

Database Management System
Prof. Partha Pratim Das
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

Lecture - 38
Query Processing and Optimization/1: Processing

Welcome to module 38 of database management systems, in this module and the next we will talk about query processing and optimization of that in the current module we will talk about query processing.

(Refer Slide Time: 00:29)

Module Recap PPD

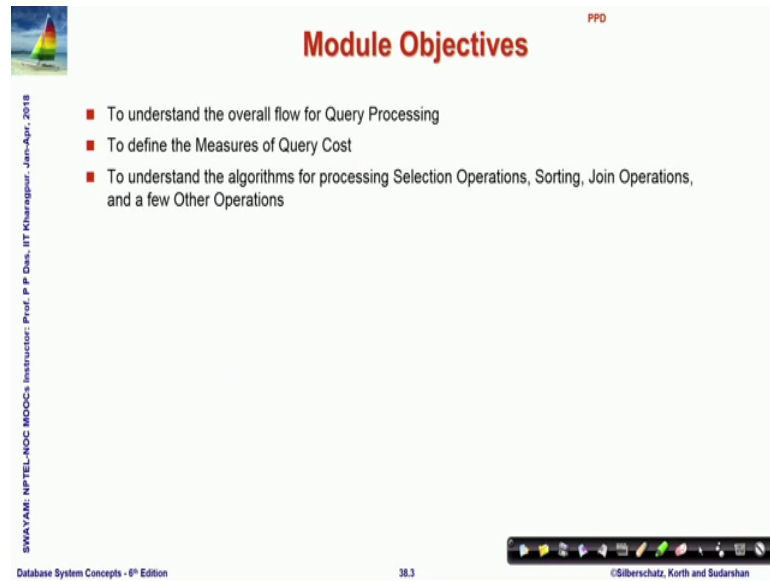
- Failure Classification
- Storage Structure
- Recovery and Atomicity
- Log-Based Recovery

SWAYAM: NPTEL-NOC MDOCS Instructor: Prof. P P Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 6th Edition 38.2 ©Silberschatz, Korth and Sudarshan

So, in the last module we had done talked about database recovery.

(Refer Slide Time: 00:36)



PPD

Module Objectives

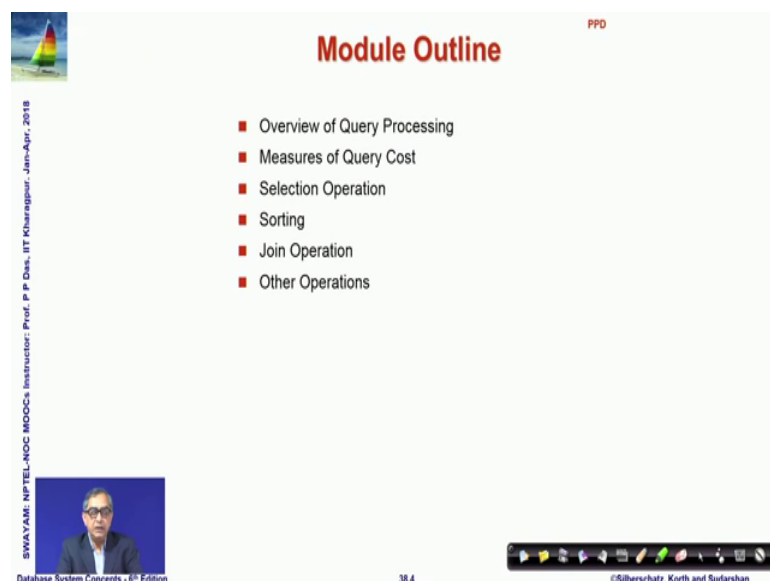
- To understand the overall flow for Query Processing
- To define the Measures of Query Cost
- To understand the algorithms for processing Selection Operations, Sorting, Join Operations, and a few Other Operations

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Khargpur, Jan-Apr, 2018

Database System Concepts - 8th Edition 38.3 ©Silberschatz, Korth and Sudarshan

And now we will try to understand the overall flow of processing queries. So, if I fire a query like select from where how will that actually access the database files the b trees and indexes and so, on and compute the result is what we would like to discuss. And a query can be processed in multiple ways giving rise to different kinds of costs in terms of the time required for processing that query. So, we will define certain measures of query cost and then we will take a quick look into a set of sample algorithms for processing simple selection operation, sorting, joint operation and few of the other operations like aggregation.


(Refer Slide Time: 01:26)



PPD

Module Outline

- Overview of Query Processing
- Measures of Query Cost
- Selection Operation
- Sorting
- Join Operation
- Other Operations

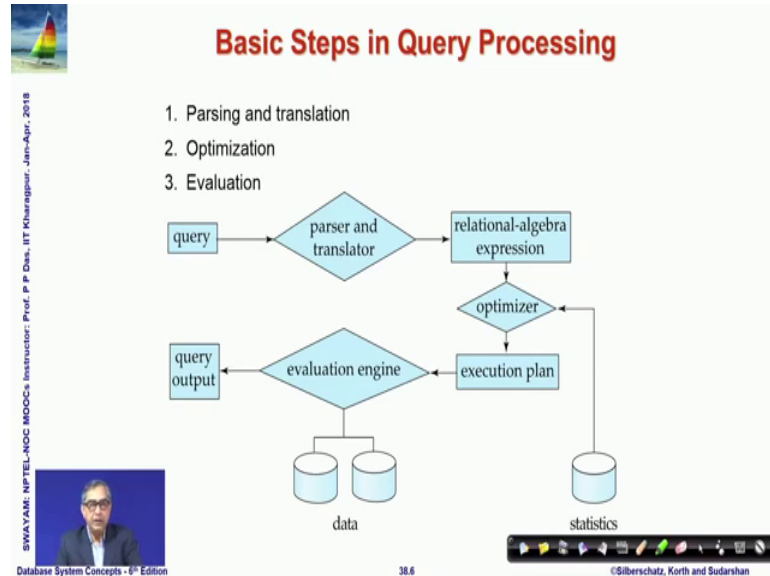


SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Khargpur, Jan-Apr, 2018

Database System Concepts - 8th Edition 38.4 ©Silberschatz, Korth and Sudarshan

So, first let us so, these are the topics to talk about and first we will take a look at the overall query processing algorithm.

(Refer Slide Time: 01:35)



So, this is the flow so, this is a way the a query will get processed. So, here what you have is this is where the input query comes in naturally it is written in terms of a in terms of SQL which is kind of a programming language.

So, you need a parser and translator. So, the it is parse translated and it is translated into a relational algebra expression as we have shown at the very beginning discussed at the very beginning that SQL is basically derived or developed based on relational algebra. So, corresponding to every SQL query there is a one or more relational algebra expression. So, you express in terms of that, then you optimize you try to see how the relational algebra expression can be made efficient and to optimize this we might use some information about the statistics.

Statistics in terms of we might use the information past history information of say what is the what are the attributes on which more often the where conditions are port we might want to use statistics like how many tuples actually exist in the relation right now and so, on. And based on that we will decide on an execution plan, execution plan is how we actually want to what are the actions that we actually want to do in terms of accessing the different indexes and the different b plus tree nodes to evaluate the query and that is the

job of the evaluation engine is you can see that it will access the data for that and finally, out of that the query output will be generated.

So, in this module and the next we will take a quick look into so, it is glimpses of these steps one by one and try to understand how query processing and optimization can happen.

(Refer Slide Time: 03:40)

Basic Steps in Query Processing (Cont.)

- Parsing and translation
 - translate the query into its internal form
 - ▶ This is then translated into relational algebra
 - Parser checks syntax, verifies relations
- Evaluation
 - The query-execution engine takes a query-evaluation plan, executes that plan, and returns the answers to the query

SWAYAM: NPTEL-NOC MOOC's Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018
Database System Concepts - 8th Edition 38.7 ©Silberschatz, Korth and Sudarshan

So, beyond a parsing and translation there is evaluation as we have talked of.

(Refer Slide Time: 03:48)

Basic Steps in Query Processing : Optimization

- A relational algebra expression may have many equivalent expressions
 - E.g., $\sigma_{\text{salary} < 75000}(\Pi_{\text{salary}}(\text{instructor}))$ is equivalent to $\Pi_{\text{salary}}(\sigma_{\text{salary} < 75000}(\text{instructor}))$
- Each relational algebra operation can be evaluated using one of several different algorithms
 - Correspondingly, a relational-algebra expression can be evaluated in many ways
- Annotated expression specifying detailed evaluation strategy is called an **evaluation-plan**.
 - E.g., can use an index on *salary* to find instructors with salary < 75000,
 - or can perform complete relation scan and discard instructors with salary ≥ 75000

*Select Salary
From instructor
where salary < 75000*

SWAYAM: NPTEL-NOC MOOC's Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018
Database System Concepts - 8th Edition 38.8 ©Silberschatz, Korth and Sudarshan

Now, if we take in terms of say a query a where select from where clause has been translated. So, for example, if this is we can we can simply write out say if we have select and what are we selecting here? We are selecting salary and where are we selecting it from? The from is instructor and under what conditions are we doing that? Where, where is salary is less than 75000.

