor is roughly the same in both, so, you can choose. So, among the 2 choices, you can choose the file which has smaller number of blocks in the and the outer loop. So that this will be compare to the other.

(Refer Slide Time: 09:57)



So, here we have b 1 has 5600 blocks and b 2 has 120 blocks, and the buffer has some 52 blocks, then you can see that if you substitute into these expressions you will get either this many number of disk accesses or if you take the S is the outer loop then 120 + 120 / 50 times, quite often 600. But when you have this, you can verify this later, but there will be a slight advantage. So, it is 190 seconds, versus 169 seconds. So, this is the situation when we really do not have any support of indices for the files.

(Refer Slide Time: 10:58)



Now, let us look at a situation where we have an index on one of these files. So this is called single loop join or index loop join algorithm. So again, we have 2 data files. Let us say R has b 1 blocks and S has b 2 blocks. And we need to compute the equi join with R dot A = S dot B equi-

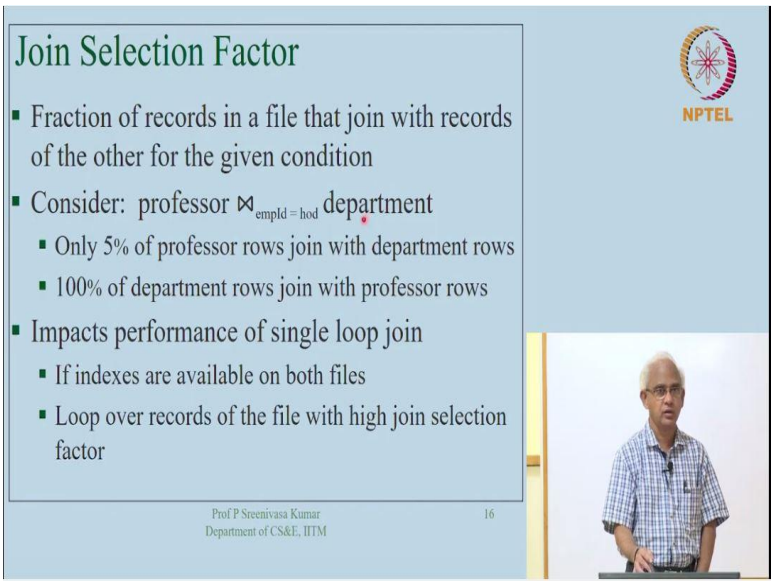
join. So let us now assume that we have an index on S. So the index of course has to be on this attribute B some arbitrary attribute will not help us. So on this join attribute, the let us say we have an index on S, what does it mean?

It means that given a value for the attribute B, I will be able to get hold of all the records that have that particular value in the S file in some number of few number of block access, depending on the indexing technique that were used. So this is much better, because we really can avoid it a loop over this S so what is the algorithm? So the algorithm is, for each record x of R, that we read in, in loop this or does it use the index on B form for the record or the other file S get all the matching records having B = x dot A.

So, x dot A you can pick up from the record that you have hold so, that is the value that is a value that matches the producers you know. So, if a record in S has exactly this value, then it will produce a result for the equi-join. So, now you can use this value, console the index get hold of all the records that have a specific rule value in the S file and then produce the join. So this way, we are able to avoid a looping over S. So, we will do a single loop which is on R for each record except R producers.

So, the time taken, obviously, time taken is easy to find out here b 1 is the number of block accesses that we have to do in order to read all records of R that reader and each record when we read that we have to do number of block accesses, you know for those matching records you have to find matching records, so the block accesses using the index will have to, well at least be making this many we will be making at least this many number of block accesses. Now, essentially having a index will greatly help us in reducing the time from quadratic to something some kind of linear.

(Refer Slide Time: 14:30)



Now there is one other important issue that comes up which is called join selection factor what is the fraction of records in a file the join with records in the other file for this specific given condition so let us illustrate it with an example. So, consider this join professor relation joined with department relations with the join commission employee id = hod. An oftentimes, we may have to find out details about the HOD of the department. So the details of the employ are there in the professor table.