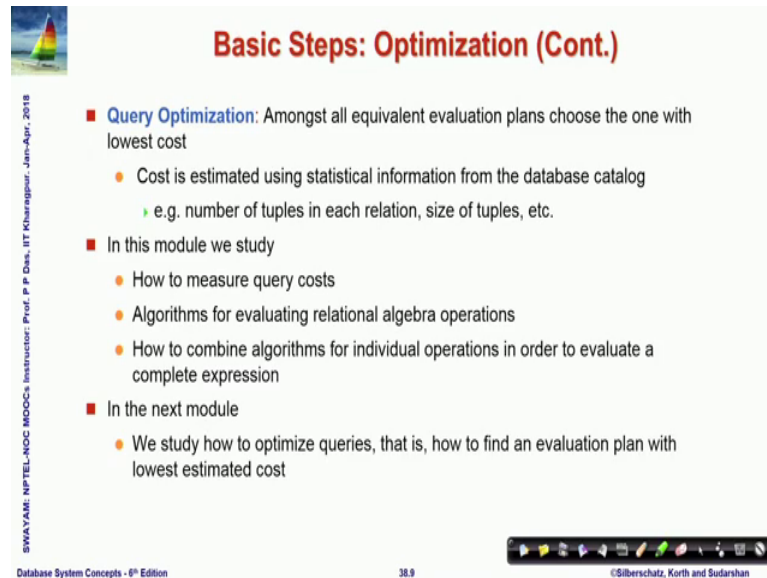
So, if you had a query like this then you know then it will be it will get translated to some kind of a relational algebra expression like this where you do a selection on the salary and then you do you do a projection on the salary and you do a selection based on that condition. Now, it is also clear to say that this particular relational algebra expression can be equivalently written by swapping that these 2 conditions, that is we can first do a selection and then do the projection they are actually equivalent and that could be multiple equivalents.

So, we know that given a query there could be multiple relational algebra expressions and then the relational algebra expression the operations can be evaluated also in one of the by using one or more different algorithms.

So, basically what you have you have different options given a SQL query, you have different possible relational algebra expressions that are equivalent given every relational algebra expressions you have different strategies different algorithms to actually execute and evaluate that. And we would like to based on these we would like to annotate the expression we would like to mark out as to whether from the above to say whether we first project on salary and then do the selection or we first do the selection on salary less than 75000 and then project.

And with that annotation we will build up a total evaluation strategy which is known as the evaluation plan and for example, here we can have different strategies like we can use an index on salary and if you use that that will be it will be pretty efficient to find tuples which satisfy salary less than 75000 or we can scan the whole relation and discard all those instructors for which salary is greater than equal to 75000. So, there could be different ways in which we can do this evaluation and that is what optimally has to be decided in every case.

(Refer Slide Time: 06:50)



Basic Steps: Optimization (Cont.)

- **Query Optimization:** Amongst all equivalent evaluation plans choose the one with lowest cost
 - Cost is estimated using statistical information from the database catalog
 - ▶ e.g. number of tuples in each relation, size of tuples, etc.
- In this module we study
 - How to measure query costs
 - Algorithms for evaluating relational algebra operations
 - How to combine algorithms for individual operations in order to evaluate a complete expression
- In the next module
 - We study how to optimize queries, that is, how to find an evaluation plan with lowest estimated cost

SWAYAM: NPTEL-NOC MDOCS Instructor: Prof. P. P. Das, IIT Khargpur, Jan-April, 2018

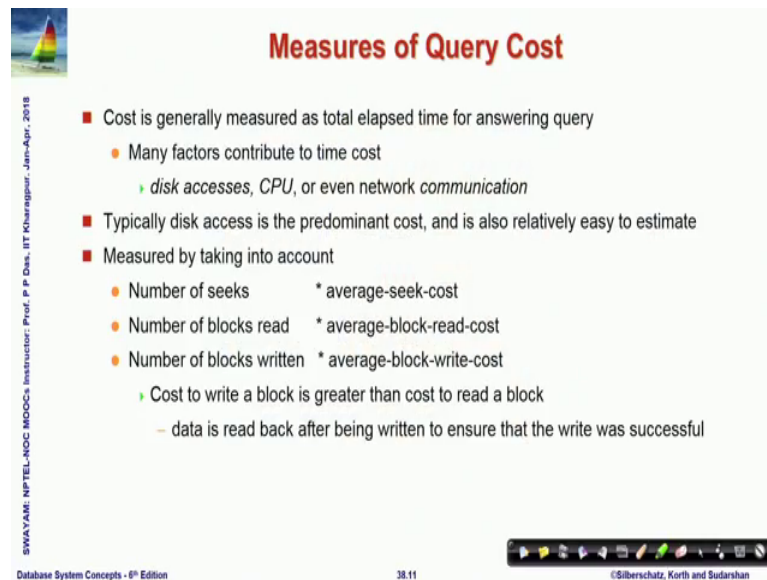
Database System Concepts - 8th Edition 38.9 ©Silberschatz, Korth and Sudarshan

So, in terms of query optimization out of all these equivalent evaluation plans we try to choose the one that gives some minimum cost the lowest cost evaluation.

So, the cost will have to be estimated based on certain statistical information from the database catalog for example, number of tuples in the relation, the size of the tuples, the attributes on which frequently condition are tested and so, on. So, this is what we would in totality try to understand out of that in this module we will first define what is the measures of cost and look at the algorithms for evaluating some of the relational algebra operations and then you can combine them to do bigger operations and in the next module we will talk about optimization.

So, first let us see how we define the cost because if we want to say that we can do it you say in 2-3 different ways, evaluate the same query in 2-3 different ways then we must assess as to what is the best way of doing it the best way is whatever gives us the least cost.

(Refer Slide Time: 08:02)



Measures of Query Cost

- Cost is generally measured as total elapsed time for answering query
 - Many factors contribute to time cost
 - ↳ *disk accesses, CPU, or even network communication*
- Typically disk access is the predominant cost, and is also relatively easy to estimate
- Measured by taking into account
 - Number of seeks * average-seek-cost
 - Number of blocks read * average-block-read-cost
 - Number of blocks written * average-block-write-cost
 - ↳ Cost to write a block is greater than cost to read a block
 - data is read back after being written to ensure that the write was successful

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Khargpur, Jan-Apr, 2018

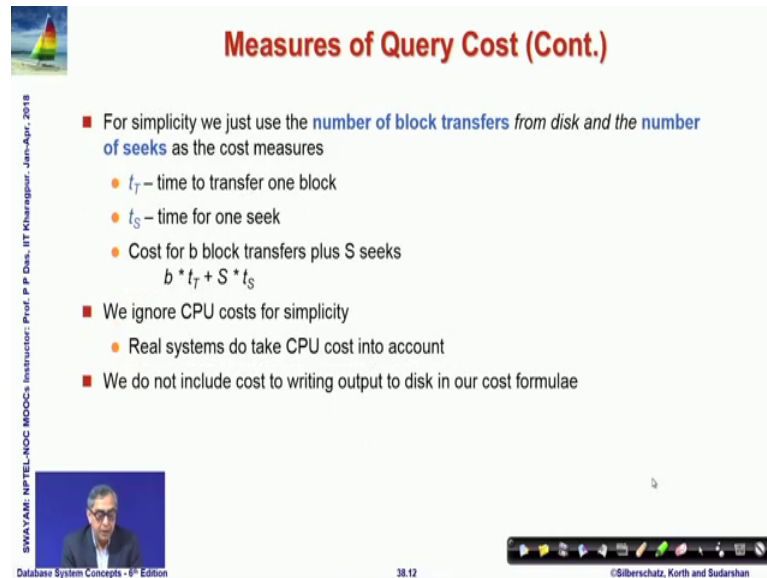
Database System Concepts - 8th Edition 38.11 ©Silberschatz, Korth and Sudarshan

So, measures of cost will be in absolute terms it is in terms of the elapsed time how much time does it take and there could be many factors which may dictate that because in terms of evaluating this we will have to access the disk. So, access time of the disk will be involved, the computing time in CPU may be involved even some network communication may get involved.

So, out of these if we assume that there is no network communication cost just for simplicity that is everything is connected to a very you know high speed network then between the disk cost and the accesses and the CPU processing cost the disk access is a predominant cost. Because and it is relatively easy to estimate that because as we have looked at the storage structure we know that it is a typically a magnetic disk which where the head has to move to the correct cylinder to find the block where the records can be located. So, there is this process is called the seek.

So, we will need to find out how many estimate how many seek operations we need and multiply that by the average cost of seeking a block. Similarly, while we are reading that we need to estimate how many blocks to read and average cost to read a block, number of blocks to write average cost to write the block, cost to write the block is usually greater than the cost to read actually often when we write a write some data after writing we also usually read it back to make sure that the right was successful.

(Refer Slide Time: 09:53)



Measures of Query Cost (Cont.)

- For simplicity we just use the **number of block transfers** from disk and the **number of seeks** as the cost measures
 - t_T – time to transfer one block
 - t_S – time for one seek
 - Cost for b block transfers plus S seeks
 $b * t_T + S * t_S$
- We ignore CPU costs for simplicity
 - Real systems do take CPU cost into account
- We do not include cost to writing output to disk in our cost formulae

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Khargpur, Jan-April, 2018

Database System Concepts - 9th Edition

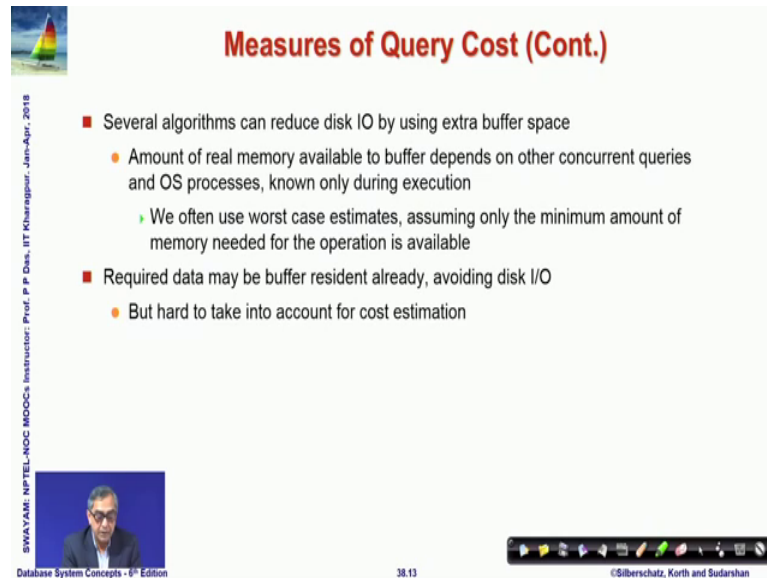
38.12

©Silberschatz, Korth and Sudarshan

So, these are the typical cost factors that will dominate. So, if we say that if we just count the number of block transfers and the number of seeks and if the time to transfer one block is t_T and time for seek is one seek is t_S , then the cost of transferring b blocks doing and doing S seek will be given by this expression you can easily understand that.

So, every block transfer is t_T and the b blocks being transferred. So, this is the transfer cost and if there are S number of seeks and every seek time is t_S , then this is the seeking cost and adding them together we get the total cost of seek and transfer. For simplicity we will ignore the CPU cost and we will also for now not consider the cost of finally, writing the result back to the disk we will simply check as to what will it take to actually compute the result.

(Refer Slide Time: 10:54)



Measures of Query Cost (Cont.)

- Several algorithms can reduce disk I/O by using extra buffer space
 - Amount of real memory available to buffer depends on other concurrent queries and OS processes, known only during execution
 - ▶ We often use worst case estimates, assuming only the minimum amount of memory needed for the operation is available
- Required data may be buffer resident already, avoiding disk I/O
 - But hard to take into account for cost estimation

SWAYAM: NPTEL-NOC MDOCS Instructor: Prof. P. P. Das, IIT Khargpur, Jan-Apr, 2018

Database System Concepts - 9th Edition

38.13

©Silberschatz, Korth and Sudarshan

So, there are also it has to be noted that there are also several algorithms to reduce the disk I/O we can do that by using extra buffer space for example, one block has been read in and we are just using one record of that if in the next operation we have to use some record which is already existing in that block and if that block is maintained in that buffer then we do not need to go back to the disk and actually read the block once again.

So, the more of the buffer space that we can provide naturally the performance would become better, but certainly; that means, that the memory required for keeping the buffer would be higher and it is also often difficult to decide as to I mean estimate a query as to if I am looking for a particular block whether it is already there in the buffer.

So, that the I/O can be avoided or it needs to be actually read back from the disk, but these are some of the you know cost measures that are used in a more sophisticated cost function, but we will simply use the seek and read cost from the disk in terms of blocks to estimate our cost of the different operation. So, let us look at sample algorithms for different basic SQL operation. So, the first and most common SQL operation is selection as you all know.

(Refer Slide Time: 12:29)

Selection Operation

- **File scan**
- **Algorithm A1 (linear search)**. Scan each file block and test all records to see whether they satisfy the selection condition
 - Cost estimate = b_r block transfers + 1 seek
 - ▶ b_r denotes number of blocks containing records from relation r
 - If selection is on a key attribute, can stop on finding record
 - ▶ cost = $(b_r/2)$ block transfers + 1 seek
 - Linear search can be applied regardless of
 - ▶ selection condition or
 - ▶ ordering of records in the file, or
 - ▶ availability of indices
- Note: binary search generally does not make sense since data is not stored sequentially

Handwritten notes: $br/2 * tr + ts$, $tr + ts$

So, the selection for selection we discuss in multiple algorithms for different situations the first algorithm is here we are calling it as algorithm A1 is a linear search. So, what we do we just need to do some selection. So, we scan the say we are looking for a result of couple of records and or a single record then we just scan the file from one end to the other we look for all the records and check whether they satisfy the selection condition.

So, the cost for that would be b_r block transfers if there are if b_r is a number of blocks containing records from relation r then b_r blocks have to be read and one seek has to happen. Now, if the selection is on a key attribute and we can stop find on finding the record and on the average we can expect that we will be able to find it by having read half of the record. So, $b_r/2$ block transfers plus 1 seek so, if I if I write it in the notation that we had used earlier this $b_r/2$ into the transfer cost plus one seek cost.

So, this should give us the cost of the finding out the particular record from any file if we are doing a linear search if we are doing a linear scan. So, the advantage of this is it can be applied irrespective of the condition, ordering of the records whether or not the index is available and so, on.

So, this could be the fallback in any case when we want to do that and just you may note that in memory when we search we say that we will keep the data sorted and binary search is efficient, but that is not the case for us here because as you know the data is not stored sequentially it is in terms of a tree structure. So, when the index is available we

will do the index based search otherwise we will have to do some kind of a linear scan alone. So, this was the first algorithm that we can think of.

(Refer Slide Time: 14:44)

Selections Using Indices

- **Index scan** – search algorithms that use an index
 - selection condition must be on search-key of index
 - h_i = height of B+ Tree
- **A2 (primary index, equality on key)**. Retrieve a single record that satisfies the corresponding equality condition
 - $Cost = (h_i + 1) * (t_T + t_S)$
- **A3 (primary index, equality on nonkey)** Retrieve multiple records
 - Records will be on consecutive blocks
 - ▶ Let b = number of blocks containing matching records
 - $Cost = h_i * (t_T + t_S) + t_S + t_T * b$

SWAYAM: NPTEL-NOC MOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Now, if now let us assume that it is the situation is such that we have some index on the b plus tree that we are using to going to do the selection on and let us assume that h_i is the height of that b plus tree. So, the second algorithm is good if we are using a primary index and we are looking for equality on a key that whether it matches certain key. So, what will have to do we know that in b plus t if the if the height is h_i then we will be able to find the leaf node surely by h_i number of block transfers because we will be a h_i is the height of the tree.

So, this is a number of nodes the number of blocks that we will need to read. So, if each one of that will take a transfer time plus a seek time because they are not consecutively located. So, everything they will have to be will need to seek them. So, that will be h_i times t_T plus t_S and we will need one additional block transfer to actually get the data get the record read it. So, that will give us cost that we have shown here.

In a variant of this algorithm A3 we may be using a primary index, but we are looking for equality on a non key. So, if you are looking for equality on a non key since is a non key then certainly in the result we may have multiple records, but the records will be on consecutive blocks because we are using a primary index.

So, if b is the number of blocks containing matching records then we will need to have say this is a cost to find out the first one and then they from consecutively they are on. We will need to locate the next record and the b blocks for transferring all the matching records this is the kind of cost that will need to go through natural you can see that in these cases all these cost expressions are better than what we were getting in terms of doing a linear search.