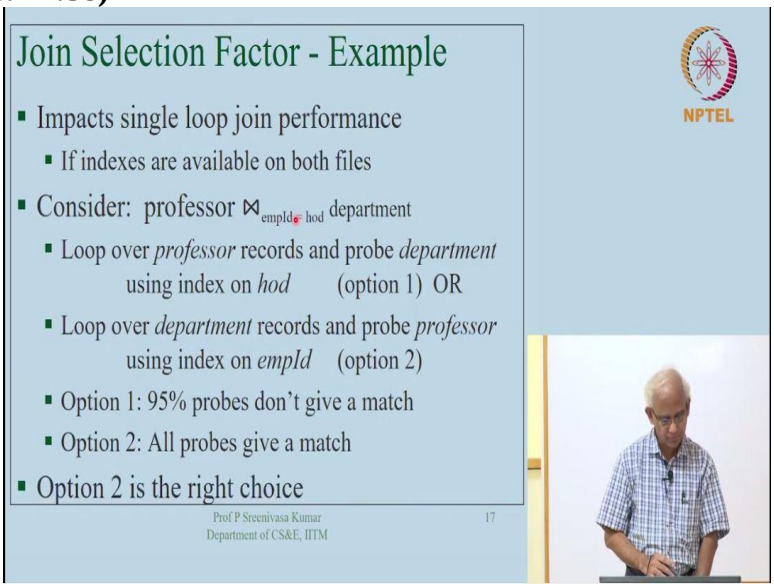
So we want to do an equi join with professor, employee id equal structure so that we can now pick up for each department we can pick up details about the head of the department on this join result of join you see that for the from the domain we have some information about this equi join. It is that every department has to have a HOD and that has to be professors and every professors need not be a HOD of some department.

And so, if you have some 500 number of professors and only 25 number of departments, you know that for only 5% of the professor records will actually have a matching department report, only 5% of these professors are actually HODs rest of the 95% people are not actually. So, if you try to join an employee ID with a HOD ID here 95% of the time its fake where a 100% of the professor departments will join with professor rows for each department has to have a HOD has to have a professor as a HOD.

So this is purely actually domain knowledge the relationship between this professor and department so these are called the join selection factors. If you know them, then it is really helpful because it is going to impact the performance of a single loop single joint. Let us say the indexes are available on both files, let us say indexes are available on professor file on employee ID and department file on HOD.

Let us assume that for a moment then it is an obvious choice that we should actually loop over the records of the file that has a high join selection. So, this particular file has high join selection factor because 100% of these records with join with other file whereas this one has low join factor because only 5% of these things will so let me know back here.

(Refer Slide Time: 17:58)



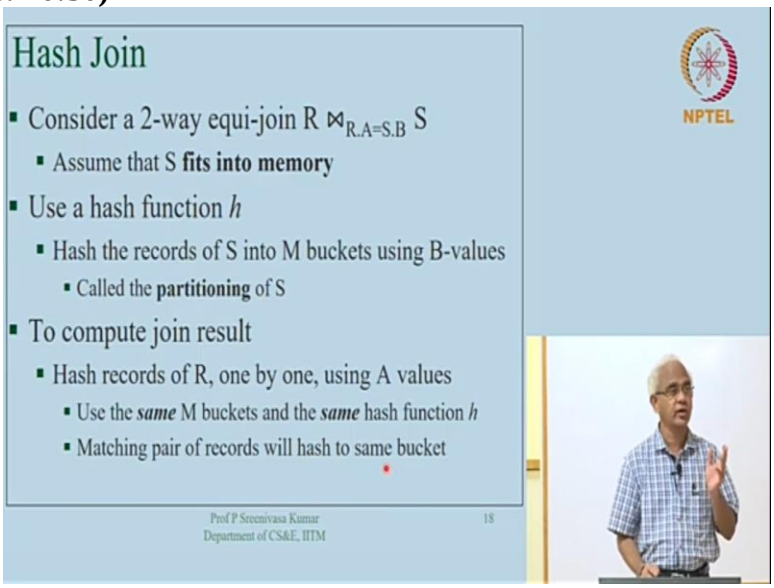
So, the joint selection factor does impact the performance of the single loop join. So, loop over professor records and probe the department we call probing the department because you will get a value for employee ID from professor record, and then you use an index on HOD to probe the appropriate department. So that is option 1 and option 2 is loop over department records and probe the professor for using the index on employee.

So, in the option 2, you can see that whenever you do a probe, you will succeed 100% successes whereas here in whenever you do a probe, only 95% of the only 5% of the time you will find the corresponding HOD corresponding department. So 95% of the prompts do not give a match and so it is a lot of wasted usage of the index. So, in order to figure out this, it will have to do some

disk accesses, because this index is in this data structure, it is lying in the on the disk, so it will spend some destruction it is a wasted effort.

So, this join section factor does will influence as to what is the correct way of doing a join which one is to be used as a in the outer loop and whose index has to be used. Of course, this is assuming that indexes are available on both file. And so what happens is that for the popular join predicates that the system is encountering. It will maintain this join selection practice for a period of time. It will accumulate knowledge about join selection factors and store them in the catalog so that for future uses of these predicates join predicates, it will make use of them and then make appropriate choices.

(Refer Slide Time: 20:30)



Now let us see how we can make use of the good old idea of hash hashing. We have seen hashing can be used for index, you know building in the last few images earlier. So now it turns out that the very beautiful idea of hashing can also be used to construct a good algorithm for join so let us see how hash joins work the couple of variations for this hash join. Let me try to cover them essentially using the idea of hashing. So these what is the facts of algorithm.

Again, let us consider some a 2 way equi join, R joins that S with join condition R dot A equals is going to be. Now let us make a we will relax this assumption little later. But let us now make an assumption so saying that the size of the file S is such that it fits into the memory. We have

enough space in the memory or the file is small. That fits into let us make this assumption for the moment. So if you do now what you do is read the records of S.

Use a hash function patch to the hash function patch and then hash this records of S on the attribute B, on the attribute values, B, attribute values, B will be there. So use them and supply them to the hashing function that will give you a bucket number and put them into m number of buckets or whatever be the number of buckets that requires hash function requires. So remember that we made an assumption that S fits into the memory. So all these m buckets are actually sitting in the RAM now.

There is there is in the memory. So the entire file is in memory now, but it is sitting in such a way that the records are distributed, hopefully evenly among all the M buckets. That depends on the hash function. So we will not go you have studied hash functions earlier in data structures because we are not go deep into that. But basically choose the appropriate hash function, hash the records of S using the B values and distribute them all this m buckets. So this is called the partitioning of S, we basically partition S records of S into buckets.

Now, in order to produce a join, what we do is now start reading the records of R R does not fit into the memory or does not fit into the memory but you can read block by block records of R and then uses A values using A values and then use the same hash function and hash these records onto the same buckets. Not for the purpose of story, but for the purpose of finding out as which bucket it is going.

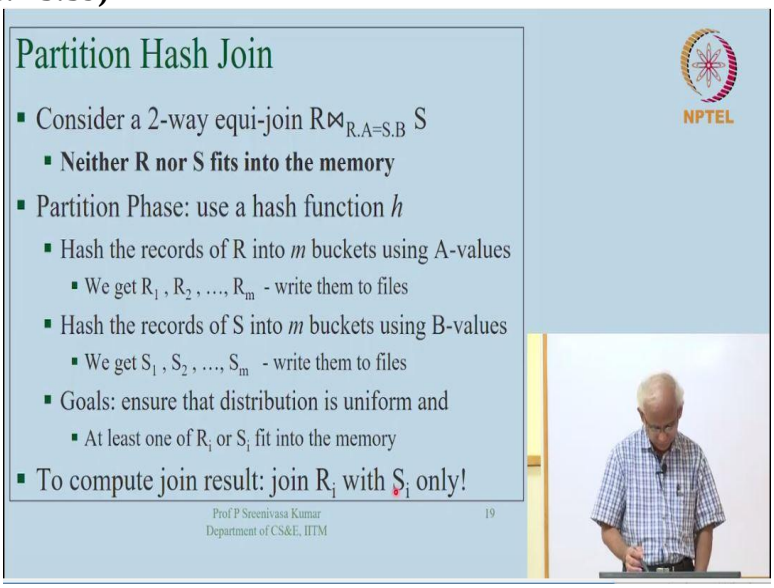
Suppose you read in a record R and it goes to the second bucket then you know that that is the only place where matching records can be found because the value if the value of R and the value of S are indeed same, then the hash will produce the same bucket number the hash function will produce the same bucket number applied to A as well as when applied to A or when applied to B the hash function will produce the same bucket number.