(Refer Slide Time: 17:08)

Selections Using Indices

■ **A4 (secondary index, equality on nonkey).**

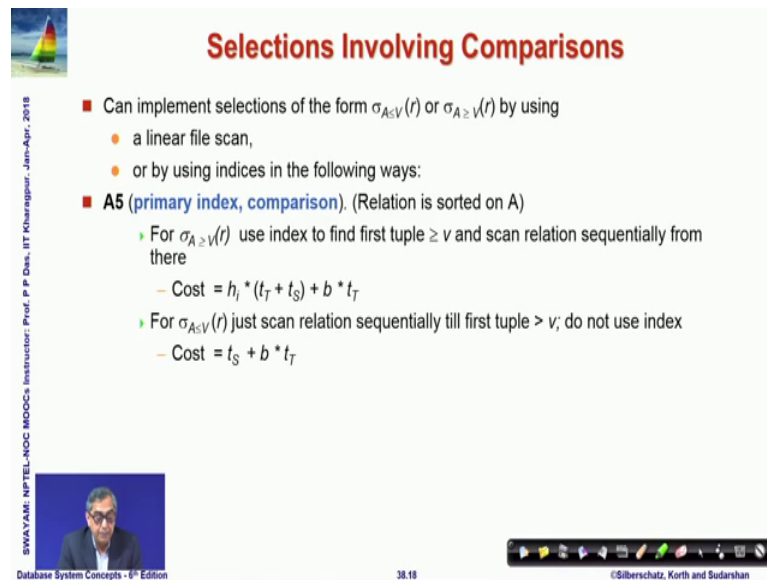
- Retrieve a single record if the search-key is a candidate key
 - ▶ $Cost = (h_i + 1) * (t_T + t_S)$
- Retrieve multiple records if search-key is not a candidate key
 - ▶ each of n matching records may be on a different block
 - ▶ $Cost = (h_i + n) * (t_T + t_S)$
 - Can be very expensive!

NPTEL-NOC MCOCA Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018
 Database System Concepts - 6th Edition
 38.17
 ©Silberschatz, Korth and Sudarshan

If we look into a few of the other situations for example let us say instead of primary index if I have a secondary index and you are looking for a equality or non key then you can retrieve a single record if the search key is candidate key. If it is a candidate key then we know that even though it is not a primary key, but certainly 2 tuples can never match on them and still exist. So, it is a candidate key we will need to have we will get only a single record and therefore, this is a cost expression that you will get, but if it is not a candidate key then there could be multiple records that will have to be finally, retrieved.

So, if there are n records then first you will need h_i to locate the first record and times of course, h_i times the transfer plus seek cost and if there are n records then you will need to every time each one of them because they are on secondary index and non key. So, you have to retrieve them one by one and every time you will need a search you will need seek and transfer time and this can as you can understand could be quite expensive if n turns out to be large which will often be the case.

(Refer Slide Time: 18:18)



Selections Involving Comparisons

- Can implement selections of the form $\sigma_{A \leq v}(r)$ or $\sigma_{A \geq v}(r)$ by using
 - a linear file scan,
 - or by using indices in the following ways:
- **A5 (primary index, comparison)**. (Relation is sorted on A)
 - ▶ For $\sigma_{A \geq v}(r)$ use index to find first tuple $\geq v$ and scan relation sequentially from there
 - Cost = $h_i * (t_I + t_S) + b * t_T$
 - ▶ For $\sigma_{A \leq v}(r)$ just scan relation sequentially till first tuple $> v$; do not use index
 - Cost = $t_S + b * t_T$

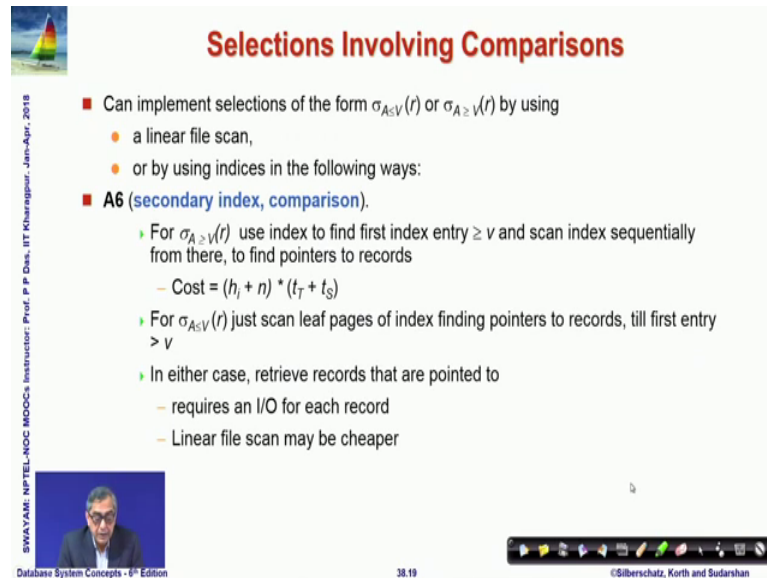
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P.P. Das, IIT Khargpur, Jan-Apr, 2018
Database System Concepts - 9th Edition
©Silberschatz, Korth and Sudarshan

We can implement different this is a very common selection condition where we have these kind of conditions that we are selecting on less than equal to or greater than equal to kind of condition.

So, we can implement this using a linear file scan or by using indices in a certain way using indices we will certainly have a better performance. So, if we have a if we have a primary index and we are using comparison then what we will we can certainly decide is we need to find out the first tuple which matches this condition that is a is greater than equal to v and then once we have found that in a primary index the following ones will all be ordered in that manner. So, I can scan sequentially from there and get them.

So, finding the first one will give me finding the first one will give will take this cost because I am doing a search on the primary index and then if there are b blocks containing the result records then this is the cost that will need. Whereas, we can do something different also we can just start sequentially based on the primary index from the beginning and check for the till we get a tuple which is greater than v and then we do not use the index we can simply we know because these are all ordered in terms of the primary index. So, that will be the search result that can be easily produced. So, here we are using a linear scan and that itself will give a good result.

(Refer Slide Time: 20:03)



Selections Involving Comparisons

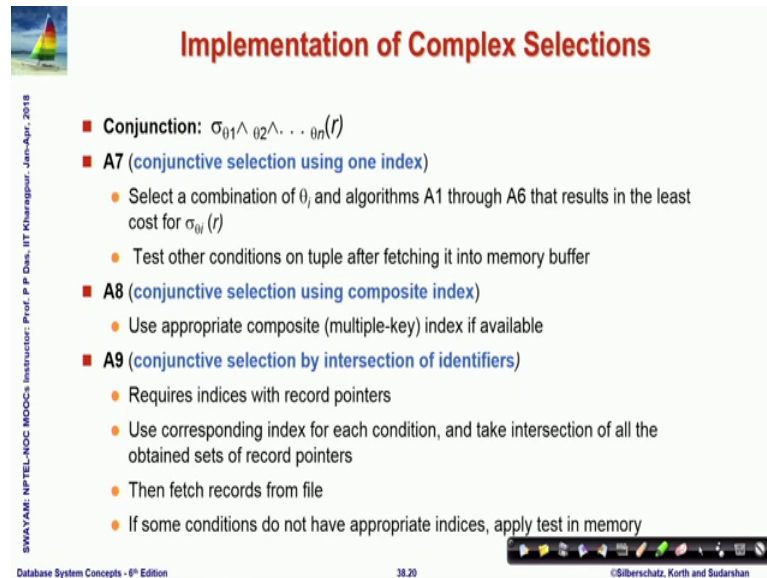
- Can implement selections of the form $\sigma_{A \leq V}(r)$ or $\sigma_{A \geq V}(r)$ by using
 - a linear file scan,
 - or by using indices in the following ways:
- **A6 (secondary index, comparison).**
 - ▶ For $\sigma_{A \geq V}(r)$ use index to find first index entry $\geq v$ and scan index sequentially from there, to find pointers to records
 - Cost = $(h_i + n) * (t_r + t_s)$
 - ▶ For $\sigma_{A \leq V}(r)$ just scan leaf pages of index finding pointers to records, till first entry $> v$
 - ▶ In either case, retrieve records that are pointed to
 - requires an I/O for each record
 - Linear file scan may be cheaper

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P.P. Das, IIT Khargpur, Jan-Apr, 2018
Database System Concepts - 9th Edition
38.19
©Silberschatz, Korth and Sudarshan

But if we are doing a similar operation based on a on a secondary index we are doing composition on a secondary index then we will again use the index to find the first index entry greater than equal to the key value and then scan the index sequentially.

So, we will get a cost which is similar to what we saw earlier and in the other condition we can just scan the leaf pages of index finding the pointers to record still the first entry so, we are doing more a sequential one. So, in either case retrieving records that are pointed to requires I/O for each record because they are on a secondary index. So, they are not necessarily consecutive and residing on the on the same block. So, they may be all distributed across different blocks and in such cases it may turn out that actually is doing a simple linear scan may turn out to be cheaper.

(Refer Slide Time: 20:59)



Implementation of Complex Selections

- **Conjunction:** $\sigma_{\theta_1 \wedge \theta_2 \wedge \dots \wedge \theta_n}(r)$
- **A7 (conjunctive selection using one index)**
 - Select a combination of θ_i and algorithms A1 through A6 that results in the least cost for $\sigma_{\theta_i}(r)$
 - Test other conditions on tuple after fetching it into memory buffer
- **A8 (conjunctive selection using composite index)**
 - Use appropriate composite (multiple-key) index if available
- **A9 (conjunctive selection by intersection of identifiers)**
 - Requires indices with record pointers
 - Use corresponding index for each condition, and take intersection of all the obtained sets of record pointers
 - Then fetch records from file
 - If some conditions do not have appropriate indices, apply test in memory

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Khargpur, Jan-Apr, 2018

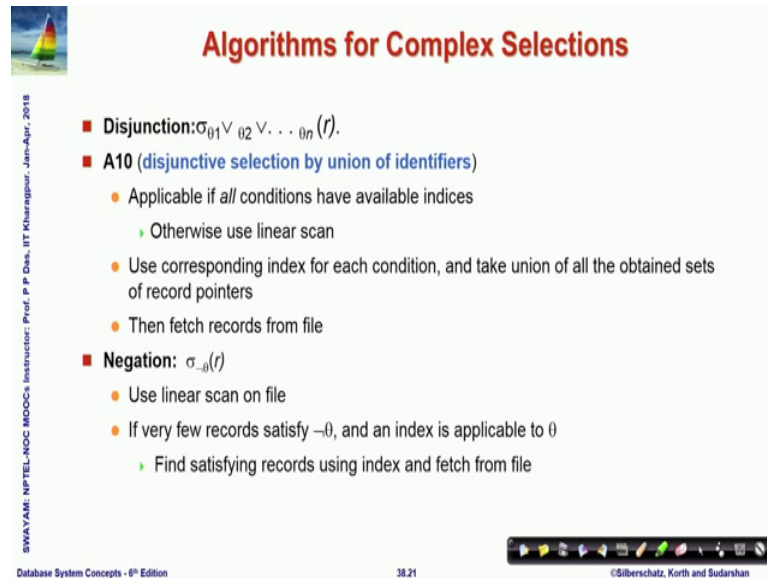
Database System Concepts - 6th Edition 38.20 ©Silberschatz, Korth and Sudarshan

Often we have select conditions which are conjunction. So, it could be conjunctive select and may be using only one index in that case it depends on if there are n conditions then we will need to depending on the combination of this condition theta n and the algorithms that we have seen here we can evaluate as to which strategy will give that least cost for this condition.

So, we will do the access based on that and then once we have accessed that tuple then we will try out the other conditions on the tuples that have been fetched into the memory buffer. You can also do conjunctive selection using composite index we can there are depending on the attributes involved in theta 1, theta 2, theta n we may have a multi key index and that decision of course, as to whether I have a multi key index or what is that multi key index is of course, dependent on the earlier statistics.

But if we have some multi key index which are appropriate composite index then we can use that and more directly get the result which will be more efficient. Or we can do conjunctive selection by intersection of identifiers which require indices with record pointers and will use corresponding index for each condition and then fetch the records which is simple to understand.

(Refer Slide Time: 22:29)



Algorithms for Complex Selections

- **Disjunction:** $\sigma_{\theta_1 \vee \theta_2 \vee \dots \vee \theta_n}(r)$.
- **A10 (disjunctive selection by union of identifiers)**
 - Applicable if *all* conditions have available indices
 - Otherwise use linear scan
 - Use corresponding index for each condition, and take union of all the obtained sets of record pointers
 - Then fetch records from file
- **Negation:** $\sigma_{\neg\theta}(r)$
 - Use linear scan on file
 - If very few records satisfy $\neg\theta$, and an index is applicable to θ
 - Find satisfying records using index and fetch from file

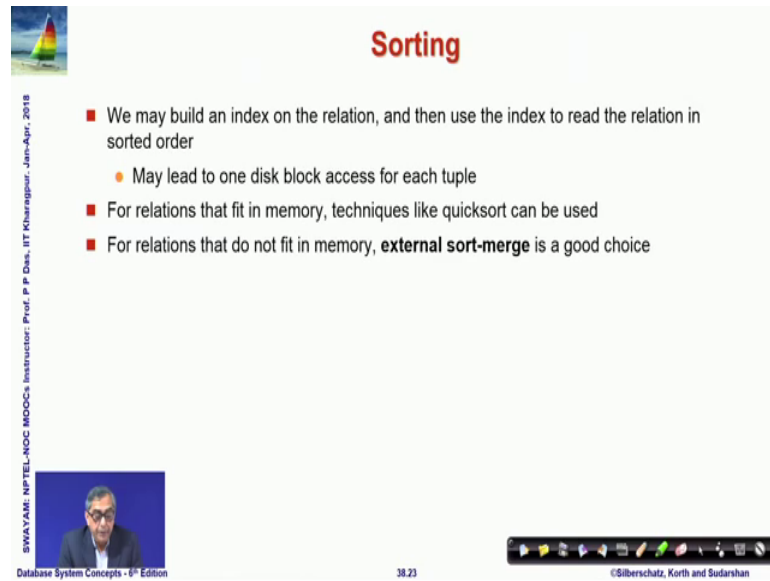
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P.P. Das, IIT Khargpur, Jan-April, 2018

Database System Concepts - 6th Edition 38.21 ©Silberschatz, Korth and Sudarshan

Disjunction this that was conjunction if we want to do disjunction then if we have all conditions all of these conditions have index on them if it is available index then we can do something better. Otherwise if we do not have that then it is better to do a linear scan because the conditions all triples which satisfies theta 1 will be there in the result, those which satisfy theta 2 may or may not satisfy the others will also be there and so, on.

So, what we can do is if we have index on each one of these based on each one of these conditions then they can use corresponding index for each condition get the results and take their union and then fetch these records. So, these are some of the so, I i just gave you a quick outline in terms of some of the different algorithms that selection could use. Negation of a condition could also be done, but it usually requires a linear scan on the file that is there is not much optimization that you can think of here. The next operation which is often required may not be explicitly, but in terms of doing other operations is sorting.

(Refer Slide Time: 23:47)



Sorting

- We may build an index on the relation, and then use the index to read the relation in sorted order
 - May lead to one disk block access for each tuple
- For relations that fit in memory, techniques like quicksort can be used
- For relations that do not fit in memory, **external sort-merge** is a good choice

SWAYAM: NPTEL-NOC MOCs Instructor: Prof. P. P. Das, IIT Khargpur, Jan-Apr, 2018

Database System Concepts - 9th Edition

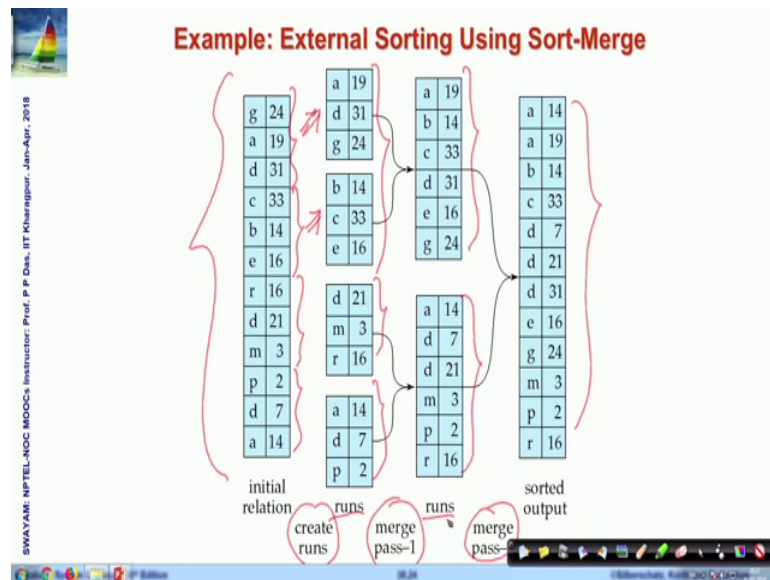
38/23

©Silberschatz, Korth and Sudarshan

So, if we may build an index on the relation then we can use that index to read the relation in sorted order, this is what we have already discussed that b plus tree in the in order traversal will always give you the sorted order. So, that may lead to one disk access for each tuple at times. Now that if the relation can totally if all the records can totally fit into the memory then we can use some in memory algorithm like quicksort, but often that will not be the case relations are much bigger.

So, what will have to do is will have to take recourse to external sort and merge strategy which is a very old strategy, but very effective.