So, you have already done the hashing on B values so, the records are all there in the buckets. Now, use the records hash the records of R hash the records of R so, if they are using the same

hash function use the same hash function and use the same M buckets. So, the then you will find that the matching pair of records will hash the same bucket. So you read in regard of R you hash it and then you go to the bucket 10 then in that bucket, you can search for matching the cost.

You will not find these matching records anywhere else; you will only find it because that is a property we had. And there is a small space so you can search there and then figure out all of it so, this idea clear is a very interesting nice idea, using which we can make use of hash functions in order to compute joins. But the only problem now is that we have assumed that S fits into the memory.

(Refer Slide Time: 25:59)



So, this what is about this clear? So if one of them fits into the memory, then we can use this hash idea in order to produce the join reserves are there any questions on that? Can I proceed? So now let us look at a variation of this algorithm called partition hash join. Here we will relax this assumption that S fits into the memory we will not assume that so either nor neither or nor S fits into the memory then accurate to large files, and we are supposed to compute an equi join on them. So a 2 way equi join R join R dot A = S dot B.

This is what we are supposed to do R is big S is big. Both of them do not fit into the memory. Now we do what is this we will do it in 2 phases. We will have a partition phase and a probe phase in a partition phase, what will do is choose a hash function h appropriately and then hash the records of R into some M number of buckets using this A values. Now, the R file R is big.

So you can use memory buffers for each of these buckets. And then as an as soon as those buckets are getting filled or something like that, you may have to write them to the disk. , so write all these R 1 R 2 R m all of them back as files is that clear? So we basically, what we did is to use a hash function on values on the A values and partition the records of R, into m files into. So let us assume that this, the hash function produces M buckets. So we will now going to get m sub files.

The only property that these files have is that all of those that have you know, the, all those records that have a particular hash value will go into one particular point. I can choose this M to be large here. Now do the same thing for records of S using same thing for the records of S, so we have written all these files back onto the disk. So we are basically replaced R / R 1 R 2 R m do the same thing for S S 1 S 2 S m now of course use the same hash function.

But then use B values and the id i here is that choose the hash function such that the distribution is uniform and at least one of R i and S i fit into the memory, at least one of R i and S i how do you ensure this, this you can ensure by usually we use hash functions of having mod m kind of things. So, you can introduce you can increase that value M and then that will produce a large number of buckets and then if it produce large number of buckets and then records are uniformly distributed all this bucket then the bucket size will be small.

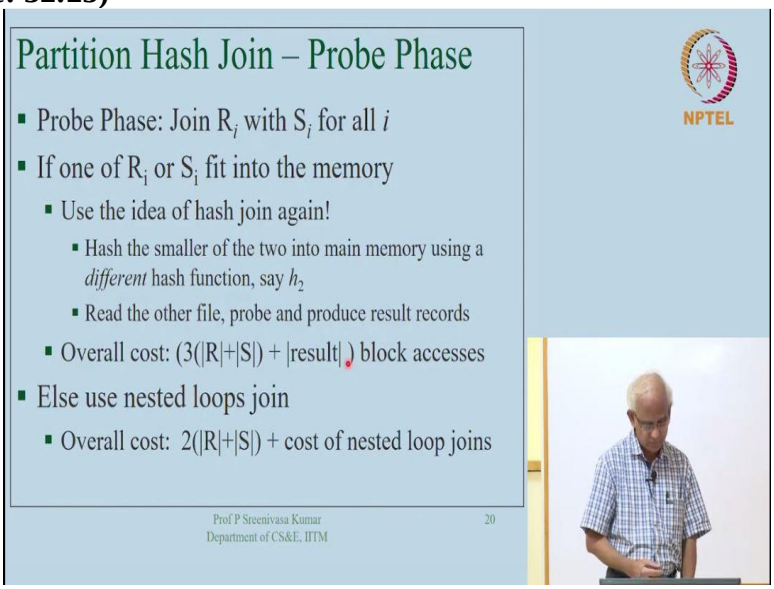
So you can choose the number of buckets appropriately so that the file fits into the memory individual files fit in to the memory (())(30:18). So this will be the goal of choosing the hash function. So now you have got an idea as why we are doing all this. Now, in order to do actual a join between R and S all that we have to do now is to only do join of R i with S i. So to compute join deserved, all we had to do is to simply compute the R i and join S i.