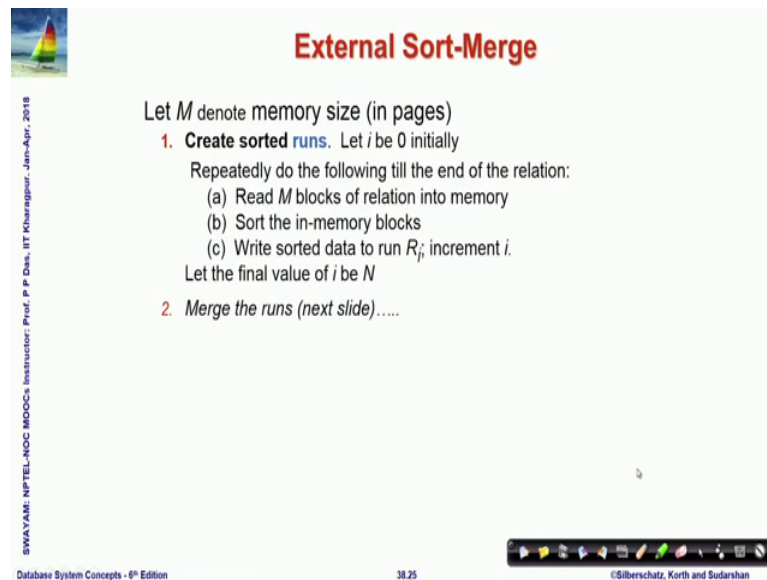
(Refer Slide Time: 24:30)



So, just to illustrate that suppose sorry so, suppose these are this is the initial relation so, what we do certainly we cannot read that whole relation in terms of memory into a memory. So, what we do we take different parts and say we are taking in groups of 3 just for illustration and make them and sort them in memory. So, take them so, take that money records which you can fit into the memory and sort them.

So, once you have sorted them then you can these are 2 sorted sub lists of the original set of records. So, now, you can merge them according to the merge strategy so, this is the sample merge saw strategy and again you write this back you do the similar things again here write them back. So, now, you have 2 bigger short sorted lists so, so these are called runs. So, the first step creates the runs and now after you have done merging once you get longer runs then again you merge them into a bigger run and depending on the on the actual size of the file and the size of the memory that directly fits in you might be doing multiple such runs till you get to the sorted output.

(Refer Slide Time: 25:52)



External Sort-Merge

Let M denote memory size (in pages)

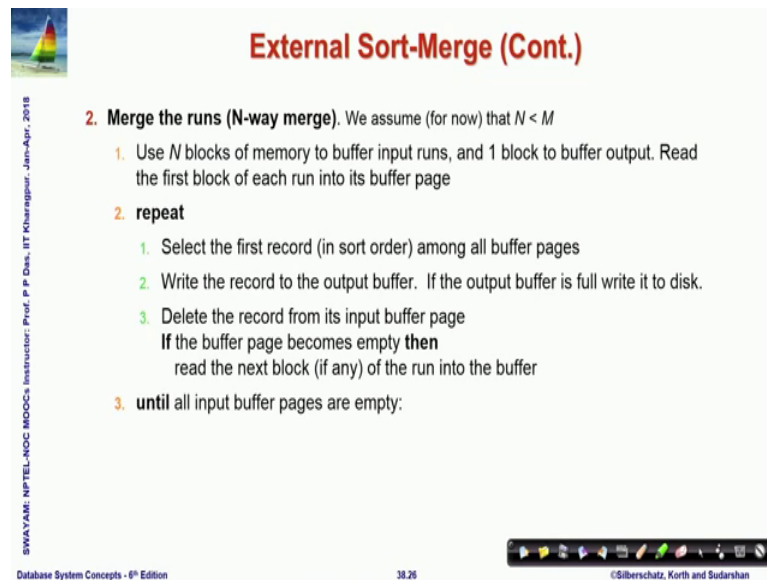
- 1. Create sorted runs.** Let i be 0 initially
Repeatedly do the following till the end of the relation:
 - (a) Read M blocks of relation into memory
 - (b) Sort the in-memory blocks
 - (c) Write sorted data to run R_i ; increment i .Let the final value of i be N
- 2. Merge the runs (next slide).....**

SWAYAM: NPTEL-NOC MOOC's Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 6th Edition 38.25 ©Silberschatz, Korth and Sudarshan

So, that is a very external sort merge is a very effective strategy that the databases will always use and the efficiency of that or the cost of that depends on the size of the memory in terms of pages as to what can fit a complete one run data. So, this is whatever I have described is simply given here in steps of algorithm.

(Refer Slide Time: 26:16)



External Sort-Merge (Cont.)

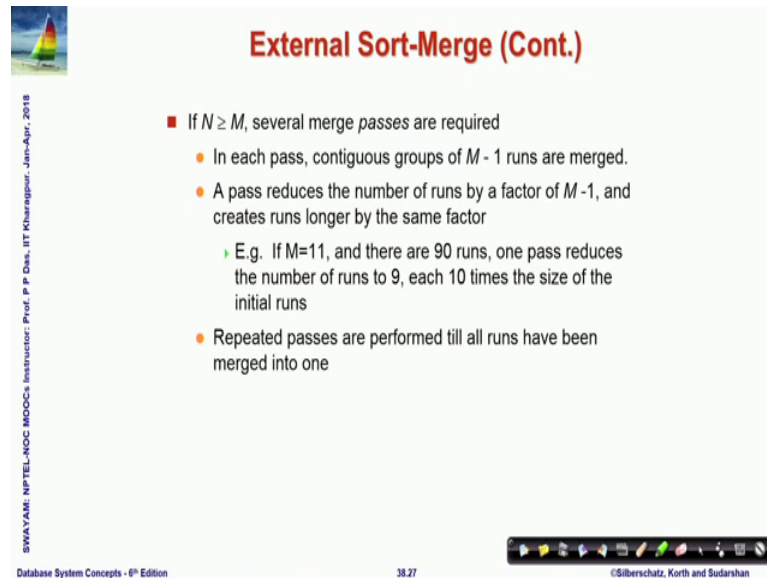
- 2. Merge the runs (N-way merge).** We assume (for now) that $N < M$
 - 1. Use N blocks of memory to buffer input runs, and 1 block to buffer output.** Read the first block of each run into its buffer page
 - 2. repeat**
 - 1. Select the first record (in sort order) among all buffer pages**
 - 2. Write the record to the output buffer.** If the output buffer is full write it to disk.
 - 3. Delete the record from its input buffer page**
If the buffer page becomes empty then
read the next block (if any) of the run into the buffer
 - 3. until all input buffer pages are empty:**

SWAYAM: NPTEL-NOC MOOC's Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 6th Edition 38.26 ©Silberschatz, Korth and Sudarshan

So, that is a sort that is that here is a merge so, you can go through that and convince yourself that this is what algorithm is actually doing.

(Refer Slide Time: 26:24)



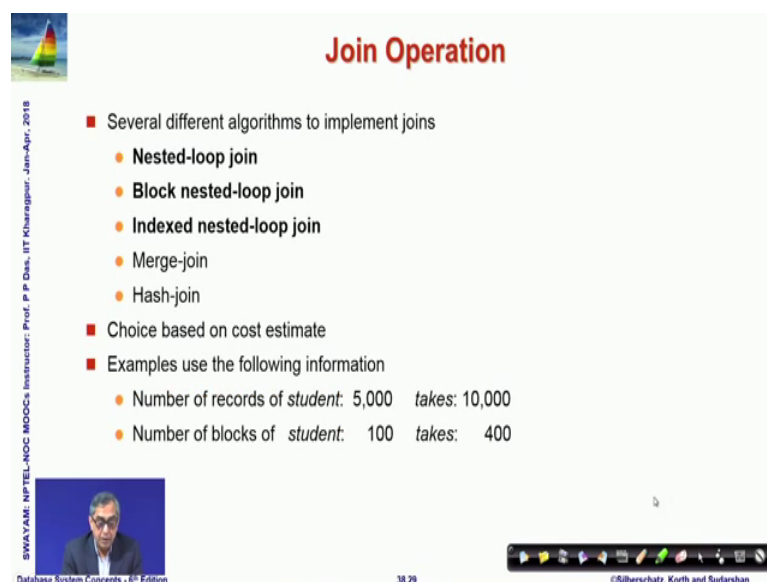
External Sort-Merge (Cont.)

- If $N \geq M$, several merge passes are required
 - In each pass, contiguous groups of $M - 1$ runs are merged.
 - A pass reduces the number of runs by a factor of $M - 1$, and creates runs longer by the same factor
 - ▶ E.g. If $M=11$, and there are 90 runs, one pass reduces the number of runs to 9, each 10 times the size of the initial runs
 - Repeated passes are performed till all runs have been merged into one

Database System Concepts - 9th Edition 38.27 ©Silberschatz, Korth and Sudarshan

And there are 2 cases you have to consider whether your data fits into the memory otherwise if your it does not fit into the memory then multiple passes are required and these are the steps of the algorithm that will be followed. Now next to sorting certainly we have often talked about that join is a very required operation in relational database in terms of SQL.