You will get this idea this is very important that the records of R i cannot join with anything other than records of S i. So, you can give a simple proof by contradiction for that in case it is so, then you know the matching record you know, let us assume that you take in a record and R i. That means the hash function sent it to some bucket I. So if there is a matching record for this,

how can the same hash function send that matching value to something other than i in the other case, it has to send it only to i the bucket i.

So, that is why basically join of R and S as has now got reduced to joining now, R i S i for all values of i corresponding R 1 join R 2 R 1 join S 1 R 2 join S 2 etc. it finish all of them, then you are done. That is a good idea.

(Refer Slide Time: 32:25)



Now we are back to the sub problem. So, now we can use a hash join, well, we have assumed that one of them fits into the memory. So, in the we call this a probe phase. So in the probe phase, basically what we do is join R i with S i for all i we have produce equal number of equal pairs R i S i so there are again so we try to fit in we try to choose this hash function such that either R i or S i fits into the memory, but we might sometimes fail also.

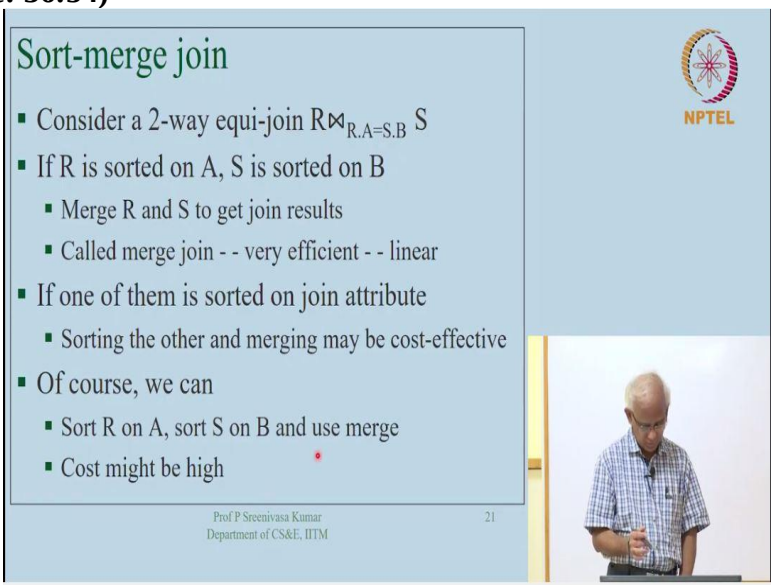
So, there are 2 cases that might arise. So, if one of R i and S i fit into the memory, then in order to produce this join, we can basically make use of the hash join that we will discuss as well. So, hash this hash the smaller of the 2 into the main memory using a obviously, you should now use a different hash function. You should not choose a different hash function, you should not choose the same hash as we have done for partitioning hash because we are now 2 separate files we are now hashing them we have to partition them now.

So, you have to use a different hash function, say h 2 and then distribute the records of one of this model file into buckets. And then we have got the file problem producing just like we did for hash join. And the overall cost will be 3 times the total number of blocks that are there in R and S (())(34:26) actually obvious know because we did a partitioning phase. So for partitioning phase we had read the whole of R whole of S and then write them back.

That is why it goes 2 block accesses R + S. Again, R + S for writing read and write. And then for this hash join, again you have to read S. And then apply a different hash function and then read all of R to start probing. So 3 times, but it is linear. Of course, you also have to take into consideration that we result you start producing result records. So they are they are in a bubble so you have to keep on writing them back to the result. So that also will cause some block accesses.

So, plus the size of the records that will be the number of block accesses. So partition hash join is not quadratic. So it certainly is linear. So it is useful. In practice so the other case here is neither one or both of them do not fit into the memory but then now we have to basically fall back onto nested loop joins and then the overall complexity will be 2 times R + S plus the cost of a nested loop join on that particular R i and of course, these cases might differ from each I actually so we should probably put this cost separate. So these are the ideas using hashing that we can actually produce nice algorithms for giving joint results.