(Refer Slide Time: 26:56)



Join Operation

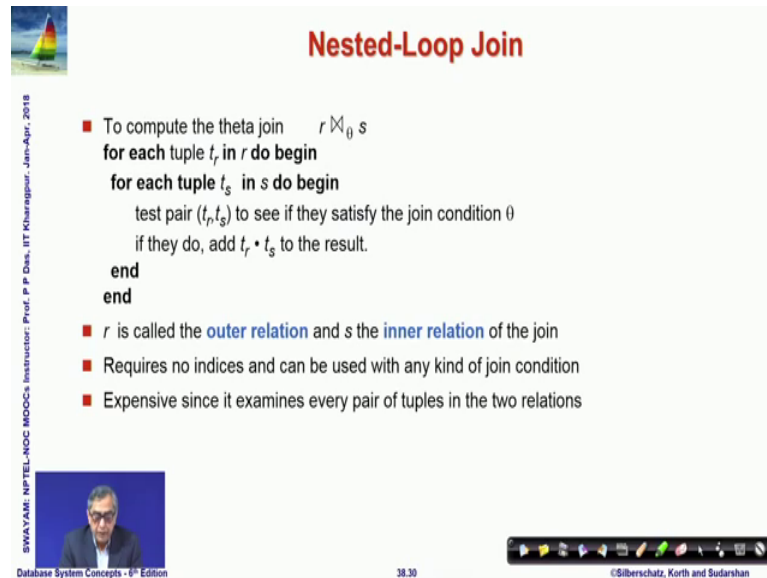
- Several different algorithms to implement joins
 - Nested-loop join
 - Block nested-loop join
 - Indexed nested-loop join
 - Merge-join
 - Hash-join
- Choice based on cost estimate
- Examples use the following information
 - Number of records of *student*: 5,000 takes: 10,000
 - Number of blocks of *student*: 100 takes: 400

Database System Concepts - 9th Edition 38.29 ©Silberschatz, Korth and Sudarshan

So, let us see what will it take to do a join so, first we talk about so, the join could be done in several ways nested loop join, block nested loop join, indexed nested loop join,

merge join, hash join. So, these are different strategies of doing join we will just illustrate the algorithms for the first 3 strategies.

(Refer Slide Time: 27:15)



Nested-Loop Join

- To compute the theta join $r \bowtie_{\theta} s$
for each tuple t_r in r do begin
 for each tuple t_s in s do begin
 test pair (t_r, t_s) to see if they satisfy the join condition θ
 if they do, add $t_r \cdot t_s$ to the result.
 end
end
- r is called the **outer relation** and s the **inner relation** of the join
- Requires no indices and can be used with any kind of join condition
- Expensive since it examines every pair of tuples in the two relations

SWAYAM: NPTEL-NOC MOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 9th Edition

38.30

©Silberschatz, Korth and Sudarshan

So, nested loop join what we are trying to do is very simple we have 2 relations we have 2 relations here r and s and we have a condition theta and we are doing a theta join. So, what needs to be done in terms of theta join in the relational algebra what do we do we do a Cartesian product and then in the Cartesian product we check out this theta condition.

So, the basic Cartesian product is all records of r will have to be matched will have to be connected to all record of s . So, that naturally can be done using a nested for loop so, for each tuple in our you try out each tuple n s take the t_r, t_s pair and if they satisfy the condition theta then they go to the output otherwise you leave that otherwise you discard that and here we say r is the outer relation and this s is the inner relation. So, naturally since you have to examine every pair this could be quite expensive to perform and the cost may be quite high.

(Refer Slide Time: 28:19)

Nested-Loop Join (Cont.)

- In the worst case, if there is enough memory only to hold one block of each relation, the estimated cost is $n_r * b_s + b_r$ block transfers, plus $n_r + b_r$ seeks
- If the smaller relation fits entirely in memory, use that as the inner relation.
 - Reduces cost to $b_r + b_s$ block transfers and 2 seeks
- Example of join of *students* and *takes*:
 - Number of records of *student*: 5,000 *takes*: 10,000
 - Number of blocks of *student*: 100 *takes*: 400
- Assuming worst case memory availability cost estimate is
 - with *student* as outer relation:
 - ▶ $5000 * 400 + 100 = 2,000,100$ block transfers,
 - ▶ $5000 + 100 = 5100$ seeks
 - with *takes* as the outer relation
 - ▶ $10000 * 100 + 400 = 1,000,400$ block transfers and 10,400 seeks
- If smaller relation (*student*) fits entirely in memory, the cost estimate will be 500 block transfers
- Block nested-loops algorithm (next slide) is preferable

SWAYAM: NPTEL-NOC MDOCS Instructor: Prof. P. P. Das, IIT Khargpur, Jan-April, 2018
Database System Concepts - 8th Edition 38.31 ©Silberschatz, Korth and Sudarshan

So, if we look at what could be the possible cost. So, if n_r is the number of records in relation r and b_r is the number of blocks in which they exist then for every record you have to actually access all the blocks of the other every for every tuple of relation r you have to actually access all the blocks of relation s . So, you get this and you have to access all the blocks of relation r .

So, this is the kind of block transfer that you will get you will require and naturally you will require so many seek because every time you have to find out you have to go and seek for that. So, one optimization that is very common is what you can do is if the smaller relation can entirely fit into the memory then you do not need to do this repeated read for that both the relations.

So, if it fits into that then the cost will significantly reduce to b_r plus b_s block transfers because you want the smaller one has already fit. So, you just need to access one relation once you need to read the smaller relation and put it in the memory and then you just need to read the other relation one after the other. So, b_r blocks of that and so, you are seeking only twice one for reading r , one for reading s there is a 2 seeks.

So, here I have just shown a simple example of computing the join of *student* and *takes* let us say *student* has 5000 records and spread over 100 blocks *takes* relation has 10000 records spread over 400 blocks then if you apply the formula above you will find that if *student* is the outer relation you have so, many block transfer and so, many seeks.

Whereas, if takes is the outer relation then you have so many block transfers and so many seeks.

So, you can understand you can see here that if you make student as a outer relation then you have much larger number of block transfers though you need to do less number of seek, but taking, but using takes as a outer relation you have much less block transfers, but more number of seek usually seek is less expensive than less costly than the block transfer.

So, will possibly in with this kind of a statistics if it is available then we will possibly take takes as outer relation and student as the inner one. You can refine this.

(Refer Slide Time: 31:07)

Block Nested-Loop Join

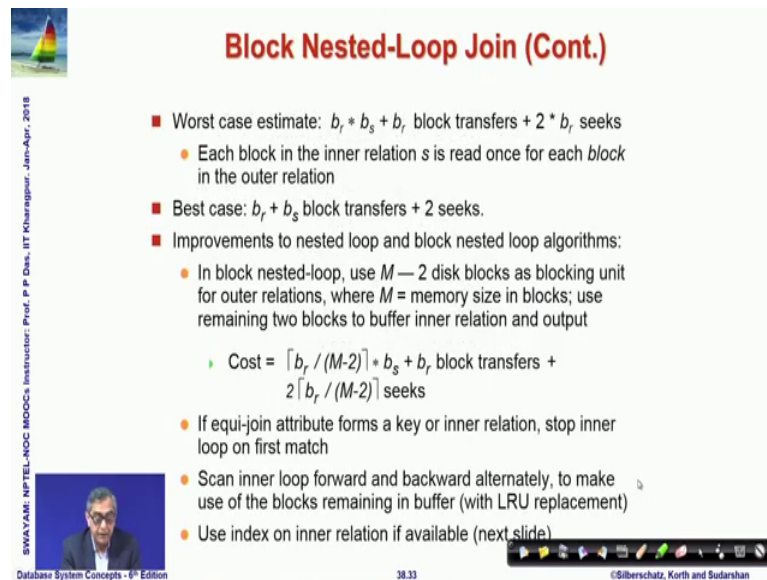
- Variant of nested-loop join in which every block of inner relation is paired with every block of outer relation

```
for each block  $B_r$  of  $r$  do begin
  for each block  $B_s$  of  $s$  do begin
    for each tuple  $t_r$  in  $B_r$  do begin
      for each tuple  $t_s$  in  $B_s$  do begin
        Check if  $(t_r, t_s)$  satisfy the join condition
        if they do, add  $t_r \cdot t_s$  to the result.
      end
    end
  end
end
```

Database System Concepts - 8th Edition 38.32 ©Silberschatz, Korth and Sudarshan

Strategy by doing a block nesting that is instead of taking every tuple of the relation you can take every block of the relation. So, for every block of relation r you try to match with you try to combine with every block of relation s and then within every block of relation r the block b r you take tuple and within b s you take t S and then you do whatever we are doing earlier, but naturally you get a much better performance.