(Refer Slide Time: 36:54)



So now we can also use the idea of merging the idea of merge start merging in order to produce a joint result actually. So, this is applicable in some selected cases. So, let us see what is that so let is again consider a 2 way join like this R join with R dot A - S dot B. And let us say R is sorted on A and this S sorted on B. So, if you have sequential or sorted files on these respective attributes, then we are very lucky.

Well this is not also uncommon also see sequential order or sorted file organization is a common file organization. And let us say you have choose doing A, a join on the primary index of one file and the primary index of other file and both of them happen to be sorted file then you can export this. So R is sorted on A S is sorted on B. Then what we can essentially do is to actually make use of the idea of merging.

So while merging, the idea is to actually produce them one after the other, if they have equal values, but here, what we will do is instead of doing that, we will simply say that if these have equal values, then they are going to produce a joint result. So, we will use the process of merging, but we will produce a different result we will produce join result so but the advantage situation here is that if you are considering a particular value x here in the sorted file, and approximate you, all the matching files will be in the same place the other file that is also sorted.

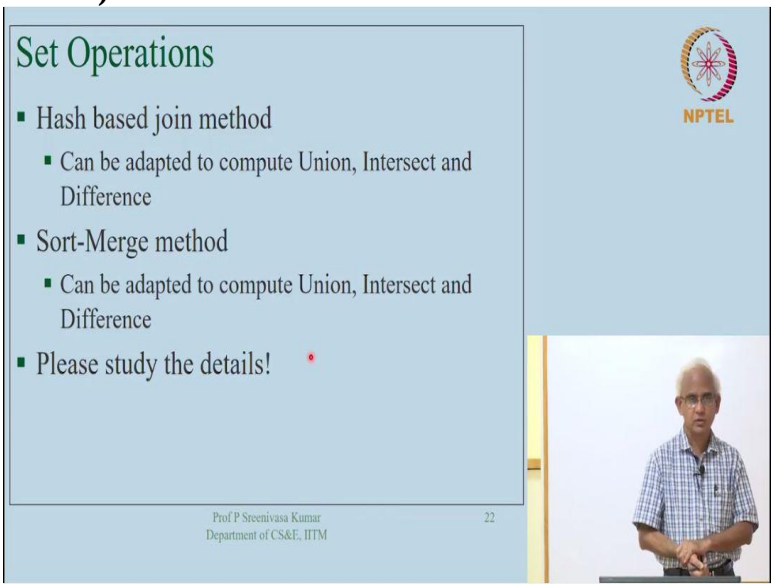
You do not have to go all over the file in order to figure out imagine so this is called a merge join. And it is really very efficient because it is also linear in the number of block accesses that you will do on R will be proportional to its size and then the same with the other file. So, you basically do a linear you pay a linear cost for this merge join. So merge join can be also made use of for producing (())(39:30).

Now in a situation where you do have a data file, that is one of them is sorted, the other one is not actually sorted then maybe, you know, sorting the other file temporarily on this join attribute, and then using a merge might actually be a better way of producing a join result. So you have to wait the cost there so sorting the other and then using the merging might be a cost-effective you know technique to produce the join result.

If one of them is sorted B of course you can make use of this sort merge join on sort on 2 data files by first joining sorting R on A and sorting S on B and then remaining the balance. For that it like to be costly because its sorting itself is costly. So you will have to these are the various options that we have while producing joins so if nothing works the best case is the nest loop join if an index and one of them you can use single loop in the join the competing joins are hash joins so you can make use of hash joins and partition hash join etcetera.

So for each of them for each of this algorithms you could also have some ideas what is the cost you are going to pay so cost estimates so depending on the kind of indexes available sizes available be nature hash functions etc. you will have to estimate the before you actually jump into chooses the algorithm you should do an estimation of the cost of that using that algorithm. And then choose (())(41:44) appropriate and this actually this idea of estimation applies to all most all the algorithm have to be used because the everywhere we have choices of algorithms.

(Refer Slide Time: 41:56)



Now we set we can also use this hash base join method you can slightly compute them you actually compute the union intersect and difference also union is all most like you know join so basically the join producing algorithm can be slightly weak to produce these even the sort merge method also can be adopted so I will not going to we stop that today so in the next class I am going to actually talk about take a example query then look at various issues in processing the query touch up on what is called query (())(42:48) its be a flavor of algorithms.