(Refer Slide Time: 31:46)



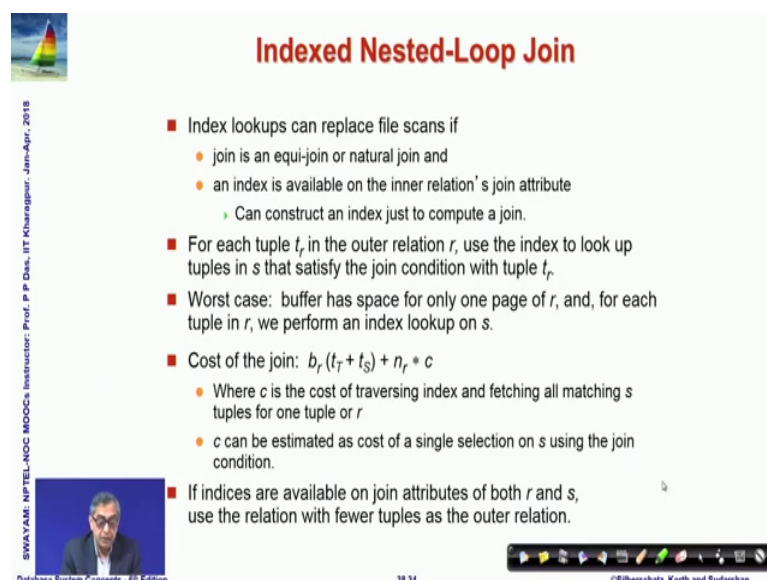
Block Nested-Loop Join (Cont.)

- Worst case estimate: $b_r * b_s + b_r$ block transfers + $2 * b_r$ seeks
 - Each block in the inner relation s is read once for each *block* in the outer relation
- Best case: $b_r + b_s$ block transfers + 2 seeks.
- Improvements to nested loop and block nested loop algorithms:
 - In block nested-loop, use $M - 2$ disk blocks as blocking unit for outer relations, where M = memory size in blocks; use remaining two blocks to buffer inner relation and output
 - Cost = $\lceil b_r / (M-2) \rceil * b_s + b_r$ block transfers + $2 \lceil b_r / (M-2) \rceil$ seeks
 - If equi-join attribute forms a key or inner relation, stop inner loop on first match
 - Scan inner loop forward and backward alternately, to make use of the blocks remaining in buffer (with LRU replacement)
 - Use index on inner relation if available (next slide)

SWAYAM: NPTEL-NOC MDOCS Instructor: Prof. P. P. Das, IIT Khairagpur, Jan-Apr, 2018
Database System Concepts - 9th Edition
38.33
©Silberschatz, Korth and Sudarshan

Because you are now optimizing based on the block reads only you are not reading every tuple every time you need. So, I will not go through this you know simple algebra to show that if you have a memory size of M blocks that your cost will significantly decrease and, but the larger the M your cost will come down by a factor of this M . So, block nested loop join will usually be far more efficient than the simple nested loop join.

(Refer Slide Time: 32:14)



Indexed Nested-Loop Join

- Index lookups can replace file scans if
 - join is an equi-join or natural join and
 - an index is available on the inner relation's join attribute
 - Can construct an index just to compute a join.
- For each tuple t_r in the outer relation r , use the index to look up tuples in s that satisfy the join condition with tuple t_r .
- Worst case: buffer has space for only one page of r , and, for each tuple in r , we perform an index lookup on s .
- Cost of the join: $b_r * (t_r + t_s) + n_r * c$
 - Where c is the cost of traversing index and fetching all matching s tuples for one tuple or r
 - c can be estimated as cost of a single selection on s using the join condition.
- If indices are available on join attributes of both r and s , use the relation with fewer tuples as the outer relation.

SWAYAM: NPTEL-NOC MDOCS Instructor: Prof. P. P. Das, IIT Khairagpur, Jan-Apr, 2018
Database System Concepts - 9th Edition
38.34
©Silberschatz, Korth and Sudarshan

The third strategy which we will use very often is efficiently applicable if you are if your join is an equijoin or a natural join as we have seen that we often need to do a natural

join. So, there are 2 attributes between these 2 relations on which during join the values that they match are retained, the values that they do not match are not retained in the natural join.

So, if we now assume that we have an index available on the inner relation then every time we go with the outer relation will be able to access the inner relation very efficiently because for each tuple in the outer relation the index to look up the tuples in nests will satisfy the condition will be found very efficiently because they are index. So, they will occur if through the index I can find them in terms of the consecutivity.

So, there the cost in that case will turn out to be very simply the cost of the b_r which is the outer relation the number of blocks in the outer relation the seek and transfer cost of that and then the number of record times, the estimated cost of a single selection using the join condition. So, we often use the nested loop join when we have to do equal join or natural join.

(Refer Slide Time: 33:44)

Example of Nested-Loop Join Costs

- Compute $student \bowtie takes$, with $student$ as the outer relation.
- Let $takes$ have a primary B⁺-tree index on the attribute ID , which contains 20 entries in each index node.
- Since $takes$ has 10,000 tuples, the height of the tree is 4, and one more access is needed to find the actual data
- $student$ has 5000 tuples
- Cost of block nested loops join
 - $400 * 100 + 100 = 40,100$ block transfers + $2 * 100 = 200$ seeks
 - ▶ assuming worst case memory
 - ▶ may be significantly less with more memory
- Cost of indexed nested loops join
 - $100 + 5000 * 5 = 25,100$ block transfers and seeks.
 - CPU cost likely to be less than that for block nested loops join

SWAYAM NPTEL-NOC MCOCA Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018
Database System Concepts - 6th Edition
38.35
©Silberschatz, Korth and Sudarshan

So, here is an example with the same students and takes example. So, it is shows that the cost of block nested join if you work out for the block nested join then you have so, many 40,100 block transfers and 200 seek. Whereas, if you do index to one on the assuming that the smaller relation the inner relation has a index then you have 25,000 block transfer and seek. So, this will turn out to be a naturally more efficient way of implementing the join.

So, these are there are several other strategies particularly hashing based strategies, merging based strategies which we are not discussing here, but there are different strategies through which you can do join in more and more efficient manner.

(Refer Slide Time: 34:36)

The slide is titled "Other Operations" in red text. It features a vertical sidebar on the left with a small sailboat icon at the top and a video thumbnail of a man at the bottom. The sidebar contains the text: "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018" and "Database System Concepts - 8th Edition". The main content area lists five operations with red square bullet points: Duplicate elimination, Projection, Aggregation, Set Operations, and Outer Join. At the bottom of the slide, there is a navigation bar with various icons and the text "©Silberschatz, Korth and Sudarshan".

- Duplicate elimination
- Projection
- Aggregation
- Set Operations
- Outer Join

Couple of other operations which are often required is duplicate elimination because if they we know that there are duplicate records cannot be kept, duplicate in the sense the records which match in the in the key field and the duplicate will often happen in terms of the result, they will happen in terms of when we do projection, we will need to do aggregation set operations outer join and so, on. So, the first 3 we will quickly out like.

(Refer Slide Time: 35:05)

Other Operations

- **Duplicate elimination** can be implemented via hashing or sorting
 - On sorting duplicates will come adjacent to each other, and all but one set of duplicates can be deleted
 - *Optimization*: duplicates can be deleted during run generation as well as at intermediate merge steps in external sort-merge
 - Hashing is similar – duplicates will come into the same bucket
- **Projection**:
 - perform projection on each tuple
 - followed by duplicate elimination

So, duplicate naturally can be very easily eliminated through sorting, they can be done through hashing also because if we sort they will come on side by they will come consecutively after the sort, if we hash then they will necessarily hash to the same value which becomes easier to check whether they are identical or not. If whenever we are doing projection we can project on each tuple and then you can perform a duplicate elimination to actually get to the final result. Aggregation group that is whatever you do group by kind of.

(Refer Slide Time: 35:37)

Other Operations : Aggregation

- **Aggregation** can be implemented in a manner similar to duplicate elimination
 - Sorting or hashing can be used to bring tuples in the same group together, and then the aggregate functions can be applied on each group
 - *Optimization*: combine tuples in the same group during run generation and intermediate merges, by computing partial aggregate values
 - ▶ For count, min, max, sum: keep aggregate values on tuples found so far in the group
 - When combining partial aggregate for count, add up the aggregates
 - ▶ For avg, keep sum and count, and divide sum by count at the end

So, aggregation is certainly will be efficiently done if you have again done sorting because if you are grouping by something then if you have sorted on that those elements will come together and or if you are hashing then they will also come together. So, you can easily do the computation on that.

And what you can do is instead of for example, you are doing a count or you are doing a minimum, maximum, sum this kind of all of these are associative operations. So, you can do it in parts that if you have done in this sorted order, in the hashed order if you have done the sum of 10 records then you can actually do not need these 10 records when you do the sum for the next 10 records and so, on. So, in this manner the aggregation can be efficiently implemented. So, we can for average keep sum and count and divide the sum by count at the end and so, on.

(Refer Slide Time: 36:39)

Module Summary

- Understood the overall flow for Query Processing and defined the Measures of Query Cost
- Studied the algorithms for processing Selection Operations, Sorting, Join Operations and a few Other Operations

SWAYAM: NPTEL-NOC MOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 9th Edition

38.40

©Silberschatz, Korth and Sudarshan

So, we have in this module we have just given a very brief outline of what is what are the steps involved in query processing and what are the measures that define a query cost typically and we have been talked about some of the simple algorithms for selection, sorting, join and aggregation operations.

In the next module we will talk about elementary optimization strategies for processing of queries.